

Sloman 83



# **SOFTWARE HANDBOOK**

**1984**



Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a) (9). Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, <sup>+</sup>i, ICE, iCS, iDBP, iDIS, i<sup>2</sup>ICE, iLBX, i<sub>m</sub>, iMMX, Insite, Intel, int<sub>e</sub>l, int<sub>e</sub>lBOS, Intelelevision, int<sub>e</sub>l<sub>i</sub>gent Identifier, int<sub>e</sub>l<sub>i</sub>gent Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, MICRO-MAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

The following are trademarks of the companies indicated and may only be used to identify products of the owners.

CP/M is a trademark of Digital Research, Inc.

DEC, DEC-10, DEC-20, PDP-11, DECnet, DECwriter, RSTS, and VAX are trademarks of Digital Equipment Corporation.

MDS is an ordering code only and is not used as a product name or trademark.

MDS® is a registered trademark of Mohawk Data Sciences Corporation.

Microsoft is a trademark of Microsoft, Inc.

## Table of Contents

### CHAPTER 1 OVERVIEW

Introduction .....	1-1
--------------------	-----

### CHAPTER 2

#### OPERATING SYSTEMS

Introduction .....	2-1
--------------------	-----

#### 8080/8085 Microprocessor Family

##### DATA SHEET

Digital Research Inc. CP/M 2.2 Operating System .....	2-2
---	-----

#### 8086/8088 Microprocessor Family

##### DATA SHEETS

iRMX 86 Operating System .....	2-5
--------------------------------	-----

iRMX 88 Real-Time Multitasking Executive .....	2-25
--	------

Preconfigured iRMX 86 Operating System .....	2-31
--	------

iOSP 86 iAPX 86/30 and iAPX 88/30 Support Package .....	2-37
---	------

iMMX 800 MULTIBUS Message Exchange Software .....	2-41
---	------

##### FACT SHEET

XENIX 286 Operating Systems .....	2-45
-----------------------------------	------

##### APPLICATION NOTE

AP-130 Using Operating Systems Processor's to Simplify Microcomputer Designs ....	2-51
---	------

##### ARTICLE REPRINTS

AR-236 Let Operating Systems Aid in Component Designs .....	2-102
---	-------

AR-286 Software That Resides in Silicon .....	2-110
---	-------

AR-287 Putting Real-Time Operating Systems to Work .....	2-116
--	-------

AR-288 Intel's Matchmaking Strategy: Marry iRMX Operating System with Hardware .....	2-128
--	-------

AR-289 iRMX 86 Has Functionality, Configurability .....	2-131
---	-------

### CHAPTER 3

#### TRANSLATORS AND UTILITIES FOR PROGRAM DEVELOPMENT

Introduction .....	3-1
--------------------	-----

#### MCS<sup>®</sup>-80/85 Microprocessor Family

##### DATA SHEETS

PL/M 80 High Level Programming Language .....	3-3
---	-----

FORTRAN 80 8080/8085 ANS FORTRAN 77 Intellec Resident Compiler .....	3-6
--	-----

Microsoft, Inc. MACRO-80 Utility Software Package .....	3-10
---	------

Microsoft, Inc. BASIC-80 Interpreter Software Package .....	3-12
---	------

Microsoft, Inc. Pascal-80 Software Package .....	3-15
--	------

iAPX 86, 88 Software Development Packages for Series II/PDS .....	3-18
---	------

#### iAPX 86/88/186/188/286 Microprocessor Family

##### DATA SHEETS

PL/M 86/88/186/188 Software Package .....	3-28
---	------

Pascal 86/88 Software Package .....	3-32
-------------------------------------	------

FORTRAN 86/88 Software Package .....	3-35
--------------------------------------	------

C-86 C Compiler for the 8086 .....	3-39
------------------------------------	------

8087 Software Support Package .....	3-43
-------------------------------------	------

8087 Support Library .....	3-46
----------------------------	------

8089 IOP Software Support Package .....	3-50
---	------

iAPX 286 Software Development Package .....	3-53
---	------

iAPX 286 Evaluation Package .....	3-58
-----------------------------------	------

PL/M 286 Software Package .....	3-60
---------------------------------	------

VAX/VMX Resident iAPX 86/88/186 Software Development Packages .....	3-64
---	------

iSDM 86 System Debug Monitor .....	3-71
------------------------------------	------

iSDM 286 iAPX 286 System Debug Monitor .....	3-76
--	------

##### FACT SHEETS

iRMX Languages .....	3-79
----------------------	------

iRMX Operating Systems .....	3-84
------------------------------	------

XENIX Languages .....	3-90
-----------------------	------



**Single Chip Microcontroller Software**

**DATA SHEETS**

2920 Software Support Package .....	3-94
MCS-48 Diskette-Based Software Support Package .....	3-105
8051 Software Development Package .....	3-107
PL/M 51 Software .....	3-110
MCS-96 Software Support Package .....	3-114

**CHAPTER 4**

**PRODUCTIVITY TOOLS AND COMMUNICATION SOFTWARE**

Introduction .....	4-1
--------------------	-----

**Program Development and Management Tools**

**DATA SHEETS**

PSCOPE High-Level Program Debugger .....	4-2
Program Management Tools .....	4-7
ISIS-II Software Toolbox .....	4-10
8086 Software Toolbox .....	4-12
AEDIT Text Editor .....	4-14
CREDIT CRT-Based Text Editor .....	4-16

**Communication Software**

**DATA SHEETS**

Mainframe Link for Distributed Development .....	4-20
Intel Asynchronous Communications Link .....	4-23
iNA 960 Network Software .....	4-26
NDS-II Electronic Mail .....	4-38

**CHAPTER 5**

**SYSTEM AND APPLICATIONS SOFTWARE**

**FACT SHEETS**

XENIX Productivity Software Tools .....	5-1
iTPS Transaction Processing Systems Terminal Application Processing System (iTAPS) .....	5-9
iTPS Transaction Processing Systems Communications .....	5-12
System 2000 Database Management System Sperry (Univac) 1100 Series .....	5-16

**CHAPTER 6**

**COMPONENT SOFTWARE**

**DATA SHEETS**

80130/80130-2 iAPX 86/30, 88/30, 186/30, 188/30 iRMX 86 Operating System Processors .....	6-1
80150/80150-2 iAPX 86/50, iAPX 88/50, 186/50, 188/50 CP/M 86 Operating System Processors .....	6-23

**CHAPTER 7**

**USER LIBRARY**

Introduction .....	7-1
--------------------	-----

**User Library**

Insite User's Program Library .....	7-2
Insite Submittal Requirements .....	7-3
Insite Index of Program .....	7-5

**APPENDIX A**

Software Standards .....	A-1
--------------------------	-----

**APPENDIX B**

Software Support .....	B-1
------------------------	-----

---

# Software Handbook Overview

---







## **SOFTWARE HANDBOOK OVERVIEW**

Welcome to the Intel Software Handbook. This handbook is a complete guide to the software products and services offered by Intel.

Intel's software products follow the open systems strategy that allows Intel products to be purchased at the customers' desired level of integration. Hence these products are available for component, board, or systems applications. This open systems philosophy is backed by software standards that insure that the software can operate at numerous levels of integration. These software standards are described in the appendix.

Software for Intel's products is available both from Intel and from Independent Software Vendors (ISVs). For a complete listing of software available from ISVs, see the Intel Yellow Pages which is published annually by Intel. This handbook describes software products that are available through Intel, consisting of Intel-developed and ISV-developed products. Products that are offered by Intel have all been evaluated and tested to meet Intel's quality standard. They are backed by an extensive support organization described in the appendix.





---

# Operating Systems

2

---





# OPERATING SYSTEMS

## INTRODUCTION

The ability to convert advanced microprocessor technology into solutions for modern day problems begins with effective and efficient designs for new hardware products and architecture. However, a most critical element in the success of any microcomputer solution is the availability of a high quality, reliable operating system. Without this software counterpart, the technological advances cannot be fully implemented, nor their benefits fully realized.

The classic role of the microcomputer operating system can be outlined by viewing its major functions and purposes. The functions of the microcomputer operating system are threefold: 1) to manage system resources and the allocation of these resources to users; 2) to provide automatic functions such as initialization and start-up procedures; and 3) to provide an efficient, straightforward and consistent method for user programs to interface with the hardware subsystems, including a simple and friendly human interface. Typically, the operating systems have one of two main purposes. First, they can be used to develop a new software system that runs on another machine. These systems are usually large and fairly sophisticated. ISIS and \*XENIX are examples of such developmental operating systems. The second purpose for microcomputer operating systems is directed toward the execution of software programs for targeted application. The largest number of operating systems are of this type, including the RMX systems. The most critical requirement is for these systems to be effective and efficient since they are usually small, fast systems dedicated to a specific real-time application.

This rather neat and simple categorization of microcomputer operating systems, which has been useful in the past, is quickly becoming blurred. The rapid developments in microcomputer technology have increased the power and decreased the cost of microcomputers, allowing them to become applicable to the solution of a broader variety and more sophisticated set of problems. Microcomputer systems must increasingly provide such capabilities as multiprogramming, multitasking, multiprocessing, networking, as well as scheduling and priority determination. As systems become more complex, they must still remain responsive to real-time applications. Operating systems must be able to capitalize on the trends toward placing more and more software into silicon. This trend is blurring the distinction between the hardware and software subsystems. Microcomputer systems are also evolving to encompass both the developmental and target application purposes into one system.

These dramatic changes in technology place additional demands on operating systems. We see operating systems undergoing changes to consider the need for: 1) modularity and ease of configurability; 2) evolutionary, not revolutionary, path of growth; and 3) standardization in languages, networks and the operating system itself. The first need is required to allow the system to be a powerful development tool yet configurable to more specialized applications. The last two items are needed to provide protection of a firm's software investment, including the option to move toward silicon software.

The operating systems and executives in this section are state-of-the-art microcomputer systems that have taken to task the challenges posed by advancing microprocessor technology. These operating systems offer the widest range of solutions with the highest quality and most future-oriented software available today. Consequently, our customers can select the appropriately optimized option to achieve their price/performance goals and give them time-to-market advantage over their competitors.

\*XENIX is a trademark of Microsoft Corp.



## DIGITAL RESEARCH INC. CP/M\* 2.2 OPERATING SYSTEM

- High-performance, single-console operating system
- Simple, reliable file system matched to microcomputer resources
- Table-driven architecture allows field reconfiguration to match a wide variety of disk capacities and needs
- Extensive documentation covers all facts of CP/M applications
- More than 1,000 commercially available compatible software products
- General-purpose subroutines and table-driven data-access algorithms provide a truly universal data management system
- Upward compatibility from all previous versions

CP/M 2.2 is a monitor control program for microcomputer system and application uses on Intel 8080/8085-based microcomputer (see the *CP/M-86 \* Operating System* data sheet for information on CP/M for Intel 8086/8088-based systems). CP/M provides a general environment for program execution, construction, storage, and editing, along with the program assembly and check-out facilities.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this system, a large number of distinct programs can be stored in both source- and machine-executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Nearly all personal software programs can be bought configured to run under CP/M, several of which are available from Intel.

### FEATURES

CP/M is logically divided into four distinct modules:

#### BIOS—Basic I/O System

- Provides primitive operations for access to disk drives and interface to standard peripherals (teletype, CRT, paper tape reader/punch, bubble memory, and user-defined peripherals)
- Allows user modification for tailoring to a particular hardware environment

#### BDOS—Basic Disk Operating System

- Provides disk management for one to sixteen disk drives containing independent file directories
- Implements disk allocation strategies for fully dynamic file construction and minimization of head movement across the disk

—Uses less than 4K of memory allowing plenty of memory space for applications programs

—Uses less than 4K of memory

—Makes programs transportable from system to system

—Entry points include the following primitive operations which can be programmatically accessed:

- |        |   |
|--------|---|
| SEARCH | Look for a particular disk file by name               |
| OPEN   | Open a file for further operations                    |
| CLOSE  | Close a file after processing                         |
| RENAME | Change the name of a particular file                  |
| READ   | Read a record from a particular file                  |
| WRITE  | Write a record to a particular file                   |
| SELECT | Select a particular disk drive for further operations |

### CCP—Console Command Processor

- Provides primary user interface by reading and interpreting commands entered through the console
- Loads and transfers control to transient programs, such as assemblers, editors, and debuggers
- Processes built-in standard commands including:
 

ERA	Erase specified files
DIR	List file names in the directory
REN	Rename the specified file
SAVE	Save memory contents in a file
TYPE	Display the contents of a file on the console

### TPA—Transient Program Area

- Holds programs which are loaded from the disk under command of the CCP
- Programs created under CP/M can be checked out by loading and executing these programs in the TPA
- User programs, loaded into the TPA, may use the CCP area for the program's data area
- Transient commands are specified in the same manner as built-in commands
- Additional commands can be easily defined by the user
- Defined transient commands include:
 

PIP	Peripheral Interchange Program —implements the basic media transfer operations necessary to load, print, punch, copy, and combine disk files; PIP also performs various reformatting and concatenation functions. Formatting options include parity-bit removal, case conversion, Intel hex file validation, subfile extraction, tab expansion, line number generation, and pagination
ED	Text Editor—allows creation and modification of ASCII files using extensive context editing commands: string substitution, string search, insert, delete and block move; ED allows text to be located by context, line number, or relative position with a macro command for making extensive text changes with a single command line

- |        |  |
|--------|--|
| ASM    | Fast 8080 Assembler—uses standard Intel mnemonics and pseudo operations with free-format input, and conditional assembly features  |
| DDT    | Dynamic Debugging Tool—contains an integral assembler/disassembler module that lets the user patch and display memory in either assembler mnemonic or hexadecimal form and trace program execution with full register and status display; instructions can be executed between breakpoints in real-time, or run fully monitored, one instruction at a time |
| SUBMIT | Allows a group of CP/M commands to be batched together and submitted to the operating system by a single command   |
| STAT   | Lists the number of bytes of storage remaining on the currently logged disks, provides statistical information about particular files, and displays or alters device assignments   |
| LOAD   | Converts Intel hex format to absolute binary, ready for direct load and execution in the CP/M environment  |
| SYSGEN | Creates new CP/M system disks for back-up purposes   |
| MOVCPM | Provides regeneration of CP/M systems for various memory configurations and works in conjunction with SYSGEN to provide additional copies of CP/M  |

### BENEFITS

- Easy implementation on any computer configuration which uses an Intel 8080/8085 Central Processing Unit (see the CP/M-86 data sheet for CP/M applications on the iAPX86 CPU)
- iPDS version supports bubble memory option as an additional diskette drive. Also allows diskette duplication with a single drive
- Extensive selection of CP/M-compatible programs allows production and support of a comprehensive software package at low cost
- Field programmability for special-purpose operating system requirements
- Upward compatibility from previous versions of CP/M release 1





- Provides field specification of one to sixteen logical drives, each containing up to eight megabytes
- Files may contain up to 65,536 records of 128 bytes each but may not exceed the size of any single disk
- Each disk is designed for 64 distinct files—more directory entries may be allocated if necessary
- Individual users are physically separated by user numbers, with facilities for file copy operations from one user area to another
- Relative-record random-access functions provide direct access to any of the 65,536 records of an eight-megabyte file

## SPECIFICATIONS

### Hardware Required

- Model 800 with 720 kit
- DS 235 kit or MDS 225 with 720 kit (integral drive supported except as system boot device)
- iPDS Personal Development System
  - Optional:
    - RAM up to 64K
    - Additional floppy disk drives
    - Single density via 201 controller
    - Bubble memory and optional Shugart 460 5¼" disk drive for iPDS

### Documentation Package

Title
CP/M 2.2 documentation consisting of 7 manuals:
An Introduction to CP/M Features and Facilities
CP/M 2.2 User's Guide
CP/M Assembler (ASM) User's Guide
CP/M Dynamic Debugging Tool (DDT) User's Guide
ED: A Context Editor for the CP/M Disk System User's Manual
CP/M 2 Interface Guide
CP/M 2 Alteration Guide

### Shipping Media

(Specify by Alpha Character when ordering.)

- A—single density (IBM 3740/1 compatible)
- B—double density
- F—double-sided, double density 5¼" floppy (iPDS format)

Order Code	Product Description
See Price List	CP/M (Control Program for Microcomputers) is a disk-based operating system for the Intel 8080/8085-based systems. CP/M provides a general environment for program development, test, execution and storage. CP/M storage is available via a comprehensive, named-file structure supporting both sequential and random access. CP/M support tools include a Text Editor, a debugger, and an 8080/8085 assembler.

## SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.

\*CP/M is a registered trademark of Digital Research, Inc.

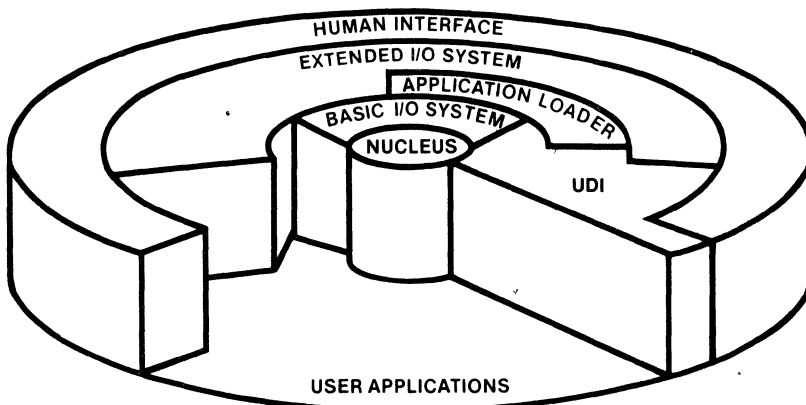
\*CP/M-86, MP/M, CP/NET and MP/NET are trademarks of Digital Research, Inc.



## iRMX™ 86 OPERATING SYSTEM

- Real-time processor management for time-critical iAPX 86 and iAPX 88 applications
- On-target system development with Universal Development Interface (UDI)
- Configurable system size and function for diverse application requirements
- All iRMX™ 86 code can be (P)ROM'ed to support totally solid state designs
- Compatible operating system services for iAPX 86/30 and 88/30 Operating System Processors (iOSP™ 86)
- Multi-terminal support with multi-user human interface
- Broad range of device drivers included for industry standard MULTIBUS® peripheral controllers
- Expandable to multi-processor systems with iMMX™ 800 Message Exchange Software
- Extendable to iAPX 286 systems with iRMX™ 286R option
- Powerful utilities for interactive configuration and real-time debugging

The iRMX™ 86 Operating System is an easy-to-use, real-time, multi-tasking and multi-programming software system designed to manage and extend the resources of iSBC® 86 and iSBC 88 Single Board Computers, as well as other iAPX 86- and iAPX 88-based microcomputers. iRMX 86 functions are available in silicon with the iAPX 86/30 and 88/30 Operating System Processors, in a user configurable software package, and fully integrated into the SYSTEM 86/300 Family of Microcomputer Systems. The Operating System provides a number of standard interfaces that allow iRMX 86 applications to take advantage of industry standard device controllers, hardware components, and a multitude of software packages developed by Independent Software Vendors (ISVs). Many high-performance features extend the utility of iRMX 86 Systems into applications such as data collection, transaction processing, and process control where immediate access to advances in VLSI technology is paramount. These systems may deliver real-time performance and explicit control over resources; yet also support applications with multiple users needing to simultaneously access terminals. The configurable layers of the System provide services ranging from interrupt management and standard device drivers for many sophisticated controllers, to data-file maintenance commands provided by a comprehensive multi-user human interface. By providing access to the standard Universal Development Interface (UDI) for each user terminal, Original Equipment Manufacturers (OEMs) can pass program development and target application customization capabilities to their users.



iRMX™ 86 VLSI Operating System

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iOSP, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

INTEL CORPORATION, 1983

The iRMX 86 Operating System is a complete set of system software modules that provide the resource management functions needed by computer systems. These management functions allow Original Equipment Manufacturers (OEMs) to best use resources available in microcomputer systems while getting their products to market quickly, saving time and money. Engineers are relieved of writing complex system software and can concentrate instead on their application software.

This data sheet describes the major features of the iRMX 86 Operating System. The benefits provided to engineers who write application software and to users who want to take advantage of improving microcomputer price and performance are explained. The first section outlines the system resource management functions of the Operating System and describes several system calls. The second section gives a detailed overview of iRMX 86 features aimed at serving both the iRMX 86 system designer and programmer, as well as the end users of the product into which the Operating System is incorporated.

## FUNCTIONAL DESCRIPTION

To take best advantage of iAPX 86 and 88 microprocessors in applications where the computer is required to perform many functions simultaneously, the iRMX 86 Operating System provides a multiprogramming environment in which many independent, multi-tasking application programs may run. The flexibility of independent environments allows application programmers to separately manage each application's resources during both the development and test phases.

The resource management functions of the iRMX 86 System are supported by a number of configurable software layers. While many of the functions supplied by the innermost layer, the Nucleus, are required by all systems, all other functions are optional. The I/O systems, for example, need not be included in systems having no secondary storage requirement. Each layer provides functions that encourage application programmers to use modular design techniques which aid in quick development of easily maintainable programs.

The components of the iRMX 86 Operating System provide both implicit and explicit management of system resources. These resources include processor scheduling, up to one megabyte of system memory, up to 57 independent interrupt sources, all input and output devices, as well as directory and data files contained on mass storage devices and accessed by a number of independent users. Management of each of these system resources and how the resources can be shared between multiple processors and users is discussed in the following sections.

## Process Management

To implement multi-tasking application systems, programmers require a method of managing the different processes of their application, and for allowing the processes to communicate with each other. The Nucleus layer of the iRMX 86 System provides a number of facilities to efficiently manage these processes, and to effectively communicate between them. These facilities are provided by system calls that manipulate data structures called tasks, jobs, semaphores, regions, and mailboxes. The iRMX 86 System refers to these structures as "objects".

**Tasks** are the basic element of all applications built on the iRMX 86 Operating System. Each task is an entity capable of executing CPU instructions and issuing system calls in order to perform a function. Tasks are characterized by their register values (including those of an optional 8087 Numeric Processor Extension), a priority between 0 and 255, and the resources associated with them.

Each iRMX 86 task in the system is scheduled for operation by the iRMX 86 nucleus. Figure 1 shows the five states in which each task may be placed, and some examples of how a task may move from one state to another. The iRMX 86 nucleus ensures that each task is placed in the correct state, defined by the events in its external environment and by the task issuing system calls. Each task has a priority to indicate its relative importance and need to respond to its environment. The nucleus guarantees that the highest priority ready-to-run task is the task that runs.

**Jobs** are used to define the operating environment of a group of tasks. Jobs effectively limit the scope of an application by collecting all of its tasks and other objects into one group. Because the environment for execution of an application is defined by an iRMX 86 job, separate applications can be efficiently developed by separate development teams.

The iRMX 86 Operating System provides two primary techniques for real-time event synchronization in multi-task applications: regions and semaphores.

**Regions** are used to restrict access to critical sections of code and data. Once the iRMX 86 Operating System gives a task access to resources guarded by a region, no other tasks may make use of the resources, and the task is given protection against deletion and suspension. Regions are typically used to protect data structures from being simultaneously updated by multiple tasks.

**Semaphores** are used to provide mutual exclusion between tasks. They contain abstract "units" that are sent between the tasks, and can be used to implement the cooperative sharing of resources.

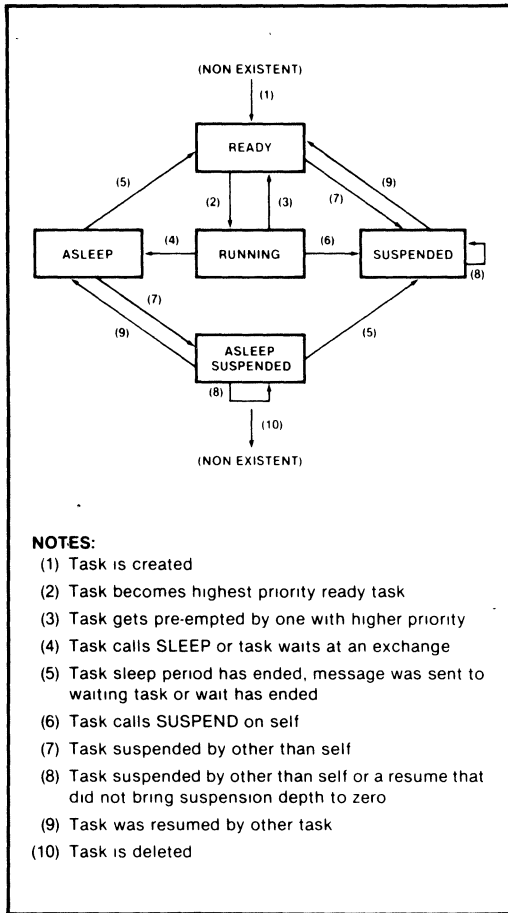


Figure 1. Task State Diagram

Multi-tasking applications must communicate information and share system resources among cooperating tasks. The iRMX 86 Operating System assigns a unique 16-bit number, called a token, to each object created in the System. Any task in possession of this token is able to access the object. The iRMX 86 Nucleus allows tasks to gain access to objects, and hence system resources, at run-time with two additional mechanisms: mailboxes and object directories.

**Mailboxes** are used by tasks wishing to share objects with other tasks. A task may share an object by sending the object's token via a mailbox. The receiving task can check to see if a token is there, or can wait at the mailbox until a token is present.

**Object Directories** are also used to make an object available to other tasks. An object is made public by cataloging its token and name in a directory. In this manner,

any task can gain access to the object by knowing its name, and job environment that contains the directory.

Three example jobs are shown in Figure 2 to demonstrate how two tasks can share an object that was not known to the programmers at the time the tasks were developed. Both Job 'A' and Job 'B' exist within the environment of the 'Root Job' that forms the foundation of all iRMX 86 systems. Each job possesses a directory in which tasks may catalog the name of an object. Semaphore 'RS', for example, is accessible by all tasks in the system, because its name is cataloged in the directory of the Root Job. Mailbox 'AN' can be used to transfer objects between Tasks 'A2' and 'A3' because its token is accessible in the object directory for Job 'A'

Table 1 lists the major functions of the iRMX 86 Nucleus that manage system processes.

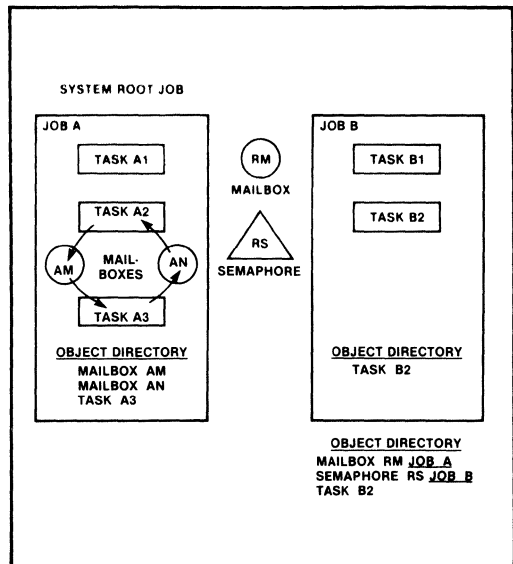


Figure 2. Object Directories

### Memory Management

Each job in an iRMX 86 System defines the amount of the one megabyte of addressable memory to be used by its tasks. The iRMX 86 Operating System manages system memory and allows jobs to share this critical resource by providing another object type: segments.

**Segments** are contiguous pieces of memory, between 16 Bytes and 64K Bytes in length, that exist within the environment of the job in which they were created. Segments form the fundamental piece of system memory used for task stacks, data storage, system buffers, loading programs from secondary storage, passing information between tasks, etc.

Table 1. Process Management System Calls

System Call	Function Performed
RQ\$CREATE\$JOB	Creates an environment for a number of tasks and other objects, as well as creating an initial task and its stack.
RQ\$DELETE\$JOB	Deletes a job and all the objects currently defined within its bounds. All memory used is returned to the job from which the deleted job was created.
RQ\$OFFSPRING	Provides a list of all the current jobs created by the specified job.
RQ\$CATALOG\$OBJECT	Enters a name and token for an object into the object directory of a job.
RQ\$UNCATALOG\$OBJECT	Removes an object's token and its name from a job's object directory.
RQ\$LOOKUP\$OBJECT	Returns a token for the object with the specified name found in the object directory of the specified job.
RQ\$GET\$TYPE	Returns a code for the type of object referred to by the specified token.
RQ\$CREATE\$MAILBOX	Creates a mailbox with queues for waiting tasks and objects with FIFO or PRIORITY discipline.
RQ\$DELETE\$MAILBOX	Deletes a mailbox.
RQ\$SEND\$MESSAGE	Sends an object to a specified mailbox. If a task is waiting, the object is passed to the appropriate task according to the queuing discipline. If no task is waiting, the object is queued at the mailbox.
RQ\$RECEIVE\$MESSAGE	Attempts to receive an object token from a specified mailbox. The calling task may choose to wait for a specified number of system time units if no token is available.
RQ\$DISABLE\$DELETION	Prevents the deletion of a specified object by increasing its disable count by one.
RQ\$ENABLE\$DELETION	Reduces the disable count of an object by one, and if zero, enables deletion of that object.
RQ\$FORCE\$DELETE	Forces the deletion of a specified object if the disable count is either 0 or 1.
RQ\$CREATE\$TASK	Creates a task with the specified priority and stack area.
RQ\$DELETE\$TASK	Deletes a task from the system, and removes it from any queues in which it may be waiting.
RQ\$SUSPEND\$TASK	Suspends the operation of a task. If the task is already suspended, its suspension depth is increased by one.
RQ\$RESUME\$TASK	Resumes a task. If the task had been suspended multiple times, the suspension depth is reduced by one, and it remains suspended.
RQ\$SLEEP	Causes a task to enter the ASLEEP state for a specified number of system time units.
RQ\$GET\$TASK\$TOKENS	Gets the token for the calling task or associated objects within its environment.
RQ\$SET\$PRIORITY	Dynamically alters the priority of the specified task.
RQ\$GET\$PRIORITY	Obtains the current priority of a specified task.
RQ\$CREATE\$REGION	Creates a region, with an associated queue of FIFO or PRIORITY ordering discipline.
RQ\$DELETE\$REGION	Deletes the specified region if it is not currently in use.
RQ\$ACCEPT\$CONTROL	Gains control of a region only if the region is immediately available.
RQ\$RECEIVE\$CONTROL	Gains control of a region. The calling task may specify the number of system time units it wishes to wait if the region is not immediately available.
RQ\$SEND\$CONTROL	Relinquishes control of a region.
RQ\$CREATE\$SEMAPHORE	Creates a semaphore.
RQ\$DELETE\$SEMAPHORE	Deletes a semaphore.
RQ\$SEND\$UNITS	Increases a semaphore counter by the specified number of units.
RQ\$RECEIVE\$UNITS	Attempts to gain a specified number of units from a semaphore. If the units are not immediately available, the calling task may choose to wait.

The example in Figure 2 also demonstrates when information is shared between Tasks 'A2' and 'A3'; 'A2' only needs to create a segment, put the information in the memory allocated, and send it via the Mailbox 'AM' using the RQ\$SEND\$MESSAGE system call (see Table 1). Task 'A3' would get the message by using the RQ\$RECEIVE\$MESSAGE system call. The Figure also shows how the receiving task could signal the sending task by sending an acknowledgement via the second Mailbox 'AN'.

Each job is created with both maximum and minimum limits set for its memory pool. Memory required by all objects and resources created in the job is taken from this pool. If more memory is required, a job may be allowed to borrow memory from the pool of its containing job (the job from which it was created). In this manner, initial jobs may efficiently allocate memory to jobs they subsequently create, without exactly knowing their requirements.

The iRMX 86 Operating System supplies other memory management functions to search specific address ranges for available memory. The System performs this search at system initialization, and can be configured to ignore non-existent memory and addresses reserved for I/O devices and other application requirements.

Table 2 lists the major system calls used to manage the system memory.

### Interrupt Management

Real-time systems, by their nature, must respond to asynchronous and unpredictable events quickly. The iRMX 86 Operating System uses interrupts and the event-driven nucleus described earlier to give real-time response to events. Use of a pre-emptive scheduling technique ensures that the servicing high priority events always take precedence over other system activities.

The iRMX 86 Operating System gives applications the flexibility to optimize either interrupt response time or interrupt response capability by providing two tiers of Interrupt Management. These two distinct tiers are managed by Interrupt Handlers and Interrupt Tasks.

**Interrupt Handlers** are the first tier of interrupt service. For small, simple functions, interrupt handlers are often the most efficient means of responding to an event. They provide faster response than interrupt tasks, but must be kept simple since interrupts (except the iAPX 86 and 88 non-maskable interrupt) are masked during their execution. When extended interrupt service is required, interrupt handlers "signal" a waiting interrupt task that, in turn, performs more complicated functions.

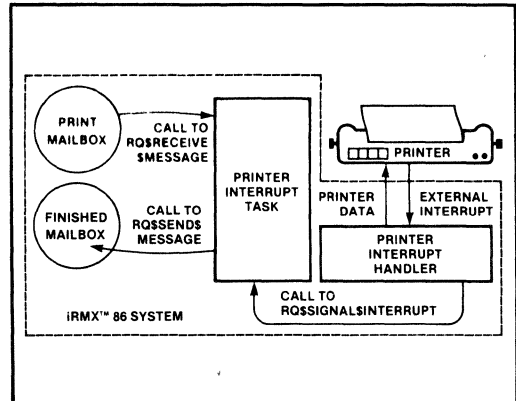
**Interrupt Tasks** are distinct tasks whose priority is associated with a hardware interrupt level. They are permitted

to make any iRMX 86 system call. While an interrupt task is servicing an interrupt, interrupts of lower priority are not allowed to pre-empt the system.

Table 3 shows the iRMX 86 System Calls provided to manage interrupts.

### INTERRUPT MANAGEMENT EXAMPLE

Figure 3 illustrates how the iRMX 86 Interrupt System may be used to output strings of characters to a printer. In the example, a mailbox named 'PRINT' is used by all tasks in the system to queue messages to be printed. Application tasks put the characters in segments that are transmitted to the printer interrupt service task via the PRINT Mailbox. Once printing is complete, the same interrupt task passes the messages on to another application via the FINISHED Mailbox so that an operator message can be displayed.



**Figure 3. Interrupt Management Example**

### BASIC I/O SYSTEM

The Basic I/O System (BIOS) provides the direct access to I/O devices needed by real-time applications. The BIOS allows I/O functions to overlap other system functions. In this manner, application tasks make asynchronous calls to the iRMX 86 BIOS, and proceed to perform other activities. When the I/O request must be completed before an application can continue, the task waits at a mailbox for the result of the operation.

Some system calls provided by the BIOS are listed in Table 4.

The Basic I/O System communicates with peripheral devices through device drivers. These device drivers provide the System with four basic functions needed to control and communicate with devices: Initialize I/O, Finish I/O, Queue I/O, and Cancel I/O. Using the device driver interface, users of non-standard devices may write custom drivers compatible with the I/O System.



**Table 2. Memory Management System Calls**

System Call	Function Performed
RQ\$CREATE\$SEGMENT	Dynamically allocates a memory segment of the specified size.
RQ\$DELETE\$SEGMENT	Deletes the specified segment by deallocating the memory.
RQ\$GET\$POOL\$ATTRIBUTES	Returns attributes such as the minimum and maximum, as well as current size of the memory in the environment of the calling task's job.
RQ\$GET\$SIZE	Returns the size (in bytes) of a segment.
RQ\$SET\$POOL\$MIN	Dynamically changes the minimum memory requirements of the job environment containing the calling task.

**Table 3. Interrupt Management System Calls**

System Call	Function Performed
RQ\$SET\$INTERRUPT	Assigns an interrupt handler and, if desired, an interrupt task to the specified interrupt level. Usually the calling task becomes the interrupt task.
RQ\$RESET\$INTERRUPT	Disables an interrupt level, and cancels the assignment of the interrupt handler for that level. If an interrupt task was assigned, it is deleted.
RQ\$GET\$LEVEL	Returns the number of the highest priority interrupt level currently being processed.
RQ\$SIGNAL\$INTERRUPT	Used by an interrupt handler to signal the associated interrupt task that an interrupt has occurred.
RQ\$WAIT\$INTERRUPT	Used by an interrupt task to SLEEP until the associated interrupt handler signals the occurrence of an interrupt.
RQ\$EXIT\$INTERRUPT	Used by an interrupt handler to relinquish control of the System.
RQ\$ENABLE	Enables the hardware to accept interrupts from a specified level.
RQ\$DISABLE	Disables the hardware from accepting interrupts at or below a specified level.

**Table 4. Key BIOS I/O Management System Calls**

System Call	Function Performed
RQ\$ATTACH\$FILE	Creates a Connection to an existing file.
RQ\$CHANGE\$ACCESS	Changes the types of accesses permitted to the specified user(s) for a specific file.
RQ\$CLOSE	Closes the Connection to the specified file so that it may be used again, or so that the type of access may be changed.
RQ\$CREATE\$DIRECTORY	Creates a Named File used to store the names and locations of other Named Files.
RQ\$CREATE\$FILE	Creates a data file with the specified access rights.
RQ\$DELETE\$CONNECTION	Deletes the Connection to the specified file.
RQ\$GET\$FILE\$STATUS	Returns the current status of a specified file.
RQ\$OPEN	Opens a file for either read, write, or update access.
RQ\$READ	Reads a number of bytes from the current position in a specified file.
RQ\$SEEK	Moves the current data pointer of a Named or Physical file.
RQ\$WRITE	Writes a number of bytes at the current position in a file.
RQ\$WAIT\$I/O	Synchronizes a task with the I/O System by causing it to wait for I/O operation results.

The iRMX 86 Operating System includes a number of device drivers to allow applications to use standard USART serial communication devices, multiple CRTs and keyboards, bubble memories, diskettes, disks, a Centronics-type parallel printer, and many of Intel's iSBC and iSBX™ device controllers (see Table 9). If an application requires use of a non-standard device, users need only write a device driver to be included with the BIOS, and access it as if it were part of the standard system. For most random-access devices, this job is further simplified by using standard routines provided with the System. Use of this technique ensures that applications can remain device independent.

### MULTI-TERMINAL SUPPORT

The iRMX 86 Terminal Support provides line editing and terminal control capabilities. The Terminal Support communicates with devices through simple drivers that do only character I/O functions. Dynamic terminal reconfiguration is provided so that attributes such as terminal type and line speed may be changed without modifying the application or the Operating System. Dynamic configuration may be typed in, generated programmatically or stored in a file and copied to a terminal I/O connection.

The iRMX 86 Terminal Support provides automatic translation of control characters to specific control sequences for each terminal. This translation enables applications using standard control characters to function with non-standard terminals. The translation requirements for each terminal can be stored in terminal description files and copied to a connection, as described above.

### DISK I/O PERFORMANCE

Table 5 shows iRMX 86 performance obtained using the iSBC 215 Winchester Disk and iSBX 218 Diskette Controllers under the specified conditions.

Each device driver can be used to interface to a number of separate and, in some cases, different devices (See Figure 4). The iSBC 215 Device Driver, supplied with the system, is capable of supporting the iSBC 215 Winchester Disk Controller, the iSBC 220 SMD Disk Controller, and the iSBX 218 Flexible Disk Controller (when mounted on an iSBC 215 board). Each device controller may, in turn, control a number of separate device units. In addition, each driver may control a number of like device controllers. This capability allows the use of large storage systems with a minimum of I/O system code to write or maintain.

### EXTENDED I/O SYSTEM

The iRMX 86 Extended I/O System (EIOS) adds a number of I/O management capabilities to simplify access to

files. Whereas the BIOS provides users with the basic system calls needed for direct management of I/O resources, many users prefer to have the system perform all the buffering and synchronization of I/O requests automatically. The EIOS allows users to access I/O devices without having to write procedures for buffering data, or to specify particular devices with constant device names.

By performing device buffering automatically, the iRMX 86 EIOS optimizes accesses to disks and other devices. Often, when an application task asks the System to READ a portion of a file, the System is able to respond immediately with the data it has read in advance of the request. Similarly, the EIOS will not delay a task for writing data to a device unless it is specifically told to, or if its output buffers are filled.

Logical file and device names are provided by the EIOS to give applications complete file and device independence. Applications may send data to the 'line printer' (:LP:) without needing to know which specific device will be used as the printer. This logical name may, in fact, not be a printer at all, but it could be a disk file that is later scheduled for printing.

The EIOS uses the functions provided by the BIOS to synchronize individual I/O requests with results returned by device drivers. Most EIOS system calls are similar to the BIOS calls, except that they appear to suspend the operation of the calling task until the I/O requests are completed.

**Table 5. BIOS Typical Performance**

Function	Average Character Throughput Bytes per Second*	
	Winchester Disk	Diskette
Single File Read	42,000	15,800
Two File Read (Same Device)	36,800	5,700
Single File Write	23,800	5,400
Two File Write (Different Devices)	36,200	6,900
Read/Write Two Files (Different Devices)	38,900	6,000

\* These measurements were made in the following environment: Entire iRMX™ 86 operating system and application code and data located in on-board RAM of a 8-MHz iSBC® 86/30 Single Board Computer. Named files, each with a file size of 128 KBytes, were used with a device and volume granularity of 1 KBytes and six 1 KByte buffers. The disk interleave factor was 2. The iSBC 215 Winchester Controller was attached to two 20-Mbyte drives, and supported the iSBX™ 218 Diskette Controller that, in turn, was attached to two double density 8" diskette drives. This performance is, to a large part, restricted by the mechanical speed of the devices.

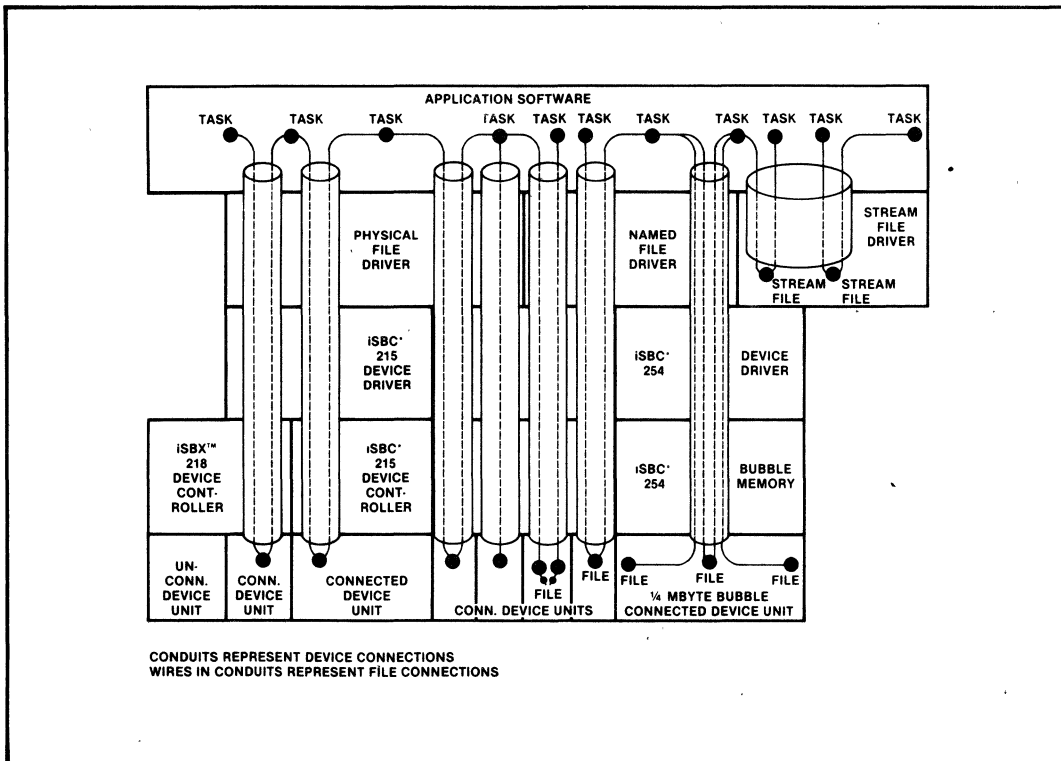


Figure 4. Device Driver and Controller Relationships

## File Management

The iRMX 86 Operating System provides three distinct types of files to ensure efficient management of both program and data files: Named Files, Physical Files, and Stream Files. Each file type provides access to I/O devices through the standard device drivers mentioned earlier. The same device driver is used to access physical and named files for a given device.

### NAMED FILES

Named files allow users to access information on secondary storage by referring to a file with its ASCII name. The names of files stored on a device are stored in special files called directories. As directories are themselves named files, the iRMX 86 File System allows directories to contain the names of other directories. Figure 5 illustrates the resulting hierarchical file structure. This structure is useful for isolating file names to particular user applications, and for tailoring system data to the requirements of users and applications sharing storage devices. Using different branches on the directory tree, different users do not have to coordinate in naming their files to ensure unique names.

Whenever a request is made involving a file name, the System will search the appropriate directory in order to find the necessary information about the file's size, access rights, and specific location on the storage device.

The iRMX 86 BIOS uses an efficient format for writing the directory and data information into secondary storage. This standard iRMX 86 format is fully compatible with the ISO Media standard, and other Intel systems such as the iRMX 88 Operating System. This structure enables the system to directly access any byte in a file, often without having to do additional I/O to access space allocation information. The maximum size of an individual file is  $2^{32}$  (4.3 billion) bytes.

### EASE OF ACCESS

The hierarchical file structure is provided to isolate and organize collections of named files. To give operators fast and simple access to any level within the file tree, an ATTACHFILE command is provided. This command allows operators to give a logical name to a point in the tree so that a long sequence of characters need not be typed each time a file is referred to.

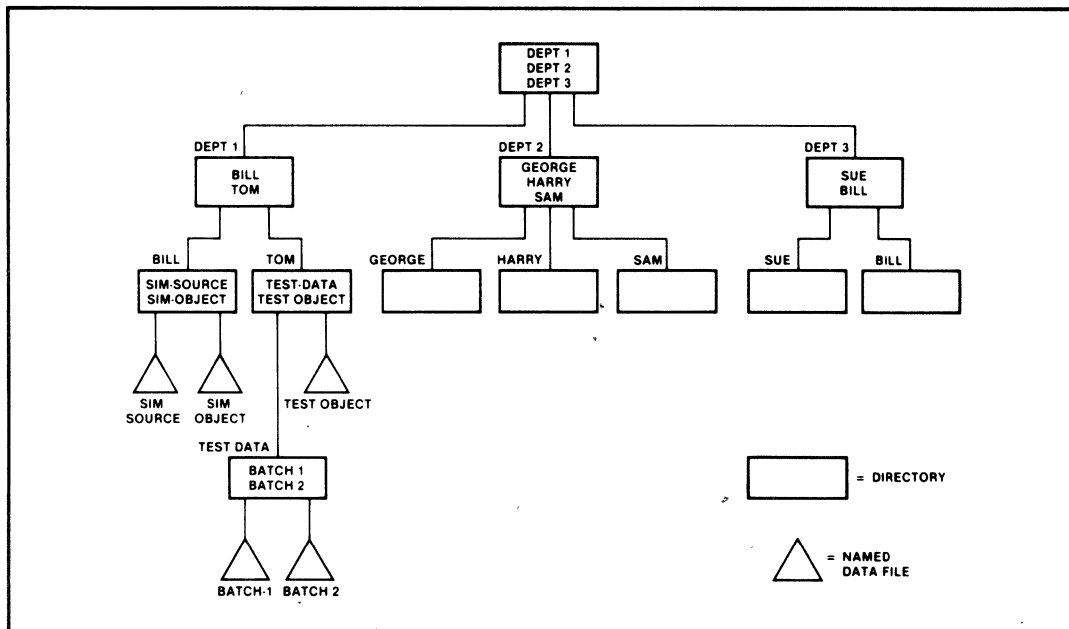


Figure 5. Hierarchical Named File Structure

## ACCESS PROTECTION

Access to each Named File is protected by the rights assigned to each user by the owner of the file. Rights to read, append, update, and delete may be selectively granted to other users of the system. In general, users of Named Files are classified into one of three categories: User, Group, and World. Users and Groups are used when different programmers and programs need to share information stored in a file. The World classification is used when rights are to be granted to all who can use the system.

## PHYSICAL FILES

Physical Files allow more direct device access than Named Files. Each Physical File occupies an entire device, treated as a single stream of individually accessible bytes. No access control is provided for Physical Files as they are typically used for such applications as driving a printing device, translating from one device format to another, driving a paper tape device, and controlling analog mechanisms.

## STREAM FILES

Stream Files provide applications with a method of using iRMX 86 file management methods for data that does not need to go into secondary storage. Stream Files act as direct channels, through system memory, from one task to another. These channels are very useful to pro-

grams, for example, wishing to preserve file and device independence allowing data sent to a printer one time, to a disk file another time, and to another program on a different occasion.

## BOOTSTRAP AND APPLICATION LOADERS

Two utilities are supplied with the System to load programs and data into system memory from secondary storage devices:

The **iRMX 86 Bootstrap Loader** can be configured to a size of less than 600 bytes of P(ROM), and is typically used to load the initial system from the system disk into memory, and begin its execution.

The **Application Loader** is typically used by application programs already running in the system to load additional programs and data from any secondary storage device. The Human Interface layer, for example, uses the Application Loader to load the non-resident Human Interface Commands. The Application Loader is capable of loading both relocatable and absolute code, as well as program overlays.

## Human Interface

The flexibility of the interface between computer controlled machines and their users often determines the usability and ultimate success of the machines. Table 12 lists iRMX 86 Human Interface functions giving users

and applications simple access to the file and system management capabilities described earlier. The process, interrupt, and memory management functions described earlier, are performed automatically for Human Interface users.

### MULTI-USER ACCESS

Using the multi-terminal support provided by the BIOS, the iRMX 86 Human Interface can support several simultaneous users. The real-time nature of the system is maintained by providing a priority for each user, and using the event-driven iRMX 86 Nucleus to schedule tasks. High-performance interrupt response is guaranteed even while users interact with various application packages. For example, multi-terminal support allows one person to be using the iRMX 86 Editor, while another compiles a FORTRAN 86 or PASCAL 86 program, while several others load and access applications.

Each terminal attached to the iRMX 86 multi-user Human Interface is automatically associated with a user, a memory pool, and an initial program to run when the terminal is connected. This association is made using a file that may be changed at any time. Changes are effective the next time the system is initialized.

The initial program specified for each terminal can be a special application program, a custom Human Interface, or the standard iRMX 86 Command Line Interpreter (CLI). For example, you may choose to use the Microsoft Basic Interpreter as this initial program. After system start-up, each terminal user would be able to run the interpreter without asking for it to be loaded. From the BASIC interpreter an operator, for example, could run a data collection program, written in BASIC, that communicates with several laboratory instruments, and prints charts and reports based on certain test results. When finished entering, changing, or running a BASIC program, the terminal would remain in BASIC for the next user.

Specifying an application program as a terminal's initial program makes the interface between operators and the computer system much simpler. Each operator need only be aware of the function of a particular application; not needing to interact with any unfamiliar functions also available on his application system.

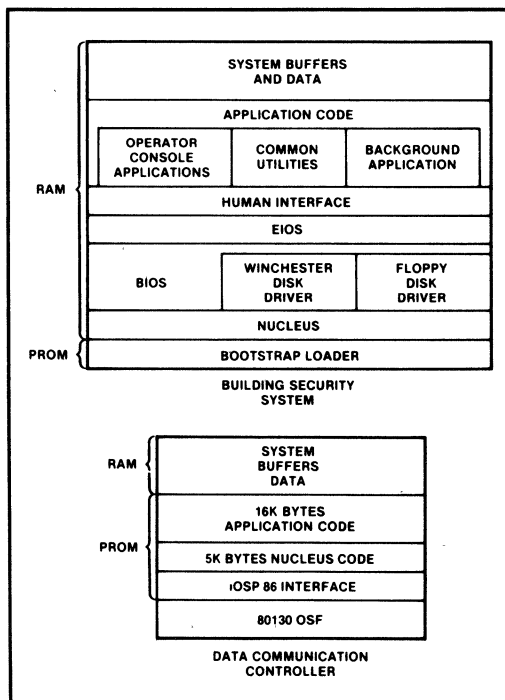
Specifying the standard iRMX 86 Human Interface CLI as the initial program enables users of the terminals to access all iRMX 86 functions. This CLI makes it easy to manage iRMX 86 files, load and execute Intel-supplied and custom programs, and submit command files for later execution.

**Table 6. iRMX™ Real-Time Performance**

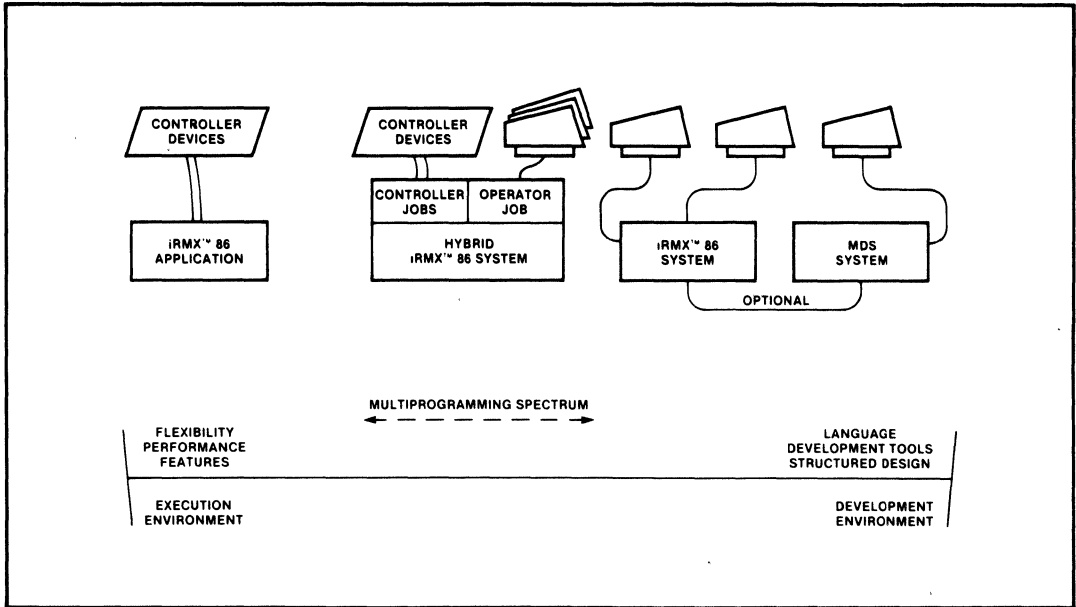
Real-Time Function	Execution Time (msec)
SUSPEND TASK	0.45
INTERRUPT LATENCY (to Handler)	0.20 (Max)
INTERRUPT LATENCY (to Handler)	0.03 (Typical)
CONTEXT SWITCH CAUSED BY INTERRUPT	0.68 (Max)
SEND MESSAGE (no context switch)	0.30
SEND MESSAGE (with context switch)	0.57
SEND CONTROL (no context switch)	0.19
SEND CONTROL (with context switching)	0.50
RECEIVE CONTROL (no waiting)	0.25

Context switch time is the time between executing in the context of a task, and the first instruction to execute in the context of another task.

These times were measured using an 8 MHz iSBC<sup>®</sup> 86/30 Single Board Computer with the standard configuration supported by the Pre-configured System, and all program and data stored in on-board dynamic RAM.



**Figure 6. Typical iRMX™ 86 Configurations**



## FEATURE OVERVIEW

The iRMX 86 Operating System is well suited to serve the demanding needs of real-time applications executing on complex microprocessor systems. The iRMX 86 System also provides many tools and features needed by real-time system developers and programmers. The following sections describe features useful in both the development and execution environments. The description of each feature outlines the advantages given to hardware and software engineers concerned with overall system cost, expandability with custom and industry standard options, and long-term maintenance of iRMX 86-based systems. The development environment features also describe the ease with which the iRMX 86 Operating System can be incorporated into overall system designs.

### Execution Environment Features

#### REAL-TIME PERFORMANCE

The iRMX 86 Operating System is designed to offer the high performance, multi-tasking functions required by real-time systems. Designers can make use of the latest VLSI devices such as the 8087 Numeric Processor Extension, and the 80130 Operating System Firmware Component to improve their system cost/performance ratio, or the iMMX™ 800 MULTIBUS® Message Exchange

software package to divide and coordinate various system activities among multiple processors. Typical iRMX 86 system performance characteristics are shown in Table 6.

Many real-time systems require high-performance operation. To meet this requirement, all of iRMX 86 (except for the Human Interface commands) can be put into zero wait-state P(ROM). This approach eliminates the possibility of disk access times slowing down performance, while allowing system designers to take advantage of high performance memory devices.

#### CONFIGURABILITY

The iRMX 86 Operating System is configurable by system layer, and by system call within each layer. In addition, all the I/O port addresses used by the System are configurable by the user. This flexibility gives designers the freedom to choose configurations of hardware and software that best suit their size and functional requirements. Two example configurations are shown in Figure 6.

Most configuration options are selected during system design stages. Others may be selected during system operation. For example, the amount of memory devoted to queues within a Mailbox can be specified at the time the Mailbox is created. Devoting more memory to the Mailbox allows more messages to be transmitted to other tasks without having to degrade system performance to allocate additional memory dynamically.

The chart shown in Table 7 indicates the actual memory size required to support these different configurations of the iRMX 86 System. Systems requiring only Nucleus level functions may require no more than 13 KBytes for the Operating System (Use of the iAPX 86/30 requires only 4K Bytes of RAM). Other applications, needing I/O management functions, may select portions of additional layers that fit their needs and size constraints.

This configurability also applies to the Terminal Handler and Debugger layers. They need be included only when the iRMX 86 Debugger is needed (usually only during system development) or when a serial terminal interface is needed in a system that otherwise doesn't need an I/O System.

### MULTI-PROCESSING

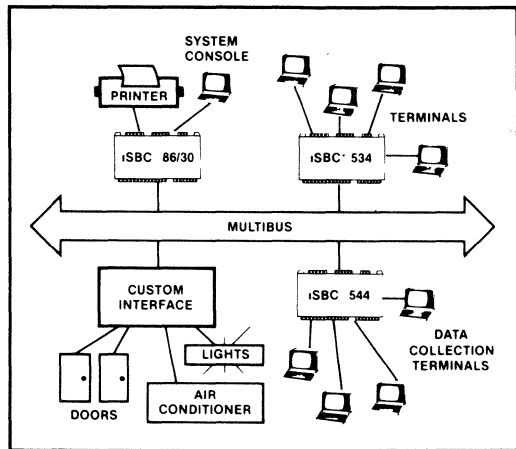
The resources provided by a single processor are often not enough to perform certain functions. With the standard interfaces provided by the iMMX 800 MULTIBUS Message Exchange package, the iRMX 86 Operating System supports a loosely-coupled multi-processing environment. Tasks running on one processor may communicate with tasks running on other processors, even if they operate under different operating systems. The iMMX 800 software is capable of sending messages over the MULTIBUS to tasks operating under either the iRMX 80 8-bit Multi-Tasking Executive, the iRMX 88 Executive, or the iRMX 86 Operating System. Using this message exchange mechanism, applications may increase their system performance quite easily, improve overall interrupt response, gain access to the iSBC 550 Ethernet Controller, and leave room for future product enhancements.

### MULTI-USER ACCESS

Many real-time systems must provide a variety of users access to system control functions and collected data.

The iRMX 86 System provides easy-to-use support for applications to access multiple terminals. It also enables multiple and different users to access different applications concurrently.

Figure 7 illustrates a typical iRMX 86 application simultaneously supporting multi-terminal data collection and real-time environments. Shown is a group of terminals used by machinists on a shop floor to communicate with a job management program, a building security system that constantly monitors energy usage requirements, a system operator console capable of accessing all system functions, and a group of terminals in the Production Engineering department used to monitor job costs while developing new device control specifications and instructions. The iSBC 544 Intelligent Terminal Interface supports multiple user terminals without degrading system performance to handle character I/O.



**Figure 7. Multi-Terminal and Multi-User Real-Time System**

**Table 7. iRMX™ 86 Configuration Size Chart**

System Layer	Min. ROMable Size	Max. Size	Data Size
Bootstrap Loader	0.5K	1.5K	6K*
Nucleus	10.5K	24K	2K
BIOS	26K	78K	1K
Application Loader	4K	10K	2K
EIOS	10.5K	12.5K	1K
Human Interface	22K	22K	15K
UDI	11K	11K	0
Terminal Handler	3K	3K	0.3K
Debugger	28.5K	28.5K	1K
Human Interface Commands			116K
Interactive Configuration Utility			308K

\* Usable by System after bootloading.

**EXTENDABILITY**

The iRMX 86 Operating System provides three means of extensions. This extendability is essential for support of OEM and volume end user value added features. This ability is provided by: User-defined operating system calls, user-defined objects (similar to Jobs, Tasks, etc.), and the ability to add functions later in the product life cycle. The modular, layered structure of the System easily facilitates later additions to iRMX 86 applications. User-defined objects are supported by the functions listed in Table 8.

Using standard iRMX 86 system calls users may define custom objects, enabling applications to easily manipulate commonly used structures as if they were part of the original operating system.

**EXCEPTION HANDLING**

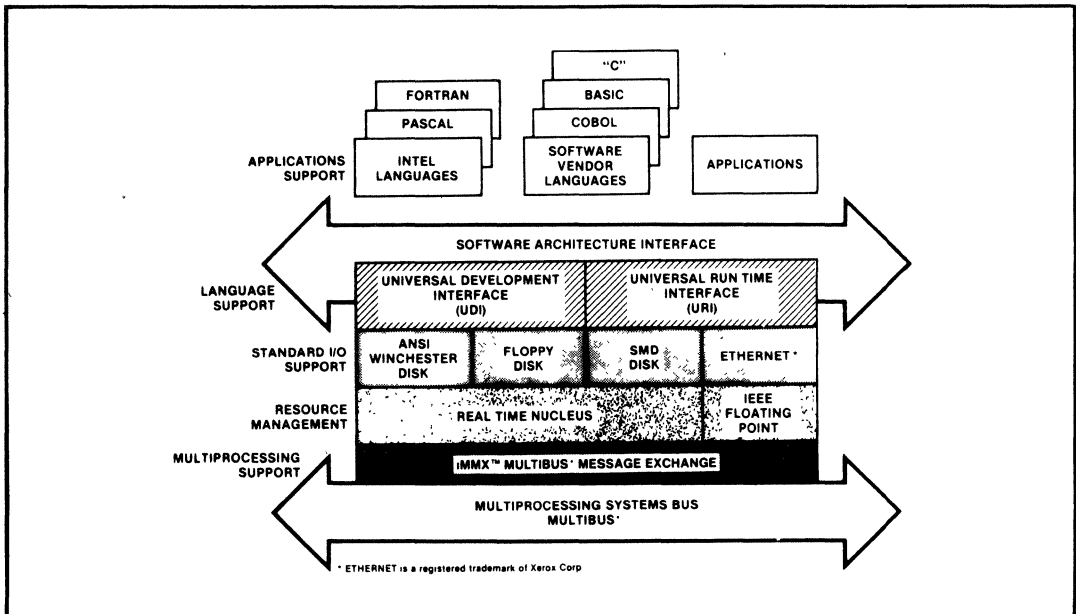
The System includes predefined exception handlers for typical I/O and parameter error conditions. The error handling mechanism is both configurable and extendable.

**SUPPORT OF STANDARDS**

The iRMX 86 Operating System supports the many hardware and software standards needed by most application systems to ensure that commonly available hardware and software packages may be interfaced with a minimum of cost and effort. The iRMX 86 System supports the iSBC family of products built on the Intel MULTIBUS (IEEE Standard 796), and a number of standard software interfaces such as the UDI and the common device driver interface (See Figure 8). The procedural interfaces of

**Table 8. User Extension System Calls**

System Call	Function Performed
RQ\$CREATE\$COMPOSITE	Creates a custom object built of previously defined objects.
RQ\$DELETE\$COMPOSITE	Deletes the custom object, but not the various objects from which it was built.
RQ\$INSPECT\$COMPOSITE	Returns a list of Token Identifiers for the component objects from which the specified composite object is built.
RQ\$ALTER\$COMPOSITE	Replaces a component object of a composite object
RQ\$CREATE\$EXTENSION	Creates a new type of object and assigns a mailbox used for collecting these objects when they are deleted.
RQ\$DELETE\$EXTENSION	Deletes an extension definition.



**Figure 8. iRMX™ 86 Standard Interfaces**



the UDI, a software analogy to the MULTIBUS, are listed in Table 10.

The Operating System includes support for the proposed IEEE 80-bit extended real-variable format of the 8087 Numeric Data Processor, the IEEE 796 (MULTIBUS) hardware interface, and the Intel Universal Run-time Interface (URI). Other standards such as the iMMX 800 MULTIBUS Message Exchange, and an Ethernet\* communication interface, are supported by optional software packages available to run on the iRMX 86 System.

### SPECTRUM OF CPU PERFORMANCE

The iRMX 86 System supports 8086 and 8088 based systems directly at a variety of processor clock speeds. With the iRMX 286R Operating System option, completely compatible systems can be built around the iAPX 286 processor. By choosing the appropriate CPU, designers can select from a wide range of performance without having to change application software.

### COMPONENT LEVEL SUPPORT

The iRMX 86 System may be tailored to support specific hardware configurations. In addition to system memory, only an iAPX 86 or iAPX 88 microprocessor, an 8259A Programmable Interrupt Controller, and either an 8253 or 8254 Programmable Interval Timer are required. In addition, the iRMX 86 Operating System may be used to augment the functions of the 80130 Operating System Firmware Component that not only provides these hardware functions, but eliminates the need for approximately 14 KBytes of the iRMX 86 Nucleus code (see Figure 6). For systems requiring extended mathematics capability, an 8087 Numeric Data Processor may be added to perform these functions up to 100 times faster than equivalent software. For applications servicing more than 8 interrupt sources, additional 8259A's may be configured as slave controllers.

### BOARD LEVEL SUPPORT

The iRMX 86 Operating System includes device drivers to support a broad range of MULTIBUS device controllers. The particular boards and types of devices supported are listed in Table 9. The device controllers all adhere to industry standard electrical and functional interfaces.

In addition to the on-CPU board terminal drivers, the iRMX 86 BIOS includes two iSBC board-level device drivers to support multiple terminal interfaces:

The iSBC 544 Intelligent Four-Channel Terminal Interface Device Driver provides support for multiple controllers each supporting up to 4 standard RS232 terminals. The iSBC 544 driver takes advantage of an on-board 8085 processor to greatly reduce the system processor time required for ter-

minal I/O by locally managing input and output buffers. The iSBC 544 firmware provided with the operating system can off-load the system CPU by as much as 75%.

The iSBC 534 Four-Channel USART Controller Device Driver also provides support for multiple controller boards each supporting up to 4 standard RS232 terminals.

**Table 9. Supported Devices**

<b>iSBC® Device Controller</b>	<b>Supported Devices</b>
iSBC® 86,88	Serial Port to CRT, Parallel Port to Centronics-type Printer, Interval Timer, and Interrupt Controller
iSBC® 204	Single Density Diskette
iSBC® 206	Cartridge-type Hard Disk
iSBC® 208	Single & Double Density, Single & Double Sided, 8" & 5.25" Diskette
iSBC® 215	Standard Winchester Disks
iSBC® 220	Standard Storage Module Disks
iSBC® 254	Bubble Memory Board
iSBC® 534,544	4-Channel Serial Ports to CRTs, Modems
iSBX™ 218	Single & Double Density, Single & Double Sided, 8" & 5.25" Diskette (When used on an iSBC® 215 Winchester Controller)
iSBX™ 270	Black and White CRT's and full ASCII keyboards

### Development Environment Features

The iRMX 86 Operating System supports the efficient utilization of programming time by providing important tools for program development. Some of the tools necessary to develop and debug real-time systems are included with the Operating System. Others, such as language compilers, are available from Intel and from leading independent Software Vendors.

### LANGUAGES

The iRMX 86 Operating System supports a group of 31 standard system calls known as the Universal Development Interface (UDI). Figure 8 shows that the additional features of this standard interface provide iRMX 86 systems the capability of using many compilers and language translators. These include the iAPX 86 and 88 Macro Assembler, and the Pascal 86/88, PL/M 86/88, and FORTRAN 86/88 compilers available from Intel. They also include a number of other Intel development tools,

\* Ethernet is a trademark of Xerox Corporation.

and language translators and utilities available from other software vendors. A subset of the UDI System Calls provides another standard interface called the Universal Runtime Interface (URI). The URI calls are those required to execute a compiled program, while the full set of UDI calls is required to run a compiler.

These standard software interfaces (the URI and the UDI) ensure that users of the iRMX 86 Operating System may transport their applications to future releases of the iRMX 86 Operating System and other Intel and independent vendor software products. The calls available in the URI and UDI are shown in Table 10.

**Table 10. URI and UDI System Calls**

System Call	Function Performed
<b>Memory Management:</b>	
DQ\$ALLOCATE	Creates a Segment of a specified size.
DQ\$FREE	Returns the specified segment to the System.
DQ\$GET\$SIZE*	Returns the size of the specified Segment.
DQ\$RESERVE\$I/O\$MEMORY*	Reserves memory to OPEN and ATTACH files.
<b>File Management:</b>	
DQ\$ATTACH	Creates a Connection to a specified file.
DQ\$CHANGE\$ACCESS*	Changes the user access rights associated with a file or directory
DQ\$CHANGE\$EXTENSION	Changes the extension of a file name in-memory
DQ\$CLOSE	Closes the specified file Connection
DQ\$CREATE	Creates a Named File.
DQ\$DELETE	Deletes a Named File.
DQ\$DETACH	Closes a Named File and deletes its Connection.
DQ\$OPEN	Opens a file for a particular type of access
DQ\$GET\$CONNECTION\$STATUS*	Returns the current status of the specified file Connection
DQ\$FILE\$INFO*	Returns data about a file Connection
DQ\$READ	Reads the next sequence of bytes from a file.
DQ\$RENAME*	Renames the specified Named File
DQ\$SEEK	Moves the position pointer of a file.
DQ\$TRUNCATE	Truncates a file.
DQ\$WRITE	Writes a sequence of bytes to a file.
<b>Process Management:</b>	
DQ\$EXIT	Exits from the current application job.
DQ\$OVERLAY*	Causes the specified overlay to be loaded
DQ\$SPECIAL	Performs special I/O related functions on terminals with special control features.
DQ\$TRAP\$CC	Captures control when CNTRL/C is typed.
<b>Exception Handling:</b>	
DQ\$GET\$EXCEPTION\$HANDLER	Returns a pointer to the program currently being used to process errors.
DQ\$DECODE\$EXCEPTION	Returns a short description of the specified error code
DQ\$TRAP\$EXCEPTION	Identifies a custom exception processing program for a particular type of error
<b>Application Assistance:</b>	
DQ\$DECODE\$TIME	Returns system time and date in binary and ASCII character format.
DQ\$GET\$ARGUMENT*	Returns the next argument from the character string used to invoke the application program
DQ\$GET\$SYSTEM\$ID*	Returns the name of the underlying operating system supporting the UDI.
DQ\$GET\$TIME*	Returns the current time of day as kept by the underlying operating system.
DQ\$SWITCH\$BUFFER	Selects a new buffer from which to process commands.

\* Calls available only through the UDI

The high performance of the iRMX 86 Operating System enhances the throughput of compilers and other development utilities. Table 11 indicates the average performance of typical development environment functions operating in the same configuration described in Table 5.

**Table 11. Development Environment Performance**

Function	Average Execution Time
Directory Command (S Format with 25 files)	5.3 sec
Load the COPY Command	1.2 sec
Copy a 1K Byte File (Winchester to Winchester)	1.0 sec
Copy a 16K Byte File	1.7 sec
Copy a 64K Byte File	3.9 sec
Copy a 1K Byte File (Winchester to Diskette)	1.4 sec
Compile PL/M 86	393 lpm
Compile PASCAL 86 Program	453 lpm

**TOOLS**

Certain tools are necessary for the development of microcomputer applications. The iRMX 86 Human Interface includes many of these tools as non-resident commands. They can be included on the system disk of an application system, and brought into memory when needed to perform functions as listed in Table 12.

**Table 12. Major Human Interface Utilities**

Command	Function
BACKUP	Copy directories and files from one device to another.
COPY	Copy one or more files to one or more destination files.
CREATEDIR	Create a directory file to store the names of other files.
DIR	List the names, sizes, owners, etc. of the files contained in a directory.
ATTACHFILE	Give a logical name to a specified location in a file directory tree.
PERMIT	Grant or rescind user access to a file.
RENAME	Change the name of a file.
SUBMIT	Start the processing of a series of commands stored in a file.
SUPER	Change operator's ID to that of the System Manager with global access rights and privileges.

**Table 12. Major Human Interface Utilities (Con't.)**

Command	Function
TIME	Set the system time-of-day clock.
VERIFY	Verify the structure of an iRMX™ 86 Named File volume, and check for possible disk data errors.

**INTERACTIVE CONFIGURATION UTILITY**

The iRMX 86 Operating System is designed to provide OEMs the ability to configure for specific system hardware and software requirements. The Interactive Configuration Utility (ICU) builds iRMX 86 configurations by asking appropriate questions and making reasonable assumptions. It runs on either an Intellec® Series III Development System or iRMX 86 System supporting the UDI and a hard disk. Table 13 lists the hardware and support software requirements of different iRMX 86 development system environments.

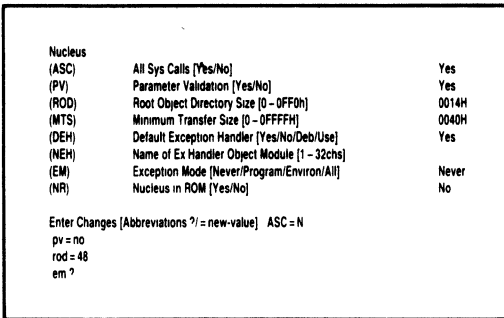
**Table 13. iRMX™ 86 Development Environment**

Intellec® Series III: MDS 313 PL/M 86/88 Compiler One hard disk and one diskette drive
iRMX™ 86 Preconfigured System iRMX™ 860 Utility iRMX™ 863 PL/M 86/88 Compiler iSBC® 957B Monitor 448K Bytes of RAM 5M Byte On-Line Storage and one double-density diskette drive
SYSTEM 86/330 Microcomputer System Basic configuration

Figure 9 shows one of the many screens displayed during the process of defining a configuration. It shows the abbreviations for each choice on the left, a more complete description with the range of possible answers in the center, and the current (sometimes default) choice on the right. The bottom of the screen shows three changes made by the operator (lower case lettering), and a request for help on the Exception Mode question. In response to a request for help, the ICU displays an additional screen outlining possible choices and some overall system effects.

The ICU requests only information required as a result of previous choices. For example, if no Extended I/O System functions are required, the ICU will not ask any further questions about the EIOS. Once a configuration session is complete, the operator may save all the information in a file. Later, when small changes are neces-

sary, this file can be modified. A completely new session is not required.



**Figure 9. ICU Screen for iRMX™ 86 Nucleus**

### REAL-TIME DEBUGGING TOOLS

The iRMX 86 Operating System supports three distinct debugging environments: Static, Dynamic, and Post-Mortem. While the iRMX 86 Operating System does support a multi-user Human Interface, these real-time debugging aids are usually most useful in a single-user environment where modifications made to the system cannot affect other users.

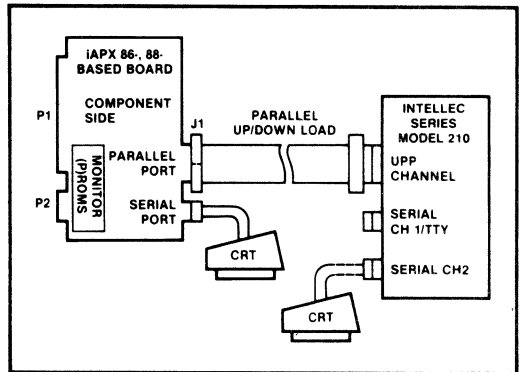
The static debugging aid is the iSBC 957B Monitor (included in the first shipment of some iRMX 86 options). The Monitor provides a basic debugging capability for both system and application code. The iRMX 86 Debugger provides a dynamic system debugging tool for testing and debugging real-time systems. The Debugger allows programmers to stop and inspect one task while the rest of the system continues to operate. The iRMX 86 Crash/Dump Analyzer enables programmers to inspect a system's structure after a problem has caused it to stop normal operation. Each of these debugging facilities are described below.

### iSBC® 957B Monitor

The iSBC 957B Monitor can be used either as a stand-alone monitor for static debugging and system start-up, or as a communication link to an Intellec Development System. A number of PROMs are included along with the necessary cables to control a hardware configuration such as is pictured in Figure 10. All programs necessary for the Intellec system and the target system are included. Configuration tools for users wishing to support different hardware configurations are also included.

Debugging of any iAPX 86 or 88 application is accomplished in an interactive manner via either of the two terminals shown in Figure 10. If an Intellec Development System is not present, all debugging instructions necessary to view and modify register and memory contents, set execution breakpoints and provide single instruction

execution are accessible from a terminal connected directly to the iAPX 86 or 88 system.



**Figure 10. Typical iSBC® 957B Configuration**

### iRMX™ 86 Debugger

The iRMX 86 Debugger runs as part of an iRMX 86 application. It may be used at any time during program development, or may be integrated into an OEM system to aid in the discovery of latent errors. The Debugger can be used to search for errors in any task, even while the other tasks in the system are running. The iRMX 86 Debugger communicates with the developer via a terminal handler that supports full line editing.

### System Crash/Dump Analyzer

The often difficult job of debugging real-time applications is made much simpler with the System Crash/Dump Analyzer. The analyzer allows program developers to record system memory for later analysis even if the system has halted. This analysis lists such vital information as which jobs have active tasks, which system queues contain which tasks, and what segments contain which data.

The information used by the Analyzer is obtained from a copy of iRMX 86 data structures after a fault has caused an unexpected halt (crash). The processor also may be halted deliberately to perform a system analysis. The system information is created by a two-step process:

1. Transferring an image of iRMX 86 system memory to a disk file on an Intellec Series III Microcomputer Development system.
2. Later printing an analyzed and formatted printout description of the state of the system.

Figure 11 shows a portion of a Crash/Dump Analyzer display for an iRMX 86 Mailbox. The display identifies the mailbox by its token and shows its key attributes. This information is followed by a list of tokens for objects (if any) queued at the mailbox.



iRMX 862 FORTRAN 86/88 Compiler  
 iRMX 863 PL/M 86/88 Compiler  
 iRMX 864 TX Screen-oriented Editor  
 iMMX 800 MULTIBUS Message Exchange software package for iRMX 80, 86, and 88 application systems  
 iOSP 86 Support Package for iAPX 86/30 and 88/30 Operating System Processors

Introduction to the iRMX 86 Operating System (9803124-04)  
 iRMX 86 Operator's Manual (144523-001)  
 Master Index for iRMX 86 Release 5 Documentation (145015-001)  
 Getting Started With The Release 5 iRMX 86 System (145073-001)  
 iRMX 86 Installation Guide (9803125-05)  
 iRMX Configuration Guide (9803126-05)  
 iRMX 86 Nucleus Reference Manual (9803122-04)  
 iRMX 86 Terminal Handler Reference Manual (143324-002)  
 iRMX 86 Debugger Reference Manual (143323-002)  
 iRMX 86 Basic I/O System Reference Manual (9803123-05)  
 iRMX 86 Loader Reference Manual (143318-002)  
 iRMX 86 Extended I/O System Reference Manual (143308-002)  
 iRMX 86 Human Interface Reference Manual (9803202-003)  
 Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-004)  
 iRMX 86 Programming Techniques (142982-003)  
 User's Guide For The iSBC 957B iAPX 86, 88 Interface and Execution Package (143979-002)  
 iRMX 86 Disk Verification Utility Reference Manual (144133-002)  
 Runtime Support Manual for iAPX 86, 88 Applications (121776-002)  
 iRMX 86 Crash Analyzer Reference Manual (144522-001)

## Supported Hardware Products

### COMPONENTS

iAPX 86 and 88 Microprocessors  
 iAPX 286 Microprocessors (with iRMX 286R)  
 8087 Numeric Data Processor Extension  
 iAPX 86/30 (80130) Operating System Firmware Component (with iOSP 86)  
 8253 and 8254 Programmable Interval Timers  
 8259A Programmable Interrupt Controller  
 8251A USART  
 8255 Programmable Parallel Interface

### iSBC® MULTIBUS® BOARD AND SYSTEM PRODUCTS

iSBC 86/12A, 86/05, 86/14, 86/30, 88/25, and 88/40 Single Board Computers  
 iSBC 286/10 Single Board Computer (With iRMX 286R)  
 iSBC 204 Diskette Controller  
 iSBC 206 Hard Disk Controller  
 iSBC 208 Diskette Controller  
 iSBC 215 Winchester Disk Controller  
 iSBC 220 SMD Disk Controller  
 iSBC 254 Bubble Memory System  
 iSBC 534 4-Channel Terminal Interface  
 iSBC 544 Intelligent 4-channel Terminal Interface and Controller  
 iSBX 218 Diskette Controller (with iSBC 215)  
 iSBX 350 Parallel Port (Centronics-type Printer Interface)  
 iSBX 351 Serial Communications Port  
 iSBX 270 CRT, Light Pen and Keyboard Interface  
 SYSTEM 86/330 Computer System  
 SYSTEM 86/380 Computer System

### OPTIONAL REFERENCE MATERIALS

Edit Reference Manual (143587-002)  
 Guide to Using iRMX 86 Languages (142907-001)

### APPLICATION NOTES

Ap Note 86 — iRMX 86 Realtime Multitasking Operating System  
 Ap Note 130 — Using Operating System Processors to Simplify Microcomputer Designs

### TRAINING COURSES

Introduction to the iRMX 86 Operating System  
 Advanced iRMX 86 Operating System Concepts

### CUSTOMER SEMINARS

Contact Local Intel Sales Office for details on available video-tape and slide presentations.

## Available Literature

The iRMX 86 Documentation Set is comprised of following reference manuals. Each is also be available under the order numbers shown.

## ORDERING INFORMATION

The iRMX 86 Operating System is available under a number of different licensing options as noted here. Except for source listings (available on microfiche) all options are provided on either single or double density ISIS-formatted diskettes, or on double density iRMX 86-formatted diskettes. ISIS-format diskettes may be used on Intel Intellec Development Systems. The iRMX 86-format may be used on any iRMX 86-based system supporting the appropriate compilers and development environment.

The OEM license options listed here allow users to incorporate the iRMX 86 Operating System into their applications. Each use requires payment of an Incorporation Fee.

<b>Order Code</b>	<b>Description</b>
-------------------	--------------------

RMX 86 KIT ARO:	Single density OEM license.
-----------------	-----------------------------

RMX 86 KIT BRO:	Double density OEM license.
-----------------	-----------------------------

RMX 86 KIT ERO: Double density iRMX 86-Format OEM license for use on iRMX 86-based environments.

Other licensing options include prepayment of all future incorporation fees, single use rights for a single machine, use at a second development site, one-year support service extensions, the right to make copies for additional development systems, and source listing materials.

Each option includes 90 days of support service that provides a periodic NEWSLETTER, Software Problem Report Service, and copies of System updates that occur during this period. Except for source listings, all initial licenses include the iSBC 957B iAPX 86 and 88 System Monitor, and a complete set of iRMX 86 Documentation.

As with all Intel software, purchase of any of these options requires the execution of a standard Intel Master Software License. The specific rights granted to users depend on the specific option and the License signed.



## iRMX™ 88 REAL-TIME MULTITASKING EXECUTIVE

- Event-driven multitasking executive software supports iSBC® 86/05, 86/12A, 86/14, 86/30, 88/25, 88/40, 88/45 or iAPX 86, 88 based applications
- Small, high-performance, PROMable executive supports high sample rates
- Provides simple, intertask communications and synchronization
- Supports the 8087 Numeric Processor Extension (NPX) for arithmetic applications
- Supports component or iSBC™-based system generation through Interactive Configuration Utility
- I/O system provides compatible iRMX™ 86 files and device independent I/O interface
- I/O system supports the User Run-time Interface (URI) for PL/M, PASCAL and FORTRAN coded application tasks
- Memory management of full megabyte iAPX 86, 88 memory

The iRMX 88 Real-Time Multitasking Executive is a small, event-driven single-user executive system. Designed for dedicated computer applications using iSBC 86/05, 86/12A, 86/14, 86/30, 88/25, 88/40, 88/45 or iAPX 86, 88 custom products, the modular software package provides real-time application support for PASCAL, FORTRAN, PL/M and assembler coded tasks. Application tasks utilize intertask communications, synchronous I/O control, priority-based resource allocation and file support for the iSBC 204, 206, 208, 215/218, and 220 Disk Controllers, and the iSBC 254 Bubble Memory product.

The small, high performance iRMX 88 Executive can be located in EPROM or bootstrapped into RAM memory. The iRMX 88 Executive offers features that are suitable for performance-critical process control applications, production test stand units, sophisticated laboratory analysis, instrumentation, specialized data acquisition systems or monitoring stations. The iRMX 88 design, based upon the iRMX 80 Real-Time Executive, offers iRMX 80-like interfaces for those 8-bit applications which are upgrading to 16-bit solutions for the 1 Megabyte addressing, expanded application functions, and higher performance data sampling requirements.

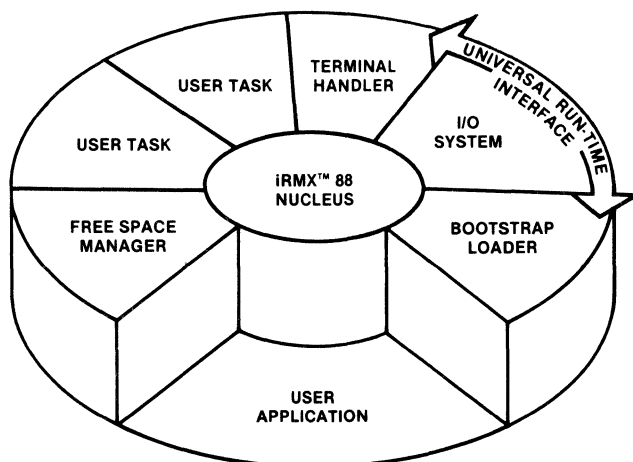


Figure 1. Module Representation

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

October, 1981  
Order Number: 143130-002



## FUNCTIONAL DESCRIPTION

The iRMX 88 Real-Time Multitasking Executive Software package provides facilities for executing tasks concurrently, managing resources and servicing asynchronous events to users of Intel's single board computers and custom iAPX 86, 88-based products. The foundation modules support real-time dedicated computer applications with priority-based task scheduling, interrupt dispatching, real-time clock control with 1 ms resolution, multiple event monitoring and control, and file services for flexible, hard, Winchester, SMD disk units and bubble memory devices. The software package includes the primary modules: Nucleus, Free Space Manager, Terminal Handler, I/O System and Bootstrap Loader. The Interactive Configuration Utility (ICU) executes on a Series III Intellec System, or iRMX 86 Operating System with a Universal Development Interface (UDI).

## FEATURE OVERVIEW

### Event-Driven Multitasking

The iRMX 88 Executive provides a control software foundation called a Nucleus. The iRMX 88 Nucleus provides two major functions: first, the facility for concurrent task execution; secondly, the facility for handling simultaneous asynchronous events.

The structured multitasking environment permits segmenting of the application tasks. The number of tasks, managed by the Nucleus, is limited only by the available 1 Megabyte memory space. The tasks are prioritized such that the highest-ranked task is executing, e.g., an alarm event preempts the lower priority executing task. The Nucleus supports 255 priority levels.

Since internal or external events (interrupts) occur randomly, the Nucleus synchronizes the event with a task. The Nucleus supports either an interrupt service routine or an interrupt task. The interrupt service routine offers high-speed perform-

ance flexibility since it masks all interrupts and supports burst-rate data sample gathering. The interrupt task is useful for lower frequency interrupts, masking only lower priority interrupts.

### Small High-Performance Executive

The iRMX 88 Executive software utilizes a simple, straightforward architecture which minimizes the memory requirements, as shown in Table 1. In addition, the modules are designed to be totally EPROM resident for those systems where mass storage devices cannot be used because of the danger of contamination.

Real-time microcomputer solutions require the recognition of interrupts. The performance of the system is with respect to data sample rates, if there is no activity in progress when an interrupt occurs, the time to handle that interrupt is dependent on the number of instructions executed, e.g., 52 microseconds interrupt latency time on an iSBC 86/12A board. Most real-time solutions have multiple events occurring and background operations in progress. Seldom does a background task have critical sections of code which cannot be interrupted.

### Intertask Communications

The iRMX 88 Nucleus provides a simple, easy-to-use intertask communications mechanism based upon a message. Messages are transferred between tasks with two basic procedure calls, a send (RQSEND) and a wait (RQWAIT). Task "A" requests the Nucleus to RQSEND the pointer to a message buffer to Task "B" (see Figure 2). The Nucleus controls the message flow by activating the higher-priority Task B, or queuing the message if a lower-priority Task B is not waiting for the message. The receiving task does an RQWAIT to get the message pointer and can now access the data which may be for synchronization or real-time control operations.

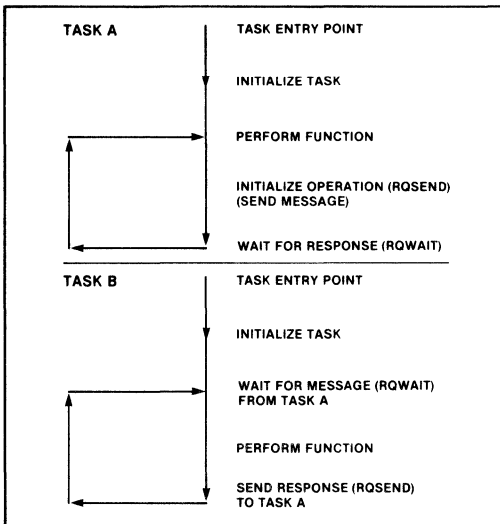
**Table 1. iRMX™ 88 Module Memory Requirements**

MODULE	NUCLEUS	TERMINAL HANDLER	FREE SPACE MANAGER	I/O SYSTEM		
				PHYSICAL**	NAMED**	BOOTSTRAP***
EPROM* (K bytes)	4.0	2.5	1.5	20.0	32.0	1.5

\* amount of code configured in EPROM; all numbers are approximate

\*\* includes one 3K byte device driver (named file plus physical file is 34.0K bytes)

\*\*\* includes an 0.5K byte device driver


**Figure 2. Intertask Communications**

## Numeric Data Processor

The iRMX 88 Nucleus fully supports the 8087 Numeric Processor Extension (NPX) functions for high-speed arithmetic functions of real-time applications. High-performance numeric processing applications, which utilize 8-, 16-, 32- and 64-bit integers, 32-, 64- and 80-bit floating point or 18-digit BCD operations, are accelerated up to 100 times over a iAPX 86, 88 software solution. The NPX functions, including trigonometric, logarithmic and exponential functionals, are essential in scientific, engineering, navigational or military applications.

## Nucleus Primitives

The Nucleus performs other functions as shown in Table 2, in addition to the message communications management. Some primitives like CREATE TASK and DELETE TASK allow dynamic creation/deletion of tasks during run-time. This dynamic capability allows the Nucleus tables to

**Table 2. Nucleus Primitives**

NAME	FUNCTION
ACCEPT	Accept a message from specified exchange. Returns message address if available, zero otherwise.
CREATE TASK	Create task by building new Task Descriptor based on specified Static Task Descriptor.
CREATE EXCHANGE	Create exchange at specified RAM address.
DISABLE INTERRUPT	Disable specified interrupt level.
DELETE EXCHANGE	Delete specified exchange.
DELETE TASK	Delete the task specified.
ENABLE INTERRUPT	Initialize message portion of the Interrupt Exchange Descriptor associated with the specified interrupt level (the first time called only), and enable specified interrupt level.
END INTERRUPT	Signals specific end-of-interrupt for the specified interrupt exchange in a user-supplied interrupt service routine.
INTERRUPT SEND	Send an interrupt message to the specified interrupt exchange.
RESUME	Resume a task that has previously been suspended.
SEND	Send the message located at "msg-addr" to the exchange specified by "exch-addr."
SET INTERRUPT	Set interrupt vector address. An interrupt is to be serviced by the user-supplied routine starting at the address, thus bypassing Nucleus interrupt software.
SUSPEND TASK	Suspend execution of the task specified by the Task Descriptor.
WAIT	Wait at the specified exchange until a message is available or time limit expires. Return address of system timeout message or user message.

expand and accommodate infrequently used tasks which are loaded into memory from a mass storage device.

### Interactive System Generation

The iRMX 88 Executive is constructed in a thoroughly modular manner with the full range of facilities being offered in library modules. By selecting the appropriate features and combining them with the user-written application tasks the generated system is tailored to the application's requirements minimizing memory overhead for unused features.

An Interactive Configuration Utility provides a query-based tool that configures the iRMX 88-based application. Responding to questions from the ICU utility program executing on a Series III Intellec Microcomputer Development System or an iRMX 86-based system, the user quickly tailors the real-time application system.

### I/O System

The iRMX 88 I/O System provides an extensive facility for device-independent I/O. Through a series of supplied iRMX 86 compatible device drivers, the I/O System supports a wide-range of iSBC peripheral controllers. Custom peripheral controllers are supported through user-written device drivers which are integrated with the I/O System at system configuration time. The device-independent nature of the system allows use of different devices without application redesign.

The I/O System (IOS) procedures manage real-time file operations supporting both sequential and random access (see Table 3). The IOS maximizes system throughput by allowing multiple disk operations to proceed in parallel. For example,

files can be "double buffered" so that the task can be processing data in one buffer while the IOS is filling another.

The IOS provides access to two types of files:

- **Named Files** allow applications to refer to collections of bytes (files) by using a name. These names are cataloged in a directory which allows files to be accessed by different tasks.
- **Physical Files** allow applications to make a physical connection to a storage device. Typically used for simple devices such as printers, terminals or sequential data logging where file structures are not necessary.

The file types are a compatible subset of the iRMX 86 Basic I/O System with a flat (non-hierarchical) directory.

### Bootstrap Loader

The iRMX 88 IOS has a Bootstrap Loader which loads a file from mass storage into system memory. The configurable Bootstrap Loader loads the file from a specific device, automatically from the first-ready device of a designated device list, or accepts the file name from a terminal. Storing the system software on disk allows easier future changes to the application system.

### Run-Time Interface

The iRMX 88 Executive provides the User Run-time Interface (URI). This URI interface, in addition to encompassing the I/O System services, provides additional functionality for tasks. The additional functionality includes a trap function and memory management routines which provide the run-time foundation for PASCAL-86, FORTRAN-86, or PL/M-86 coded application tasks.

**Table 3. I/O System Services**

	SERVICE	FUNCTION
<b>Data Transfer Services</b>	CLOSE OPEN READ SEEK TRUNCATE WRITE	<b>Closes</b> a file connection. <b>Opens</b> a file connection for access. <b>Reads</b> a number of bytes from a file. <b>Seeks</b> to the indicated position within a file. <b>Truncates</b> a file. <b>Writes</b> a number of bytes to that file.
<b>File Connection Services</b>	ATTACH CREATE CONNECTION STATUS DELETE DETACH RENAME	<b>Attaches</b> to a file connection. <b>Creates</b> a file and returns a file connection. Returns the file <b>connection status</b> . Marks the file for <b>deletion</b> . <b>Detaches</b> a file connection. <b>Renames</b> an existing file.
<b>Volume Preparation</b>	FORMAT	<b>Formats</b> the disk for files.

## SPECIFICATIONS

### Intellec® System Configuration and Generation Requirements

Series III Intellec Microcomputer Development System with UDI support and a minimum of 2 diskette drives.

### iRMX™-Based Configuration and Generation Requirements

iRMX 86-based system with UDI support and a minimum of 2 diskette drives.

### Supported Hardware

#### ISBC™ SUPPORTED MICROCOMPUTERS

iSBC 86/05 Board  
iSBC 86/12A Board  
iSBC 86/14 Board  
iSBC 86/30 Board  
iSBC 88/25 Board  
iSBC 88/40 Board  
iSBC 88/45 Board

#### MASS STORAGE

iSBC 204 Flexible Diskette Controller  
iSBC 206 Flexible Disk Controller  
iSBC 208 Flexible Disk Controller  
iSBC 215A Winchester Disk Controller  
iSBC 215B Winchester Disk Controller  
iSBC 220 SMD Disk Controller  
iSBC 254 Bubble Memory Board

## MULTIMODULE™ BOARDS

iSBX 218 Flexible Disk Controller (when used with the iSBC 215 Controller)

iSBC 337 Numeric Data Processor

iSBX 351 Serial I/O Board

## CUSTOM iAPX 86, 88-BASED SYSTEMS REQUIREMENTS

8253 or 8254 Programmable Interval Timer

8259A Programmable Interrupt Controller

8251A USART or iSBX 351 board (when the Terminal Handler is configured into the system).

8087 Numeric Processor Extension (when NPX tasks are configured into the system).

## Reference Manuals (supplied)

**143238** — Introduction to the iRMX 80/88 Real-Time Multitasking Executives

**143241** — iRMX 88 Installation Instructions

**143232** — iRMX 88 Reference Manual

**142603** — iRMX 80/88 Interactive Configuration User's Guide

**142926** — Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems

**ORDERING INFORMATION**
**Part Number Description**

RMX 88	A licensed product which includes Nucleus, Terminal Handler, Free Space Manager, and I/O System object modules. Package also includes UDI-compatible Interactive Configuration Utility program for system generation and a complete set of manuals. Purchase price includes an iRMX 88 Customer Training Course credit.
RMX 88 ARO	Single Density ISIS media. Requires derivative work incorporation fee.
RMX 88 BRO	Double Density ISIS media. Requires derivative work incorporation fee.
RMX 88 DRO	Single Density RMX-86 media. Requires derivative work incorporation fee.

**Part Number Description**

RMX 88 ABY	Single Density ISIS media. Includes incorporation fee buyout.
RMX 88 BBY	Double Density ISIS media. Includes incorporation fee buyout.
RMX 88 DBY	Single Density RMX-86 media. Includes incorporation fee buyout.
RMX 88 AWX	One year Single Density ISIS media update service.
RMX 88 BWX	One year Double Density ISIS media update service.
RMX 88 DWX	One year Single Density RMX-88 media update service.
RMX 88 LST	Human readable source listings for iRMX 88 software.
RMX 88 LWX	Update service for human readable source listings.
RMX 88 RF	Incorporation fee.



## PRECONFIGURED iRMX™ 86 OPERATING SYSTEM

- Ready-to-run Preconfigured iRMX™ 86 Operating System for iSBC® systems
- Efficient realtime multitasking scheduler with 255 priority levels
- Complete support of 8087 numeric processor extension
- Direct support of independent software vendor compilers and applications
- Direct support for Intel on-target compilers and development tools
- Simple program load and debug with Bootstrap and Monitor in 2732A EPROMs
- Device drivers included for diskettes, Winchester hard disks, serial terminal interface, and parallel line printer
- A complete, high-performance, execution engine for UDI applications

The Intel Preconfigured iRMX 86 Operating System is a flexible, realtime, and multitasking system which is configured to run on a low-cost, iSBC 86-based hardware system. The iRMX 86 Operating System is designed to provide a structured and efficient environment for many time- and performance-critical applications such as factory automation, business data and text processing, medical electronics, data communications and process control. The Preconfigured System provides this environment without requiring specific hardware and software configurations. Based on the UDI software interface architecture for optional compilers and interpreters, the iRMX 86 PC System supports development of sophisticated applications using the target hardware. A ready-to-use comprehensive human interface provides advanced services including creating and maintaining a hierarchical file system, entering the debug monitor and backing-up diskette volumes.

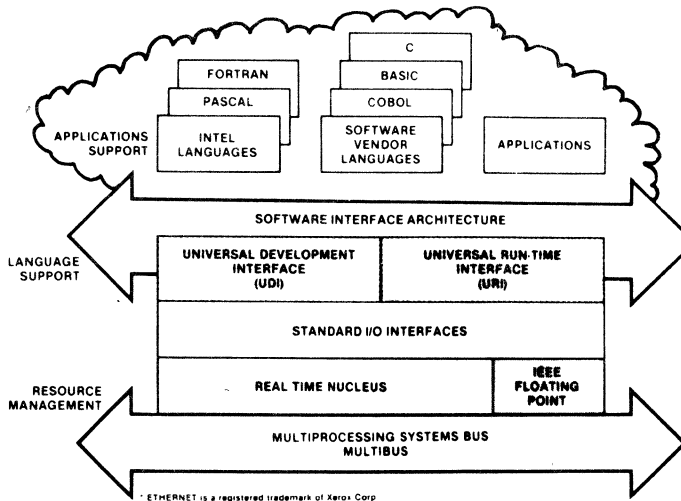


Figure 1. iRMX™ 86 PC Support for Standard Interfaces

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Intellec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI, µScope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1982

March, 1982  
Order Number: 210422-001

The Preconfigured iRMX 86 Operating System is a complete set of system software modules that are ready-to-run in a simple MULTIBUS system consisting of an iSBC 86 computer, memory, and a diskette controller board. All the features of the iRMX 86 Operating System are provided along with a bootstrap monitor to load the system diskette into the system.

The Preconfigured iRMX 86 System provides both implicit and explicit management of system resources. These resources include the processor's time and registers, up to one megabyte of system memory, independent interrupt sources, all input and output devices, as well as directory and data files contained on diskettes or 8" Winchester disks.

### FUNCTIONAL DESCRIPTION

In applications where computers are required to perform many functions simultaneously, the iRMX 86 Operating System provides a multiprogramming environment in which many independent, and optionally multitasking, applications may run. Each application environment may be treated separately to allow application programmers the flexibility to separately manage each application's resources. A complete description of the iRMX 86 Operating System can be found in the iRMX 86 Data Sheet (Order Number: 210330).

### User Commands

The iRMX 86 PC System provides a number of powerful tools necessary for the development of microcomputer applications. They are included on the system disk and brought into memory when needed to perform the functions listed in Table 1.

These commands are especially useful for managing user programs and data stored on diskettes.

### File Management

The iRMX 86 PC file management system allows users to access information on diskettes by referring to a file with its ASCII name. The names of files stored on a disk are catalogued in special files called directories. As directories are themselves named files, the iRMX 86 file system allows directories to contain the names of other directories. This leads to a hierarchical file structure as illustrated in Figure 2. This structure is useful for isolating file names of particular applications, and for tailoring the system's data to the requirements of users and applications sharing storage devices.

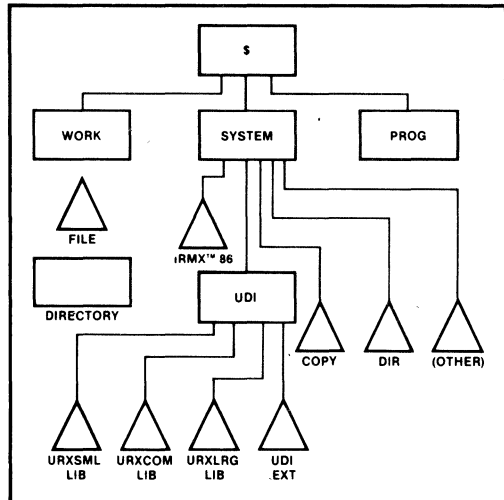


Figure 2. iRMX™ 86 PC System Disk Directory Tree

Table 1. iRMX™ 86 PC Commands

Command	Function
ATTACHDEVICE	Gives a logical name to a specific disk, CRT, or Printer device
BACKUP	Copy directories and files from one device to another
COPY	Copy one or more files to one or more destination files
CREATEDIR	Create a directory file to store the names of other files
DATE	Set the system calendar
DELETE	Delete a file or directory
DEBUG	Enter the System Monitor
DETACHDEVICE	Remove a device from the system
DIR	List the names, sizes, owners, etc. of the files contained in a directory
FORMAT	Prepare a new diskette volume for use
RENAME	Change the name of a file
RESTORE	Recreates a volume saved by BACKUP
SUBMIT	Start the processing of a series of commands stored in a file
TIME	Set the system time-of-day clock
VERIFY	Verify the structure of an iRMX 86 Named File volume, and check for possible disk data errors

Figure 2 also shows the structure of the directories on the iRMX 86 PC system diskette. It contains all the programs and commands that make up the iRMX 86 PC System. Users may add other files and directories anywhere in the structure. Whenever an operator makes a request to use one of these files, the System will search the appropriate directory tree in order to find the necessary information about the file's size, access rights, and specific location on the diskette. Applications may also refer to a specific file or group of files by specifying the directory from which to start the search.

### Standard Interfaces

The iRMX 86 PC System supports a group of 31 easy-to-use standard system calls known as the

Universal Development Interface (UDI). Figure 1 shows how this interface provides iRMX 86 systems the capability of using many compilers and language translators. These include the iAPX 86 and 88 Macro Assembler, and the PASCAL 86/88, PL/M 86/88, and FORTRAN 86/88 compilers available from Intel. They also include a number of other Intel development tools, and language translators and applications available from independent software vendors.

The standard UDI software interface establishes a path to future Intel software products and opens the door to a host of compilers, interpreters, and application programs available from independent software vendors. These UDI calls are easy-to-use and are listed in Table 2. A more complete list of all the system calls provided by the iRMX 86 PC System can be found in the iRMX 86 Data Sheet.

**Table 2. UDI System Calls**

System Call	Function Performed
<b>Memory Management:</b> DQ\$ALLOCATE DQ\$FREE DQ\$GET\$SIZE DQ\$RESERVE\$10\$MEMORY	Creates a segment of a specified size for use by the application. Returns the specified segment to the system. Returns the size of the specified segment. Reserves memory for use by I/O operations.
<b>File Management:</b> DQ\$ATTACH DQ\$CHANGE\$EXTENSION DQ\$CLOSE DQ\$CREATE DQ\$DELETE DQ\$DETACH DQ\$OPEN DQ\$READ DQ\$RENAME DQ\$SEEK DQ\$TRUNCATE DQ\$WRITE DQ\$FILE\$INFO DQ\$CHANGE\$ACCESS	Creates a connection to a specified file. Changes or adds an extension to a file name. Closes the specified file connection. Creates a Named File for use by the application. Deletes a Named File. Closes a file and deletes its connection. Opens a file for a particular type of access. Reads the next sequence of bytes from a file. Renames the specified Named File. Moves the current position pointer of a file. Truncates a file. Writes a sequence of bytes to a file. Returns information about the specified file. Changes the access rights of the specified file.
<b>Process Management:</b> DQ\$EXIT DQ\$GET\$CONNECTIONS\$STATUS DQ\$OVERLAY DQ\$SPECIAL	Exits from the current application job. Returns the current status of the specified file connection. Causes the specified overlay to be loaded. Performs special I/O related functions on terminals with special control features.
<b>Exception Handling:</b> DQ\$GET\$EXCEPTION\$HANDLER DQ\$DECODE\$EXCEPTION	Returns a pointer to the program currently being used to process errors. Returns a short description of the specified error code.



**Table 2. UDI System Calls (con't.)**

System Call	Function Performed
<b>Exception Handling (con't.)</b>	
DQ\$TRAP\$EXCEPTION	Identifies a custom exception processing program for a particular type of error.
DQ\$TRAP\$CC	Identifies a custom handler for processing CNTL/C keyboard inputs.
<b>Application Assistance:</b>	
DQ\$GET\$ARGUMENT	Returns the next argument from the character string used to invoke the application program.
DQ\$GET\$SYSTEM\$ID	Returns the name of the underlying operating system supporting the UDI.
DQ\$GET\$TIME	Returns the current time of day as kept by the underlying operating system.
DQ\$SWITCH\$BUFFER	Selects a new buffer from which to process commands.
DQ\$DECODE\$TIME	Returns date and time in ASCII characters.

### Simple System Start-Up

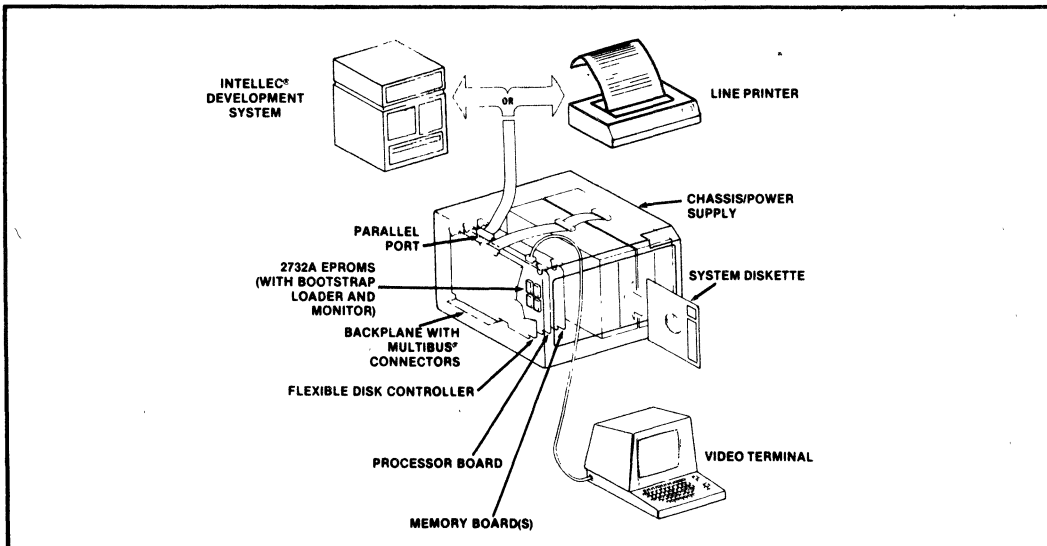
The iRMX 86 PC system includes a comprehensive Monitor and Bootstrap Loader in four 2732A EPROMs. These programs have been configured to support the hardware shown in Figure 3. As shown, the Monitor is capable of communicating with an Intellec Microcomputer Development System. This communications link can be used to transfer programs and data between an iRMX 86 System and the Intellec Development System.

This start-up system provides a perfect environment for the development and efficient execution of applications programs. When these programs require different I/O devices or a different software configuration, they can be moved to any other

iRMX 86 System directly. The iRMX 86 PC System includes a separate diskette with the complete set of iRMX 86 multitasking system call declarations for those programmers requiring more function than is supplied by the UDI.

### Debugging Aids

The iRMX 86 PC System includes a System Monitor that provides the capability of debugging one task at a time. The monitor includes instructions for examining and modifying the contents of all 8086 and 8087 registers, setting system breakpoints, single-stepping, examining and modifying system memory, executing CPU I/O, and disassembling program instructions.


**Figure 3. Hardware Configuration of PC System**

## SPECIFICATIONS

### Optional Intel® Software Products

- iRMX 86 Fully configurable iRMX 86 Realtime Operating System
- iRMX 860 iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, and Macro Assembler, Librarian, and the iRMX 86 Editor
- iRMX 861 PASCAL 86/88 Compiler for execution on iRMX 86 Systems
- iRMX 862 FORTRAN 86/88 Compiler for execution on iRMX 86 Systems
- iRMX 863 PL/M 86/88 Compiler for execution on iRMX 86 Systems
- iSBC 957B iAPX 86 System Monitor and Microcomputer Development System Communications Link

### Supported Hardware Products

#### iSBC® MULTIBUS® PRODUCTS

- iSBC 86/12A, 86/14, and 86/30 Single Board Computers
- iSBC 208 Flexible Disk Controller

#### PERIPHERAL DEVICE

- CRT** — RS232 at 9600 Baud
- Printer** — Centronics-type Parallel Interface
- Diskettes** — 2 to 4 Single- or Double-Density, Single- or Double-Sided

### Memory Requirements

- 200K Bytes to support applications less than 16K Bytes.
- 384K Bytes to support Intel's PASCAL 86 Compiler.
- 256K Bytes to support Microsoft's Basic Interpreter and a 32K Byte user program and data space.

### Reference Material

- iRMX 86 Operating System Data Sheet (210330)

- Getting Started with the iRMX 86 System (144340-001) (Included in PC System Package)
- Introduction to the iRMX 86 Operating System (9803124-03)
- iRMX 86 Installation Guide (9803125-04)
- iRMX 86 Configuration Guide (9803126-04)
- iRMX 86 NUCLEUS Reference Manual (9803122-03)
- iRMX 86 Terminal Handler Reference Manual (143324-01)
- iRMX 86 Debugger Reference Manual (143323-01)
- iRMX 86 Basic I/O System Reference Manual (9803123-04)
- iRMX 86 Loader Reference Manual (143318-01)
- iRMX 86 Extended I/O System Reference Manual (143318-001)
- iRMX 86 Human Interface Reference Manual (9803202-002)
- iRMX 86 System Programmer's Reference Manual (142721-003)
- Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-003)
- iRMX 86 Programming Techniques (142982-002)
- User's Guide for the iSBC 857B iAPX 86,88 Interface and Execution Package (143979-002)
- iRMX 86 Disk Verification Utility Reference Manual (144133-001)
- iRMX 86 Pocket Reference (142861-002)
- Edit Reference Manual (143587-001)
- Runtime Support Manual for iAPX 86,88 Applications (121776-001)
- Guide to Using iRMX 86 Languages (143907-001)
- Reference material may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

### Training Courses

- Introduction to the iRMX 86 Operating System
- iRMX 86 I/O System Concepts

## ORDERING INFORMATION

The iRMX 86 PC System is provided on a double-density, iRMX 86 compatible system diskette (format type E). The iRMX 86 PC System is shipped with a comprehensive users' manual ("Getting Started With The iRMX 86 System), Bootloader and Monitor EPROMs, and the complete iRMX 86 Interface Libraries contained on a second diskette. A full year of Intel Support Level D (Software Problem Report Service) is included. This Intel copyrighted system is licensed as a single-use software product as defined by Intel's Master Software Licenses.

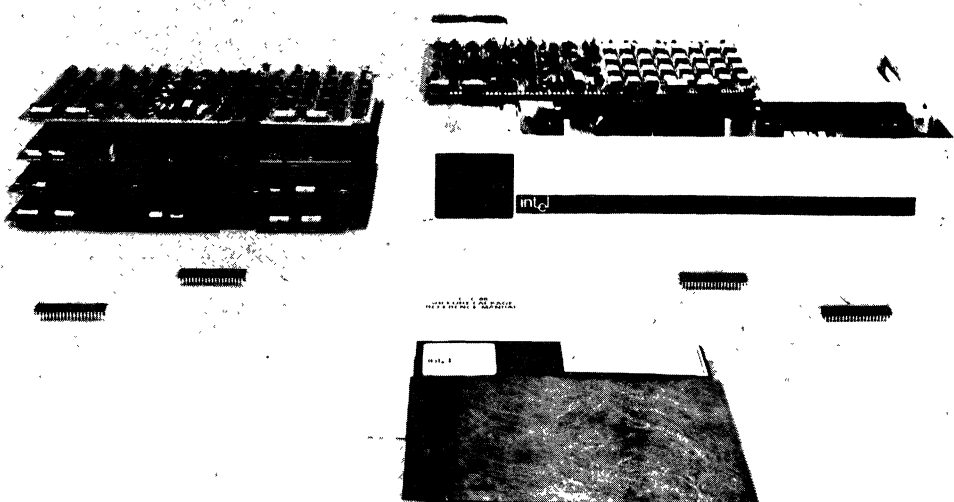
<b>Order Code</b>	<b>Description</b>
RMX 86PC E	Complete Preconfigured iRMX 86 Operating System with interface libraries, bootstrap monitor, and user documentation.



## iOSP™ 86 iAPX 86/30 AND iAPX 88/30 SUPPORT PACKAGE

- Development and run-time support for iAPX 86/30 and 88/30 Operating System Processors
- Total iRMX™ 86 Operating System software compatibility
- Extendable with iRMX™ 86 Operating System calls
- Compatible with Intel PL/M 86/88, PASCAL 86/88, FORTRAN 86/88, and iAPX 86/88 ASSEMBLER
- Supports (P)ROM or RAM based system
- Complete system initialization aids
- Complete system configuration aids
- OSP Interactive Configuration Utility

The Intel iOSP 86 Support Package for the iAPX 86/30 and 88/30 Operating System Processors contains a comprehensive set of easy-to-use tools necessary to develop (P)ROM or RAM-based applications that use the 80130 Operating System Firmware component. All of the system initialization and run-time facilities are provided in libraries that may be configured to specific requirements, and linked to application programs written in either iAPX 86 or iAPX 88 Assembler or a high level programming language such as PASCAL 86 and PL/M 86. The iOSP 86 Package provides users with the basic initialization and interface routines needed to build application software based on the fundamental operating system functions of the iAPX 86/30 and 88/30 Operating System Processors. The iOSP 86 Package also enables users to add higher level I/O functions from the fully compatible iRMX 86 Operating System, or to form custom, real-time systems.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, CREDIT, Index, Insite, Inteltec, Library Manager, Megachassis, Micromap, MULTIBUS, PROMPT, UPI,  $\mu$ Scope, Promware, MCS, ICE, iRMX, iSBC, iSBX, MULTIMODULE, iOSP and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

© INTEL CORPORATION, 1981

October, 1981  
Order Number: 210236-001

## FUNCTIONAL DESCRIPTION

The iAPX 86/30 and iAPX 88/30 Operating System Processors (OSPs) provide an easy-to-use foundation on which many real-time applications may be built. They provide the functions and system support needed to implement both simple and complex applications that require multiple tasks to run concurrently (see Figure 1). These services are made possible by the addition of the five new data types integrated into the 80130 Operating System Firmware (OSF) component. The 80130 OSF extends the basic data types of the CPU (integer, byte, character, etc.) by adding new system data types (JOB's, TASK's, MAILBOX's, SEGMENT's, and REGION's), and extensive timer, interrupt, memory, and error management designed to give real-time response to multitasking and multi-programming applications. As shown in the second half of the figure, other operating system functions such as mass storage I/O services and an easy-to-use Human Interface can be added easily, by using modules from the complete operating system services of the iRMX 86 Operating System. The iOSP 86 Support Package provides both an interface between application software and the Operating System Processors, and development tools designed to make the implementation and initialization of real-time, multitasking systems much simpler.

The iOSP 86 Support Package provides system developers with the configuration options necessary to tailor the iAPX 86/30 and 88/30 Operating System Processors to custom applications. Central to the entire configuration process is the OSP Interactive Configuration Utility (OSPICU). This utility is an easy-to-use tool which allows you to make configuration decisions by responding to screen-oriented displays. Using the ICU, users can form easy-to-use initializa-

tion routines, and support code. The interface libraries form a simple interface between application software and the operating system primitives of the 80130 OSF component. The various configuration options include:

### Memory and I/O Addressing

The 80130 OSF requires a 16K byte block of memory address space to be reserved for accessing internal functions. The iOSP 86 Support Package is used to specify the base address of the 80130 and the beginning of the initialization routines.

All Interrupt and Timer management of the OSF is controlled via a reserved 16-byte I/O address block that may be selected by the user. In addition, from 1 to 7 slave 8259A interrupt controllers can be specified in order to provide the system with up to 57 priority interrupt sources. The OSF baud rate generator may also be configured to support an optional terminal interface.

### Extending the 80130 OSF

The 80130 OSF allows users to add their own operating system extensions. These extensions may take advantage of the detailed and efficient intertask communication and synchronization primitives already provided by the 80130, and/or may utilize custom functions tailored to specific applications. The Support Package also enables users to extend the OSF with the extensive services of Intel's iRMX 86 Operating System, thereby allowing applications to grow without having to change or alter application software already written, or having to write other operating system software. Use of the 80130 with the iRMX 86 Operating system greatly reduces the amount of memory needed for the iRMX 86 Nucleus layer, and enables applications to take advantage of the increased

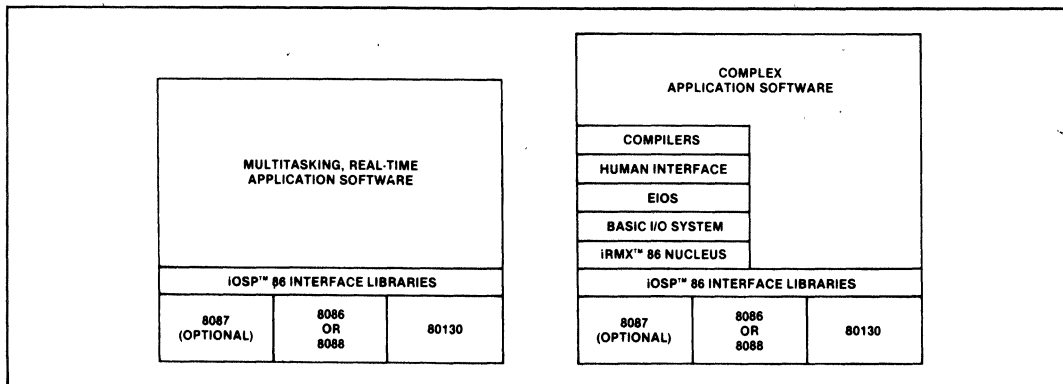


Figure 1. Structure of Typical Systems

performance and reduced size requirements inherent in the iAPX 86/30 and 88/30 VLSI Operating System Processors. As each of the services provided by the 80130 is completely iRMX 86-compatible, applications have an automatic upward path to support complete file systems and multiple processor environments.

### Application Interfaces

Two interface libraries are included in the iOSP 86 Support Package. The first allows programmers to write application software modules in the Compact Model of computation supported by Intel's compilers. The second provides an interface to program segments written in either the Medium or Large Models.

The interface libraries provide the means of accessing all of the primitives supported by the Operating System Processors. With this interface, and all the memory management primitives of the OSPs, applications have full access to 1M byte of memory, and all of the addressing modes of the CPU.

The iAPX 86/30 and 88/30 OSPs allow applications to take full advantage of the Compact, Medium, and Large models of computation afforded by the segment model of the CPU's.

These libraries are fully compatible with object modules produced by the MACRO 86/88 Assembler, and the PASCAL 86/88 and FORTRAN 86/88 and PL/M 86/88 Compilers.

### Application Initialization

The iOSP 86 Support Package provides for the configuration of the system Root JOB, and all user application JOB's that require initialization when the system is started. The user may also specify the

configuration of the interrupt system (including slave 8259A interrupt controllers) and the clock rate used for system timing. These choices are automatically programmed into the various devices when the system is initialized.

### Parameter Validation

Parameter validation is a configurable option of an OSP-based system. The OSP can check the Parameters of the Primitive that you invoke either on a system-wide basis or on a per job basis.

### Operating System Calls

The 80130 OSF performs a total of 37 operating system primitives all of which are completely compatible with the equivalent iRMX 86 Operating System calls. The iOSP 86 Support Package provides user-level interfaces to these primitives to enable applications to create, delete, control, and exchange the new data types provided by the 80130 OSF. In general, these interfaces allow application software to manage all of the resources of an iAPX 86/30 or 88/30 OSP (and an optional 8087 Numeric Processor Extension) system via any of the 37 system calls shown in Figure 2.

### Required Development Hardware

Use of the iOSP 86 Support Package requires an Intel MDS Development System which supports Series III (either single or double density diskettes) or any iRMX 86 system supporting a standard floppy diskette drive and the iRMX 860 Assembler, Linker, and Locator Package. Use of the 80130 requires only a minimal system including either the iAPX 86/30 or 88/30 Operating System Processor, and enough system memory to contain the application programs and initialization and interface software provided in the iOSP 86 Support Package.

JOB GROUP	SEGMENT GROUP	INTERRUPT MANAGEMENT GROUP
CALL RQ\$CREATE\$JOB	CALL RQ\$CREATE\$SEGMENT	CALL RQ\$SET\$OS\$EXTENSION
	CALL RQ\$DELETE\$SEGMENT	CALL RQ\$SET\$INTERRUPT
<b>TASK GROUP</b>	<b>REGION GROUP</b>	CALL RQ\$ENTER\$INTERRUPT
CALL RQ\$CREATE\$TASK	CALL RQ\$CREATE\$REGION	CALL RQ\$EXIT\$INTERRUPT
CALL RQ\$DELETE\$TASK	CALL RQ\$DELETE\$REGION	CALL RQ\$WAIT\$INTERRUPT
CALL RQ\$SUSPEND\$TASK	CALL RQ\$SEND\$CONTROL	CALL RQ\$SIGNAL\$INTERRUPT
CALL RQ\$RESUME\$TASK	CALL RQ\$RECEIVE\$CONTROL	CALL RQ\$RESET\$INTERRUPT
CALL RQ\$SLEEP	CALL RQ\$ACCEPT\$CONTROL	CALL RQ\$ENABLE
CALL RQ\$GET\$TASK\$TOKENS		CALL RQ\$DISABLE
CALL RQ\$SET\$PRIORITY	<b>OBJECT MANAGEMENT GROUP</b>	CALL RQ\$GET\$LEVEL
<b>MAILBOX GROUP</b>	CATALOG OBJECT	<b>ERROR CONTROL GROUP</b>
CALL RQ\$CREATE\$MAILBOX	LOOKUP OBJECT	CALL RQ\$SET\$EXCEPTION
CALL RQ\$DELETE\$MAILBOX	CALL RQ\$DISABLE\$DELETION	CALL RQ\$SIGNAL\$EXCEPTION
CALL RQ\$SEND\$MESSAGE	CALL RQ\$ENABLE\$DELETION	CALL RQ\$GET\$EXCEPTION
CALL RQ\$RECEIVE\$MESSAGE	CALL RQ\$GET\$TYPE	

**Figure 2. Operating System Primitives**

**ORDERING INFORMATION**

Each of the ordering options listed below include all the necessary initialization and interface procedures needed to use the iAPX 86/30 and iAPX 88/30 Operating System Processors. Purchase of the iOSP 86 Package requires verification of an Intel Master Software License. Each package also includes an iOSP 86 User's Manual (Document Number 145393-001), and a one-year update service.

<b>Part Number</b>	<b>Description</b>
OSP 86 A	iOSP 86 Support Package contained on an ISIS-II compatible, single density diskette.
OSP 86 B	iOSP 86 Support Package contained on an ISIS-II compatible, double density diskette.
OSP 86 E	iOSP 86 Support Package contained on an iRMX 86 format, double density diskette.

---



## iMMX™ 800 MULTIBUS® MESSAGE EXCHANGE SOFTWARE

- Supports use of multiple processors on the MULTIBUS® system bus
- Increases total system throughput
- Implements Intel-standard multi-processing protocol
- Supports combination of 8- and 16-bit boards in one design
- Helps solve critical response-time problems
- Includes Ethernet device driver
- Provides hardware-independent application interface
- Supports iRMX™ 80, iRMX™ 86, and iRMX™ 88 applications

The iMMX MULTIBUS Message Exchange provides an Open Multiprocessing System. It allows tasks-executing on separate processors to communicate by sending messages. By providing an off-the-shelf implementation of the MULTIBUS Interprocessor Protocol, it cuts many man-months off the typical development schedule. Loosely coupled multiprocessing makes multiple-microcomputer applications simple: programs request message transfer by means of a small set of systems calls — the iMMX software takes care of providing reliable message transfer via shared MULTIBUS memory.

The iMMX product is open to high-performance applications, encourages modular design practices, and supports multiple operating systems. By making it easy to use multiple processors it increases total system throughput and allows processing power to be optimized for both I/O handling and data processing. Once an initial application design is complete, it can be easily enhanced by adding new tasks and/or processors. The iMMX product allows the engineer to choose from a wide range of 8- and 16-bit iSBC microcomputers — from the iSBC 80/24 board to the iSBC 86/30 single board computer. The net result is a combination of performance and flexibility that meets the needs of a diverse set of multiple-micro-computer applications.

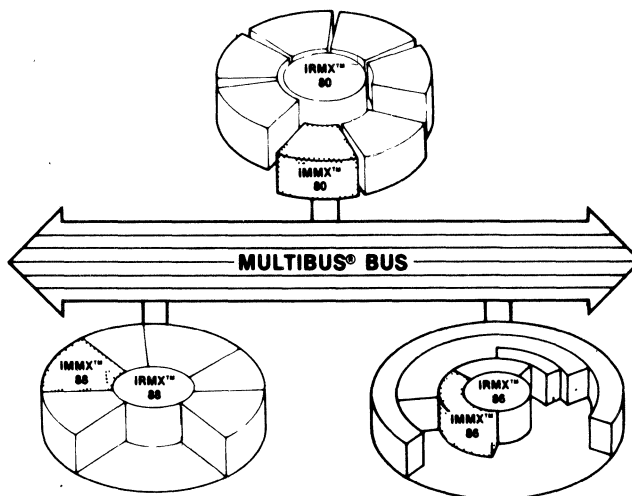


Figure 1. iMMX 800 Real-Time Executive Interface

The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. \* ETHERNET is a trademark of Xerox Corporation.

© INTEL CORPORATION, 1982

August, 1982  
Order Number: 143875-003



## FUNCTIONAL DESCRIPTION

### Open Multiprocessing System

#### OPEN TO HIGH PERFORMANCE APPLICATIONS

The iMMX supports high performance applications in two ways. First, it increases the total system throughput by allowing multiple processors to be easily incorporated in an application. Second, critical response time requirements can be met by placing computing power close to each critical input. Application programmers can concentrate on added-value functions while iMMX software takes care of variable length transfers, shared memory management, mutual exclusion, interprocessor interrupts, and hardware details.

#### OPEN TO MODULAR DESIGN

By supporting modular design, iMMX software provides four key benefits. First, each hardware module can be selected or designed according to needs of a subsystem; the iMMX 800 software takes care of the integration. Second a whole range of products can be created from a few hardware/software modules. Third, the breadth of products available for the industry-standard MULTIBUS dramatically reduces the amount of custom-design work required to complete a system. Finally, as customers, new markets, or competition re-

quires, performance can be enhanced or new features can be added by adding new modules.

#### OPEN TO MULTIPLE OPERATING SYSTEMS

iMMX software supports both standard Intel iRMX operating systems and custom systems. Off-the-shelf support is provided for iRMX 80, iRMX 86, and iRMX 88 applications — allowing the engineer to choose the best match for each problem. In addition, the underlying MULTIBUS Interprocessor Protocol (MIP) is completely specified so that custom operating systems and other subsystems can be integrated with iRMX-based subsystems.

#### Loosely-Coupled Multiprocessing

The iMMX 800 software supports loosely-coupled multiprocessor systems. The software interface is composed of simple, easy-to-use, modules. By supporting the addressing, data transfer, control, and memory management functions, the software as shown in Figure 2, divides the operation into three functions: the virtual interface, the logical protocol, and the physical protocol.

The **virtual interface** is the application task's access to the iMMX services. Using this interface, a task can request a connection to a particular port. Using the connection, the task can request that messages be transferred to the task(s) that are requesting messages from the same port.

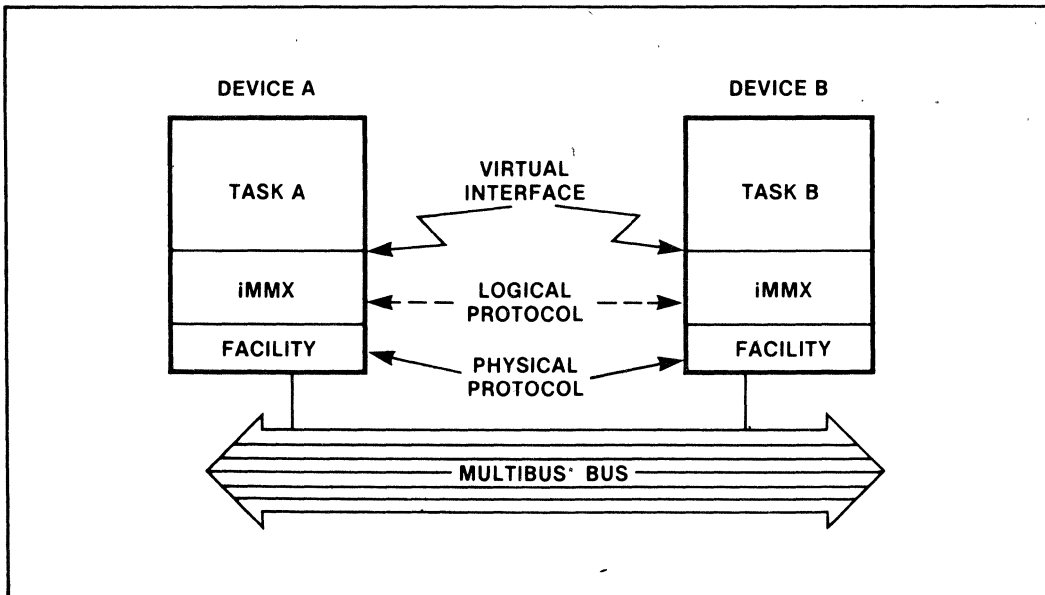


Figure 2. Inter-Device Task-to-Task Communications

The **logical protocol** supports a message manager function. The Message Manager prepares the message for delivery to a specific destination port based on the connection specified. In addition, the logical protocol returns status information about the transfer.

The **physical protocol** is implemented by VLSI and associated circuitry. This level of protocol includes data flow control, mutual exclusion mechanics, address recognition and interactive signalling requirements.

iMMX software supports four different inter-device signalling mechanisms: MULTIBUS interrupts, memory-mapped interrupts, I/O-port-mapped interrupts and polling.

### iRMX™ Uniform Interface

The iMMX 800 software provides a uniform interface across all iRMX based software environments. The iMMX software services are provided as a set of tasks, system procedures, and interrupt drives.

Support is supplied for the iAPX 86/88-based microcomputers that support the iRMX 86 and the iRMX 88 Operating Systems. In addition, software support is provided Intel 8085-based products via the iRMX 80 Operating System. Table 1 shows the code size requirements of each of the iMMX configurations. Table 2 gives a complete list of the boards that are supported.

**Table 1. iMMX 800 Software Memory Requirements**

Executive	K Bytes
iRMX™ 80 Operating System	3.7K Bytes
iRMX™ 88 Operating System 128K support 1MB support "Compact"	4.8K Bytes
"Large"	5.5K Bytes
	6.3K Bytes
iRMX™ 86 Operating System	6.6K Bytes

**Table 2. Supported Single Board Computers**

iRMX™ 80 Operating System	iRMX™ 88 Operating System	iRMX™ 86 Operating System
iSBC® 80/24	iSBC® 86/05	iSBC® 86/05
iSBC® 80/30	iSBC® 86/12A	iSBC® 86/12A
iSBC® 544	iSBC® 86/14	iSBC® 86/14
iSBC® 569	iSBC® 86/30	iSBC® 86/30
	iSBC® 88/25	iSBC® 88/25
	iSBC® 88/40	iSBC® 88/40
	iSBC® 88/45	iSBC® 88/45

### Message Transfer Mechanism

iMMX multiprocessing is based on a message-passing model. Tasks on each processor communicate with each other by sending and receiving messages to and from ports.

Table 3 shows five iMMX system calls: Find Port, Activate Port, Transfer Message, Deactivate Port and Lose Port.

### Shared Memory Space

The iMMX software manages the message passing in such a way that a task that receives a message can address it even if the message originated in the private memory of another processor. This means that, when appropriate, the message is copied into memory that can be addressed by the receiver.

### Interprocessor Protocol Architecture

The Intel MULTIBUS Interprocessor Protocol (MIP) specifies an architecture by which processes executing on different MULTIBUS single board computers can communicate with one another in a reliable, controlled manner within that system. A system can consist of a heterogeneous set of processors, executing a heterogeneous set of real-time executives and application software.

Based on a simple internal structure, the MIP specification defines a functional consistency across several product lines and provides the

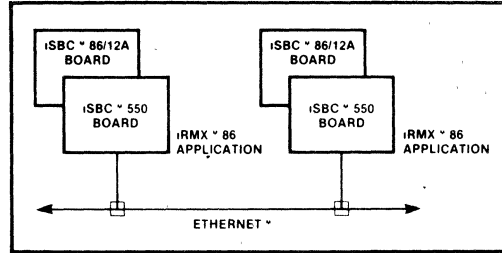
**Table 3. System Calls**

Function	Name	Description
FIND PORT	CQFIND	<b>Find</b> a port and return a connection-ID
ACTIVATE PORT	CQACTV	<b>Activate</b> a port for receiving messages from other tasks
TRANSFER MESSAGE	CQXFER	<b>Transfer</b> a message to a port identified by the connection-ID.
DEACTIVATE PORT	CQDACT	<b>Deactivate</b> port. Further messages are returned to the sender
LOSE	CQLOSE	<b>Loses</b> a connection to a port

means to support efficient operation in multiple processor environments.

**Ethernet Device Driver**

The iMMX 800 package provides an iSBC 550 Ethernet Communications Controller device driver. This device driver uses iMMX routines to communicate to the iSBC 550 controller (see Figure 3). This same approach can be used to write other iRMX 88 and 86 device drivers.



**Figure 3. Ethernet Communications**

**SPECIFICATIONS**

**iSBC™ Supported Hardware**

**SINGLE BOARD COMPUTERS**

- iSBC 80/24
- iSBC 80/30
- iSBC 86/05
- iSBC 86/12A
- iSBC 86/14
- iSBC 86/30
- iSBC 88/25

- iSBC 88/40
- iSBC 88/45

**INTELLIGENT CONTROLLERS**

- iSBC 544 (Communications)
- iSBC 569 (Digital)
- iSBC 550 (Communications)  
via Ethernet driver

**Reference Manual (Supplied)**

- iMMX 800 MULTIBUS Message Exchange Reference Manual

**ORDERING INFORMATION**

**Description**

The iMMX 800 MULTIBUS Message Exchange Software is a licensed product that provides users of Intel Single Board Computers using the iRMX 80, iRMX 86, and iRMX 88 Operating Systems a standardized, memory-based, task-to-task communication protocol. This protocol provides the fundamental capabilities needed to exchange data between multiple 8-bit and 16-bit microcomputers residing on the same MULTIBUS system bus.

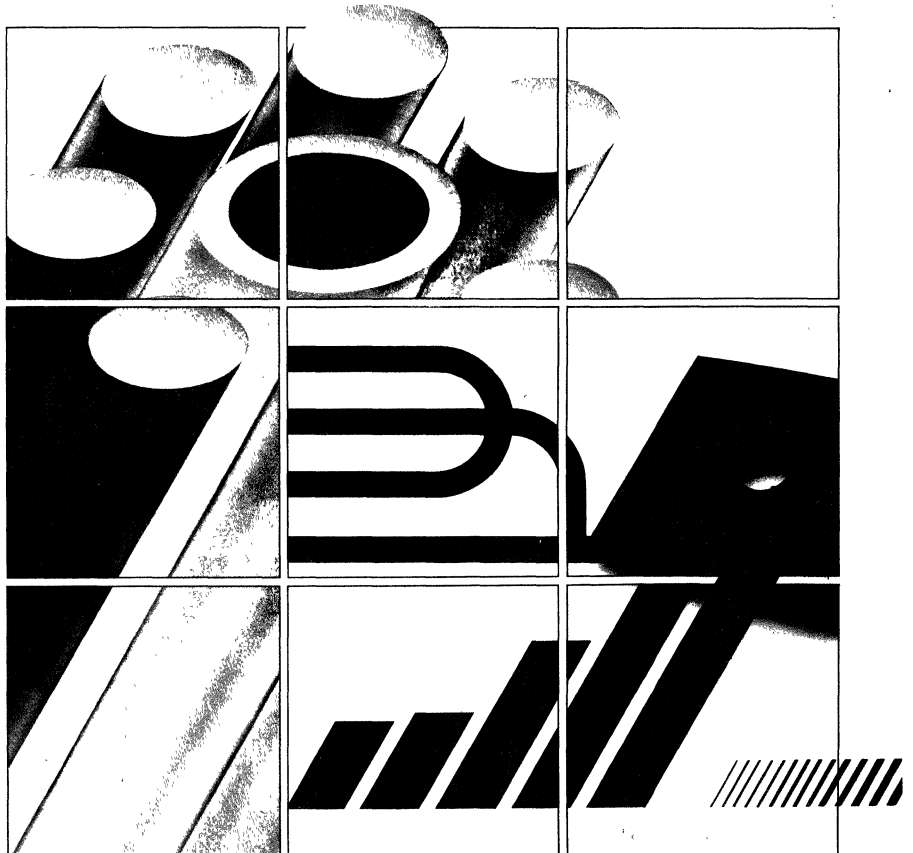
**Part Number Description**

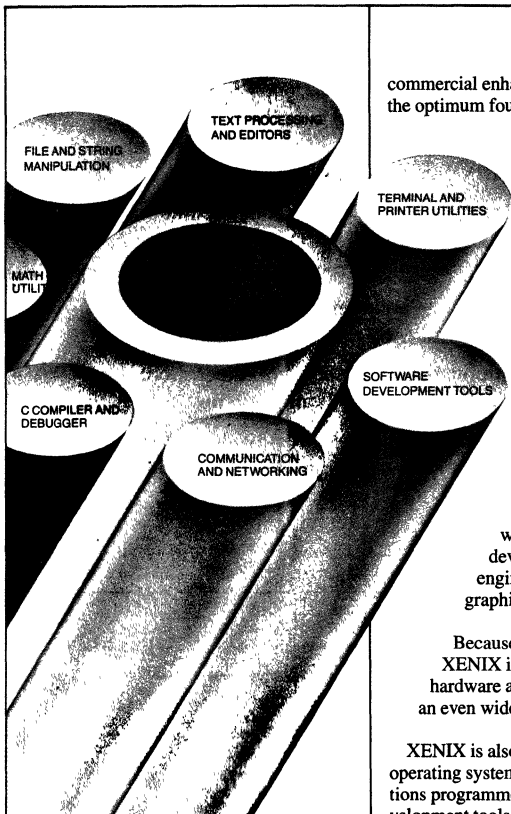
- MMX 800 ARO Single Density Media. Requires incorporation fee for each derivative work.
- MMX 800 BRO Double Density Media. Requires incorporation fee for each derivative work.

- MMX 800 ABY Single Density Media. Includes incorporation fee buyout.
- MMX 800 BBY Double Density Media. Includes incorporation fee buyout.
- MMX 800 AWX Single Density Media. Update service for an additional year.
- MMX 800 BWX Double Density Media. Update service for an additional year.
- MMX 800 LST Human readable source listings for the iMMX 800 software modules.
- MMX 800 LWX Extends source listing updates for an additional year.

## **XENIX\* 286 OPERATING SYSTEMS**

- Fully licensed version of the UNIX† operating system optimized for the Intel iAPX 286 processor
- Fastest microprocessor implementation of UNIX, fastest floating point performance on a microprocessor
- Important commercial OEM enhancements
- Supports multiple levels of integration: components, boards and systems
- Supported by Intel's worldwide post-sales service and support organizations





FILE AND STRING  
MANIPULATION

TEXT PROCESSING  
AND EDITORS

TERMINAL AND  
PRINTER UTILITIES

MATH  
UTILITIES

C COMPILER AND  
DEBUGGER

SOFTWARE  
DEVELOPMENT TOOLS

COMMUNICATION  
AND NETWORKING

**A XENIX Operating System Especially for the iAPX 286**

Intel's XENIX 286 Operating System is a fully licensed derivation of Bell Laboratories' Version 7 UNIX operating system for the iAPX 286 processor. XENIX 286 includes not only all the functionality of UNIX Version 7, but powerful enhancements from Microsoft and Intel that meet the needs of the commercial OEM (Original Equipment Manufacturer).

**The Best Foundation for Building OEM Solutions**

XENIX 286 provides the OEM with a complete software base on which to build value-added functionality. It includes the operating system, the C language, text processors, development tools, system accounting and security features, and

commercial enhancements that make it the optimum foundation for OEM application software solutions.

**XENIX: Portable, Flexible, Powerful**

XENIX has become the industry-standard microcomputer operating system for interactive, multi-user applications. It has gained wide popularity in applications such as distributed data processing, business data processing, word processing, software development, scientific and engineering applications, and graphics.

Because of its standardization, XENIX is portable to a variety of hardware and therefore able to run an even wider variety of software.

XENIX is also an extremely powerful operating system, providing the applications programmer with a wealth of development tools and utilities for bringing OEM products to market quickly.

**XENIX 286: Faster than any other UNIX on a Micro**

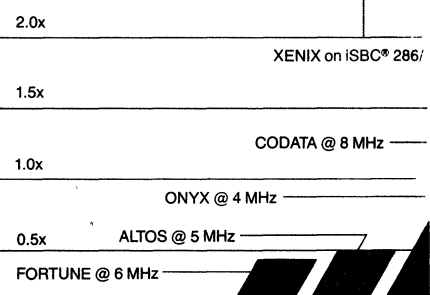
XENIX 286 stands head and shoulders above other microprocessor versions of UNIX, because it runs on the fastest, most advanced microprocessor on the

market: the Intel iAPX 286. As the first UNIX operating system derivative optimized for the iAPX 286, XENIX 286 alone can take full advantage of the 80286's unique features:

**On-chip memory management and protection** is a key advantage of XENIX 286 over other microprocessor UNIX implementations. On-chip memory management reduces the overhead in accessing system memory as compared to the usual separate memory management unit. With memory management functionality right on the chip, the operating system works more smoothly and efficiently.

**Advanced microprocessor architecture** provides pipeline processing, wherein a continual flow of instructions is kept in the CPU queue, results in throughput several times faster than the fastest competing microprocessor.

**Fast floating point processing** is due to XENIX 286 support of the Intel iAPX 287 math coprocessor. Floating point processing delivers throughput that is an order of magnitude faster than non-floating point processing. Extra high processing speeds are needed in applications such as data base processing, commercial data reduction and graphics.



**Faster, More Reliable Still When Teamed with Other Intel Systems Components**

The throughput enhancements in the XENIX 286 software are pushed to even greater speeds by special hardware architecture in Intel's systems and board products.

### MULTIBUS® System Architecture

is the industry-standard system bus. It accommodates any of the special-purpose Intel iSBC® boards, as well as standard peripherals, for easy system expansion.

### iLBX™ (Local Bus Exchange)

is an Intel hardware innovation that increases the amount of local memory accessible by the operating system to significantly improve system throughput.

### Error Correction Circuitry (ECC)

automatically detects and corrects soft errors in RAM. This on-board, self-correction facility reduces errors and further underscores data integrity.

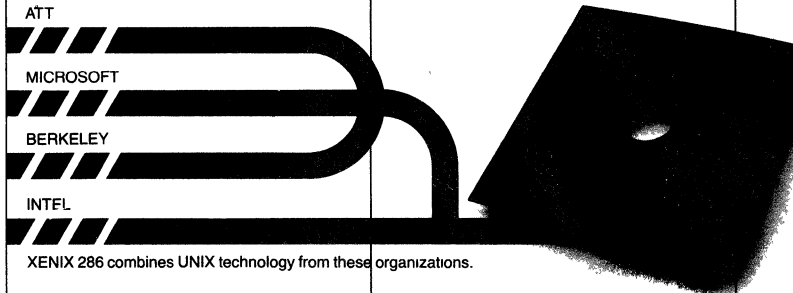
### A Faster Operating System Means Market Leadership

The combination of the industry's most widely accepted operating system for multi-user, interactive applications with the industry's fastest and most advanced microprocessor gives the OEM a far superior price/performance ratio than is

@ 5 MHz

See Intel benchmark series order no. 230676-001

available through other options. The result for the OEM: market leadership due to the ability to more attractively price products based on superior performance.



### XENIX 286: The Best of Everything

The XENIX 286 Operating System contains the best of many vendors' UNIX/XENIX development efforts during the last ten years (see Figure above). We have taken the best features of many UNIX versions—ease of use, flexibility, performance, security, reliability—and added our own enhancements (not the least of which is compatibility with the iAPX 286) to make XENIX 286 the optimum software development tool for the commercial OEM.

### Superior Data Reliability and Integrity

XENIX 286 contains enhancements to provide extremely high data reliability and integrity, particularly important to the OEM who is adding value to a system product. The following enhancements in XENIX 286 contribute to uniformly reliable data at all stages of application development.

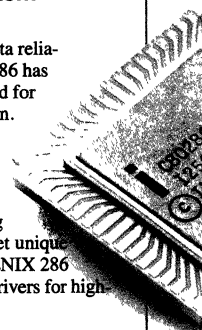
**Automatic disk recovery** is an improvement of the UNIX file system that allows automatic recovery of the file system in the event of unexpected system shutdown.

**Record and file locks** arbitrate multiple-access requests to the same record or file, allowing the programmer to extend locks to a single record, group of records or the entire file. This is important in multi-user applications to prevent two or more users accessing and updating the same information simultaneously.

**XENIX System Analysis Test (XSAT)** is a complete hardware-software diagnostic package included with all Intel integrated system products. XSAT provides a total analysis of a XENIX-based system, ensuring reliability even after the OEM configures new drivers into the system.

### Tools for Easy System Configuration

In addition to increased data reliability measures, XENIX 286 has been functionally enhanced for easier system configuration. An interactive configuration utility allows the user to specify device drivers, disk buffers, memory size, etc., making it easy for the OEM to meet unique design requirements. XENIX 286 includes over six device drivers for high-speed controllers.



## Friendlier Interface

The standard UNIX human interface has been enhanced in XENIX 286, with the addition of vi, a full-screen editor, for easier and faster application development.

The XENIX C shell augments the capabilities of the standard UNIX shell with the ability to maintain histories of invoked processes and provide the alias feature, saving re-keying of often-used commands.

## Intel's Open Systems Approach

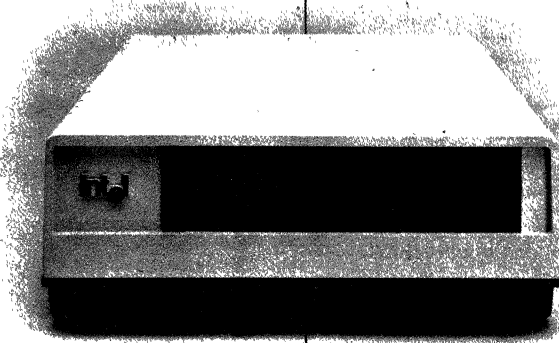
Intel believes that system components—hardware or software—should be fully compatible with other family members at any level of integration and open to future VLSI advancements. XENIX 286 was designed to be part of the Open Systems concept.

## Portability from Chip to Board to System

Intel's XENIX 286 Operating System is available for and fully compatible across Intel component, board and system designs, something that no other XENIX version offers.

Such portability gives OEMs the flexibility to choose the most appropriate and profitable level of integration for their applications. Component-level integration allows the OEM to meet unique design requirements; board and system-level integration afford reduced time to market.

There is no loss in software development investment as your needs change, since you can port XENIX-based applications from the chip to the system level or even from one Intel processor to another. For instance, code developed on XENIX 86 can be fully ported to a XENIX 286 system.



## Open to Still Greater Configurability through Third-Party Software and Hardware

XENIX 286 users can tap into an extensive base of existing third-party languages and application packages for almost endless versatility in system configurability. There are hundreds of such packages available today with many more on the way.

## Worldwide Support and Service

XENIX 286 customers can take advantage of Intel's worldwide staff of trained hardware and software engineers in contracting for application design assistance. A liberal warranty, including software updates and a technical newsletter, follows the sale. Once the warranty expires customers can choose from a variety of support contracts.

Intel offers complete training on the XENIX 286 Operating System as well as the iAPX 286 processor and associated hardware.

## Intel, The Technological Leader...

Intel is committed to pushing the frontiers of VLSI design to their ultimate limits. In the process, we move our customers along the technology curve

without interruptions in application development or expensive mid-stream architecture changes.

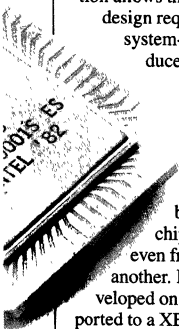
Intel started the micro revolution with the 4004 and has been the market leader with every generation of advanced processors since.

Systems and system software are a natural for us: who better knows the pieces and how to make them work together?

## ...In Total Solutions

The XENIX 286 Operating System fully exploits the iAPX 286, the fastest and most sophisticated microprocessor on the market. No other processor/operating system combination will give OEMs a faster and more economical path to getting systems and applications on the market.

Intel has always been first with the latest and most advanced VLSI and now with system software tailor-made for Intel VLSI. Because we're there first, our customers are first in their respective markets with state-of-the-art OEM and end-user products.



# XENIX 286

## Specifications

The XENIX 286 Operating System includes the following utilities, commands and subroutines:

System Administration	Documentation	File and String Manipulation	Math Utilities and Sub-Routines	Software Development	System Status	Text Processing and Editors
Boot Utilities	learn	basename	abs	Libraries	asktime	checker
UNIX code	manuals	cat	cabs	adb	date	col
ac		cd	ceil	arcv	df	creek
accton	<b>Graphics</b>	chgrp	cos	awk	du	deroff
ar	curses	chmod	cosh	ctags	file	ed
clri	graph	chown	dc	false	iostat	eqn
config	plot	cmp	exp	gets	l	ex
dcheck	spline	comm	fabs	ld	lc	look
dump		copy	floor	lorder	ls	ms
dumpdir	<b>Language</b>	cp	hypot	make	ps	negn
finger	as	crypt	log	mkstr	pwd	nroff
fsck	bc	dd	rand	nm	pstat	prep
haltsys	cb	diff3	sin	od	quot	ptx
icheck	cc	egrep	sinh	printenv	tty	refer
mkconf	lex	fgrep	srand	prof	who	rev
login	lint	grep	tan	SCCS +		sed
mkfs	m4	head*	tanh	size		spell
mknod	ranlib	ln		strings	<b>Terminal and Printer Utilities</b>	t300
mount	ratfor	mkdir	<b>Program Execution</b>	strip	disable	t300s
ncheck		mknod	at	time	enable	t45
newgrp	<b>Communication and Networking</b>	more*	chon	tr	lpr	tbl
passwd	calendar	mv	csh*	truct	pr	troff
restore	cu	rm	echo	true	stty	typo
sa	mail	rmdir	expr	units	tabs	vi*
sddate	mesg	sed	kill	xstr	termcap	
settime	rmail	sort	nice	yacc	tset	<b>Miscellaneous</b>
shutdown	uucp	split	nohup	yes	vpr	backgammon
su	uux	tail*	read			cal
sync	wall	tr	sh			fortune
tar	write	tsort	sleep			hangman
touch		uniq	tee			quiz
tp		wc	test			scms
umount			wait			units
XSAT °						wump

° Intel XENIX Operating System Enhancement

\* Berkeley UNIX 4.1 BSD Enhancement

+ Bell Laboratories UNIX System III Enhancement

### XENIX 286 includes support for the following Intel Systems, single board computers and processors.

- System 286/310
- System 286/380
- iSBC® 286/10 Processor Board
  - 16mb of addressing
  - On-chip memory protection
- New CX Series RAM board
  - ECC (Error Correction Circuitry)
  - iLBX™ (Local Bus Extension)
- iSBC 215 Winchester Controller
- iSBX 218 Floppy Controller
- iSBC 534 Serial I/O Expansion Board
- iSBC 544 Intelligent Serial I/O Expansion Board
- 80286 Central Processor
- 80287 Fast Floating Point Processor

### Documentation

XENIX Operating System Documentation Includes:

- XENIX Fundamentals
- XENIX Installation Guide
- XENIX Operating Guide
- XENIX Reference Manual
- XENIX Software Development Manual
- XENIX Text Processing Manual

### Industry Standard Text Books

*The C Programming Language*, Kernigan & Ritchie  
*A User Guide to the UNIX System*, Yates and Thomas





## Ordering Information

XNX 286 H	XENIX Object Software (8" double side, double density)
XNX 286K	XENIX Object Software (5¼" double-sided, double density)
XNX 286 RO	Software License Rights Extension
XNX 286 RF	Software Incorporation Fee
173258	XENIX Documentation Package
CTW 14PP	XENIX Customer Training
SPRTECHREP	XENIX Support Subscription Services
HOTLINE	XENIX Hotline Phone Service
SP86 330 XINSTALL	XENIX Software Installation
CONSULT-FIELD	XENIX Onsite Field Consulting
CONSULT-LT	XENIX Onsite Field Consulting for extended time periods.

March 1982

**Using Operating System  
Firmware Components  
to Simplify Hardware  
and Software Design**

**John Wharton**  
Microprocessor Applications

## INTRODUCTION

Intel recently introduced a new set of extensions to its microprocessor product line. The iAPX 86/30 and iAPX88/30 Operating System Processors (OSPs) augment the general-purpose instruction set of the well-known 8086/8088 architecture to include common, real-time, operating system capabilities. A single device, the 80130 Operating System Firmware component (OSF), now provides hardware support for functions previously relegated to software.

The 80130 introduces new concepts in the areas of both hardware and software. At first glance, traditional component-level hardware designers could feel somewhat intimidated by the esoteric concepts and unfamiliar buzzwords encountered in the software world. Even the experts in conventional operating system (OS) design may initially find it strange that what used to be "soft" software routines are now cast in silicon.

This application note is intended for readers at both levels. The first section reviews the development of processor extensions in general and operating system firmware in particular. Later sections should help you understand what a real-time operating system can do, how the 80130 provides these capabilities, and how to

design system hardware and software to take advantage of such features.

The note also documents a complete (albeit simple) system, including schematics and listings. The reader may wish to reconstruct this system to get started with OSFs. Finally, a step-by-step description of the so-called "configuration" process shows how physical system parameters are incorporated into the software as the software is "installed" in memory. Throughout the note are a number of "exercises"—questions relating to concepts just presented. Please take a few moments to think about these questions before reading on.

The reader need not have worked with operating systems previously, though such background would be helpful. The reader should also know something about microprocessor hardware—at a minimum, how the 8086 or 8088 devices operate. For simplicity, most of the software examples are written in PL/M-86, so the reader should be familiar with PL/M-80 or some other block-structured language. Finally, be forewarned that the configuration steps make use of several ISIS utility programs, including EDIT, SUBMIT, ASM86, LINK86, and LOC86. Readers who wish to brush up on any of the above should consult the appropriate Intel reference manuals.

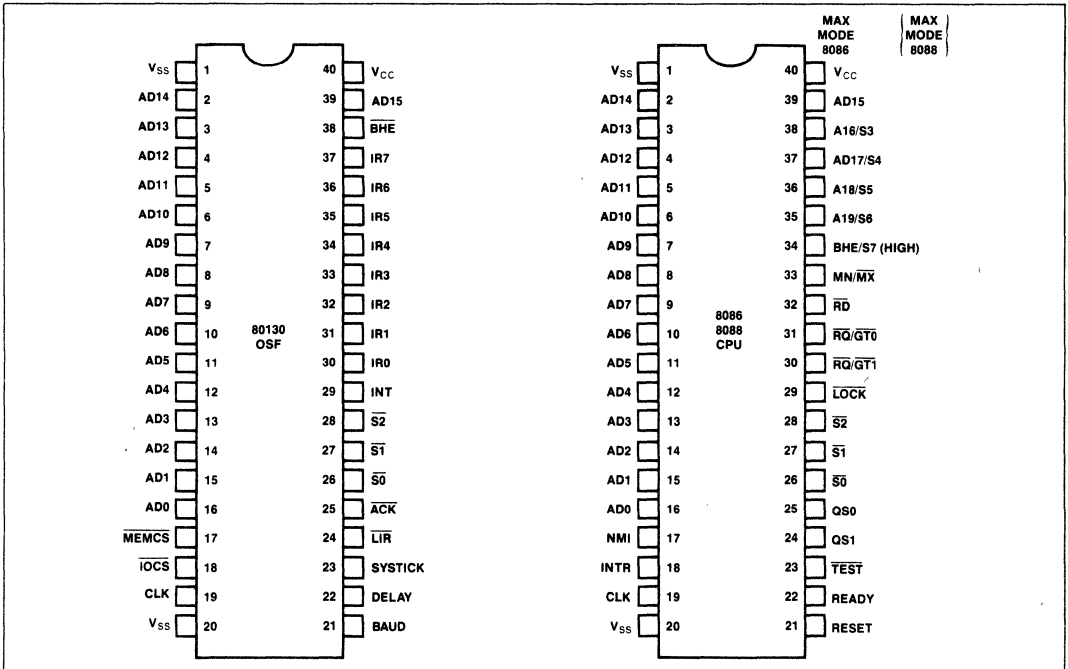


Figure 1. 8086 and 80130 Pinout Diagrams

## EVOLUTION OF PROCESSOR EXTENSIONS

In the early days of microcomputing (circa 1974), things were simple. The first microprocessors comprised just the central processing unit of a simple computer. Systems built up from these processors were generally small, dedicated-purpose device controllers—often replacing the random logic of an earlier design. The system designer had responsibility for the development of the hardware and all application software.

Semiconductor technology has progressed rapidly since then. Devices have become more sophisticated, as have the applications in which they are used. System functions today are more complex than they used to be, and are demanding more in the way of both system hardware and software.

To help designers cope with this complexity, semiconductor vendors are building increasingly more “functionality” into their standard product lines. Whereas the general arithmetic functions of the 8080 and 8085 were limited to addition and subtraction of eight-bit unsigned (ordinal) values, for example, the Intel® 8088 and 8086 now add, subtract, multiply, or divide eight- or 16-bit, signed or unsigned variables—an obvious improvement.

The evolution of floating-point arithmetic provides another example of technology growth. Initially, designers of numeric and process-control systems each developed the floating-point arithmetic routines they needed. Intel eased this task considerably in 1977 when it introduced a standard floating-point format and a floating-point arithmetic software library, FPAL-80. In 1978, the iSBC 310 High-Speed Mathematics Unit implemented these same functions with dedicated hardware and executed them an order-of-magnitude faster.

The 8231A Arithmetic Processor Unit (introduced in 1979) provided similar functionality in one chip at much lower cost. To accommodate the needs of today's world, the Intel RealMath™ software standard and the 8087 numeric coprocessor perform 80-bit floating-point arithmetic for high-performance 8088 and 8086 systems.

This evolution of floating-point hardware illustrates two recurring themes in the microcomputer industry. First, there is a natural trend toward componentization:

1. New applications reveal a need for new types of functionality (in this case, floating-point arithmetic).
2. As common requirements become evident, vendors develop software to serve these needs.

3. Specialized hardware is developed to support the established functions more simply and effectively than software alone.

In time, everything ends up in silicon.

The second theme is this: different functions should be implemented in different ways to fit the customer's needs. “Universal” requirements—like 16-bit multiplication—are best incorporated into the CPU. Functions needed only by certain applications—like high-speed, extended-precision square roots—should be provided as optional Processor Extensions so that their expense is incurred only by those who need them. In keeping with this philosophy, Intel currently offers several processor extension products (see “What's in a Name?”).

### What's in a Name?

The 80130 Operating System Firmware (OSF) device is only the latest member of an extremely flexible family of Intel microprocessors. Its siblings include the 8086 and 8088 Central Processing Units (CPUs), the 8089 I/O Processor (IOP), and a floating-point math coprocessor, the 8087 Numeric Processor Extension (NPX). These individual standard components may be mixed and matched in numerous ways to create combinations optimized for widely varying applications.

To make it easier to discuss the most common configurations, Intel has defined an “Advanced Processor Series” (iAPX) numbering scheme, something akin to those used in the minicomputer and mainframe worlds. The 8086 CPU by itself, for instance, is called the iAPX 86/10. The 8086/8087 combination is dubbed the iAPX 86/20. An 8086/80130 pair has the name iAPX 86/30. The 8086, 8087, and 80130 together would form an iAPX 86/40.

When each of these combinations uses an 8088 in lieu of the 8086, each of the numbers above substitutes “88” for the “86”. An 8088 teamed with an 80130 is therefore called the iAPX 88/30. Finally, adding an 8089 to any system changes the final zero to a one. So, an iAPX 88/41 system would be one using the 8088/8087/8089/80130 chip set.

### Real-Time Operating Systems

Let's turn our attention now to the subject of microcomputer operating system software—an area steadily growing in importance. The trends toward standardized functions with specialized implementations will become evident.

But first, what is an operating system? The phrase means different things to different people. In 20 words or less: An OS is a tool, a set of programs or routines which reduce and simplify the problem of managing system resources. (Well, 21, actually . . .)

Most microcomputer programmers have encountered single-user diskette operating systems, Intel's ISIS-II®, and CP/M® and CP/M-86® from Digital Research Incorporated among them. In essence, an OS of this sort is a collection of run-time subroutines which perform device I/O operations and give application programs access to a disk-based file system. Along with these are routines to supervise the loading and execution of application programs. Historically, this type of OS is oriented toward user-interactive applications: software development, business computing, and the like.

In the mainframe world, the goal of an operating system is to use expensive equipment as efficiently as possible. Batch processing systems ensure that programs waste as little CPU time as possible, though each monopolizes the CPU until it has completed. A time-sharing OS allots short periodic "slices" of time to each of several independent users, during which each has access to the CPU, memory, and other system resources.

A step above the traditional time-sliced OS are "real-time, multitasking operating systems." But what is a "real-time" application? ("Don't all programs execute in real time?")

A real-time system is one in which the CPU must do many different things (tasks), all more-or-less simulta-

neously. Unlike the sequential time-sharing of mainframe OSs, though, the tasks are prioritized. Low-priority tasks are preempted if any of higher priority have work to do. The higher-priority task then runs until it must wait for some external event to occur or no longer needs the CPU for some other reason. Thus, the CPU services tasks in their order of importance.

A computer controlling factory machinery, for instance, might perform five separate tasks:

1. Monitor input switches to detect emergency conditions, determine intended operating mode, or update indicator lights showing machine status;
2. Drive a stepper motor to position a tool;
3. Keep track of the time of day;
4. Send output to the console (e.g., CRT), either in response to explicit commands or as part of some other task;
5. Read and process characters entered from a console keyboard.

These tasks seem largely unrelated, though the first few may be more important to system operation than the others. Let's consider some alternate ways to accomplish these functions with today's microcomputers.

Conceptually, the most straightforward approach might be to dedicate a separate computer to each. The program for each would then be quite simple: an initialization phase followed by an endless loop performing the dedicated function. Algorithms for the first four tasks are flowcharted in Figure 2.

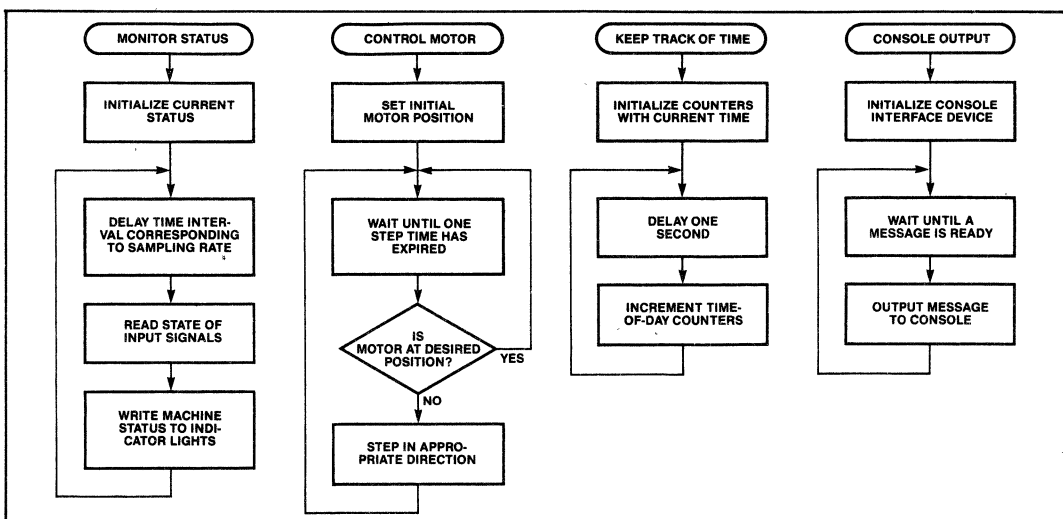


Figure 2. Flowcharts for Concurrent Machine-Tool Tasks

What's wrong with this approach? Ignoring cost, the need for multiple CPUs becomes physically unrealistic for more than a few tasks—60, say, or 600. And tasks are rarely fully independent; note that the switches monitored by task 1 could affect task 2, and that tasks 4 and 5 interact with the rest of the system in as yet undefined ways. So, some sort of communications would have to be set up between the micros.

**Exercise 1.** Suppose five tasks are all interrelated. How many communications channels would have to be set up between different processors? If each channel requires two dedicated communication

chips, how would the number of peripheral devices compare with the number of CPUs?

In each task, the CPU spends most of its time waiting for time to pass or for something to happen. One CPU would be able to implement all five tasks if its time were properly divided among them. An alternate approach, then, might be for a single processor to attend to each task in turn, performing the actions called for by each. Figure 3 shows a flowchart for this scheme. Only one CPU is required and the tasks can communicate between themselves and share physical resources like the console.

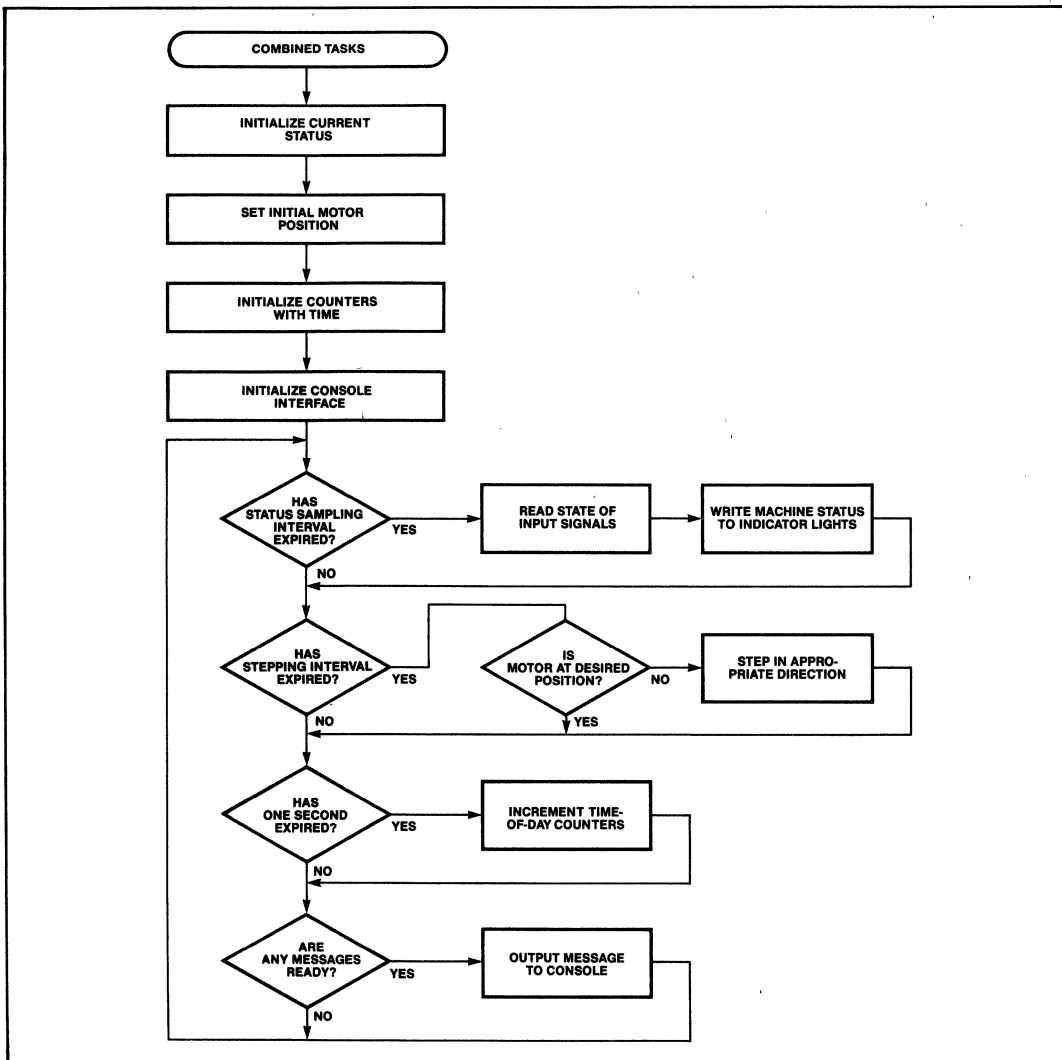


Figure 3. Machine-Tool Tasks Implemented Via Polling Scheme

The problem here is the heavy interaction between tasks. Before it can be serviced, an important task may have to wait for many other less critical tasks to complete. This imposes a constraint that each task release the CPU as quickly as possible. Also, lumping tasks together obscures the boundaries between them. Initialization sequences must be grouped with each other, rather than with the sections of code affected. Adding to or deleting any task may affect the others. It's not clear how to structure the program such that programmers could cooperate on such a program.

Moreover, the various tasks can interfere with each other. Suppose on a given pass through the processor loop, three tasks each send one new character of a message to the console display screen. The resulting output would be most interesting.

The third, and optimal approach, would be one which combined the advantages of the first two approaches, while avoiding the pitfalls. Each function of the overall system could be designed, written, and tested separately, as in the first approach, yet all the software would run on a single computer system as in the second. Tasks could therefore communicate with each other easily, and share peripherals such as CRTs. This multitask control and communication function could be performed largely through software.

The key is finding a way to properly budget CPU time between the various tasks. Early pioneers of complex, real-time, control system design found that they needed special routines, apart from the application tasks themselves, to supervise the execution of application tasks. It was (at best) an inconvenience for so many engineers to independently define, design, document, test and debug software with the same general purpose. At worst, schedules slipped or projects were cancelled for the lack of reliable executive software.

To help avoid these hazards and free up the designers to concentrate on more immediate goals, Intel developed iRMX 80, the first real-time, multitasking, executive operating system for microprocessors. iRMX 86 was introduced to the 16-bit world two years later in 1980.

Because of the critical real-time nature of such operating systems, they require certain hardware capabilities in the host system, such as special timer logic clocked at certain frequencies to measure the passing of time, and interrupt controllers to monitor assorted asynchronous events. Combine all this with a handful of memory chips to house just the OS software, and the address decode and control logic needed by all of the above, and you'll find you need the equivalent of a single-board computer system just to support a multitasking environment.

Until now, that is. The current trend is to integrate OS software and hardware functions into silicon. Intel's iAPX 432 32-bit MicroMainframe™ system does this within the CPU. For the 16-bit world, however, Intel provides a separate chip, the 80130, which contains operating system firmware as well as timer and interrupt control functions.

What is the 80130 OSF? It is an extremely sophisticated integrated circuit, fabricated using Intel's high-performance HMOS technology, which contains over 160,000 devices. In one 40-pin package (Figure 4), the 80130 combines several timers, multiple-mode interrupt control logic, and a large control store memory—plus buffers, decoders and the like—to form the integrated heart of a multitasking operating system. Compared with the iRMX 86 Nucleus, for example, the 80130 replaces an 8259A PIC, an 8253 PIT, a special oscillator, 16K bytes' worth of memory, and associated control logic.

The 80130 operates in conjunction with the 8086 CPU. Together, the two chips are called the iAPX 86/30 OSP. The same device may be paired just as easily with an 8088 forming the iAPX 88/30. From here on, though, references to the 8086 or "host processor" apply to both CPUs. Due to the high speed of HMOS, the 80130 currently runs at system clock rates up to 8 MHz without inserting any wait states. Firmware in the 80130 supports the 35 primitive functions listed in Table 1. Many of these are discussed in Chapter IV.

## SYSTEM HARDWARE DESIGN

The 80130 supports a wide range of system architectures, from compact to quite complex. Most, however, have in common the functional blocks represented in Figure 5. After a brief review of iAPX 86/30 systems in general, we'll examine 80130 requirements in greater detail.

### Basic Functional Blocks

In addition to the 80130, the central processing "core" of a typical OSP system would include an 8088 or 8086 operating in maximum mode, an 8284A clock generator, and an 8288 system controller, all connected according to the standard rules. More on the 80130-specific interconnects later.

Address latches (e.g., 8282s or 8283s) are generally needed to demultiplex the processor address bus for standard memory devices and for memory and I/O device-select logic. The number (from zero to three octal latches) depends on the host processor, memories, and the addressing scheme employed. Data



**Table 1. Operating System Primitives Supported by 80130**

<p><b>Task Management</b>                  Suspend Task                  Resume Task                  Sleep                  Create Task                  Delete Task                  Set Priority                  Get Task Tokens</p>	<p><b>Interrupt Management</b>                  Set Interrupt                  Signal Interrupt                  Reset Interrupt                  Enter Interrupt                  Wait Interrupt                  Exit Interrupt                  Enable                  Disable                  Get Level</p>
<p><b>Intertask Communications and Synchronization</b>                  Send Message                  Receive Message                  Create Mailbox                  Delete Mailbox</p>	<p><b>Free Memory Management/System Partitioning</b>                  Create Segment                  Delete Segment                  Create Job</p>
<p><b>Mutual Exclusion Control</b>                  Receive Control                  Accept Control                  Send Control                  Create Region                  Delete Region</p>	<p><b>Misc. Support</b>                  Signal Exception                  Get Type                  Disable Deletion                  Enable Deletion                  Set O.S. Extension                  Get Exception Handler                  Set Exception Handler</p>

transceivers (8286s or 8287s) may also be needed for increased bus buffering.

Any complete microprocessor system must also have some combination of I/O peripherals and memory, collectively indicated by the box labeled "Local Resources." As we shall see, some of the system RAM and ROM (or EPROM) must be reserved for OSP itself. Additional logic decodes the latched address lines to generate chip-select signals for the memory and I/O devices.

This note only discusses simple, single-processor systems. More sophisticated architectures may incorporate a multimaster system bus, in addition to a local processor bus. This would require additional system controllers, address latches, and bus transceivers for bus isolation, and address mapping logic (not shown) to select between the various busses, enable the respective transceivers, generate a System Ready signal, and so forth. For design information on such techniques, refer to application note AP-67 in the *iAPX 86,88 User's Manual*.

**80130 Pin Functions**

Back to the 80130. Certain pins on the 80130 (in particular, AD15-AD0) attach directly to the CPU. The AD pins are bidirectional, accepting addresses from the host and returning instructions or data. By monitoring the system clock and status signals,  $\overline{S2}$ - $\overline{S0}$ , the 80130 can decode the processor status internally and respond automatically to the appropriate bus cycles. The  $\overline{BHE}$  input lets the 80130 determine the width of data transfers and distinguishes an 8088 host from an 8086. If you refer back to Figure 1, you'll notice that these 80130 pin assignments were selected to simplify P.C. board layout.

Because of the 80130's location on the CPU side of any latches or data transceivers (on what is sometimes called the "pin bus"), the transceivers (if used) must be disabled when the 80130 is driving the processor bus. Whenever the 80130 is responding to any type of bus cycle, it generates an  $\overline{ACK}$  signal. As Figure 4 suggests, one way to avoid contention is to simply disable the transceivers when  $\overline{ACK}$  is active.  $\overline{ACK}$  can also be used to prevent the insertion of wait states.

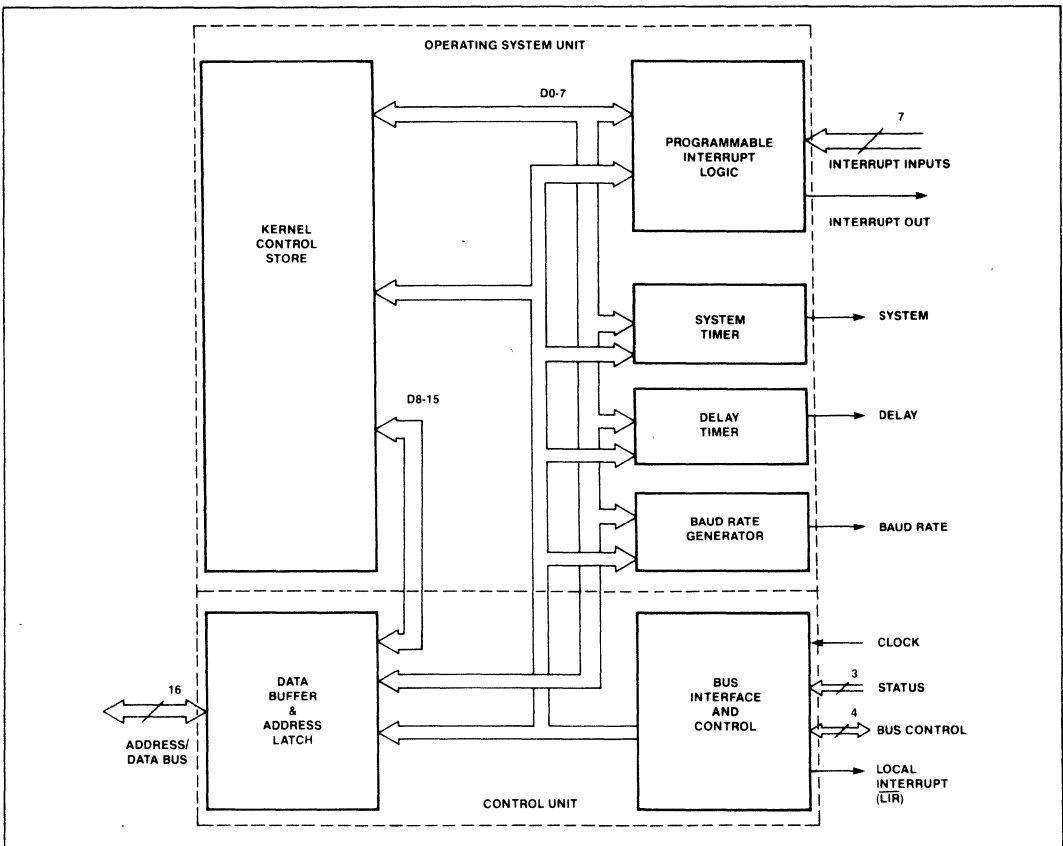


Figure 4. 80130 Internal Block Diagram

Additional pins on the 80130 include eight interrupt-request inputs. Internal interrupt control logic provides many of the functions of the 8259A. During system configuration (Chapter V), each of the eight may be individually defined as a direct level-sensitive or edge-triggered interrupt request, or each may be cascaded with a standard 8259A in slave mode.

The INT output must be connected to the host CPU to inform it of an enabled interrupt request. In very large systems with multiple, cascaded interrupt controllers, Local Interrupt Request (LIR) indicates to the bus contention logic whether a requesting slave is local, or must be accessed via a multimaster bus.

The 80130 also contains dedicated timer logic to provide the OS time base, which is output on SYSTICK. Software operating in conjunction with the 81030 assumes one of the interrupt inputs (INT2 in this case) is

driven by SYSTICK, so this connection must be made externally. Routines within the 80130 initialize and perform all bit-level control of the interrupt and timer logic, according to options and parameters specified during the configuration process. Freeing the programmers from this tedium allows them to devote more thought to solving their own unique problems.

An additional, independent timer generates a user-programmable, square-wave output signal called BAUD to clock an off-chip USART.

Since the 80130 displays some of the characteristics of both memory and I/O, it requires chip-select signals for both the memory (MEMCS) and I/O (IOCS) address spaces. These are discussed at length below. Finally, Intel has reserved one output pin (called "DELAY") for use in future designs. Leave it unconnected in iAPX 86/30 systems.

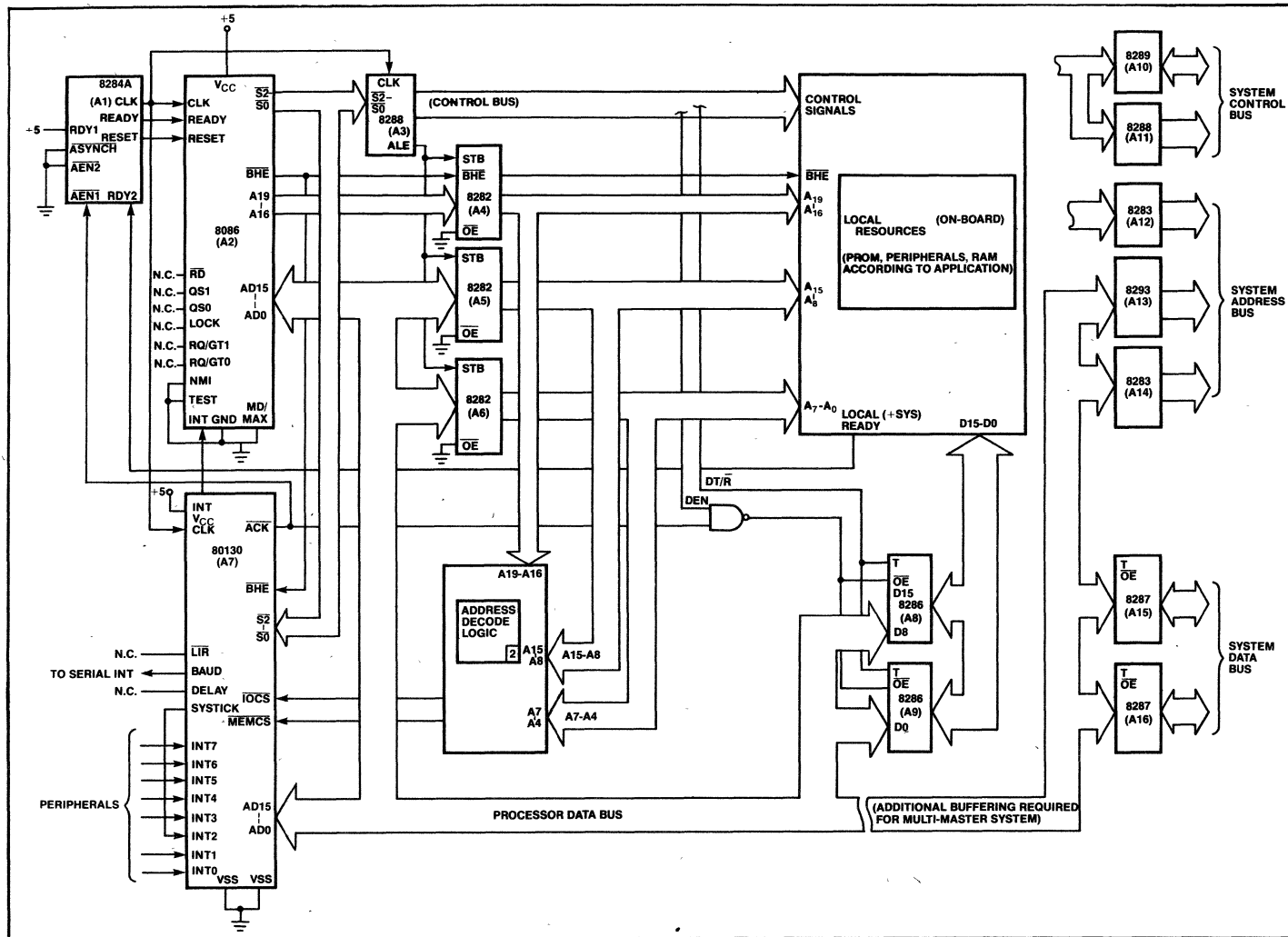


Figure 5. Basic iAPX 86/30 Microcomputer System Block Diagram

2-60

AFN-00358A

AP-130

### Additional System Requirements

The OSP requires a certain amount of off-chip memory for its own operation. The system must provide at least 1K bytes of RAM at address 00000H for the CPU interrupt vectors, plus another 1500<sub>10</sub> bytes for OSP system variables, data structures, stacks, and the like. This RAM may reside anywhere in the 8086 megabyte address space, although it is often contiguous with the interrupt vector up front. Application tasks must each have their own stack, so allow at least an additional 300 bytes of RAM for each.

Any iAPX 86 system must have ROM or EPROM at the upper end of memory to hold the CPU restart vector. About 3400 more bytes are consumed by code to initialize and access the OSP. This code is generated automatically from libraries on a diskette provided with a product called the iAPX 86/30 and iAPX 88/30 Operating System Processor Support Package (iOSP 86). Space left in the initialization EPROMs is available for application tasks.

As code is being written, the system designer should count on another 1500 bytes of code from the support

libraries being added to his application during the linking and system configuration steps. These memory requirements are shown in Figure 6. In practice, the separate blocks in this figure would be grouped together for more efficient use of RAM and EPROM chips.

The 80130 occupies a 16K-byte block of addresses in the host-processor memory space, so external logic should decode address bits  $A_{19}-A_{14}$  to generate MEMCS. Similarly, the timer and interrupt control logic occupy a 16-byte block of addresses in the I/O space; at least some of the bits  $A_{15}-A_4$  must be decoded to generate IOCS. The 80130 decodes all the lower-order address bits (14 for memory, four for I/O internally).

Firmware in the 80130 leaves a great deal of flexibility in decoding the chip-select signals, to be compatible with whatever decode logic is already present in the system. The I/O starting address may be on any 16-byte boundary in the full CPU I/O space. The memory block has only two restrictions: the off-chip initialization and interface code memory must be placed immediately above the MEMCS block, so the 80130 may not occupy the extreme top of memory, nor may the 80130 reside at address 00000H since this area is reserved for interrupt vectors.

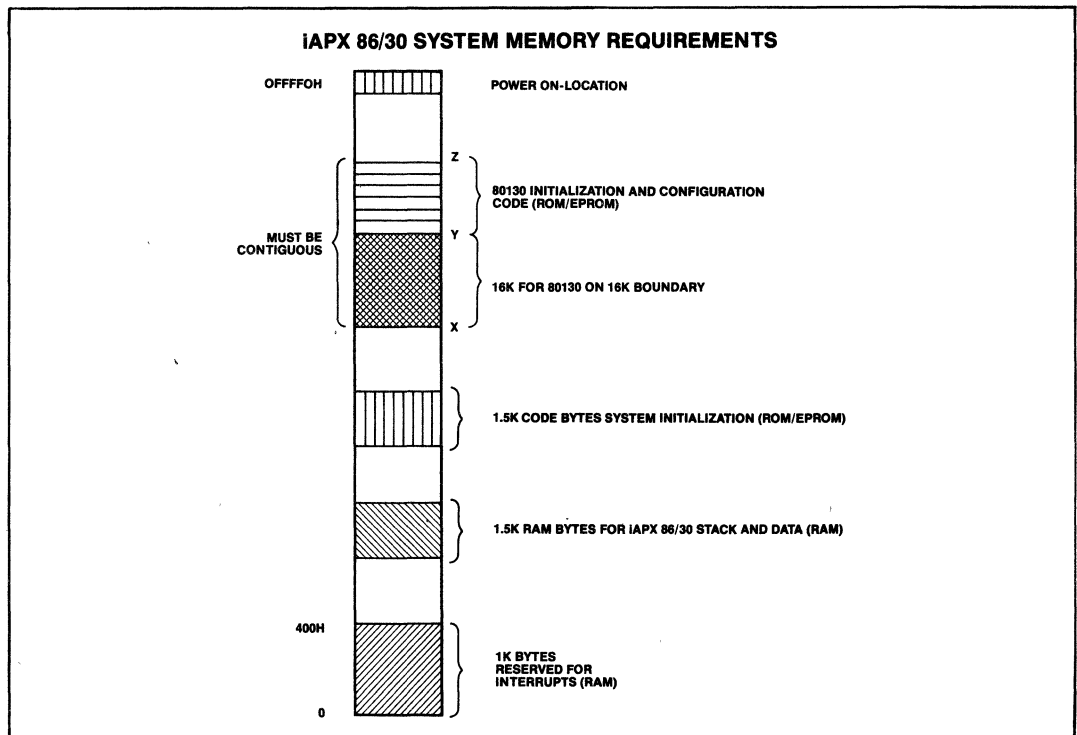


Figure 6. Operating System Processor System Memory Requirements



The propagation delay numbers plugged into the equation are worst-case values from the appropriate Intel data sheets. The CPU is an 8086-2 operating at 8 MHz. This means the address decode logic must produce stable CS outputs within 140 nanoseconds.

**Exercise 2.** Using standard, low-power Schottky TTL, does it make sense for a circuit to take longer than 140 nsec. to decode 6 program or 12 I/O address bits? Even if the rather liberal setup specs are not met, the 80130 would still work fine. Wait states would be needed until the chip-select signal was active, however, so performance would degrade some.

The second point of concern relates to ready signal timing. The 80130's acknowledge output signal,  $\overline{ACK}$ , can be used to control the CPU's ready signal. For this case, the chip-select signal must be active early in a memory or I/O cycle to allow activation of  $\overline{ACK}$  early enough to prevent wait states. There are two schemes for implementing ready signals; "normally ready" and "normally not ready." (For more details, refer to AP-67, "8086 System Design.") Chip-select timing is more critical in some "normally not ready" systems.

In a "normally not ready" design, acknowledge signals are generated when each resource is accessed. The individual acknowledgements are combined to form a system-wide ready signal which is synchronized by the 8284A clock generator via the RDY and AEN inputs. The 8284A can be strapped to accept asynchronous ready signals (asynchronous operation) or to accept synchronous ready signals (synchronous operation). Synchronous 8284A operation provides more time for address latch propagation and chip-select decoding. In addition, inverting ACK off chip produces an active-high ready signal compatible with the 8284A RDY inputs, which have shorter set-up requirements than AEN inputs. (As a side benefit, a NAND gate used like this can combine ACK with the active-low acknowledge signals from other parts of the system.) Based on these assumptions, the time available for address latch propagation and chip-select decoding at 8 MHz is:

$$\begin{aligned} T_{CLAV} + T_{OVCS} + T_{CSAK} + R_{RIVCL} &\leq T_{CLCL} + T_{CLCL} \\ T_{OVCS} &\leq 2 T_{CLCL} - T_{CLAV} - T_{CSAK} - T_{RIVCL} \\ &\leq 250 - 60 - 110 - 35 \\ &\leq 45 \text{ nsec.} \end{aligned}$$

The circuit in Figure 8 which uses Schottky TTL components leaves about 15 nsec. to produce  $\overline{MEMCS}$  from

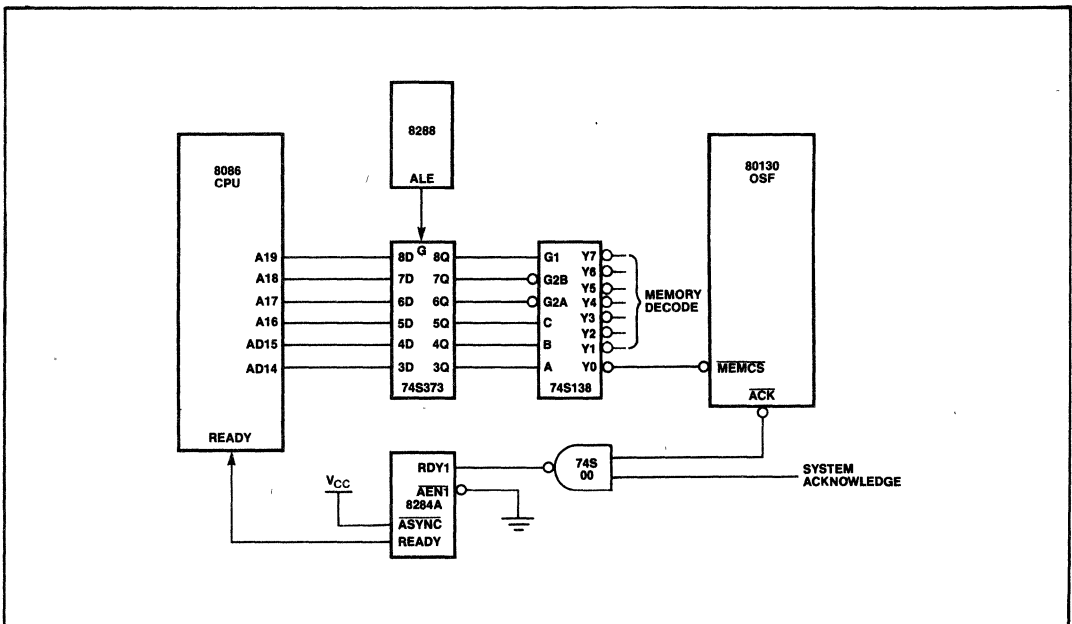


Figure 8. High-Speed Address Decoding Circuit

the high-order address bits—more than enough for the 74S138 one-of-eight decoder shown.

Granted, this does not leave much leeway to fully decode the I/O address bits. A 12-input NAND gate on AD15–AD4 could be used, introducing only a single propagation delay but forcing the I/O register block to start at 0FFF0H. Incomplete decoding is also legal: it is safe to drive  $\overline{IOCS}$  with the (latched) AD15 signal directly, provided all other ports in the system are disabled when this bit is low. In this case, the effective address of the I/O block (which must be specified during the system configuration step) could be 0000H, or any other multiple of 16 between 0000H and 7FF0H.

Again, the OSP system will still operate even if the memory or I/O decoding is slow. The acknowledge signal returned to the host CPU would just be delayed accordingly, so unnecessary wait states would be inserted in access cycles, but the 80130 would *not* malfunction. Only rarely does the OSP access resources in its I/O space. Even if slow decode logic were to insert several wait states into every I/O cycle, the overall effect on system performance would be insignificant.

A few words of caution, though. If the 8284A is strapped for synchronous operation, external circuitry must guarantee that ready-input transitions don't violate the latch set-up requirements. Also, the chip-select signal must *not* remain low so long after the address changes that the 80130 could respond to a non-80130 access cycle.

**Exercise 3.** Suppose the typical timing values for a particular decoder would easily meet the ready-input set-up requirements presented above for asynchronous 8284A operation, but pathological worst-case figures were just a little slow. Could that circuit still be used safely in most applications? What would happen if the worst-case combination of worst-case conditions ever actually did occur? These occasional extra wait states would probably not cause a hard system failure.

**Exercise 4.** Earlier it was mentioned that the acknowledge signal could also be used to avoid bus contention. Prove that with any decode logic which meets the above requirements,  $\overline{ACK}$  would disable the bus transceivers before the host CPU samples the bus.

## Example System Design

Appendix A includes full schematics for a complete iAPX 86/30 system providing considerable functionality with only 27 chips. In addition to the OSP, the

system has 4K bytes of 2114 RAM (with sockets for another 4K), from 8K to 32K bytes of 2732A or 2764 EPROM, an 8251A USART operating at 9600 baud, and an 8255A Programmable Peripheral Interface with 24 parallel I/O lines. Eight of the inputs read logic values off DIP switches; eight outputs drive small LEDs. Four more outputs connect to the coil drivers of a four-phase stepper motor. A layout diagram of the prototype appears in Figure 9.

The system is even simpler than the discussion of "typical" requirements implied. The 8086 direct-bus drive capability is adequate to make the data transceivers unnecessary. (To equalize the bus loading, the 8255A is connected to the upper half of the bus.) Address decoding logic was minimized by making the high-order address bits "don't-cares." Moreover, the part count could have been reduced to 16 using an 8088 and multiplexed-bus 8185 RAMs and 8755A EPROMs. (The reader may be surprised to learn that, except for wire-wrapping mistakes, the prototype system hardware worked when it was first powered up. The author certainly was!)

## APPLICATION SOFTWARE DEVELOPMENT

Like other well-structured programs, application software to run on the iAPX 86/30 is written as a number of separate procedures or subroutines. In conventional programs, though, execution begins with a section of code (the *program body*) at the outermost level. The program calls application procedures, which may call other procedures, but which eventually run to completion and return to the program body.

In an OSP application, though, there is no "outermost level" in the traditional sense; rather, the procedures are started, suspended, and resumed as situations warrant under the control of the OSP. The term "task" refers to the execution of such a procedure in this way. While an instruction stream is suspended, the OSP keeps track of the task state (instruction counter, CPU register contents, etc.) so that it may be resumed later.

Each task is assigned a relative priority by the programmer, on a scale of 0 (high priority) to 255 (low). Tasks with higher (numerically lower) priority are given preferential treatment by the OSP; the task actually controlling the CPU at any given instant will be the one with the highest priority which is not waiting for some event to occur. (If all this sounds confusing, examples coming later may help.)

A task which operates independent of other tasks can be written without knowing anything about the others.

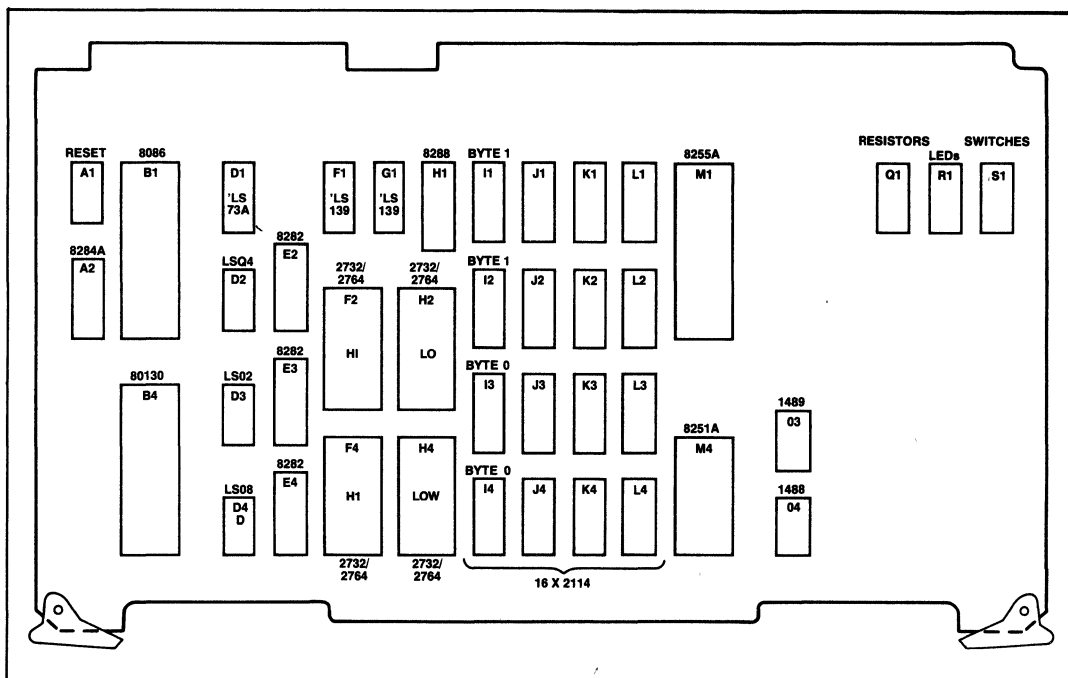


Figure 9. Example System Prototype Layout

This makes it easy to divide a very large programming job among a team of programmers, each writing the code for some of the tasks. Moreover, a task need not even know if other tasks exist. They may be tested and debugged before others have even been written. As an application evolves, new tasks may be added or unnecessary ones removed without affecting the rest.

The number of tasks in an application may need to be quite large. The number of tasks allowed in one application is essentially unlimited, as is the number of other objects—regions, mailboxes, segments, and the like. (The term “object” relates to different types of data structures maintained internally by the OSP.) Each object is internally identified by a unique 16-bit “token,” which means the theoretical maximum total is over 65,000. The more pragmatic issue of physical memory consumption limits the number of simultaneous concurrent tasks to “only” several thousand.

(When a number of tasks cooperate to accomplish some common goal, the collection of tasks is referred to as an application “job.” The OSP also allows for an unlimited number of application jobs, though only one is illustrated in the example discussed here. A second similar machine, with different status switches, a differ-

ent motor, and a different console might make up a second job.)

All OSP application jobs must have one special initialization task (often called INIT\$TASK) just to get started; this one may, in turn, create other tasks as it executes. The initialization task for this example is discussed at the end of this chapter.

## Hardware Initialization

The life of any task can be broken into three phases: start-up, execution, and termination. The start-up phase initializes variables, data structures, and other objects needed by the task. During the execution phase the task performs its useful work. Depending on the application, this may be a single sequence of actions, or a loop executed repeatedly. When the task completes, it must terminate itself so as not to use any more CPU time. One or more phases may be omitted. For example, some tasks are intended to execute “forever,” in which case the termination phase is not required.

This life cycle is suggested by Example 1, a segment of code called HARDWARE\$INIT\$TASK. This task first



programs the 80130 internal timer logic to generate a square-wave cycle on the BAUD pin every 52 system clock cycles, which corresponds to a system console data rate of 9600 baud. The task then sets the system's 8255A PPI and 8251A USART devices to operate in the desired modes, and outputs a short sign-on message to the CRT. For the sake of reader's unfamiliar with the protocol for interfacing with the 8251A, simple input and output routines (C\$IN and C\$OUT) are reproduced in Example 2.

```
HARDWARE$INIT$TASK PROCEDURE,
  DECLARE HARD$INIT$EXCEPT$CODE WORD,
  DECLARE PARAM$1 (*) BYTE DATA (40H, 8DH, 00H, 40H, 4EH, 27H),
  DECLARE PARAM$1$INDEX BYTE,
  DECLARE SIGNON$MESSAGE (*) BYTE DATA
  (CR, LF, 'iAPX 86/30 HARDWARE INITIALIZED', CR, LF),
  DECLARE SIGNON$INDEX BYTE,

  OUTPUT (PPI$CMD)=90H,
  OUTPUT (TIMER$CMD)=0B6H,
  OUTPUT (BAUD$TIMER)=33; /*GENERATES 9600 BAUD FROM 5 MHZ*/
  OUTPUT (BAUD$TIMER)=0,
  DO PARAM$1$INDEX=0 TO (SIZE(PARAM$1)-1),
  OUTPUT (CMD$1)=PARAM$1(PARAM$1$INDEX),
  END; /*OF USART INITIALIZATION DO-LOOP*/
  DO SIGNON$INDEX=0 TO (SIZE(SIGNON$MESSAGE)-1),
  CALL C$OUT (SIGNON$MESSAGE(SIGNON$INDEX)),
  END; /*OF SIGN-ON DO-LOOP*/
  CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, @HARD$INIT$EXCEPT$CODE),
  CALL RQ$DELETE$TASK (0, @HARD$INIT$EXCEPT$CODE),
  END HARDWARE$INIT$TASK,
```

### Example 1. System Hardware Initialization Task

```
C$OUT PROCEDURE (CHAR),
  DECLARE CHAR BYTE,
  DO WHILE (INPUT(STAT$51) AND 01H)=0,
  /* NOTHING */
  END;
  OUTPUT (CHAR$51)=CHAR;
  END C$OUT;

C$IN PROCEDURE BYTE,
  DO WHILE (INPUT(STAT$51) AND 02H)=0,
  /* NOTHING */
  END,
  RETURN INPUT(CHAR$51),
  END C$IN,
```

### Example 2. Simple 8251A Input and Output Routines

The baud timer should be initialized by a code sequence like that shown here. The 80130 logic is actually compatible with the initialization sequence which would be needed to configure timer 2 of an 8253A as a programmable rate generator. The baud rate parameter loaded into the timer is simply the system clock frequency divided by the desired output frequency. No other timers should be affected by user programs.

When the hardware has been initialized, the task calls an operating system procedure called RQ\$RESUME\$TASK. This signals the OSP that the task's start-up phase has completed, and that the initialization task (which in this case suspended itself after creating HARD\$INIT\$TASK) may continue. Since its function is hardware initialization only, HARD\$INIT\$TASK has no execution phase *per se*. It terminates by calling

the procedure RQ\$DELETE\$TASK, suicidally specifying itself as the task to be deleted.

**Exercise 5.** Beginners may make two common programming errors when developing OSP tasks. The first is when a task deletes itself without ever resuming the suspended task that created it. The second is to not terminate a task properly, with the result that the processor executes a return instruction when the task's work is done. (However, execution of the task did not originate with a call from the OS.) As with all computers, an OSP will do exactly what it is told. How do you suppose the system would react in each case? (Hint: only one of the two failure modes is predictable.)

You may have noticed three things from this short example and Table 1. First, every OSP call begins with the letters RQ. (PL/M compilers totally ignore dollar signs within symbols; they serve only to split long symbol names to make them easier for humans to read.) The letters RQ don't mean anything in particular; their purpose is to make sure OSP routine names don't conflict with any user symbols. These particular letters were chosen to be compatible with the historical naming convention used by iRMX 86. It may be useful, though, to think of RQ as an abbreviation for REQUEST, implying that the OSP provides useful services at the bidding of application code.

The second thing to notice is that the OSP routine names imply pretty well what each routine does. On the one hand, long procedure names take a little longer to type; on the other, they make code listings much easier to read and understand. In effect, the long names help make OSP code self-documenting. The long names shouldn't hinder code development; rarely can programmers think faster than they can type. If they could, programmer productivity would be measured in thousands of lines per day.

The third thing is that the last parameter in every OSP system call points to a word in which the OSP procedure will return an exception code to the application task. The procedure will return a non-zero exception code in this word if it cannot do its job correctly. This does not always imply that an error occurred; sometimes it just means another task isn't ready to cooperate yet. Sometimes an exception value indicates whether the OSP request was processed immediately or delayed for some reason. In fact, some OSP routines are guaranteed never to return a non-zero exception code, yet the pointer is still required for the sake of consistency. For a full explanation of the other parameters for the OSP procedures and details on what the different exception codes mean, consult the *iAPX 86/30, 88/30 User's Manual*.

To illustrate how the OSP procedures are used, the following code examples implement the machine controller tasks introduced earlier. Appendix B puts all the code examples together, though not in the exact order discussed. *Be Forewarned:* the examples border on trivial. They are in this note to demonstrate how to call system routines with as few lines of code as possible, not to tax the capabilities of the OSP. In fact, none of the tasks even check for exception codes returned by the OSP, under the naive assumption that nothing will go wrong in a debugged program. If you're interested in more elaborate software examples, consult application notes AP-86 and AP-110. These notes focus specifically on iRMX 86, but their methods and much of the code apply equally to the OSP systems.

### Simple Time Delays

The STATUS\$TASK routine simply monitors eight switches through an input port, and updates eight LEDs with a pattern determined by the switch settings and task status. Specifically, the LEDs display the bitwise Exclusive-OR function of the inputs and an eight-bit software counter maintained by the task. This action will repeat twice per second. The task does nothing between iterations.

The RQ\$SLEEP routine gives application tasks a way to release the CPU when it is not needed. Any task calling this routine is "put to sleep" for the amount of time it specifies (from 1 to 65,000 SYSTICK intervals), releasing the CPU to service other tasks in the meantime. After the requested time has transpired, the OSP task will reawaken the task and resume its execution, provided a more important task is not then executing.

The 80130 timer logic generates the fundamental System Tick by dividing the system clock frequency by two, then subdividing that frequency by a 16-bit value specified during the configuration process. The period used here is 5 msec., which would result in an 5 MHz system by dividing the 2.5 MHz internal frequency by 12,500.

**Exercise 6:** At this rate, what's the longest nap that would result from a single call to RQ\$SLEEP? How could this duration be extended?

PL/M listings for the complete STATUS\$TASK routine appear in Example 3.

```
STATUS$TASK PROCEDURE,
  DECLARE STATUS$COUNTER BYTE,
  DECLARE STATUS$EXCEPT$CODE WORD;

  STATUS$COUNTER=0,
  CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, @STATUS$EXCEPT$CODE),
  DO FOREVER:
    OUTPUT (PP1$A)=INPUT (PP1$A) XOR STATUS$COUNTER,
    STATUS$COUNTER=STATUS$COUNTER+1,
    CALL RQ$SLEEP (100, @STATUS$EXCEPT$CODE),
  END,
END STATUS$TASK.
```

### Example 3. Status Polling and Reporting Task

### Stepper Motor Control

Conceptually, a stepper motor consists of four coils spaced evenly around a rotating permanent magnet. By energizing the coils in various combinations, the magnet can be induced to align itself with the coils, individually or in pairs. A microcomputer can make a stepper motor rotate, step-by-step, in either direction, by emitting appropriate coil control signal patterns at intervals corresponding to the step rate.

The stepper-motor sequencer (Example 4) is an embellished version of STATUS\$TASK. The OSP calls are intermixed with a few more statements of application code, and the task uses global variables as delay parameters. The reader may wish to adapt the command interpreter task at the end of this chapter to let the operator modify (read: "play with") these parameters to adjust the motor speed as the program runs.

```
DECLARE CW$STEP$DELAY BYTE,
  CW$STEP$DELAY BYTE,
  CW$PAUSE$DELAY BYTE,
  CCW$PAUSE$DELAY BYTE,

MOTOR$TASK PROCEDURE,
  DECLARE MOTOR$EXCEPT$CODE WORD,
  DECLARE MOTOR$POSITION BYTE,
  MOTOR$PHASE BYTE,
  DECLARE PHASE$CODE (4) BYTE
  DATA (0000101B, 0000110B, 0000101B, 00001001B),

  CW$STEP$DELAY=50, /*INITIAL STEP DELAYS = 1/4 SECOND*/
  CW$STEP$DELAY=50,
  CW$PAUSE$DELAY=200, /*PAUSES AFTER ROTATION = 1 SECOND*/
  CCW$PAUSE$DELAY=200,
  CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, @MOTOR$EXCEPT$CODE),
  DO FOREVER:
    DD MOTOR$POSITION=0 TO 100,
    MOTOR$PHASE=MOTOR$POSITION AND 0003H,
    OUTPUT (PP1$C)=PHASE$CODE (MOTOR$PHASE),
    CALL RQ$SLEEP (CW$STEP$DELAY, @MOTOR$EXCEPT$CODE),
  END,
  CALL RQ$SLEEP (CW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE),
  DD MOTOR$POSITION=0 TO 100,
  MOTOR$PHASE=(100-MOTOR$POSITION) AND 0003H,
  OUTPUT (PP1$C)=PHASE$CODE (MOTOR$PHASE),
  CALL RQ$SLEEP (CCW$STEP$DELAY, @MOTOR$EXCEPT$CODE),
  END,
  CALL RQ$SLEEP (CCW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE),
  END,
END MOTOR$TASK.
```

### Example 4. Stepper-Motor Controller Task

### Real-Time Interrupt Processing

The 80130 supports a two-tiered hierarchy of interrupt processing. The lower-level tier corresponds to the

traditional concept of hardware interrupt servicing; a routine called an "Interrupt Handler" is invoked by the 80130 internal interrupt control logic for immediate response to asynchronous external events. A short routine like this might, for example, move one character from a USART to a buffer. Interrupt handlers operate with lower-priority interrupts disabled, so it is a good idea to keep these routines as quick as possible.

"Interrupt Tasks," on the other hand, are higher-level tasks which sit idle until "released" by an interrupt handler. The task then executes along with other active tasks, under the control of the OSP. Such a task should be used to perform slower but less time-critical processing when occasions warrant, such as when the aforementioned buffer is full. Moving such additional processing outside the hardware-invoked interrupt handler reduces the worst-case interrupt processing time.

This hierarchy also decreases interrupt latency. Most OSP primitives execute in their own, private "environment" (e.g., with their own stack and data segments) rather than that of the calling task. Interrupt handlers, on the other hand, run in the same environment as the interrupted task. (In fact, the 80130 primitives may themselves be interrupted!) Leaving the CPU segment registers unchanged minimizes software overhead and interrupt response time, but also means that interrupt handlers may not call certain OS routines. An interrupt task, on the other hand, is initiated and suspended by the OSP itself, with no such restrictions.

Let's see how these capabilities would be used. The time delays introduced by the RQ\$SLEEP call are only as accurate as the crystal frequency from which they are ultimately derived. This may not be exact enough for critical time-keeping applications, since oscillators vary slightly with temperature and power fluctuation.

To keep track of the time of day, the example system uses a 60-Hz A.C. signal as its time base. (Most power utility companies carefully regulate line frequency to *exactly* 60 Hz, averaged over time.) A signal from the power supply is made TTL-compatible to drive one of the 80130 interrupt request pins. An interrupt handler responds to the interrupts, keeping track of one second's worth of A.C. cycles. An interrupt task counts the seconds by incrementing a series of variables.

Example 5 illustrates the former routine. AC\$HANDLER simply increments a variable on each 60-Hz interrupt. Upon reaching 60, it clears the counter and signals TIME\$TASK (Example 6).

```

DECLARE AC$CYCLE$COUNT BYTE;
AC$HANDLER: PROCEDURE INTERRUPT 59; /*VECTOR FOR 80130 INT3*/
DECLARE AC$EXCEPT$CODE WORD;

CALL RQ$ENTER$INTERRUPT (AC$INTERRUPT$LEVEL, @AC$EXCEPT$CODE);
AC$CYCLE$COUNT=AC$CYCLE$COUNT+1;
IF AC$CYCLE$COUNT >= 60
THEN DO;
AC$CYCLE$COUNT=0;
CALL RQ$SIGNAL$INTERRUPT (AC$INTERRUPT$LEVEL,
@AC$EXCEPT$CODE);
END;
ELSE CALL RQ$EXIT$INTERRUPT (AC$INTERRUPT$LEVEL,
@AC$EXCEPT$CODE);
END AC$HANDLER;

```

### Example 5. 60-Hz A.C. Interrupt Handler

In its initialization phase, TIME\$TASK sets up the interrupt handler by calling the RQ\$SET\$INTERRUPT routine. The body of TIME\$TASK (the execution phase) is just a series of nested loops counting hours, minutes, and seconds. When TIME\$TASK calls RQ\$WAIT\$INTERRUPT inside its inner-most loop, the OSP suspends execution of the task until AC\$HANDLER signals that another second's worth of A.C. cycles has elapsed. Thus, interrupt handlers can serve to "pace" interrupt tasks. After a day, TIME\$TASK completes and deletes itself.

```

DECLARE SECOND$COUNT BYTE;
MINUTE$COUNT BYTE;
HOUR$COUNT BYTE;

TIME$TASK PROCEDURE;
DECLARE TIME$EXCEPT$CODE WORD;

AC$CYCLE$COUNT=0;
CALL RQ$SET$INTERRUPT (AC$INTERRUPT$LEVEL, 01H,
INTERRUPT$PTR (AC$HANDLER), DATA$SEG$ADDR BASE,
@TIME$EXCEPT$CODE);
CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, @TIME$EXCEPT$CODE);
DO HOUR$COUNT=0 TO 23;
DO MINUTE$COUNT=0 TO 59;
DO SECOND$COUNT=0 TO 59;
CALL RQ$WAIT$INTERRUPT (AC$INTERRUPT$LEVEL,
@TIME$EXCEPT$CODE);
IF SECOND$COUNT MOD 5 = 0
THEN CALL PROTECTED$CRT$OUT (BEL);
END; /* SECOND LOOP */
END; /* MINUTE LOOP */
END; /* HOUR LOOP */
CALL RQ$RESET$INTERRUPT (AC$INTERRUPT$LEVEL,
@TIME$EXCEPT$CODE);
CALL RQ$DELETE$TASK (0, @TIME$EXCEPT$CODE);
END TIME$TASK;

```

### Example 6. Interrupt Task to Maintain Time of Day

**Exercise 7:** The time maintained by TIME\$TASK is consistently wrong, unless the system resets at midnight. Aside from that, how much error would accumulate per month had TIME\$TASK paced its inner loop by calling RQ\$SLEEP if the system oscillator was 00.01% off? How does this compare with a cheap digital watch? How much error will accumulate from the 60-Hz time base described?

TIME\$TASK incorporates another gimmick: every five seconds it sends an ASCII "BEL" character (07H) to the console to make it beep, by calling a routine called PROTECTED\$OUTPUT. This lead-in gives us a chance to discuss OSP provisions for task synchronization and mutual exclusion.

## Mutual Exclusion

Whenever system resources (e.g., the console) are shared among multiple concurrent tasks, the software designer must be aware of the potential for conflicts. In single-threaded (as opposed to multitasking) programs, the easiest way to transmit characters is by calling a console output routine (written by the user or supplied by the OS) which outputs the character code. (Remember the examples following the hardware initialization routine?)

This approach presents two problems in a multitasking system. One is efficiency: a high-priority task could hang up the whole system while it waits for a printer solenoid to energize, induce a magnetic field, accelerate the hammer, contact a daisy-wheel spoke, move it up to the ribbon, and press them both against the paper. This waste of time is termed "busy waiting," and should always be avoided. By OSP standards, even 1/30 of a second can seem interminable; if the printer is otherwise occupied, the whole system could shut down indefinitely.

Aside from efficiency, though, there is a more serious synchronization problem here. Assume Task A has a higher priority than Task B. Task A is asleep. Task B calls a subroutine to poll the USART and transmit a character. The USART becomes ready.\*When this is detected, the subroutine prepares to output the character to the USART . . . .

Time out! Task A just woke up and starts running. Task A wants to transmit its own character. It calls its own output routine, checks the USART, finds it available, sends it a new character, and goes back to sleep (or suspends itself, or awaits another interrupt—whatever).

Now Task B continues. It "knows" the USART is available, having dutifully monitored it earlier. Task B's character goes out to the USART. The USART goes out to lunch. (In practice, the USART will probably just transmit corrupted data; still, its operating requirements have been violated.)

In Task B's output routine, the sequence of statements from when the peripheral is found to be ready to when the next character is written constitutes a "critical region" (a.k.a. "critical section" or "non-interruptable sequence"). Recognizing such regions and handling them correctly is an important concern in any multitasking system, so the OSP provides several facilities—interrupt control, regions and mailboxes—to help handle general synchronization and mutual exclusion problems. Which one to choose depends on the circumstance.

**Exercise 8:** In this example, would it be better if Tasks A and B shared a single output routine, so that only one section of code sent data to the USART? Convince yourself that the same (or worse!) problems could still arise.

Sometimes critical sections can be protected by just disabling interrupts at appropriate points in the application software. To maintain the integrity of an iAPX 86/30 system, *application code must never execute the STI, CLI, or HLT instructions (ENABLE, DISABLE, or HALT statements in PL/M)*, nor can it access the interrupt control logic directly. Instead, the interrupt status should be controlled with the OSP RQ\$ENABLE and RQ\$DISABLE procedures; routines should be halted via RQ\$\$SUSPEND or RQ\$WAIT\$INTERRUPT.

Back to TIME\$TASK: we want to transmit BELs to the console every five seconds. The console output task will be transmitting other characters. A "clever" programmer may recognize that this will lead to a critical section and analyze the situation as follows:

1. A hazard would arise if TIME\$TASK sends out a beep when CONSOLE\$OUT\$TASK is using the USART;
2. TIME\$TASK will only execute after being signaled by ASC\$HANDLER;
3. ASC\$HANDLER only reponds to an external interrupt.

"Therefore, all CONSOLE\$OUT\$TASK has to do to be safe is disable the 60-Hz interrupt around its output routine."

Not quite. There are still potential hazards. Suppose CRT\$OUT\$TASK has the same priority as TIME\$TASK. TIME\$TASK may already have been signaled by ASC\$HANDLER and be ready to run when CRT\$OUT\$TASK completes. An otherwise unrelated event—another interrupt, for instance—could momentarily suspend CRT\$OUT\$TASK during the critical region with A.C. interrupts disabled. When the OSP returns to that level, it might resume with TIME\$TASK, *not* CRT\$OUT\$TASK. This could lead to the same malfunctions as before, so disabling 60-Hz interrupts didn't help. This series of worst-case assumptions is admittedly convoluted, but the resulting sporadic errors are among the hardest of all bugs to squash.

The problem is that this attempted solution involves too much interaction between tasks, making it confusing and error-prone. Even if some scheme of priority-level assignments and task interactions could be made to work, later modifications or simple additions to the job

could cause bugs to reappear. (The analogy of an unexploded time bomb comes to mind.)

A simpler solution would be one corresponding more closely with the problem. Accordingly, the OSP supports several primitives just to supervise and control access to critical regions.

One of the OSP "data types" is a data structure called a "Region," which can be used by application code to control access to a shared port or some other resource. A task wishing access to the resource should call the OSP procedure `RQ$RECEIVE$CONTROL` before trying to access that resource; when done it must call `RQ$SEND$CONTROL`.

The OSP keeps track of which regions are in use. As long as a region is busy (i.e., has been entered but not yet exited), the OSP will prevent other tasks from entering the region by putting them to sleep. The OSP keeps a queue of all tasks waiting for the busy region. When the region later becomes available (i.e., when the task controlling the region calls `RQ$SEND$CONTROL`), one of the sleeping tasks—either the highest priority or the most patient—will be awakened, granted control of the region, and sent on its way. (When a region is created, the OSP is told whether to awaken tasks waiting for the region based on their priority or how long they have been waiting.) Effectively, a call to `RQ$RECEIVE$CONTROL` will not return to the application task until the resource in question becomes available.

The `PROTECTED$CRT$OUTPUT` (Example 7) demonstrates this protocol. The routine is declared reentrant which means (by definition) the routine may be interrupted and restarted safely. A reentrant routine may be shared by a number of tasks, instead of replicating the same code throughout the application.

```
PROTECTED$CRT$OUT  PROCEDURE (CHAR) REENTRANT,
  DECLARE CHAR BYTE,
  DECLARE CRT$EXCEPT$CODE WORD,
  CALL RQ$RECEIVE$CONTROL (CRT$REGION$TOKEN, @CRT$EXCEPT$CODE),
  DO WHILE (INPUT (STAT$51) AND OIH)=0,
    /* NOTHING */
  END,
  OUTPUT (CHAR$51)=CHAR,
  CALL RQ$SEND$CONTROL (@CRT$EXCEPT$CODE),
  END PROTECTED$CRT$OUT;
```

### Example 7. CRT Output Routine Protected by Region Protocol

As a concession to simplicity, `PROTECTED$CRT$OUTPUT` does use a form of the busy waiting method described earlier. The maximum delay at 9600

baud is only one millisecond, however, much shorter than a system tick. Besides, tasks performing character I/O will all have low priority levels, so the OSP would just delay them if anything more urgent comes up.

**Exercise 9:** Decide whether this explanation is a feeble attempt at rationalization, or a well-justified engineering trade-off.

## Inter-Task Communication

But what if a high priority task must output a string of characters, or the peripheral response time is too long? Busy-waiting may not be acceptable. Alternatively, the output routine could buffer the data and service the USART within an interrupt routine. Another would be to simply pass the data off to a special (low-priority) output task and continue.

Tasks pass information to each other via something called a "message." A message may be the token for any type of OSP object, but the most common and most flexible type is called a "memory segment." In our example, segments will be used to carry strings of ASCII characters between tasks, so we'll examine segments first. Message formats are defined by the individual application programmer—make sure the sending and receiving tasks assume the same format!

A memory segment is just a section of contiguous-system RAM allocated (set aside) by the OSP at the request of an executing task. The OSP keeps track of a free memory "pool," which is initially all unused RAM in the system. When a task needs some RAM, it tells the `RQ$CREATE$SEGMENT` procedure how much it wants. The OSP finds a suitable memory block in the pool, and returns a 16-bit token defining its location. (If not enough memory is available, the procedure returns an exception code.)

The token is the base portion of pointer to the first usable byte of the segment, with the offset portion assumed to be zero. (The token values for all other objects have no physical significance.) Knowing this, it's possible to access elements of the segment as the application warrants.

The subroutine in Example 8 shows how to request a segment and construct a message. `PRINT$TIME` sends the ASCII values of the time-of-day counters (maintained in `TIME$TASK`) to the CRT output task described later. The message format adopted for these examples will consist of a byte giving the message

length, followed by that number of ASCII characters. Figure 10 shows this format.

```

PRINT$TOD PROCEDURE;
DECLARE TOD$MESSAGE$TOKEN WORD;
DECLARE TOD$EXCEPT$CODE WORD;
DECLARE TOD$SEGMENT$OFFSET WORD;
TOD$SEGMENT$BASE WORD;
DECLARE TOD$SEGMENT$PNTR POINTER AT (@TOD$SEGMENT$OFFSET);
DECLARE TOD$TEMPLATE (28) BYTE
DATA (27, 'THE TIME IS NOW hh:mm:ss ', CR, LF);
DECLARE TOD$STRING BASED TOD$SEGMENT$PNTR (28) BYTE;
DECLARE TOD$STRING$INDEX BYTE;

TOD$MESSAGE$TOKEN=RQ$CREATE$SEGMENT (28, @TOD$EXCEPT$CODE);
TOD$SEGMENT$BASE=TOD$MESSAGE$TOKEN;
TOD$SEGMENT$OFFSET=0;
DO TOD$STRING$INDEX=0 TO 27;
TOD$STRING (TOD$STRING$INDEX)=
TOD$TEMPLATE (TOD$STRING$INDEX);
END;
TOD$STRING (17)=ASCII$CODE (HOUR$COUNT/10);
TOD$STRING (18)=ASCII$CODE (HOUR$COUNT MOD 10);
TOD$STRING (20)=ASCII$CODE (MINUTE$COUNT/10);
TOD$STRING (21)=ASCII$CODE (MINUTE$COUNT MOD 10);
TOD$STRING (23)=ASCII$CODE (SECOND$COUNT/10);
TOD$STRING (24)=ASCII$CODE (SECOND$COUNT MOD 10);
CALL RQ$SEND$MESSAGE (CRT$MAILBOX$TOKEN,
TOD$MESSAGE$TOKEN, 0, @TOD$EXCEPT$CODE);
RETURN;
END PRINT$TOD;
    
```

**Example 8. Subroutine to Send Time-of-Day Message to Output Task**

We're coding PRINT\$TIME here (see Example 8), while TIME\$TASK is fresh in our minds. It will actually be called by (and is therefore considered a part of) KEYBOARD\$TASK. Note that while tasks are written as individual procedures, they need not be fully self-contained: outside procedures should be used to help organize and structure the code.

The first thing PRINT\$TIME does is have the OSP create a segment of suitable length, and copies a "message template" into the segment, byte by byte. Then it converts the TIME\$TASK counter values to ASCII, filling in blanks in the template. Finally, it sends the token for the message to the CRT mailbox.

To repeat, these examples are intended to illustrate use of the OSP routines assuming minimum familiarity with PL/M. Better programming practices might take advantage of PL/M literals, structures and the array LENGTH function to build the message, rather than the inflexible constants shown here. Some of these techniques are suggested by PRINT\$STATUS (Example 9), which indicates the binary status of the input switches.

```

PRINT$STATUS PROCEDURE;
DECLARE STATUS$MESSAGE$TOKEN WORD;
DECLARE STATUS$EXCEPT$CODE WORD;
DECLARE STATUS$SEGMENT$OFFSET WORD;
STATUS$SEGMENT$BASE WORD;
DECLARE STATUS$SEGMENT$PNTR POINTER
AT (@STATUS$SEGMENT$OFFSET);
DECLARE STATUS$TEMPLATE (40) BYTE DATA
(39, 'THE SWITCHES ARE NOW SET TO B', CR, LF);
DECLARE STATUS$STRING BASED STATUS$SEGMENT$PNTR (40) BYTE;
DECLARE STATUS$STRING$INDEX BYTE;
DECLARE BIT$PATTERN BYTE;

STATUS$MESSAGE$TOKEN=RQ$CREATE$SEGMENT (40,
@STATUS$EXCEPT$CODE);
STATUS$SEGMENT$BASE=STATUS$MESSAGE$TOKEN;
STATUS$SEGMENT$OFFSET=0;
DO STATUS$STRING$INDEX=0 TO 39;
STATUS$STRING (STATUS$STRING$INDEX)=
STATUS$TEMPLATE (STATUS$STRING$INDEX);
END;
BIT$PATTERN=INPUT (PPI$A);
DO STATUS$STRING$INDEX=29 TO 36;
STATUS$STRING (STATUS$STRING$INDEX)=
ASCII$CODE (BIT$PATTERN AND 01H);
BIT$PATTERN=RROR (BIT$PATTERN, 1);
END;
CALL RQ$SEND$MESSAGE (CRT$MAILBOX$TOKEN,
STATUS$MESSAGE$TOKEN, 0, @STATUS$EXCEPT$CODE);
END PRINT$STATUS;
    
```

**Example 9. Subroutine to Send Status Report Message to Output Task**

**Exercise 10:** One input port is read by both STATUS\$TASK and PRINT\$STATUS. Does this constitute a shared resource? A critical region?

**Exercise 11:** PRINT\$TIME reads the counts maintained by TIME\$TASK, but doesn't alter them. Forced mutual exclusion is generally mandatory when multiple tasks perform read/modify/write sequences on a given variable. Can PRINT\$TIME make TIME\$TASK malfunction? What about the opposite case? If this failure mode was deemed unacceptable, how could it be protected?

**Mailboxes**

The data in a message doesn't actually move or get copied from source to destination when the message is sent; this would be too slow with long messages. Rather, the OSP "carries" the message's token from task to task via a data structure cleverly termed a mailbox. If one task must send messages to another, a mailbox must be created to hold them. The sender calls the RQ\$SEND\$MESSAGE to put a message token into the mailbox. If the receiver isn't ready for the message yet, the OSP puts the message token into an ordered queue. When the receiver calls RQ\$

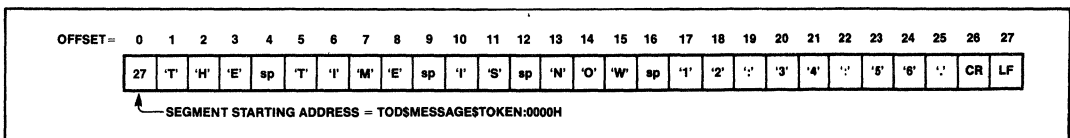


Figure 10. Message Formats Expected by Output Task

RECEIVE\$MESSAGE later, the OSP will give it the tokens one at a time.

What happens if a task tries to receive a message when the mailbox is empty? (This is quite possible, since tasks do run asynchronously.) What token would the OSP return?

In the simple case . . . it doesn't! Instead of returning right away with no data, the OSP will wait until data is available. In the meantime, the OSP puts the receiving task to sleep, remembering that it is waiting for a message at that mailbox. The next time a message is sent to that mailbox, the OSP will awaken the receiving task, give it the token, and—if its priority is high enough—resume its execution. Alternatively, receiving tasks may elect to not wait if the mailbox is empty, or to wait only a specified time.

Many tasks may actually send and receive messages through a single mailbox, with messages being queued in the order that the RQ\$SEND\$MESSAGE calls are executed. The OSP also maintains a list of tasks waiting to receive messages from an empty mailbox, analogous to the queued tasks waiting for region control. As each message is sent to the mailbox, it is passed immediately to a waiting task, either the one waiting the longest or the one with the highest priority (likewise determined by a parameter specified when the mailbox is created).

**Exercise 12:** Under what conditions could a mailbox's message queue contain messages waiting to be received, while the task queue contains tasks waiting for messages? Ignore the possibility that this may happen momentarily during the implementation of either routine. If you think of any such circumstances, please contact the author.

Example 10 shows a task which prints the messages sent above. Upon receiving a message token, CRT\$OUT\$TASK determines the message length from the first two bytes, and sequentially prints each element of the string through the PROTECTED\$CRT\$OUTPUT routine explained earlier. When done, the segment containing the message is deleted, returning its RAM to the free-memory pool.

A few words are in order about the segment accessing techniques demonstrated here. PL/M-86 has a special data type, called a "pointer," used to indirectly access other PL/M variables. OSP application programs must be compiled with the "compact" or "large" model specified. This tells the compiler to implement pointers as 32-bit double words corresponding to the two parts (base:offset) of the 8086 machine-segmented addressing scheme. PL/M-86 tries to shield the programmer

```

CRT$OUT$TASK PROCEDURE,
  DECLARE MESSAGE$LENGTH BYTE;
  DECLARE MESSAGE$TOKEN WORD;
  DECLARE RESPONSE$TOKEN WORD;
  DECLARE MESSAGE$EXCEPT$CODE WORD;
  DECLARE MESSAGE$SEGMENT$OFFSET WORD,
    MESSAGE$SEGMENT$BASE WORD;
  DECLARE MESSAGE$SEGMENT$PNTR POINTER AT
    (MESSAGE$SEGMENT$OFFSET);
  DECLARE MESSAGE$STRING$CHAR BASED MESSAGE$SEGMENT$PNTR BYTE;

  CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, MESSAGE$EXCEPT$CODE);
  DO FOREVER;
    MESSAGE$TOKEN=RQ$RECEIVE$MESSAGE (CRT$MAILBOX$TOKEN, OFFFFH,
      MESSAGE$TOKEN, MESSAGE$EXCEPT$CODE);
    MESSAGE$SEGMENT$OFFSET=0;
    MESSAGE$SEGMENT$BASE=MESSAGE$TOKEN;
    MESSAGE$LENGTH=MESSAGE$STRING$CHAR;
    DO MESSAGE$SEGMENT$OFFSET=1 TO MESSAGE$LENGTH;
      CALL PROTECTED$CRT$OUT (MESSAGE$STRING$CHAR);
    END;
    CALL RQ$DELETE$SEGMENT (MESSAGE$TOKEN, MESSAGE$EXCEPT$CODE);
  END; /* OF FOREVER-LOOP */
END CRT$OUT$TASK;

```

### Example 10. Task to Transmit Messages to the CRT

from the details, yet at times the two parts must be manipulated separately (for instance, to access data in an OSP segment knowing only the segment token/base value).

To get around this, these examples assign a pair of word variables to the same address as a PL/M pointer variable. Each representation is then an alias for the other. To determine the base or offset value of an item of data, load the pointer variable with a pointer to the item and then reference the appropriate field of the overlaid pair of word variables. To "build" an arbitrary pointer, assign computed values to the base and offset fields and then access the data item via the composite pointer.

**Exercise 13:** PL/M 86 does not have built-in functions to separate the high and low-order words of a pointer variable. Does this seem to be a weakness in the language? Bear in mind that the machine representation for pointers varies depending on which programming model is specified at compilation time. When the "small" model is selected, the compilers take advantage of a 16-bit pointer representation for faster and more compact code.

### Console Command Interpreter

If a system has a console keyboard, it's probably used to accept and interpret operator commands. For this demonstration system, the lowest priority of all tasks is a simple-minded routine which polls the USART until a character has been received, and immediately echoes it by calling—you guessed it!—PROTECTED\$CRT\$OUTPUT. Thus, the keyboard is "alive"; it responds immediately to keystrokes, so the operator can type whatever nonsense he desires while everything else is going on.

Ten of the keys (digits 0 through 9), invoke special commands which illustrate interactions between the

multiple tasks. Commands 0 and 1 print out the time and status messages; the rest suspend and resume various tasks, as shown by Table 2. The code for COMMAND\$TASK appears in Example 11.

### Initialization Task

Now that the application tasks have been written, we can write the initialization task.

All applications require a special type of task to initialize system variables and peripherals and create tasks and other objects used by the application. It, too, is written as a PL/M procedure, and can thus be divided conceptually into the same three phases.

Example 12 shows such a task for the demonstration system. The first thing INIT\$TASK does is determine the base address of the job data segment by assigning pointer DATA\$SEG\$PTR with its own address. Next it calls the RQ\$GET\$TASK\$TOKENS routine, which tells the task what token value the OSP assigned it at run time. It then initializes the system peripherals by creating the hardware initialization task discussed above; this code could have been integrated into INIT\$TASK itself just as easily. During its own "execution" phase, INIT\$TASK calls routines to create the OSP data structures shared by the application tasks: the REGION controlling access to the USART, and the MAILBOX repository for output messages. INIT\$TASK creates the application tasks themselves by calling RQ\$CREATE\$TASK.

Though not always required, it is common practice for the overall initialization task to suspend itself after creating each offspring, to let the newborn task get started. Under this convention, each offspring task must resume the initialization task by calling the

```

COMMAND$TASK PROCEDURE,
DECLARE CONSOLE$CHAR BYTE,
DECLARE COMMAND$EXCEPT$CODE WORD;

CALL RQ$RESUME$TASK (INIT$TASK$TOKEN, @COMMAND$EXCEPT$CODE),
DO FOREVER,
CONSOLE$CHAR=C$IN AND 7FH;
CALL _PROTECTED$CRT$OUT (CONSOLE$CHAR),
IF CONSOLE$CHAR=CR
THEN CALL _PROTECTED$CRT$OUT (LF),
IF (CONSOLE$CHAR >= '0') AND (CONSOLE$CHAR <= '9')
THEN DO,
CALL _PROTECTED$CRT$OUT (CR),
CALL _PROTECTED$CRT$OUT (LF),
DO CASE (CONSOLE$CHAR-'0'),
CALL PRINT$TOD,
CALL PRINT$STATUS,
CALL RQ$SUSPEND$TASK (CRT$OUT$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
CALL RQ$RESUME$TASK (CRT$OUT$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
CALL RQ$DISABLE (AC$INTERRUPT$LEVEL,
@COMMAND$EXCEPT$CODE),
CALL RQ$ENABLE (AC$INTERRUPT$LEVEL,
@COMMAND$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (MOTOR$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
CALL RQ$RESUME$TASK (MOTOR$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (STATUS$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
CALL RQ$RESUME$TASK (STATUS$TASK$TOKEN,
@COMMAND$EXCEPT$CODE),
END, /* OF CASE-LIST */
END, /* OF COMMAND PROCESSING */
END,
END COMMAND$TASK.
    
```

### Example 11. Task to Accept and Process Keyboard Commands

```

INIT$TASK PROCEDURE PUBLIC,
DECLARE INIT$EXCEPT$CODE WORD,

DATA$SEG$PTR=@INIT$TASK$TOKEN, /*LOAD DATA SEGMENT BASE*/
CRT$MAILBOX$TOKEN=RQ$CREATE$MAILBOX (0, @INIT$EXCEPT$CODE),
CRT$REGION$TOKEN=RQ$CREATE$REGION (0, @INIT$EXCEPT$CODE),
INIT$TASK$TOKEN=RQ$GET$TASK$TOKENS (0, @INIT$EXCEPT$CODE),
HARDWARE$INIT$TASK$TOKEN=RQ$CREATE$TASK
(110, @HARDWARE$INIT$TASK, DATA$SEG$ADDR BASE, 0, 300,
0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
STATUS$TASK$TOKEN=RQ$CREATE$TASK (110, @STATUS$TASK,
DATA$SEG$ADDR BASE, 0, 300, 0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
MOTOR$TASK$TOKEN=RQ$CREATE$TASK (110, @MOTOR$TASK,
DATA$SEG$ADDR BASE, 0, 300, 0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
TIME$TASK$TOKEN=RQ$CREATE$TASK (120, @TIME$TASK,
DATA$SEG$ADDR BASE, 0, 300, 0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
CRT$OUT$TASK$TOKEN=RQ$CREATE$TASK (120, @CRT$OUT$TASK,
DATA$SEG$ADDR BASE, 0, 300, 0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
COMMAND$TASK$TOKEN=RQ$CREATE$TASK (130, @COMMAND$TASK,
DATA$SEG$ADDR BASE, 0, 300, 0, @INIT$EXCEPT$CODE),
CALL RQ$SUSPEND$TASK (0, @INIT$EXCEPT$CODE),
CALL RQ$END$INIT$TASK,
CALL RQ$DELETE$TASK (0, @INIT$EXCEPT$CODE),
END INIT$TASK,
    
```

### Example 12. Task to Initialize System Software

Table 2. Special Console Commands

Key	Function
0	Send Time-of-day message to CRT.
1	Send status update message to CRT.
2	Suspend CRT output task. The OSP will automatically save messages to the task in the CRT mailbox queue.
3	Resume CRT output task. Queued messages will be displayed.
4	Disable 60-Hz interrupt-driven time base. Time-of-day clock will stop.
5	Enable 60-Hz time base to resume clock execution.
6	Suspend motor control task. Motor will stop.
7	Resume motor control task. Note that if task was suspended 17 times, it must be resumed 17 times.
8	Suspend status polling task. Lights indicating system status will freeze in current state.
9	Resume status polling task.



RQ\$RESUME\$TASK routine when its own local initialization is complete. This convention is called **synchronous initialization**; its purpose is to ensure that each task is allowed to complete its own start-up phase before the next task is created. Otherwise, there's a risk that higher-priority tasks created later could start executing before earlier tasks were ready for them, with (at best) unpredictable results.

When all the tasks have been created, INIT\$TASK has served its purpose. It must then call RQ\$SEND\$INIT\$TASK. This short procedure (actually self-contained in an OSP Support Package interface library, not built into the 80130) tells the OSP that all the off-spring tasks have been created for a given job. At this point, INIT\$TASK could continue with non-initialization activities. The code for KEYBOARD\$TASK might have been implemented here, for example. Since this example has nothing more to do, INIT\$TASK deletes itself with a final call to RQ\$DELETE\$TASK.

## Code Translation

That's all, folks. Mix together the above code fragments, declare literals and global variables, and compile until done (about four minutes). The source file name selected for this example is AP130.PLM. The compiler will produce two files: an annotated source listing (named AP130.LST) reproduced *in toto* in Appendix B, and a relocatable object file (AP130.OBJ) which will be used in the installation procedure discussed next.

## High-Level Parameter Passing Conventions

Well-designed programs generally rely on subprograms ("procedures" in PL/M terminology) for often-repeated instruction sequences, or to perform machine-level operations within High-Level Language programs. PL/M-86 and other Intel high-level languages use a standard set of conventions to pass parameters and results between procedures; assembly language programmers are advised to adhere to these conventions for software compatibility.

Before calling a subroutine or function, input parameters must be pushed sequentially onto the stack, in the order (left-to-right) they appear in the procedure parameter list. When eight-bit parameters are pushed, the high-order byte associated with them is undefined. Thirty-two-bit pointer values are pushed in two steps, offset word before base word. The stack "grows" down, so the left-most parameter will have highest-numbered address.

Functions which return a byte or word value (i.e., typed procedures) do so in the CPU AL or AX registers. Pointers are returned through the ES:AX register pair. The *PL/M Programming Manual* explains these conventions more fully.

One way to see how an assembly language routine would interface with PL/M is to first write a dummy PL/M procedure using the same parameter sequence as the desired assembly language routine. Compile this procedure with the compiler CODE switch set. The listing will then include the appropriate assembly language instruction sequence, and may be followed as a pattern for the final routine.

## SOFTWARE CONFIGURATIONS & INTEGRATION

When the application code has been written and compiled, the hardest part of program development is over. Before the code may be executed, though, the OSP must be told various things about the system hardware environment, desired software options, application job characteristics, and so forth.

This information is conveyed during a multi-phase sequence of steps collectively called the Configuration process. Though the process is somewhat lengthy and time-consuming, it is also very "mechanical"; the person doing the work does not need to understand any of the application code or even know what it does. Normally, configuration would be performed by a technician or a single member of the programming team, aided by appropriate SUBMIT command files. This chapter shows the full configuration and installation process for the demonstration system. For more details, refer to the *OSP User's Manual*.

The three phases of the configuration are:

1. Generating, linking, and locating OSP support code required for the EPROM immediately above the 80130 address space;
2. Linking and locating the object file for the application job developed in Section IV;
3. Creating, linking, and locating a short module (called the Root Job) which initializes the OSP and application jobs when system is reset.

Finally, of course, the absolute code resulting from each phase must be programmed into EPROMs or loaded into a test system before it can be executed.

Before starting, though, it is beneficial to draw up a memory map for host system hardware, to determine what sections of memory are available. This map will be filled in as each module is linked and located.

The prototype system memory space has two areas of interest: addresses 00000H through 01FFFFH contain RAM, while 0FC000H through 0FFFFFFH contain EPROM. Since the CPU uses the first 1K bytes of RAM for the CPU interrupt pointers, and the last 16 bytes for the restart sequence, these areas should be recorded on the map. For reference purposes, Figure 11 also indicates that addresses 0F8000H through 0FBFFFH enable the 80130 firmware. All this is shown in Figure 11.

### Generating the OSP Support Code

The OSP support code "customizes" the OSP firmware for a particular hardware environment, initializes the system, and supports extended software capabilities.

To define the hardware environment, the user creates a source file which invokes a series of Intel-supplied macros. Parameters for these macros specify the 80130 I/O base address, SYSTICK interval (in system clock cycles), and how the interrupt request pins will be used.

For instance, the code example in Figure 12 defines the prototype system hardware. This source file must be assembled, linked with several libraries from the OSP support disk, and located to produce the actual OSP support code. Figure 13 shows the actual sequence of commands needed. The DATA starting address specified within the LOC86 parameter list (00400H) is the first free byte of system RAM (see Figure 11); the CODE address (0F8000H) is simply the 80130 firmware starting address.

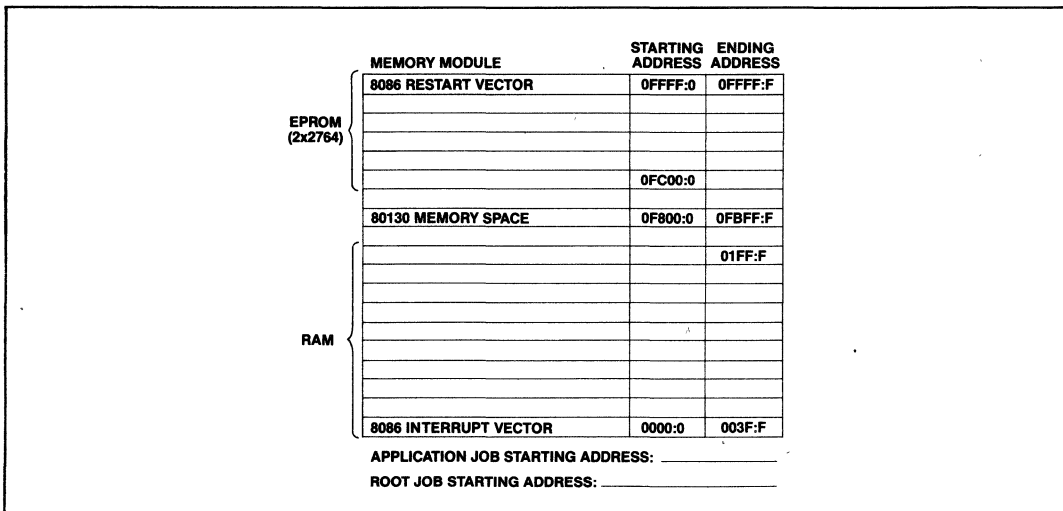


Figure 11. Example System Memory Map

```

#TITLE(80130 DEVICE CONFIGURATION TABLE)
NAMEDEVCF

#INCLUDE( F1 NDEVCF MAC)

%MASTER_PIC(80130, 2000H, 0, 0)

, SLAVE_PIC( SLAVE_TYPE, BASE_PORT, EDGE_VS_LEVEL, MASTER_LEVEL )

%TIMER(80130, 200BH, 2BH, 12500)

, NDP_SUPPORT( ENCODED_LEVEL )

END
    
```

Figure 12. 80130 Device Configuration Table

```

FO ASM86 F1 SUP130 A86 PRINT( F1 SUP130 LST) ERRORPRINT &
MACRO(80) PAGEWIDTH(132)

FO LINK86 &
F1 OSX LIB(OSX86,OSXCNF), &
F1 NUC1 LIB(NBEGIN), &
F1 ODEVCF OBJ, &
F1 OSX LIB, &
F1 NUC1 LIB, &
F1 OSX LIB, &
F1 NUC2 LIB, &
F1 OSX LIB, &
F1 NUC4 LIB, &
F1 OSX LIB, &
F1 NURSLV LIB, &
F1 OSX LIB &
TO F1 SUP130 LNK MAP PRINT( F1 SUP130 MP1) NAME(MINIMAL_BO130)

FO LOC86 &
F1 SUP130 LNK TO F1 SUP130 MAP PRINT( F1 SUP130 MP2) SC(3) &
SEGSIZE(STACK(0)) &
ADDRESSES(CLASSES(CODE(OFB000H),DATA(O0400H))) & &
ORDER(CLASSES(DATA,STACK)) &
OBJECTCONTROLS(NOLINES,NO COMMENTS, NOSYMBOLS)

```

**Figure 13. Support Code Configuration Commands**

A reliable and relatively straightforward way to perform this step is to create a file containing the exact command sequence shown in Figure 13 and execute this file using the SUBMIT utility program. Of course, the example assumes SUBMIT, ASM86, LINK86, and LOC86 are all on drive :F0:, and that the various libraries have been copied from the support disk to drive :F1:.

(An alternate, support-code configuration scheme lets the user modify the OSP software characteristics in special situations. A programmer working with iRMX 86, for instance, may wish to augment the OSP firmware to support all the iRMX Nucleus primitives. This would be done by editing and assembling file 0TABLE.A86 to select from a menu of software options, and modifying the linkage step slightly to include one of the iRMX 86 libraries. The OSP built-in features are more than sufficient for the purposes of this note, though, so only the first approach is illustrated.)

Appendix D reproduces the Locate map file produced during this phase. Near the end of file SUP130.MP2 is a table of memory usage, showing that the last bytes of RAM and ROM consumed are 00A6: FH and 0FC61: FH, respectively. Update Figure 11 with this information. (The final version of the demonstration-system memory map appears in Appendix C.) This phase needn't be repeated unless the system hardware characteristics change.

## Application Code Configuration

After compiling the application job, it must be linked with a library of interface routines from the support diskette, and located within available memory. Use RPIFC.LIB or RPIFL.LIB, depending on whether the job was compiled with the Compact or Large software model. Figure 14 is a command sequence file suggested for this purpose. Again, the starting addresses specified for LOC86 are taken from the system memory map.

Whenever the support code is reconfigured, check SUP130.MP2 to see if its memory needs have changed. If so, the application-job-configuration command file will need to be edited. This is still a lot simpler (not to mention more reliable) than retyping the whole sequence each time application jobs are revised. Readers familiar with the capabilities of the SUBMIT program may prefer to represent these variables by parameters, such that they may be easily specified each time the command file is invoked.

As in the first phase, examine the locate map ("AP130.MP2", reproduced in Appendix E) after the application code has been configured and update the memory map. Also, note the segment and offset values assigned to the initialization task. These will be needed later.

## Creating the Root Job

By now, all of the code needed to execute the application program has been prepared and is ready to run—except it has no way to get it started! The OSP hardware and system data structures must be initialized before INIT\$TASK can be created. A short module called the Root Job performs this function.

The process closely resembles the one which produced the OSP support code. First, determine various system characteristics. Then create a file defining these characteristics as macro input parameters. Finally, assemble, link, and locate the file to produce the final code.

Figure 15 is the Root Job source file for the demonstration system, dubbed RJB130.A86. It consists of just five macro calls. The %JOB macro defines certain characteristics of the application job; for a full description see the *OSP User's Manual*. One of these parameters is the initialization-task starting address (noted in the last step), which will likely change with each iteration of the application software.

The two %SAB macros define “System Address Blocks”—sections of the overall memory space which the OSP should not consider “free space.” Note that the first invocation blocks off the RAM addresses consumed so far in the memory map, plus an extra 140H bytes reserved for the Root Job initialization stack.

```

,
, SUBMIT FILE TO LINK APPLICATION JOB TO INTERFACE LIBRARY
, AND LOCATE RESULTING OUTPUT.
, REVISED 10/23/81 - JHW
,
LINKB6 F1 AP130.OBJ, F1 RPIFC LIB TO F1.AP130 LNK      &
MAP PRINT( F1 AP130 MP1)

LOCB6 F1 AP130 LNK TO :F1 AP130                        &
ORDER (CLASSES(DATA, STACK, MEMORY))                  &
SEGSIZE (STACK (0))                                   &
ADDRESSES (CLASSES (DATA (00A70H),                     &
                    CODE (0FC620H)))                   &
MAP PRINT ( F1 AP130 MP2)                              &
OBJECTCONTROLS (NOLINES, NOCOMMENTS, NOPUBLICS, NOSYMBOLS)

DHB6 F1 AP130 TO F1 AP130.H86

COPY F1 AP130 MP1 TO LP:

COPY F1 AP130 MP2 TO LP

```

Figure 14. Job Configuration Commands

```

, SOURCE PROGRAM DEFINING CHARACTERISTICS OF ROOT JOB FOR
, AP-130 DEMONSTRATION PROGRAM (JHW - 10/23/81)
,
%INCLUDE( F1 CTABLE MAC)
,
%SAB(0,00C0,U)
%SAB(0200,FFFF,U)
%JOB(0,0C0H,100H,OFFFHH,OFFFHH,1,0,0,1,0,100,0FC62,06B5,0,0,0,200H,0)
%DSX (OFB000H,N)
%SYSTEM(FB00,0,4,N,N,1)
,
END

```

Figure 15. Root Job Configuration File

(After completing this phase, examine RJB130.MP2 to confirm that 140H is the correct number.) The second %SAB invocation excludes addresses 02000H through 0FFFFFFH, all of which is non-RAM, either EPROM, 80130 firmware, or non-existent. The %SYSTEM macro defines system-wide software parameters.

Figure 16 is a command file to translate, link, and locate the root job. Once again, the LOC86 parameters come from Figure 11. The listings produced during this phase are reproduced in Appendix F. The final memory map appears in Appendix C.

## EPROM Programming

We are now ready to program EPROMs with the program modules linked and located above. Intel's Universal PROM Programmer (UPP) and a control program called the Universal Prom Mapper (UPM) will be used in this step. Particular commands to the UPM will vary with program size, memory location, and EPROM type, but the general sequence should resemble that shown here.

The first step is to invoke UPM and initialize the programming system, following a command sequence similar to that in Figure 17. The example system incorporates two 2764 devices, so 16K bytes of memory buffer are cleared.

Next, all the final code modules produced above (e.g., SUP130, AP130, and RJB130) must be loaded into the

UPM memory buffer. The three commands in Figure 18 perform this function.

When the final system is reset, execution must branch into the root job initialization sequence. When the absolute code modules have finished loading, manually patch a jump instruction into the buffer area corresponding to the CPU reset vector. The opcode for the 8086 or 8088 intersegment jump is OEAH; the instruction's address field must contain the address assigned to label RQ\$START\$ADDRESS (read from the root job locate map), the 16-bit segment offset (low byte first) followed by the segment base address (ditto). The UPM CHANGE command should be used to make this patch, as illustrated in Figure 19.

The UPM memory buffer now contains a complete image of the code needed for the system EPROMs. Up until now, all software-related steps—source code preparation, translation, linking and locating—have been the same for 8086- or 8088-based systems. At this point, however, the software installation procedures diverge slightly.

Recall that the 8086 fetches instructions 16 bits at a time, from coordinated pairs of EPROMs. One contains only even-numbered program bytes, the other, odd. To separate the linear UPM buffer into high- and low-order bytes for iAPX 86/30 designs, use the UPM STRIP command as shown in Figure 20.

Now "burn" the EPROMs with the PROGRAM command in Figure 21.

```

, LINK AND LOCATE THE iRMX 86 ROOT JOB
,
, MODIFIED FOR TWO-DRIVE OPERATION
, REVISED 10/25 - JHW
,
ASMB6 f1 RJB130 AB6 MACRO(75)
,
LINK86                                     &
      f1 root lib(root),                   &
      f1 RJB130 obj,                       &
      f1 root lib                           &
      TO f1 RJB130 lnk                      &
      MAP PRINT( f1 RJB130 mp1)
,
LOC86 f1 RJB130 lnk                       &
      TO F1 RJB130                          &
      MAP PRINT( f1 RJB130 mp2) &
      OC(nol1, nop1, nocm, nosb) &
      PC(nol1, pl, nocm, nosb) &
      SEGSIZE(stack(0)) &
      ORDER(classes(data, stack, memory)) &
      ADDRESSES(classes(code(0FD180H),    &
                          data(00AD0H))) &
,
OHB6 F1 RJB130 TO F1 RJB130 HB6
,
COPY F1 RJB130 LST TO LP
,
COPY F1 RJB130 MP1 TO LP
,
COPY F1 RJB130 MP2 TO LP
,

```

Figure 16. Root Job Configuration Commands

```
fill from 0 to 3fff with 0fff
```

Figure 17. UPM Initialization Sequence

```
read 86hex file : f1: sup130. h86 from 0 to 3fff start 0fc00h
read 86hex file : f1: ap130. h86 from 0 to 3fff start 0fc00h
read 86hex file : f1 : rjb130. h86 from 0 to 3fff start 0fc00h
```

Figure 18. UPM Commands to Load Hex Files

```
change 3ff0h-0eah, 11h, 00h, 18h, 0fdh
```

Figure 19. UPM Command to Patch Restart Vector

```
strip low from 0 to 3fff into 4000h
strip hi from 0 to 3fff into 6000h
```

Figure 20. UPM Commands to Strip High and Low Bytes

```
program from 4000h to 5fff start 0
program from 6000h to 7fff start 0
exit
```

Figure 21. UPM Commands to Program EPROMs

To save some trouble, the UPM invocation and all commands except the manual patch can be combined into a SUBMIT command file. Replace the CHANGE command with a control-E character so the operator can adjust the starting address for the iteration. Also place control-Es before each PROGRAM step to give the operator time to socket the next memory device.

## SUMMARY

The development of the 80130 marks a major milestone in the evolution of microcomputer systems. For the first time, a single VLSI device integrates the hardware facilities and operating system firmware needed by real-time multitasking applications. The 80130 offers the system hardware designer the advantages of higher integration—reduced device count, smaller boards, greater reliability—along with faster design cycles and optimal system performance.

The 80130 gives the software engineer built-in support for 35 standard operating system primitives. Application problems may now be solved at a higher level than

before. It is now possible for concurrent tasks to be dispatched, memory segments allocated, and messages relayed through mailboxes nearly as easily as subroutines, dynamic variables, and I/O ports were used in the past. In effect, Jobs, Tasks, Segments, Mailboxes, and Regions become new OSP data types, manipulated entirely by firmware in the 80130.

Yet despite standardizing these functions, the OSP does not restrict the user's flexibility. The device can accommodate a variety of hardware environments, and both the hardware and software capabilities are desired.

## ACKNOWLEDGEMENTS

The author would like to thank Peter Pederson for designing and implementing the demonstration system breadboard discussed in this note, Pam Johnson for her assistance in typing the manuscript, and Hal Kop, Lionel Smith, George Alexy, Chuck McMinn, and Sandy Wharton for their help in reviewing the drafts and providing many thoughtful comments and criticisms.

**APPENDIX A  
EXAMPLE SYSTEM SCHEMATICS**





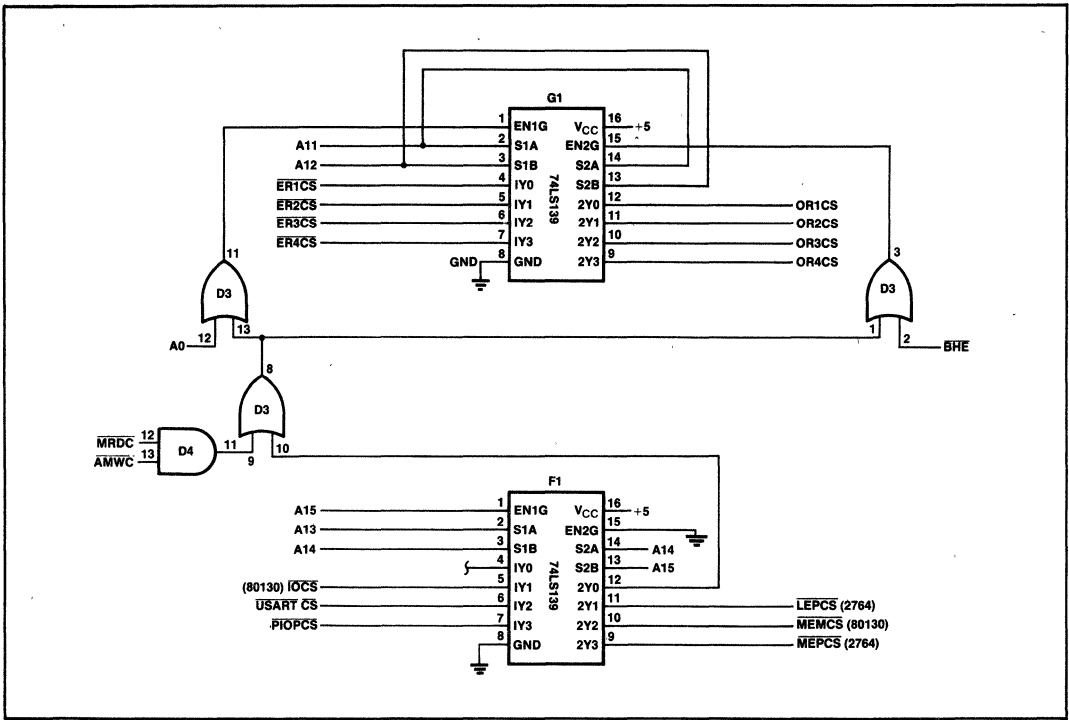


Figure A-1. Example System Schematics (continued)

**APPENDIX B  
SOURCE CODE LISTINGS**

ISIS-11 PL/M-86 V2.0 COMPILATION OF MODULE DEMO130  
 OBJECT MODULE PLACED IN :F1:AP130.OBJ  
 COMPILER INVOKED BY: PLM86 :F1:AP130.PLM DATE(12/21)

```

$DEBUG COMPACT ROM TITLE('AP-130 APPENDIX B - 12/21/81')

1      DEMO#130:  DD;

      /* SYSTEM-WIDE LITERAL DECLARATIONS:  */

2      1      DECLARE FOREVER LITERALLY 'WHILE 01H';

      /* I/O PORT DEFINITIONS:  */

3      1      DECLARE CHAR#51 LITERALLY '4000H',
      CMD#51 LITERALLY '4002H',
      STAT#51 LITERALLY '4002H';

4      1      DECLARE PPI#A LITERALLY '6001H',
      PPI#B LITERALLY '6003H',
      PPI#C LITERALLY '6005H',
      PPI#CMD LITERALLY '6007H',
      PPI#STAT LITERALLY '6007H';

5      1      DECLARE TIMER#CMD LITERALLY '200EH',
      BAUD#TIMER LITERALLY '200CH';

6      1      DECLARE AC#INTERRUPT#LEVEL LITERALLY '00111000B';

7      1      DECLARE CR LITERALLY '0DH',
      LF LITERALLY '0AH',
      BEL LITERALLY '07H';

8      1      DECLARE ASCII#CODE (16) BYTE DATA ('0123456789ABCDEF');

$EJECT

      $INCLUDE (.F1:NUCLUS.EXT)
=      $SAVE NOLIST
      $INCLUDE (.F1:NEXCEP.LIT)
=      $save nolist

      /* GLOBAL VARIABLE DECLARATIONS:  */

299    1      DECLARE DATA#SEG#PTR POINTER,
      DATA#SEG#ADDR STRUCTURE (OFFSET WORD,BASE WORD)
      AT (@DATA#SEG#PTR);

300    1      DECLARE HARDWARE$INIT$TASK$TOKEN WORD,
      STATUS$TASK$TOKEN WORD,
      MOTOR$TASK$TOKEN WORD,
      TIME$TASK$TOKEN WORD,
      AC$HANDLER$TOKEN WORD,
      CRT$OUT$TASK$TOKEN WORD,
      COMMAND$TASK$TOKEN WORD,
      INIT$TASK$TOKEN WORD;

301    1      DECLARE CRT$MAILBOX$TOKEN WORD,
      CRT$REGION$TOKEN WORD;

```

```

$EJECT

/* CODE EXAMPLE 2. SIMPLE CRT INPUT AND OUTPUT ROUTINES. */

302 1  C$OUT. PROCEDURE (CHAR);
303 2      DECLARE CHAR BYTE;
304 2      DO WHILE (INPUT(STAT$51) AND 01H)=0;
           /* NOTHING */
305 3      END;
306 2      OUTPUT(CHAR$51)=CHAR;
307 2      END C$OUT;

308 1  C$IN. PROCEDURE BYTE.
309 2      DO WHILE (INPUT(STAT$51) AND 02H)=0;
           /* NOTHING */
310 3      END;
311 2      RETURN INPUT(CHAR$51);
312 2      END C$IN;

$EJECT

/* CODE EXAMPLE 1. HARDWARE INITIALIZATION TASK. */

313 1  HARDWARE$INIT$TASK: PROCEDURE;
314 2      DECLARE HARD$INIT$EXCEPT$CODE WORD;
315 2      DECLARE PARAM$51 (*) BYTE DATA (40H, 8DH, 00H, 40H, 4EH, 27H);
316 2      DECLARE PARAM$51$INDEX BYTE;
317 2      DECLARE SIGN$ON$MESSAGE (*) BYTE DATA
           (CR, LF, 'IAPX 86/30 HARDWARE INITIALIZED', CR, LF);
318 2      DECLARE SIGN$ON$INDEX BYTE;

319 2      OUTPUT(PPI$CMD)=90H;
320 2      OUTPUT(TIMER$CMD)=0B6H;
321 2      OUTPUT(BAUD$TIMER)=33; /*GENERATES 9600 BAUD FROM 5 MHZ*/
322 2      OUTPUT(BAUD$TIMER)=0;
323 2      DO PARAM$51$INDEX=0 TO (SIZE(PARAM$51)-1);
324 3          OUTPUT(CMD$51)=PARAM$51(PARAM$51$INDEX);
325 3          END; /*OF USART INITIALIZATION DO-LOOP*/
326 2      DO SIGN$ON$INDEX=0 TO (SIZE(SIGN$ON$MESSAGE)-1);
327 3          CALL C$OUT(SIGN$ON$MESSAGE(SIGN$ON$INDEX));
328 3          END; /*OF SIGN-ON DO-LOOP*/
329 2      CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @HARD$INIT$EXCEPT$CODE);
330 2      CALL RQ$DELETE$TASK(0, @HARD$INIT$EXCEPT$CODE);
331 2      END HARDWARE$INIT$TASK;

$EJECT

/* CODE EXAMPLE 3. STATUS POLLING AND REPORTING TASK. */

332 1  STATUS$TASK: PROCEDURE;
333 2      DECLARE STATUS$COUNTER BYTE;
334 2      DECLARE STATUS$EXCEPT$CODE WORD;

335 2      STATUS$COUNTER=0;
336 2      CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @STATUS$EXCEPT$CODE);
337 2      DO FOREVER,
338 3          OUTPUT(PPI$B)=INPUT(PPI$A) XOR STATUS$COUNTER;
339 3          STATUS$COUNTER=STATUS$COUNTER+1;
340 3          CALL RQ$SLEEP(100, @STATUS$EXCEPT$CODE);
341 3          END;
342 2      END STATUS$TASK;

```

```
$EJECT
```

```
/* CODE EXAMPLE 4. STEPPER MOTOR CONTROL TASK. */
```

```
343 1  DECLARE CW$STEP$DELAY BYTE,
      CCW$STEP$DELAY BYTE,
      CW$PAUSE$DELAY BYTE,
      CCW$PAUSE$DELAY BYTE;

344 1  MOTOR$TASK: PROCEDURE;
345 2  DECLARE MOTOR$EXCEPT$CODE WORD;
346 2  DECLARE MOTOR$POSITION BYTE,
      MOTOR$PHASE BYTE;
347 2  DECLARE PHASE$CODE (4) BYTE
      DATA (00000101B; 00000110B, 00001010B, 00001001B);

348 2  CW$STEP$DELAY=50; /*INITIAL STEP DELAYS = 1/4 SECOND*/
349 2  CCW$STEP$DELAY=50;
350 2  CW$PAUSE$DELAY=200; /*PAUSES AFTER ROTATION = 1 SECOND*/
351 2  CCW$PAUSE$DELAY=200;
352 2  CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @MOTOR$EXCEPT$CODE);
353 2  DO FOREVER;
354 3  DO MOTOR$POSITION=0 TO 100;
355 4  MOTOR$PHASE=MOTOR$POSITION AND 0003H;
356 4  OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE);
357 4  CALL RQ$SLEEP(CW$STEP$DELAY, @MOTOR$EXCEPT$CODE);
358 4  END;
359 3  CALL RQ$SLEEP(CW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE);
360 3  DO MOTOR$POSITION=0 TO 100;
361 4  MOTOR$PHASE=(100-MOTOR$POSITION) AND 0003H;
362 4  OUTPUT(PPI$C)=PHASE$CODE(MOTOR$PHASE);
363 4  CALL RQ$SLEEP(CCW$STEP$DELAY, @MOTOR$EXCEPT$CODE);
364 4  END;
365 3  CALL RQ$SLEEP(CCW$PAUSE$DELAY, @MOTOR$EXCEPT$CODE);
366 3  END;
367 2  END MOTOR$TASK;
```

```
$EJECT
```

```
/* CODE EXAMPLE 5. INTERRUPT HANDLER TO TRACK 60 HZ INPUT. */
```

```
368 1  DECLARE AC$CYCLE$COUNT BYTE;

369 1  AC$HANDLER: PROCEDURE INTERRUPT 59; /*VECTOR FOR 80130 INT3*/
370 2  DECLARE AC$EXCEPT$CODE WORD;

371 2  CALL RQ$ENTER$INTERRUPT(AC$INTERRUPT$LEVEL, @AC$EXCEPT$CODE);
372 2  AC$CYCLE$COUNT=AC$CYCLE$COUNT+1;
373 2  IF AC$CYCLE$COUNT >= 60
      THEN DO;
375 3  AC$CYCLE$COUNT=0;
376 3  CALL RQ$SIGNAL$INTERRUPT(AC$INTERRUPT$LEVEL,
      @AC$EXCEPT$CODE);
377 3  END;
378 2  ELSE CALL RQ$EXIT$INTERRUPT(AC$INTERRUPT$LEVEL,
      @AC$EXCEPT$CODE);
379 2  END AC$HANDLER;
```

\$EJECT

/\* CODE EXAMPLE 7. PROTECTED CRT OUTPUT SUBROUTINE. \*/

```

380 1   PROTECTED$CRT$OUT: PROCEDURE (CHAR) REENTRANT;
381 2       DECLARE CHAR BYTE;
382 2       DECLARE CRT$EXCEPT$CODE WORD;
383 2       CALL RQ$RECEIVE$CONTROL(CRT$REGION$TOKEN, @CRT$EXCEPT$CODE);
384 2       DO WHILE (INPUT(STAT$51) AND 01H)=0;
           /* NOTHING */
385 3       END;
386 2       OUTPUT(CHAR$51)=CHAR;
387 2       CALL RQ$SEND$CONTROL(@CRT$EXCEPT$CODE);
388 2       END PROTECTED$CRT$OUT;

```

\$EJECT

/\* CODE EXAMPLE 6. INTERRUPT TASK TO MONITOR CLOCK TIME. \*/

```

389 1   DECLARE SECOND$COUNT BYTE,
           MINUTE$COUNT BYTE,
           HOUR$COUNT BYTE;

390 1   TIME$TASK: PROCEDURE;
391 2       DECLARE TIME$EXCEPT$CODE WORD;

392 2       AC$CYCLE$COUNT=0;
393 2       CALL RQ$SET$INTERRUPT(AC$INTERRUPT$LEVEL, 01H,
           INTERRUPT$PTR(AC$HANDLER), DATA$SEG$ADDR. BASE,
           @TIME$EXCEPT$CODE);
394 2       CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @TIME$EXCEPT$CODE);
395 2       DO HOUR$COUNT=0 TO 23;
396 3           DO MINUTE$COUNT=0 TO 59;
397 4               DO SECOND$COUNT=0 TO 59;
398 5                   CALL RQ$WAIT$INTERRUPT(AC$INTERRUPT$LEVEL,
           @TIME$EXCEPT$CODE);
399 5                   IF SECOND$COUNT MOD 5 = 0
                       THEN CALL PROTECTED$CRT$OUT(BEL);
401 5                   END; /* SECOND LOOP */
402 4                   END; /* MINUTE LOOP */
403 3                   END, /* HOUR LOOP */
404 2       CALL RQ$RESET$INTERRUPT(AC$INTERRUPT$LEVEL,
           @TIME$EXCEPT$CODE);
405 2       CALL RQ$DELETE$TASK(0, @TIME$EXCEPT$CODE);
406 2       END TIME$TASK;

```

```
$EJECT
```

```
/* CODE EXAMPLE 8. SUBROUTINE TO CREATE TIME-OF-DAY MESSAGE. */
```

```
407 1 PRINT$TOD: PROCEDURE;
408 2 DECLARE TOD$MESSAGE$TOKEN WORD;
409 2 DECLARE TOD$EXCEPT$CODE WORD;
410 2 DECLARE TOD$SEGMENT$OFFSET WORD,
    TOD$SEGMENT$BASE WORD;
411 2 DECLARE TOD$SEGMENT$PNTR POINTER AT (@TOD$SEGMENT$OFFSET);
412 2 DECLARE TOD$TEMPLATE (28) BYTE
    DATA (27, 'THE TIME IS NOW hh:mm:ss.', CR, LF);
413 2 DECLARE TOD$STRING BASED TOD$SEGMENT$PNTR (28) BYTE;
414 2 DECLARE TOD$STRING$INDEX BYTE;

415 2 TOD$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(28, @TOD$EXCEPT$CODE);
416 2 TOD$SEGMENT$BASE=TOD$MESSAGE$TOKEN;
417 2 TOD$SEGMENT$OFFSET=0;
418 2 DO TOD$STRING$INDEX=0 TO 27;
419 3 TOD$STRING(TOD$STRING$INDEX)=
    TOD$TEMPLATE(TOD$STRING$INDEX);
420 3 END;
421 2 TOD$STRING(17)=ASCII$CODE(HOUR$COUNT/10);
422 2 TOD$STRING(18)=ASCII$CODE(HOUR$COUNT MOD 10);
423 2 TOD$STRING(20)=ASCII$CODE(MINUTE$COUNT/10);
424 2 TOD$STRING(21)=ASCII$CODE(MINUTE$COUNT MOD 10);
425 2 TOD$STRING(23)=ASCII$CODE(SECOND$COUNT/10);
426 2 TOD$STRING(24)=ASCII$CODE(SECOND$COUNT MOD 10);
427 2 CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
    TOD$MESSAGE$TOKEN, 0, @TOD$EXCEPT$CODE);
428 2 RETURN;
429 2 END PRINT$TOD;
```

```
$EJECT
```

```
/* CODE EXAMPLE 9. SUBROUTINE TO CREATE SWITCH STATUS MESSAGE. */
```

```
430 1 PRINT$STATUS: PROCEDURE;
431 2 DECLARE STATUS$MESSAGE$TOKEN WORD;
432 2 DECLARE STATUS$EXCEPT$CODE WORD;
433 2 DECLARE STATUS$SEGMENT$OFFSET WORD,
    STATUS$SEGMENT$BASE WORD;
434 2 DECLARE STATUS$SEGMENT$PNTR POINTER
    AT (@STATUS$SEGMENT$OFFSET);
435 2 DECLARE STATUS$TEMPLATE (40) BYTE DATA
    (39, 'THE SWITCHES ARE NOW SET TO .....B', CR, LF);
436 2 DECLARE STATUS$STRING BASED STATUS$SEGMENT$PNTR (40) BYTE;
437 2 DECLARE STATUS$STRING$INDEX BYTE;
438 2 DECLARE BIT$PATTERN BYTE;

439 2 STATUS$MESSAGE$TOKEN=RQ$CREATE$SEGMENT(40,
    @STATUS$EXCEPT$CODE);
440 2 STATUS$SEGMENT$BASE=STATUS$MESSAGE$TOKEN;
441 2 STATUS$SEGMENT$OFFSET=0;
442 2 DO STATUS$STRING$INDEX=0 TO 39;
443 3 STATUS$STRING(STATUS$STRING$INDEX)=
    STATUS$TEMPLATE(STATUS$STRING$INDEX);
444 3 END;
445 2 BIT$PATTERN=INPUT(PPI$A);
446 2 DO STATUS$STRING$INDEX=29 TO 36;
447 3 STATUS$STRING(STATUS$STRING$INDEX)=
    ASCII$CODE(BIT$PATTERN AND 01H);
448 3 BIT$PATTERN=ROR(BIT$PATTERN, 1);
449 3 END;
450 2 CALL RQ$SEND$MESSAGE(CRT$MAILBOX$TOKEN,
    STATUS$MESSAGE$TOKEN, 0, @STATUS$EXCEPT$CODE);
451 2 END PRINT$STATUS;
```

```
$EJECT
```

```
/* CODE EXAMPLE 10. TASK TO RECEIVE MESSAGES AND TRANSMIT THEM TO CRT. */
```

```

452 1  CRT$OUT$TASK:  PROCEDURE;
453 2      DECLARE MESSAGE$LENGTH BYTE;
454 2      DECLARE MESSAGE$TOKEN WORD;
455 2      DECLARE RESPONSE$TOKEN WORD;
456 2      DECLARE MESSAGE$EXCEPT$CODE WORD;
457 2      DECLARE MESSAGE$SEGMENT$OFFSET WORD,
          MESSAGE$SEGMENT$BASE WORD;
458 2      DECLARE MESSAGE$SEGMENT$PNTR POINTER AT (@MESSAGE$SEGMENT$OFFSET);
459 2      DECLARE MESSAGE$STRING$CHAR BASED MESSAGE$SEGMENT$PNTR BYTE;

460 2      CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @MESSAGE$EXCEPT$CODE);
461 2      DO FOREVER;
462 3          MESSAGE$TOKEN=RQ$RECEIVE$MESSAGE(CRT$MAILBOX$TOKEN, OFFFFF,
          @RESPONSE$TOKEN, @MESSAGE$EXCEPT$CODE);
463 3          MESSAGE$SEGMENT$OFFSET=0;
464 3          MESSAGE$SEGMENT$BASE=MESSAGE$TOKEN;
465 3          MESSAGE$LENGTH=MESSAGE$STRING$CHAR;
466 3          DO MESSAGE$SEGMENT$OFFSET=1 TO MESSAGE$LENGTH;
467 4              CALL PROTECTED$CRT$OUT(MESSAGE$STRING$CHAR);
468 4          END,
469 3          CALL RQ$DELETE$SEGMENT(MESSAGE$TOKEN, @MESSAGE$EXCEPT$CODE);
470 3      END; /* OF FOREVER-LOOP */
471 2      END CRT$OUT$TASK;

```

```
$EJECT
```

```
/* CODE EXAMPLE 11. TASK TO POLL KEYBOARD AND PROCESS COMMANDS. */
```

```

472 1  COMMAND$TASK:  PROCEDURE;
473 2      DECLARE CONSOLE$CHAR BYTE;
474 2      DECLARE COMMAND$EXCEPT$CODE WORD;

475 2      CALL RQ$RESUME$TASK(INIT$TASK$TOKEN, @COMMAND$EXCEPT$CODE);
476 2      DO FOREVER;
477 3          CONSOLE$CHAR=C$IN AND 7FH;
478 3          CALL PROTECTED$CRT$OUT(CONSOLE$CHAR);
479 3          IF CONSOLE$CHAR=CR
          THEN CALL PROTECTED$CRT$OUT(LF);
481 3          IF (CONSOLE$CHAR >= '0') AND (CONSOLE$CHAR <= '9')
          THEN DO;
483 4              CALL PROTECTED$CRT$OUT(CR);
484 4              CALL PROTECTED$CRT$OUT(LF);
485 4              DO CASE (CONSOLE$CHAR-'0'),
486 5                  .CALL PRINT$TOD;
487 5                  CALL PRINT$STATUS;
488 5                  CALL RQ$SUSPEND$TASK(CRT$OUT$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
489 5                  CALL RQ$RESUME$TASK(CRT$OUT$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
490 5                  CALL RQ$DISABLE(AC$INTERRUPT$LEVEL,
          @COMMAND$EXCEPT$CODE);
491 5                  CALL RQ$ENABLE(AC$INTERRUPT$LEVEL,
          @COMMAND$EXCEPT$CODE);
492 5                  CALL RQ$SUSPEND$TASK(MOTOR$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
493 5                  CALL RQ$RESUME$TASK(MOTOR$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
494 5                  CALL RQ$SUSPEND$TASK(STATUS$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
495 5                  CALL RQ$RESUME$TASK(STATUS$TASK$TOKEN,
          @COMMAND$EXCEPT$CODE);
496 5                  END; /* OF CASE-LIST */
497 4          END; /* OF COMMAND PROCESSING */
498 3      END;
499 2      END COMMAND$TASK;

```



#EJECT

/\* CODE EXAMPLE 12. TASK TO INITIALIZE OSP SOFTWARE. \*/

```

500 1  INIT$TASK: PROCEDURE PUBLIC;
501 2  DECLARE INIT$EXCEPT$CODE WORD;

502 2      DATA$SEG$PTR=@INIT$TASK$TOKEN; /*LOAD DATA SEGMENT BASE*/
503 2  CRT$MAILBOX$TOKEN=RQ$CREATE$MAILBOX(0, @INIT$EXCEPT$CODE);
504 2  CRT$REGION$TOKEN=RQ$CREATE$REGION(0, @INIT$EXCEPT$CODE);
505 2  INIT$TASK$TOKEN=RQ$GET$TASK$TOKENS(0, @INIT$EXCEPT$CODE);
506 2  HARDWARE$INIT$TASK$TOKEN=RQ$CREATE$TASK
      (110, @HARDWARE$INIT$TASK, DATA$SEG$ADDR. BASE, 0, 300,
      0, @INIT$EXCEPT$CODE);
507 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
508 2  STATUS$TASK$TOKEN=RQ$CREATE$TASK(110, @STATUS$TASK,
      DATA$SEG$ADDR. BASE, 0, 300, 0, @INIT$EXCEPT$CODE);
509 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
510 2  MOTOR$TASK$TOKEN=RQ$CREATE$TASK(110, @MOTOR$TASK,
      DATA$SEG$ADDR. BASE, 0, 300, 0, @INIT$EXCEPT$CODE);
511 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
512 2  TIME$TASK$TOKEN=RQ$CREATE$TASK(120, @TIME$TASK,
      DATA$SEG$ADDR. BASE, 0, 300, 0, @INIT$EXCEPT$CODE);
513 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
514 2  CRT$OUT$TASK$TOKEN=RQ$CREATE$TASK(120, @CRT$OUT$TASK,
      DATA$SEG$ADDR. BASE, 0, 300, 0, @INIT$EXCEPT$CODE);
515 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
516 2  COMMAND$TASK$TOKEN=RQ$CREATE$TASK(130, @COMMAND$TASK,
      DATA$SEG$ADDR. BASE, 0, 300, 0, @INIT$EXCEPT$CODE);
517 2  CALL RQ$SUSPEND$TASK(0, @INIT$EXCEPT$CODE);
518 2  CALL RQ$END$INIT$TASK;
519 2  CALL RQ$DELETE$TASK(0, @INIT$EXCEPT$CODE);
520 2  END INIT$TASK;

521 1  END DEMO$130;

```

## MODULE INFORMATION.

```

CODE AREA SIZE      = 084CH   2124D
CONSTANT AREA SIZE  = 0000H    0D
VARIABLE AREA SIZE  = 0052H   82D
MAXIMUM STACK SIZE  = 0026H   38D
848 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-86 COMPILATION

**APPENDIX C  
SYSTEM MEMORY MAP**

EXAMPLE SYSTEM MEMORY MAP

	MEMORY MODULE	STARTING ADDRESS	ENDING ADDRESS
EPROM (2x2764)	8086 RESTART VECTOR	0FFF:0	0FFF:F
	ROOT JOB CODE AREA	0FD18:0	0FD36:6
	APPLICATION JOB CODE AREA	0FC62:0	0FD17:8
	OSP SUPPORT CODE AREA	0FC00:0	0FC61:F
	80130 MEMORY SPACE	0F800:0	0FBFF:F
RAM	(FREE SYSTEM RAM)	00C0:0	01FF:F
	ROOT JOB DATA AREA	00AD:0	00BF:F
	APPLICATION JOB DATA AREA	00A7:0	00AC:1
	OSP SUPPORT DATA AREA	0040:0	00A6:F
	8086 INTERRUPT VECTOR	0000:0	003F:F

INITIALIZATION TASK STARTING ADDRESS: FC62:06B5

ROOT JOB STARTING ADDRESS: FD18:0011

**APPENDIX D**  
**SUPPORT CODE LOCATE MAP**

# AP-130

```

ISIS-II MCS-B6 LOCATER, V1 2 INVOKED BY
FO LOC64
F1 SUP130 LNK TO F1 SUP130 MAP PRINT( F1 SUP130 MP2) SC(3) &
&
SEGSIZE(STACK(0))
ADDRESSES(CLASSES(CODE(OF8000H), DATA(00400H))) &
ORDER(CLASSES(DATA, STACK)) &
OBJECTCONTROLS(NOLINES, NOCOMMENTS, NOSYMBOLS)
WARNING 26 DECREASING SIZE OF SEGMENT
SEGMENT STACK
    
```

```

SYMBOL TABLE OF MODULE MINIMAL_80130
READ FROM FILE F1 SUP130 LNK
WRITTEN TO FILE F1 SUP130
    
```

BASE	OFFSET	TYPE	SYMBOL	BASE	OFFSET	TYPE	SYMBOL	BASE	OFFSET	TYPE	SYMBOL
0040H	0000H	PUB	INTERRUPTTASKVEC	0040H	0120H	PUB	DEFAULT_HANDLER	0040H	0144H	PUB	READYLISTROOT
0040H	0148H	PUB	INTERRORENTRY	0040H	014CH	PUB	SYSTEMEXCEPTIONH	0040H	0150H	PUB	DELETIONTASKTOME
0040H	0152H	PUB	EXTENSIONLISTROO	0040H	0154H	PUB	DELETION_OBJECT_	0040H	0156H	PUB	SYSTEMPOOLTOKEN
0040H	0158H	PUB	ROOTJOBTOKEN	0040H	015AH	PUB	MINTRANSIZE	0040H	015CH	PUB	LAST_NDP_TASK
0040H	015EH	PUB	NDP_INTERRUPT_LE	0040H	0160H	PUB	PARAM_VALIDATION	0040H	0162H	PUB	REGION_FLAGS
0040H	0164H	PUB	TASK_WAITING_FLG	0040H	0166H	PUB	REGION_TOKEN_TAB	0040H	0176H	PUB	SIGNAL_Q_INDEX
0040H	0178H	PUB	SIGNAL_Q	0040H	0168H	PUB	KERNEL_FLAG	0040H	01E9H	PUB	ACTIVATE_SIGNAL_
0040H	01EAH	PUB	FILLCHAR	0040H	01EBH	PUB	NUM_SLAVES	0040H	01ECH	PUB	OLD_SLAVE_NUM
0040H	01EDH	PUB	INTMASK	0040H	01F6H	PUB	DISABLEMASK	0040H	01FFH	PUB	LEVEL_SET_TABLE
0040H	0208H	PUB	IMR_PORT	0040H	021AH	PUB	EDI_PORT	0040H	022CH	PUB	ISR_PORT
0040H	023EH	PUB	PIC_INFO	0040H	0247H	PUB	CLOCK_SPEC_EOI	0040H	0248H	PUB	CLOCK_ON
0040H	0249H	PUB	CLOCK_OFF	0040H	024AH	PUB	CLOCK_LEVEL	0040H	05D0H	PUB	END_OF_DATA
F800H	45CCH	PUB	NDP_INTERRUPT_LE	F800H	45C2H	PUB	VALIDATE_PARAMS_	F800H	4542H	PUB	GETDESCRTOKEN
F800H	4556H	PUB	GETDESCRPOINTER	F800H	4567H	PUB	GETPDIINTER	F800H	433DH	PUB	SCANMEMORY
F800H	4538H	PUB	OVERFLOW	F800H	4533H	PUB	NENTRY_BODY	F800H	432EH	PUB	KSUSPEND
F800H	4529H	PUB	KINITIALIZE	F800H	4524H	PUB	KENABLELEVELNS	F800H	431FH	PUB	KENABLELEVEL
F800H	451AH	PUB	KCREATEREGIONNS	F800H	4515H	PUB	KCREATEDBJECTNS	F800H	4310H	PUB	KCREATEOBJECT
F800H	4508H	PUB	INITNDP	F800H	4506H	PUB	INITIALIZE	F800H	4301H	PUB	FINISHINITIALIZA
F800H	44FCH	PUB	EDI_ROUTINE	F800H	44F7H	PUB	DIVIDEBYZERO	F800H	44F2H	PUB	DECODE_LEVEL
F800H	44EDH	PUB	COMMON_ERROR	F800H	44E8H	PUB	CLOCKENTRY_BODY	F800H	44E3H	PUB	ARRAYBOUNDS
F800H	44D0H	PUB	SYSTEMEXCEPTIONH	F800H	4472H	PUB	INITIALIZE_TIMER	F800H	43AEH	PUB	INITIALIZE_PICS
F800H	435CH	PUB	INIT_INTERNAL_RE	F800H	434EH	PUB	NDP_INTERRUPT_HA	F800H	433FH	PUB	CLOCKENTRY
F800H	4336H	PUB	NENTRY	F800H	40FEH	PUB	INITIALIZENUCLEU	F800H	4086H	PUB	RQWAITINTERRUPT_
F800H	40B1H	PUB	RQSIGNALLINTERRUPT	F800H	40ACH	PUB	RQGETLEVEL_BODY	F800H	40A7H	PUB	RQEXITINTERRUPT_
F800H	40A2H	PUB	RQENTERINTERRUPT	F800H	409DH	PUB	RQDISABLE_BODY	F800H	4094H	PUB	RQWAITINTERRUPT
F800H	408AH	PUB	RQSIGNALLINTERRUPT	F800H	4080H	PUB	RQGETLEVEL	F800H	4076H	PUB	RQENTERINTERRUPT
F800H	406CH	PUB	RQEXITINTERRUPT	F800H	4062H	PUB	RQDISABLE	F800H	405DH	PUB	NUNLOCK_DELETION
F800H	4058H	PUB	NUNLOCKNS	F800H	4053H	PUB	NUNLOCK	F800H	404EH	PUB	NOPEN_DELETION_O
F800H	4049H	PUB	NOPENNS	F800H	4044H	PUB	NOPEN	F800H	403FH	PUB	NLOCK_DELETION_O
F800H	403AH	PUB	NLOCKNS	F800H	4035H	PUB	NLOCK	F800H	4030H	PUB	NCLOSE_DELETION_
F800H	402BH	PUB	NCLOSENS	F800H	4026H	PUB	NCLOSE	F800H	4021H	PUB	DELETERUNNINGTAS
F800H	401CH	PUB	DELETEDOBJECT	F800H	400AH	PUB	COPYRIGHT	F800H	4000H	PUB	NBEGIN
F800H	4000H	PUB	INIT_NUCLEUS_JUM	FC5DH	0004H	PUB	IMR_START	FC5CH	000EH	PUB	
FC5CH	000FH	PUB	INIT_CMD1	FC5CH	0010H	PUB	INIT_CMD5_MASTER	FC5CH	0011H	PUB	
FC5CH	0012H	PUB	INIT_CMD4_MASTER	FC61H	000EH	PUB	SLAVE_TABLE	FC61H	0003H	PUB	
FC61H	0005H	PUB	CLOCK_O_PORT	FC61H	0007H	PUB	CLOCK_COUNT	FC61H	000AH	PUB	
FC61H	0008H	PUB	C_CLOCK_SPEC_EOI	FC61H	000CH	PUB	C_CLOCK_ON	FC61H	0009H	PUB	
F800H	4576H	PUB	LEVEL7_HANDLER	F800H	4574H	PUB	PARAM_VALIDATION				

```

MEMORY MAP OF MODULE MINIMAL_80130
READ FROM FILE F1 SUP130 LNK
WRITTEN TO FILE F1 SUP130
    
```

### SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
00000H	003FFH	0400H	A	(ABSOLUTE)	
00400H	009EFH	05F0H	W	DATA	DATA
009F0H	009FFH	0010H	Q	INTVEC_REG_SEG	DATA
00A00H	00A0FH	0010H	Q	EXT_REG_SEG	DATA
00A10H	00A1FH	0010H	Q	JOB_REG_SEG	DATA
00A20H	00A2FH	0010H	Q	SEM_REG_SEG	DATA
00A30H	00A3FH	0010H	Q	MAIL_REG_SEG	DATA
00A40H	00A4FH	0010H	Q	DD_REG_SEG	DATA
00A50H	00A5FH	0010H	Q	POOL_REG_SEG	DATA

00A60H	00A6FH	0010H	0	DELETION_REG_S -EG	DATA	← LAST RAM BYTE USED
00A70H	00A70H	0000H	W	STACK	STACK	
00A70H	00A70H	0000H	0	?2SEG		
F8000H	FC5CDH	49CEH	W	CODE	CODE	
FC5CEH	FC5D2H	0005H	W	PIC_CNF_SEG	CODE	
FC5D4H	FC5E5H	0012H	W	_IMR_PORT	CODE	
FC5E6H	FC5F7H	0012H	W	_E0I_PORT	CODE	
FC5F8H	FC609H	0012H	W	_ISR_READ_PORT	CODE	
FC60AH	FC612H	0009H	B	_PIC_INFO	CODE	
FC613H	FC61CH	000AH	B	TIMER_CNF_SEG	CODE	
FC61EH	FC61EH	0000H	W	CSEG	CODE	
FC61EH	FC61FH	C 02H	W	SLAVE_SEG	CODE	← LAST EPROM BYTE USED
FC620H	FC620H	0000H	W	MEMORY	MEMORY	

GROUP MAP

ADDRESS	GROUP OR SEGMENT NAME
00400H	DGROUP
	DATA
	INTVEC_REG_SEG
	EXT_REG_SEG
	JOB_REG_SEG
	SEM_REG_SEG
	MAIL_REG_SEG
	DD_REG_SEG
	POOL_REG_SEG
F8000H	DELETION_REG_SEG
	CORUP
	CODE
	PIC_CNF_SEG
	_IMR_PORT
	_E0I_PORT
	_ISR_READ_PORT
	_PIC_INFO
	TIMER_CNF_SEG
	CSEG
	SLAVE_SEG

**APPENDIX E  
APPLICATION JOB LOCATE MAP**

# AP-130

```

ISIS-II MCS-86 LOCATER, V1 2 INVOKED BY
LOC86 F1 AP130 LNK TO F1 AP130
ORDER (CLASSES(DATA, STACK, MEMORY))
SEGSIZE (STACK (0))
ADDRESSES (CLASSES (DATA (00A70H),
                    CODE (0FC620H)))
MAP PRINT ( F1 AP130 MP2)
OBJECTCONTROLS (NOLINES, NOCMODULES, NOPUBLICS, NOSYMBOLS)
WARNING 26 DECREASING SIZE OF SEGMENT
SEGMENT STACK
    
```

```

SYMBOL TABLE OF MODULE DEMO130
READ FROM FILE F1 AP130.LNK
WRITTEN TO FILE F1 AP130
    
```

BASE	OFFSET	TYPE	SYMBOL	BASE	OFFSET	TYPE	SYMBOL
FC62H	0B3AH	PUB	RGENDINITTASK	FC62H	0B1CH	PUB	RQ_N_C_RETURN_40
FC62H	0B00H	PUB	RQ_N_C_RETURN_20	FC62H	0AE4H	PUB	RQ_N_C_RETURN_14
FC62H	0AC8H	PUB	RQ_N_C_RETURN_12	FC62H	0AA4H	PUB	RQ_N_C_RETURN_10
FC62H	0A90H	PUB	RQ_N_C_RETURN_8	FC62H	0A74H	PUB	RQ_N_C_RETURN_6
FC62H	0A5BH	PUB	RQ_N_C_RETURN_4	FC62H	0A3EH	PUB	RQERROR
FC62H	0A2BH	PUB	RQGETLEVEL	FC62H	0A0EH	PUB	RQSIGNALEXCEPTIO
							-N
FC62H	09F0H	PUB	RQWAITINTERRUPT	FC62H	09DAH	PUB	RQSIGNALINTERRUPT
							-T
FC62H	09D4H	PUB	RQDELETSEMAPHOR	FC62H	09CEH	PUB	RQDELETEMAILBOX
			E				
FC62H	098BH	PUB	RQEXITINTERRUPT	FC62H	09B2H	PUB	RQUNCATALOGOBJEC
							-T
FC62H	094CH	PUB	RQSENDUNITS	FC62H	09A6H	PUB	RQSPENDTASK
FC62H	09A0H	PUB	RQSETPRIORITY	FC62H	099AH	PUB	RQSETPDOLMIN
FC62H	0974H	PUB	RQSETOEXTENSION	FC62H	098EH	PUB	RQSENDMESSAGE
FC62H	098BH	PUB	RQSLLEEP	FC62H	0982H	PUB	RQSETINTERRUPT
FC62H	097CH	PUB	RQSETEXCEPTIONHA	FC62H	0976H	PUB	RQSENDCONTROL
			-NDLER				
FC62H	0970H	PUB	RQRECEIVEUNITS	FC62H	096AH	PUB	RQRESUMETASK
FC62H	093BH	PUB	RQRECEIVEMESSAGE	FC62H	0932H	PUB	RQRESETINTERRUPT
FC62H	092CH	PUB	RQRECEIVECONTROL	FC62H	0926H	PUB	RQDIFFSPRING
FC62H	0920H	PUB	RQLOOKUPOBJECT	FC62H	091AH	PUB	RQINSPECTCOMPOSI
							-TE
FC62H	0914H	PUB	RQGETTASKTOKENS	FC62H	090EH	PUB	RQGETTYPE
FC62H	090BH	PUB	RQGETSIZE	FC62H	0902H	PUB	RQGETPRIORITY
FC62H	08FCH	PUB	RQGETPOOLATTRIB	FC62H	08F6H	PUB	RQGETEXCEPTIONHA
							-NDLER
FC62H	08F0H	PUB	RQFORCEDELETE	FC62H	08EAH	PUB	RQENABLE
FC62H	08D4H	PUB	RQENTERINTERRUPT	FC62H	08CEH	PUB	RQENABLEDELETION
FC62H	08CBH	PUB	RQDELETETASK	FC62H	08C2H	PUB	RQDELETSEGMEN
FC62H	08ACH	PUB	RQDISABLE	FC62H	08A6H	PUB	RQDELETEREGION
FC62H	08A0H	PUB	RQDELETEJOB	FC62H	089AH	PUB	RQDELETEEXTENSIO
							-N
FC62H	0894H	PUB	RQDISABLEDELETIO	FC62H	088EH	PUB	RQDELETECOMPOSIT
			-N				-E
FC62H	088BH	PUB	RQCREATETASK	FC62H	0882H	PUB	RQCREATESEMAPHOR
							-E
FC62H	087CH	PUB	RQCREATESEGMENT	FC62H	0876H	PUB	RQCREATEREGION
FC62H	0870H	PUB	RQCATALOGOBJECT	FC62H	086AH	PUB	RQCREATEMAILBOX
FC62H	0864H	PUB	RQCREATEJOB	FC62H	085EH	PUB	RQCREATEEXTENSIO
							-N
FC62H	085BH	PUB	RQCREATECOMPOSIT	FC62H	0852H	PUB	RQALTERCOMPOSITE
			-E				
FC62H	084CH	PUB	RQACCEPTCONTROL	FC62H	06B5H	PUB	INITTASK
DEMO130			SYMBOLS AND LINES				
FD17H	000CH	SYM	MEMDRY	FC62H	0000H	SYM	ASCIICCODE
00A7H	0000H	SYM	DATABASEPTR	00A7H	0000H	SYM	DATABASEADDR
00A7H	0004H	SYM	HARDWAREINITTASK	00A7H	0006H	SYM	STATUSTASKTOKEN
			-TOKEN				
00A7H	0008H	SYM	MOTORTASKTOKEN	00A7H	000AH	SYM	TIMETASKTOKEN
00A7H	000CH	SYM	ACHANDLERTOKEN	00A7H	000EH	SYM	CRTOUPTASKTOKEN
00A7H	0010H	SYM	COMMANDTASKTOKEN	00A7H	0012H	SYM	INITTASKTOKEN
00A7H	0014H	SYM	CRMAILBOXTOKEN	00A7H	0016H	SYM	CRREGIONTOKEN
FC62H	00B4H	SYM	COUT	STACK	0004H	SYM	CHAR
FC62H	00A1H	SYM	CIN	FC62H	00B9H	SYM	HARDWAREINITTASK
00A7H	0018H	SYM	HARDINITEXCEPTC	FC62H	0010H	SYM	PARAMS1
			ODE				
00A7H	0040H	SYM	PARAMS1INDEX	FC62H	0016H	SYM	SIGNONMESSAGE
00A7H	0041H	SYM	SIGNONINDEX	FC62H	013BH	SYM	STATUSTASK
00A7H	0042H	SYM	STATUSCOUNTER	00A7H	001AH	SYM	STATUSXCEPTCODE
00A7H	0043H	SYM	CWSTEPDELAY	00A7H	0044H	SYM	CCWSTEPDELAY
00A7H	0045H	SYM	CWPAUSEDELAY	00A7H	0046H	SYM	CCWPAUSEDELAY
FC62H	0172H	SYM	MOTORTASK	00A7H	001CH	SYM	MOTOREXCEPTCODE
00A7H	0047H	SYM	MOTORPOSITION	00A7H	0048H	SYM	MOTORPHASE
FC62H	0039H	SYM	PHASECODE	00A7H	0049H	SYM	ACCYCLECOUNT
FC62H	0256H	SYM	ACHANDLER	00A7H	001EH	SYM	ACEXCEPTCODE
FC62H	029CH	SYM	PROTECTEDCRTOU	STACK	0006H	SYM	CHAR
STACK	0002H	SYM	CRTEXCEPTCODE	00A7H	004AH	SYM	SECONDNCOUNT
00A7H	004BH	SYM	MINUTECOUNT	00A7H	004CH	SYM	HOURCOUNT
FC62H	02CFH	SYM	TIMETASK	00A7H	0020H	SYM	TIMEXCEPTCODE
FC62H	038BH	SYM	PRINTOD	00A7H	0022H	SYM	TODMESSAGETOKEN
00A7H	0024H	SYM	TODXCEPTCODE	00A7H	0026H	SYM	TODSEGMENTOFFSET
00A7H	0028H	SYM	TODSEGMENTBASE	00A7H	0026H	SYM	TODSEGMENTPNTR
FC62H	003DH	SYM	TODTEMPLATE	00A7H	0026H	BAS	TODSTRING
00A7H	004DH	SYM	TODSTRINGINDEX	FC62H	04B9H	SYM	PRINTSTATUS
00A7H	002AH	SYM	STATUSMESSAGE	00A7H	002CH	SYM	STATUSXCEPTCODE
			-EN				



# AP-130

00A7H 002EH SYM STATUSSEGMENTOFF	00A7H 0030H SYM STATUSSEGMENTBAS	
00A7H 002EH SYM STATUSSEGMENTPNT	FC62H 0059H SYM STATUSTEMPLATE	
00A7H 002EH BAS STATUSSTRING	00A7H 004EH SYM STATUSSTRINGINDE	
00A7H 004FH SYM BITPATTERN	FC62H 052FH SYM CRTOUTTASK	
00A7H 0050H SYM MESSAGELENGTH	00A7H 0032H SYM MESSAGEOKEN	
00A7H 0034H SYM RESPONSETOKEN	00A7H 0036H SYM MESSAGEEXCEPTCOD	
00A7H 003BH SYM MESSAGESEGMENTOF	00A7H 003AH SYM MESSAGESEGMENTBA	
00A7H 003BH SYM MESSAGESEGMENTPN	00A7H 003BH BAS MESSAGESTRNGCHA	
FC62H 05AFH SYM COMMANDTASK	00A7H 0051H SYM CONSOLECHAR	
00A7H 003CH SYM COMMANDEXCEPTCOD	FC62H 0685H SYM INITTASK	← INITIALIZATION TASK STARTING ADDRESS
00A7H 003EH SYM INITEXCEPTCODE		
FC62H 0087H LIN 304	FC62H 0084H LIN 302	
FC62H 0094H LIN 306	FC62H 0093H LIN 305	
FC62H 00A1H LIN 308	FC62H 0099H LIN 307	
FC62H 00B0H LIN 310	FC62H 00A4H LIN 309	
FC62H 00B9H LIN 312	FC62H 00B3H LIN 311	
FC62H 00BCH LIN 319	FC62H 00B9H LIN 313	
FC62H 00CBH LIN 321	FC62H 00C2H LIN 320	
FC62H 00D1H LIN 323	FC62H 00CEH LIN 322	
FC62H 00EFH LIN 325	FC62H 00E4H LIN 324	
FC62H 010CH LIN 327	FC62H 00FBH LIN 326	
FC62H 011FH LIN 329	FC62H 0116H LIN 328	
FC62H 0139H LIN 331	FC62H 012CH LIN 330	
FC62H 013EH LIN 335	FC62H 013BH LIN 332	
FC62H 0150H LIN 337	FC62H 0143H LIN 336	
FC62H 015CH LIN 339	FC62H 0150H LIN 338	
FC62H 016DH LIN 341	FC62H 0160H LIN 340	
FC62H 0172H LIN 344	FC62H 0170H LIN 342	
FC62H 017AH LIN 349	FC62H 0175H LIN 348	
FC62H 0184H LIN 351	FC62H 017FH LIN 350	
FC62H 0196H LIN 353	FC62H 0189H LIN 352	
FC62H 01A5H LIN 355	FC62H 0196H LIN 354	
FC62H 01BDH LIN 357	FC62H 0180H LIN 356	
FC62H 01D6H LIN 359	FC62H 01CDH LIN 358	
FC62H 01F5H LIN 361	FC62H 01E6H LIN 360	
FC62H 020FH LIN 363	FC62H 0202H LIN 362	
FC62H 022BH LIN 365	FC62H 021FH LIN 364	
FC62H 023BH LIN 367	FC62H 023BH LIN 366	
FC62H 0259H LIN 371	FC62H 0256H LIN 369	
FC62H 0270H LIN 373	FC62H 0266H LIN 372	
FC62H 027DH LIN 376	FC62H 027BH LIN 375	
FC62H 02BDH LIN 378	FC62H 02BAH LIN 377	
FC62H 029CH LIN 380	FC62H 029AH LIN 379	
FC62H 02ACH LIN 384	FC62H 02A0H LIN 383	
FC62H 02BBH LIN 386	FC62H 02BBH LIN 385	
FC62H 02CAH LIN 388	FC62H 02C2H LIN 387	
FC62H 02D2H LIN 392	FC62H 02CFH LIN 390	
FC62H 02F3H LIN 394	FC62H 02D7H LIN 393	
FC62H 030FH LIN 396	FC62H 0300H LIN 395	
FC62H 032DH LIN 398	FC62H 031EH LIN 397	
FC62H 034EH LIN 400	FC62H 033AH LIN 399	
FC62H 035DH LIN 402	FC62H 0394H LIN 401	
FC62H 036FH LIN 404	FC62H 0366H LIN 403	
FC62H 0389H LIN 406	FC62H 037CH LIN 405	
FC62H 03BEH LIN 415	FC62H 038BH LIN 407	
FC62H 03A7H LIN 417	FC62H 039FH LIN 416	
FC62H 03BEH LIN 419	FC62H 03ADH LIN 418	
FC62H 03D9H LIN 421	FC62H 03D0H LIN 420	
FC62H 040EH LIN 423	FC62H 03F5H LIN 422	
FC62H 0440H LIN 425	FC62H 0427H LIN 424	
FC62H 0472H LIN 427	FC62H 0459H LIN 426	
FC62H 0489H LIN 429	FC62H 0487H LIN 428	
FC62H 048CH LIN 439	FC62H 0489H LIN 430	
FC62H 04A5H LIN 441	FC62H 049DH LIN 440	
FC62H 04BCH LIN 443	FC62H 04ABH LIN 442	
FC62H 04D7H LIN 445	FC62H 04CEH LIN 444	
FC62H 04EEH LIN 447	FC62H 04DFH LIN 446	
FC62H 050FH LIN 449	FC62H 050BH LIN 448	
FC62H 052DH LIN 451	FC62H 051BH LIN 450	
FC62H 0532H LIN 460	FC62H 052FH LIN 452	
FC62H 053FH LIN 462	FC62H 053FH LIN 461	
FC62H 0560H LIN 464	FC62H 055AH LIN 463	
FC62H 0573H LIN 466	FC62H 055BH LIN 465	
FC62H 0592H LIN 468	FC62H 058BH LIN 467	
FC62H 05AAH LIN 470	FC62H 059DH LIN 469	
FC62H 05AFH LIN 472	FC62H 05ADH LIN 471	
FC62H 05BFH LIN 476	FC62H 05B2H LIN 475	
FC62H 05C9H LIN 478	FC62H 05BFH LIN 477	
FC62H 05DAH LIN 480	FC62H 05D0H LIN 479	
FC62H 05F4H LIN 483	FC62H 05E0H LIN 481	
FC62H 0600H LIN 485	FC62H 05FAH LIN 484	
FC62H 0616H LIN 487	FC62H 0610H LIN 486	
FC62H 062CH LIN 489	FC62H 061CH LIN 488	
FC62H 064CH LIN 491	FC62H 063CH LIN 490	
FC62H 066CH LIN 493	FC62H 065CH LIN 492	
FC62H 068CH LIN 495	FC62H 067CH LIN 494	
FC62H 06B0H LIN 498	FC62H 069CH LIN 496	
FC62H 06B5H LIN 500	FC62H 06B3H LIN 499	
	FC62H 068BH LIN 502	

# AP-130

FC62H	06C4H	LIN	503		FC62H	06D5H	LIN	504
FC62H	06E6H	LIN	505		FC62H	06F6H	LIN	506
FC62H	071FH	LIN	507		FC62H	072CH	LIN	508
FC62H	0755H	LIN	509		FC62H	0762H	LIN	510
FC62H	078BH	LIN	511		FC62H	079BH	LIN	512
FC62H	07C1H	LIN	513		FC62H	07CEH	LIN	514
FC62H	07F7H	LIN	515		FC62H	0804H	LIN	516
FC62H	082DH	LIN	517		FC62H	083AH	LIN	518
FC62H	083DH	LIN	519		FC62H	084AH	LIN	520
FC62H	00B4H	LIN	521					

MEMORY MAP OF MODULE DEMO130  
 READ FROM FILE F1 AP130 LNK  
 WRITTEN TO FILE F1 AP130

SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
00A70H	00AC1H	0052H	W	DATA	DATA
← LAST DATA BYTE OF APPLICATION JOB					
00AC2H	00AC2H	0000H	W	STACK	STACK
00ADDH	00ADDH	0000H	G	~SEG	
FC620H	FD17BH	0B5CH	W	CODE	CODE
← LAST CODE BYTE OF APPLICATION JOB					
FD17CH	FD17CH	0000H	W	MEMORY	MEMORY

GROUP MAP

ADDRESS	GROUP OR SEGMENT NAME
FC620H	CGROUP
	CODE
00A70H	DGROUP
	DATA

**APPENDIX F  
ROOT JOB LOCATE MAP**

```
ISIS-II MCS-86 LOCATER, V1 2 INVOKED BY
LOC86 f1 RJB130 lmk &
      TO F1 RJB130 &
      MAP PRINT( f1 RJB130 mp2) &
      OC(nol1, nop1, nocm, nosb) &
      PC(nol1, pl, nocm, nosb) &
      SECSIZE(stack(0)) &
      ORDER(classes(data, stack, memory)) &
      ADDRESSES(code(OFD180H), &
                data(OOAD0H)))
```

WARNING 26 DECREASING SIZE OF SEGMENT  
SEGMENT STACK

SYMBOL TABLE OF MODULE ROOT  
READ FROM FILE F1 RJB130 LNK  
WRITTEN TO FILE F1 RJB130

BASE	OFFSET	TYPE	SYMBOL	BASE	OFFSET	TYPE	SYMBOL	
FD18H	0180H	PUB	NUC_INIT_ENTRY	FD18H	0184H	PUB	CODEDATA	
<b>FD18H</b>	<b>0011H</b>	<b>PUB</b>	<b>R0STARTADDRESS</b>	<b>← ROOT JOB STARTING ADDRESS</b>				FD18H 0010H PUB INTERROR
FD18H	0000H	PUB	CRASH	FD18H	002AH	PUB	R0ROOTJOBVERSION	
FD18H	0030H	PUB	ROTTTASK	FD18H	010CH	PUB	SYSTEMSUICIDE	
FD18H	0118H	PUB	R0CREATEJOB	FD18H	011EH	PUB	R0GETTASKTOKENS	
FD18H	0124H	PUB	R0SUSPENDTASK	FD18H	012AH	PUB	R0_N_C_RETURN_6	
FD18H	0146H	PUB	R0_N_C_RETURN_40	FD18H	0162H	PUB	R0ERRDR	
00ADH	0000H	PUB	J0BNUMBER	00ADH	0002H	PUB	ROTTTASKSTATUS	

MEMORY MAP OF MODULE ROOT  
READ FROM FILE F1 RJB130 LNK  
WRITTEN TO FILE F1 RJB130

MODULE START ADDRESS PARAGRAPH = FD18H OFFSET = 0011H  
SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
00AD0H	00AD3H	0004H	W	DATA	DATA
<b>00AD4H</b>	<b>00BFFH</b>	<b>012CH</b>	<b>W</b>	<b>INIT_STACK</b>	<b>STACK</b>
00C00H	00C00H	0000H	W	STACK	STACK
00C00H	00C00H	0000H	G	??SEG	STACK
FD180H	FD339H	01BAH	W	CODE	CODE
FD33AH	FD345H	000CH	W	SAB_DESCRIPTOR	CODE
				-S	
<b>FD346H</b>	<b>FD366H</b>	<b>0021H</b>	<b>W</b>	<b>U_U_DESCRIPTOR</b>	<b>CODE</b>
				-S	
FD368H	FD368H	0000H	W	MEMORY	MEMORY

GROUP MAP

```
ADDRESS GROUP OR SEGMENT NAME
00AD0H DGROUP
      DATA
FD180H CGROUP
      CODE
      SAB_DESCRIPTOR
      U_U_DESCRIPTOR
```

November, 1982

# Let Operating Systems Aid in Component Designs

George Helder  
OEM Microcomputer Systems  
Applications Engineering

# LET OPERATING SYSTEMS AID IN COMPONENT DESIGNS

The iRMX 86 operating system processor package offers hardware designers a set of thoroughly tested software primitives upon which to build present and future custom hardware designs

by George Heider

Component users build application systems by integrating standard and custom hardware, software, and packaging. Microprocessors and other very large scale integration components are replacing much custom hardware with larger, more powerful standard hardware modules. Microprocessors lead to powerful systems, but they often require complex system management software. While this complex software often comprises one third or less of the final system software, it may require two thirds or more of the storage development effort. Worse, bugs in system management software sometimes do not show up until late in development or after the product is at the customer's site.

One solution to this problem is to employ standard management software such as operating systems. More complex, multifunction applications in a realtime environment benefit greatly from operating systems. Examples of these applications include file subsystems, public automatic branch exchange (PABX) systems, and transaction processing systems. But implementing

*George Heider is a senior applications engineer at Intel's OEM Microcomputer Systems Div, 5200 NE Elam Young Pkwy, Hillsboro, OR 97123. He works primarily with 16-bit software applications, including the iRMX 86 operating system. Previous experience includes telecommunications systems engineering, microprocessor systems, microprocessor operating system development, and disk storage system software. Mr Heider holds an MS in computer science from the University of California, Santa Barbara and a BSEE from Oregon State University.*

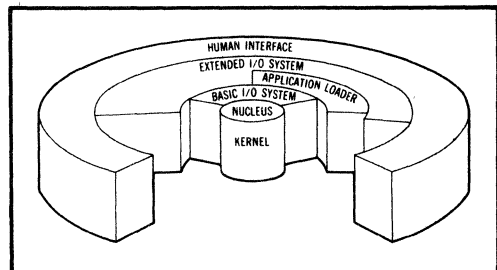


Fig 1 iRMX operating system architecture. Kernel consists of primitives also implemented in hardware in iAPX 86/30 and iAPX 88/30 OSP.

operating system functions in a component design requires new software tools, education, and expertise. Also, these functions are often specific to the particular design, so tools and expertise developed for one application are not suitable for subsequent designs.

These problems are directly addressed by the Intel iAPX 86/30 and iAPX 88/30 operating system processors (OSP) and the iRMX<sup>®</sup> 86 operating system. The iRMX 86 is a full-featured, realtime multitasking operating system for iAPX 86 or iAPX 88 based systems. The Intel OSP implements the iRMX 86 kernel functions in hardware consisting of an iAPX 86 or iAPX 88 central processor coupled with an operating system firmware (OSF) component, the Intel 80130. The OSF extends the base iAPX 86 and iAPX 88 architecture by adding 37 operating system primitive instructions to the base iAPX 86 or iAPX 88 instruction set; systems can be built directly on the OSP. System implementation time is thus decreased by having fully debugged operating system functions in hardware. Further capabilities can be added by iRMX<sup>™</sup> is a trademark of Intel Corp

extending the set of the OSP primitives or by integrating portions of the iRMX 86 on top of the OSP.

### Operating system architecture

The iRMX 86 architecture shown in Fig 1 consists of the nucleus and layers for the basic input/output (I/O) system, extended I/O system, application loader, and human interface. The system also provides a debugger, a terminal handler, a bootstrap loader, and a patch facility.

While the nucleus is the lowest layer of the operating system, fundamental system functions are handled by the nucleus kernel, which is the core of any operating system. The kernel controls memory allocation, allocates processor resources, communicates between processes, and manages interrupts. In the Intel OSP these functions are implemented in hardware. (OSP functions are described in Table 1; additional functions supported by the iRMX 86 nucleus are shown in Table 2.) Software development can be based on either the OSP or the iRMX 86, allowing software development to proceed in parallel with hardware development.

In addition to the operating system primitives, the OSP contains timers and interrupt control logic expandable from 8 to 57 interrupt levels. The timers include a system clock, Reserved delay timer, and baud rate generator. The 40-pin OSP has bus buffers and demultiplex logic, which allows it to interface directly to the iAPX 86 or iAPX 88 multiplexed bus. The OSP can be located at any 16 byte address boundary in the 1M-byte system address space. Application interface to OSP stepping and revision levels is independent. A block diagram of the 80130 is shown in Fig 2.

Minimum hardware requirements for the iRMX 86 operating system shown in Fig 3 are 1.8k bytes of random access memory (RAM), about 16k bytes of kernel code memory, and integrated circuits. By comparison, the OSP shown in Fig 4 still requires 1.8k bytes of RAM, but does not require the kernel code, the programmable interrupt controller, or the programmable interrupt timer. These are all replaced by the OSP. Approximately 1k bytes of required system configuration code are not shown in Figs 3 and 4.

### Kernel functions

Since it defines system architecture, application requests for system operations like interrupt management and memory allocation must go through the kernel. These

TABLE 1  
OSP primitives

Primitive	Description
<b>JOB</b>	
CREATE JOB	Creates a job partition including memory pool, task list, and stack area.
<b>TASK</b>	
CREATE TASK	Creates a task with specified environment and priority. Task is created in ready state. Checks for insufficient memory available within containing job.
DELETE TASK	Deletes a task from system as well as from any queues it is awaiting. Task's state and stack segment are deallocated.
SUSPEND TASK	Suspends a task (changes its status to suspended) or increases task's suspension count by 1. A sleeping task may also be suspended and will awaken suspended unless resumed.
RESUME TASK	Decreases suspension count of a task by 1. If at that point count is reduced to 0, task state is made ready. If it was suspend-asleep, it is put back to sleep.
SLEEP	Puts task in asleep state; up to 10 ms units can be specified.
GET TASK TOKENS	Gives token for a task or task's job partition.
<b>INTERRUPT</b>	
SET PRIORITY	Changes task's priority to value passed in primitive.
SET INTERRUPT	Assigns an interrupt handler to a level. Task that makes this call is made interrupt task for same level, unless call indicates there is no interrupt task.
RESET INTERRUPT	Disables an interrupt level; cancels interrupt handler; deletes interrupt task for level if assigned.
GET LEVEL	Returns number of the interrupt level for highest priority interrupt handler currently in operation (several interrupt handlers can be operating).
EXIT INTERRUPT	Completes interrupt processing and sends end of interrupt signal to hardware.
SIGNAL INTERRUPT	Invokes interrupt task assigned to a level from that level's interrupt handler.
WAIT INTERRUPT	Suspends interrupt task state pending a signal interrupt from an interrupt handler. Used by an interrupt task to signal its readiness to service an interrupt.
ENTER INTERRUPT	Sets data segment base for an interrupt handler.
ENABLE	Enables external interrupt level.
DISABLE	Disables an external interrupt level.
GET EXCEPTION HANDLER	Reads location and exception handling mode of current OSP exception handler for a task.
SET EXCEPTION HANDLER	Establishes location and exception handling mode of current OSP exception handler for task.
SIGNAL EXCEPTION	Notifies current OSP exception handler of exception.

requests are made by system calls, or primitives, which are comparable to subroutine calls for system actions. Since the kernel manages much of the system hardware, the application code need not concern itself with many hardware details. This independence is not absolute, however: system hardware or resources not managed by the kernel still require application code.

Basic kernel concepts can be explained using a general purpose system (Fig 5). Input data can be characters, analog signals, or digital signals; processing can be numerical analysis, editing, spectrum analysis, process control algorithms, or virtually any other transformation. Processed data must be sent to an interrupt driven output device—a display, a communications line,

<u>Primitive</u>	<u>Description</u>
<b>SEGMENT</b>	
CREATE SEGMENT	Dynamically allocates area of memory of specified length in 16-byte paragraph units up to 64k-byte maximum (eg, for use as buffer). Returns location token for segment allocated.
DELETE SEGMENT	Deallocates memory segment indicated by parameter token.
ENABLE DELETION	Allows deletion of system data type value indicated by location token.
DISABLE DELETION	Prevents deletion of system data type value indicated by location token.
<b>MAILBOX</b>	
CREATE MAILBOX	Creates a mailbox with specified task queuing discipline. Returns location token.
DELETE MAILBOX	Deletes a mailbox and returns its memory. If tasks are waiting for mailbox, they are awakened (ie, their state is made ready) with appropriate exception condition. If messages are waiting for tasks, they are discarded
SEND MESSAGE	Sends message segment to mailbox.
RECEIVE MESSAGE	Task is ready to receive message at mailbox. Task is placed on mailbox task queue. Task can wait for response indefinitely, wait (generally 10 ms) units, or not wait. When complete, primitive returns to task the location token of message segment received.
<b>REGION</b>	
CREATE REGION	Creates region data type value, specifying queuing discipline. Returns token for region.
DELETE REGION	Deletes region if the region is not in use.
ACCEPT CONTROL	Gains control of region if region immediately available, but does not wait if not available.
RECEIVE CONTROL	Same primitive as accept control but task that performs it may elect to wait.
SEND CONTROL	Relinquishes region.
<b>OTHER</b>	
SET OS EXTENSION	Links new primitive with kernel.
GET TYPE	Gives system type code of a system data type.

control hardware, or mass storage. In this general purpose system, input, process, and output are the only functions, or tasks, that make up the system.

#### Buffer management

Assume input data will be placed into 128-byte buffers by the input task. Without help from the operating system, the buffers must be prelocated in RAM. Software is needed to manage the buffers, which must be given to the tasks in the correct sequence and returned for reuse when empty. If the buffers are too small, or if RAM is moved, the software must handle these changes.

If an operating system or OSP is used, the locations and sizes of RAM are made known to the kernel during

system initialization. The input task requests each buffer, or memory segment, from the kernel by making the kernel system call "create segment" with 128 bytes. If a larger buffer is needed, the create segment call needs a larger value for the size parameter. When the buffer is full, the input task gives the segment to the process task. When the buffer is no longer needed, it can be returned to the system memory pool by a "delete segment" system call. Because the kernel dynamically manages memory allocation and buffer access, no additional code for these functions is necessary.

#### Communication and synchronization through mailboxes

The sample system needs a dispatching algorithm to send the segments from task to task. Such an algorithm can be written without an operating system. For example, the input task can fill a buffer and call the process task. When the process task finishes, it can call the output task; the output task can finish with the buffer and return. When control returns to the input task, system processing for that buffer is complete. Another method is to have a polling task occasionally check if buffers are ready to be sent to other tasks. Both methods are inefficient and rigid, requiring that each task finish processing data in each buffer before another task can run.

With an operating system, the buffers can be sent from task to task through "mailboxes"—places where tasks can send or receive data. (See Fig 6.) Task A sends a message (segment) to mailbox 1 and specifies mailbox 2 as a return mailbox. Task A then waits for a return message at mailbox 2. Task B receives the message (segment) from mailbox 1, then sends a return message with status to mailbox 2.

Task A receives the return message, which contains task B status, and synchronizes the two tasks.

In general, each task obtains a segment, modifies its contents, sends the segment to the next task, and waits for another segment. The input task first gets a segment using "create segment." When the segment is full, the input task uses the kernel call "send message" to send the segment to mailbox A. The process task uses the "receive message" system call to wait at mailbox A for the segment. The process task receives the segment, processes the data, puts the new data in the segment, and sends the segment to mailbox B. The process task then waits at mailbox A for the next segment from the input task. The output task takes the segment from mailbox B



TABLE 2

## Additional primitives supported by the iRMX 86 nucleus

**CATALOGING SYSTEM****DATA TYPES**

CATALOG OBJECT	Catalogs system data type token under name given by task in job partition directory.
UNCATALOG OBJECT	Removes name and token from job partition directory.
LOOKUP OBJECT	Uses name to find token cataloged in job partition directory.

**NEW SYSTEM****DATA TYPES**

CREATE EXTENSION	Notifies kernel of new system data type code for new system data type.
DELETE EXTENSION	Removes system data type code and deletes all composite system data types with that system data type code.
CREATE COMPOSITE	Creates new system data type from list of current system data types and system data type code received from create extension.
DELETE COMPOSITE	Deletes new system data type.
INSPECT COMPOSITE	Gives list of system data types that form new system data type.
ALTER COMPOSITE	Changes list of system data types that form new system data type.

**SEMAPHORES**

CREATE SEMAPHORE	Creates semaphore system data type.
DELETE SEMAPHORE	Deletes semaphore system data type
SEND UNITS	Task adds a number of units to semaphore.
RECEIVE UNITS	Task asks for a number of units from semaphore Task can wait for response indefinitely, wait (generally 10 ms), or not wait.

**OTHER****PRIMITIVES**

GET PRIORITY	Gives priority level of task.
FORCE DELETE	Deletes system data type even if disabled delete has been called for system data type.
GET SIZE	Gives byte size of memory segment.

**ADDITIONAL JOB****PRIMITIVES**

OFFSPRING	Returns child job partitions created by a task in parent job partition.
GET POOL ATTRIBUTES	Gives memory pool attributes of job partition, including pool minimum, pool maximum, initial size, number of bytes used, and number of bytes available.
SET POOL MINIMUM	Changes pool minimum for job partition.
DELETE JOB	Deletes job partition and returns its memory to parent job partition.

and outputs the data. The output task has two choices: it can either delete the segment, letting the input task create more segments, or it can send the segment to mailbox C. After sending or deleting the segment, the output task waits at mailbox B for the next segment from the process task. If the output task sent the segment to mailbox C, the input task segments from the output task, synchronizing the input task with the output task. If the output task deleted the segment, the input task creates a new segment and waits for input data. The entire process runs continuously, synchronized by mailboxes and segment availability. Additionally, the tasks can elect to wait for a specified amount of time at mailboxes, and if no segments

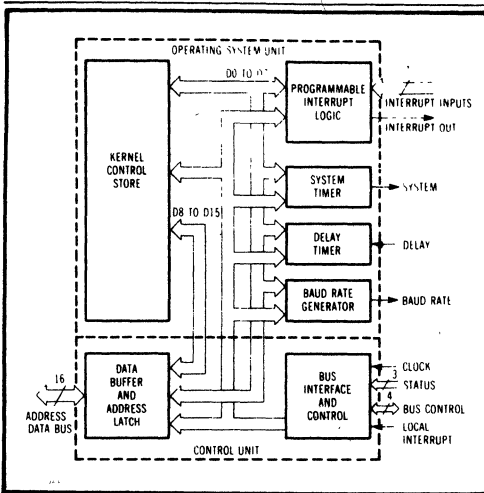
appear, an error routine can alert the system operator that processing has stopped.

The mailbox method has several advantages over synchronization algorithms and polling tasks. The entire process is synchronized by the availability of data in segments, eliminating the need for algorithms and extra code; the same process applies whether the tasks operate at the same or different speeds. Also, burst input or output rates can be handled by adding buffers. For instance, if too much data arrives for the process or output tasks to handle immediately, the input task fills multiple buffers and passes them to mailbox A. The process task takes each segment in turn. After processing is completed, the segments are all sent to mailbox C, and the process waits for the next burst of data. The only interfaces between the tasks are mailboxes and segments, so tasks can be easily replaced or added to the processing loop; the same scheme works for larger or smaller segments.

**Tasks and task scheduling**

Tasks are independent bodies of executing code, initialized and scheduled by the kernel. Therefore, tasks must have iRMX 86 parameters like priority, initial memory resources, entry address, and other iRMX 86 data. A task is like an expanded subroutine managed by an operating system. The actual application code is written much the same as it is without an operating system except that requests are made using kernel calls.

Even though the system's multiple independent tasks appear to run simultaneously, only one task actually runs at one time. Some method of scheduling is needed to decide which task receives control of the system processor; this scheduling depends on the task priority. Since data coming into a system must not be missed, the input task has the highest priority. Data going out of the system are next in importance, so the output task has second priority; the sequential process task has the lowest priority. The scheduling algorithm is simple—the highest priority task that is ready to run will get control of the processor. This is an example of preemptive priority. In this case, ready to run means that a task is complete—it has a segment to fill and data coming in (input task), data to process (process task), or data to output (output task). For instance, if input data arrives when the process task is running and the input task has a buffer waiting for data, the input task will preempt the process task to receive



**Fig 2 80130 firmware component performs clock and interrupt control functions, and supplies operating system primitives.**

the data. If the input task is not running and the hardware driven by the output task is ready to output another data value, the output task will receive control of the processor.

Since the operating system schedules the tasks, each task is designed as though it has sole control of the processor. Tasks make system calls such as receive message, which may cause another task to run because no message is waiting. In addition, interrupts will likely cause a different task to run. The kernel can schedule the tasks because only interrupts or system calls can cause a higher priority task to become ready, and both of these are handled by the kernel. Thus any time an interrupt occurs or a system call is made, the kernel runs the highest priority task that is ready. The tasks are written without any code to manage scheduling. The kernel scheduling is general purpose, so adding new tasks to the system does not require modifying the

scheduling functions. A system with work balanced among the tasks runs as though all tasks perform simultaneously.

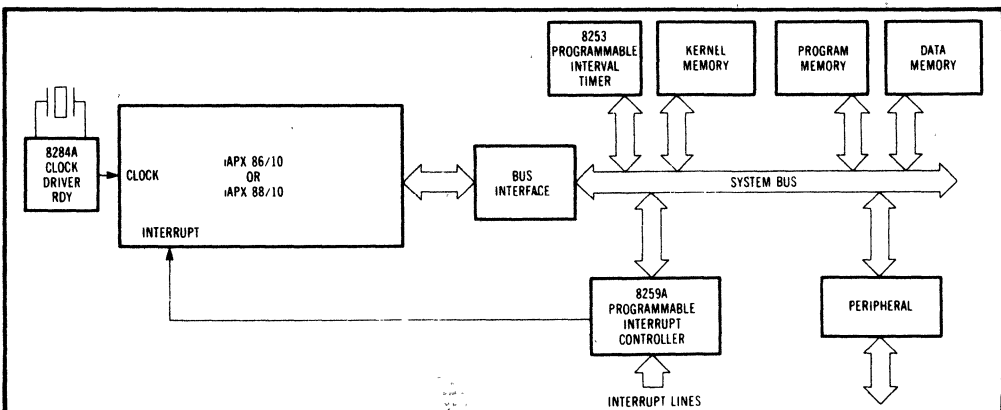
The net result of task scheduling is that the system runs as fast as it can. When data come in, the input task will always get control of the processor. The output task will execute whenever it has data to send and the input task is not running. The process task will run whenever it has data and no other tasks are running. Also, tuning the system is easier with the standardized mailbox interfaces: slower tasks can be easily removed and replaced with faster tasks, and remaining tasks will not be affected.

In a multitasking system, multiple independent tasks execute concurrently. Buffer transfers occur through mailboxes rather than through a direct interface to tasks, and system functions not related to the primary data processing functions can be handled by other tasks. For example, a supervisory task that monitors a system console for operator requests can be added to the system at a lower priority than the process task. No changes to any scheduling algorithm would be required.

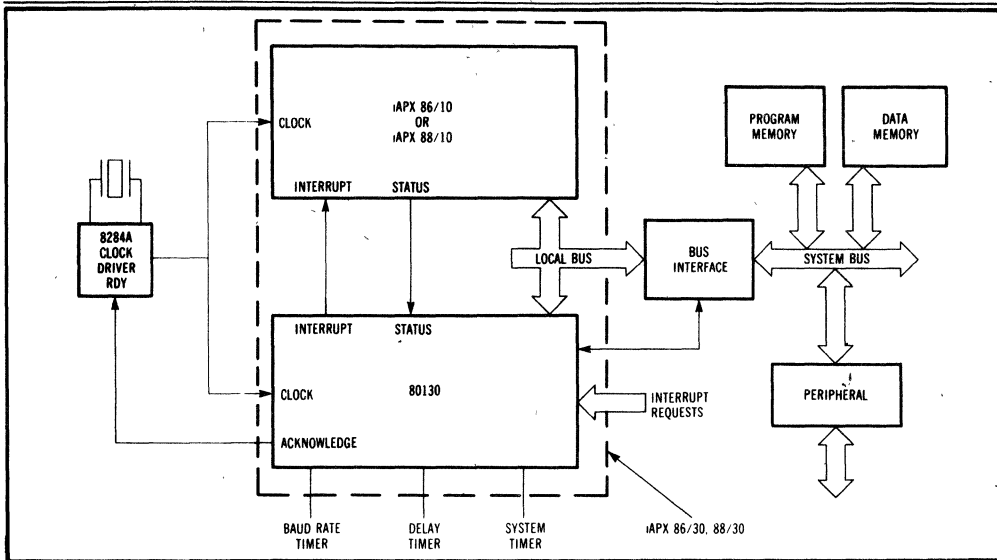
**Interrupt management**

The iRMX 86 kernel and the OSP provide two classes of interrupt management: interrupt handlers and interrupt tasks. An interrupt handler is a short procedure whose only function is to respond to the interrupt as quickly as possible. All interrupts become disabled in order to let the interrupt handler execute at top speed. Interrupt handlers can make only a few system calls. In the sample system, the interrupt procedure receives a data value, places it in a buffer, and returns. When the buffer is full, the interrupt handler notifies the interrupt task. Typical response time for an 8-MHz iAPX 86 processor, from the time an interrupt occurs until the interrupt handler gets control is 30 to 50  $\mu$ s. In the unlikely event of a worst-case time, response time is about 160  $\mu$ s.

Higher priority interrupts are enabled when an interrupt handler gets control, is 30 to 50  $\mu$ s. In the unlikely task uses a mailbox to pass the full buffer on to the next task. Since both interrupts and tasks have priorities assigned to them, the kernel uses the task priority to



**Fig 3 iRMX 86 hardware requirements. Operating system processor fits into basic hardware system for iRMX 86 and brings with it functions of kernel memory, 8259A programmable interrupt controller, and 8253 programmable interrupt timer.**



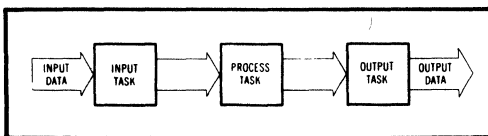
**Fig 4 Basic hardware system with iAPX OSP. OSP replaces kernel code, programmable interrupt controller, and programmable interval timer.**

determine if interrupts should be disabled or enabled. If the task priority is higher than an interrupt priority, that interrupt is disabled while the task is running. A priority level can be given to a task that disables all, some, or none of the interrupts: ie, defining a task that is more important than all interrupts (initialization task), more important than some interrupts (input task), or less important than all interrupts (processing task).

**Multiprogramming**

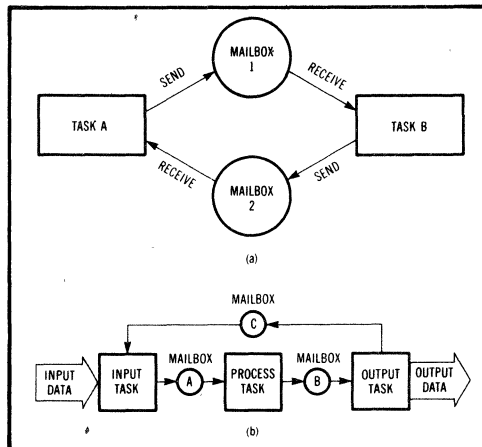
System parameters in a component system are normally well defined: RAM locations are fixed, code addresses are known, and address and I/O ports are specified. Application code usually depends on these parameters. If the system changes, substantial alterations are often needed in the application code. However, if an iRMX 86 operating system is used, the kernel is made aware of system resources during system configuration. System configuration assigns these resources to "jobs."

Jobs do not do work but instead serve as resource boundaries, containing tasks that accomplish system functions. Many component applications systems, including the sample system, will have only one job. All system resources are given to the job and all tasks are contained there. When the system is initialized, the job is created and control is passed to the first task in the job.



**Fig 5 General purpose system consists of 3 basic functions. Application code receives data, places data in buffer, then processes it. Processed data are sent to interrupt driven output devices.**

Multiprogramming occurs when a system has two or more jobs. The system boundaries provided by jobs confine errors and define limits for system resources such as memory. These boundaries limit the effect of one job on another. For instance, the system debugger is a separate job. During development, the sample processing system would look like Fig 7. After development, the debugger would be removed, leaving only the application system. The job environment of the processing system is not affected by adding or removing the debugger. The overall system will, of course, be affected



**Fig 6 Mailboxes allow intertask communication by providing places to send and receive messages (a). Synchronization is easy since tasks can poll a mailbox and wait for messages. Mailboxes also form interfaces between tasks in application system (b) so tasks can be easily added or removed without changing code.**

because removing the debugger will cause more system resources to be available for other jobs.

The jobs, tasks, segments, and mailboxes are part of a large set of system data types which are data structures managed by the operating system. System data types are manipulated only through system calls, which enforce the rules that govern their use. Together system data types and system calls form the application interface to the operating system. This interface provides not only a good boundary for error detection and debugging, but also common architecture that can be carried from application to application.

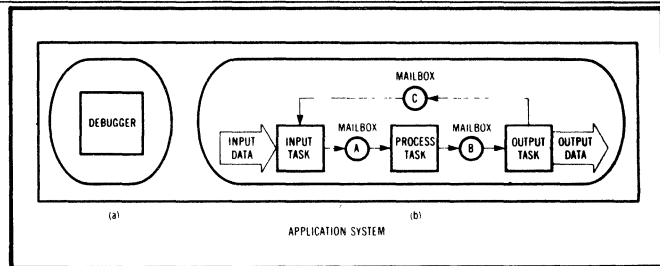
### Debugging

The iRMX 86 operating system has a debugger that interprets and uses system data types, and manipulates them to control the system. For example, the processing flow in the sample system can be halted by the debugger when a segment is sent to mailbox A. Data flow through the system can be traced by halting or breakpointing the system as the segment goes from mailbox to mailbox.

Debugging is further aided by the modularity of the tasks and jobs. Modules limit error effects; the interfaces between the modules are well defined; and the modules are easily inserted or removed. A standard system debugger can be used for all applications, avoiding the need to develop specific diagnostic tools.

### Conclusion

Multiprogramming and multitasking promote application code modularity, allowing applications to be created by adding new functions to old software. The same scheduling and kernel interfaces work for systems with only a few tasks, or systems with many tasks performing multiple processes. An entirely new process can be added to the example by adding more tasks. If the new and existing processes have nothing in common, the new process can be in a different job. If both processes can share general purpose tasks, such as output, the new



**Fig 7 Job structure for development provides distinct boundaries so that a debugger (a) or other piece of development software can be used during system development and later removed without disturbing application job (b).**

process can be in the same job and use the mailbox interfaces to send data to one output task. If system designers are careful, they can design systems whose functions can be added in the field. Thus, expensive custom software will not have to be rewritten for each new application.

Users with a wide range of applications will find that this approach allows them to implement a corresponding range of capabilities, expanding an OSP based system up to a high level human interface. A complete iRMX 86 operating system includes extensive I/O capabilities, a debugger, an application loader, a bootstrap loader, and integrated user console functions. Such a system can perform general purpose processing and still provide all iRMX 86 facilities. With these features, one operating system can be used for current projects and expanded for future ones, minimizing software learning curves for new applications.

### Bibliography

- Introduction to the iRMX 86 Operating System*, no 9803124, Intel Corp, Santa Clara, Calif, 1982
- iRMX 86 Nucleus Reference Manual*, no 9803122, Intel Corp, Santa Clara, Calif, 1981
- Using the iRMX 86 Operating System on iAPX 86 Component Designs*, Application Note AP110, Intel Corp, Santa Clara, Calif, 1981
- J. Zarella, *Operating Systems Concepts and Principles*, Microcomputer Applications, Suisun City, Calif, 1979

June 1983

**Software That  
Resides in Silicon**

**Ron Stamp  
and Jim Person  
Intel Corporation**

# Software That Resides in Silicon

Ron Stamp and Jim Person, Intel Corporation

**S**ilicon software sounds like a contradiction in terms. The casting of software in silicon implies that the software cannot be changed; yet software does and must change. For example, it must be possible to alter a microprocessor operating system so that the system will support different hardware and software designs, as well as accommodate new hardware components and applications. And if the software has been committed to silicon, then a way must exist to overcome any bugs that are discovered later.

## Design Considerations

Silicon software consists of two kinds of code: on-chip code and off-chip code (see Figure 1). In a typical case, some of the off-chip code works closely with the on-chip code, and is developed as part of the silicon software package. This special off-chip (or "support") code might contain initialization, interface, system, and version update codes. For silicon software to tolerate change and be usable in more than one system, the on-chip code must have three qualities: position independence, configuration independence and stepping independence.

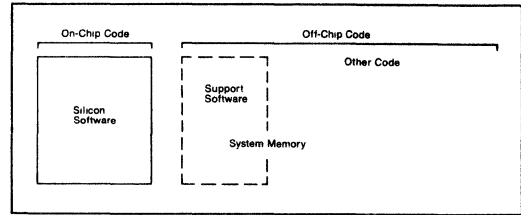
### Position Independence

Because the most advanced microprocessors address at least 1 megabyte of memory, system software that resides in silicon must work right regardless of its location in memory. Absolute addresses in the read-only, on-chip code or data restricts the configuration of the system. Because the on-chip code recognizes only offsets, absolute addresses are unacceptable. On-chip code cannot presume to know the location of any code or data, it can only presume to know the structure of the data which it accesses. It cannot know, except relatively, where in memory it (or any other code) resides. If the on-chip code is to be position independent, then any absolute addresses needed by the on-chip code must be obtained via the processor's registers.

Position independence is not a new concept; in fact, it is rather an obvious requirement for silicon software. Compilers and relocatable assemblers allow linking and locating, thus making it easier to produce position-independent code. But most of these tools can also produce code that is not position independent. Silicon software developers need to be aware of the position-independence requirement throughout the design, implementation and test phases for their products.

### Configuration Independence

The second requirement for silicon-resident software is that the on-chip code must not depend on the underlying hardware and software configuration of the system. Instead, the on-chip code must have indirect access to other code or data, and must then check the run-time data to deduce the system configuration.



**FIGURE 1.** Silicon software is divided into on-chip code and off-chip code. The off-chip code either directly supports the on-chip code or contains other applications code.

Because of the read-only nature of silicon software, constants can cause problems when they are located within the on-chip code. Values representing a hardware device must not reside on-chip if that device can be located anywhere in the system, or when values support several devices having similar functions but different programming interfaces. Indirect access is necessary for all values that vary depending on the configuration of the system.

### Stepping Independence

Stepping independence is an expansion of configuration independence, and is perhaps the most elusive of the requirements to be met by software intended for residence in silicon. A "step" is an updated version of the on-chip code. The on-chip code and the off-chip code must remain compatible, regardless of changes in either of them. Stepping independence exists when all versions of the on-chip code work with all versions of the off-chip code.

If stepping independence is taken into consideration when the silicon software is developed, then provisions can be made for the subsequent additions of options without changing the on-chip code. Otherwise, the static nature of the on-chip code might make it impossible to add options. Although configuration independence can be designed into software from the start, stepping independence can be achieved only if a system's existing silicon software does not include features that prevent it.

One type of data that is likely to change between steps is the value representing the size of a data area. If the software is to be stepping independent, it cannot know the sizes of the data areas accessed by on-chip code prior to run time. (No problems arise if on-chip and off-chip code agree on the size of the data area.)

But what happens if the on-chip code is not from the same version of the product as the off-chip code, and if the size of the data area has changed between versions? If the size of the data area is defined by a constant in the on-chip code, then that area might be smaller than the off-chip code expects it to be. This misunderstanding can lead to disaster as the off-chip code reads and writes beyond the data area.

This problem is solved when the on-chip code ascertains the size of the data area from off-chip data. Thus, the size of the data areas for the system becomes a configuration option.

**Getting the Bugs Out of Silicon Software**

Every large program contains bugs. Designers usually remove bugs by modifying the program to correct the problem, and then discarding the old program. However, a program in silicon cannot be modified without stepping the component. And even so, it is undesirable to discard the outdated component.

Software designed for silicon should include a facility for fixing bugs in on-chip code. One way to fix an on-chip bug is to prevent access to the routine containing the bug. A correct version of the routine is provided off-chip, and program execution is forced to branch to the off-chip version whenever the routine is invoked. Modular programming practices during development help reduce the cost of such off-chip duplication.

This on-chip bug-fix works well over time. Each component step has an associated collection of bug-fix modules. The collection is updated for each new version of the product, as component steps fix known bugs. During system configuration, the user specifies which component step is being used; the fixes for that step are included automatically in the off-chip code. Because of this facility, one step looks just like another to the user.

**Intel's OSF: A Software Component**

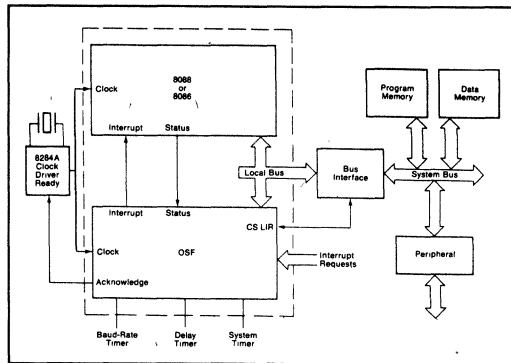
The Operating System Firmware (OSF) component consists of several hardware modules (see Figure 2). These modules provide two functions that are essential to operating systems: interrupts and timers. The OSF modules include a Control Store (16K bytes of fast ROM) to contain the silicon software, three programmable interval timers, an eight-input programmable interrupt controller, a bus interface, control logic, a data buffer, and address latch logic.

**The 80130: The iRMX™ 86 Kernel in Silicon**

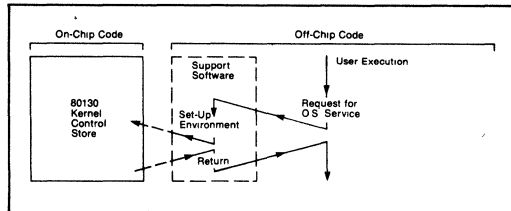
Intel's first software-on-silicon product is the 80130. It provides a functional subset of the iRMX™ 86 Nucleus, which is the heart of the iRMX 86 operating system (OS). The iRMX 86 OS is a real-time, multi-tasking, multiprogramming operating system intended for 16-bit microprocessor designs. The iRMX 86 family of standard software modules includes a nucleus, a stand-alone terminal handler, a stand-alone debugger, an asynchronous I/O system, a synchronous I/O system, a loader, a human interface, and options required for real-time applications. The nucleus manages the creation and dynamic deletion of all system architectural features (tasks, program environments, memory segments, data-communication managers, etc.). It also schedules tasks, based on priority, interrupt management, memory management, validation of parameters, management of exceptional conditions, and co-processor support.

**How the 80130 Satisfies the Silicon Software Criteria**

The iRMX 86 Nucleus provides both the on-chip and off-chip codes needed to implement the operating system. The on-chip code resides in the 16K-byte ROM space of the 80130. It is the main portion of the Nucleus code, and includes the kernel of the



**FIGURE 2. The OSF component works with systems that use the iAPX 86, 88, 186, or 188 microprocessor. Close coupling of the CPU and the OSF allows maximum zero-wait-state performance of the OSF software.**



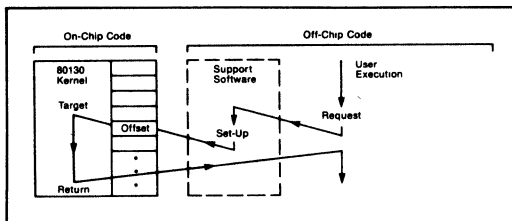
**FIGURE 3. The position-independent interface supplies data location and run-time values, and starts on-chip execution of the software.**

operating system and the primitives, which are present in the basic 80130 configuration. The off-chip code is stored in external RAM or ROM. It consists of initialization code, and code that either cannot be position independent or cannot be known before a given system is configured.

Position independence is guaranteed if entry to the on-chip code is possible only through an interface in the off-chip code that sets up the necessary registers. The off-chip position-independence interface (see Figure 3) provides an absolute data location and begins on-chip execution by the silicon-resident code. All run-time values can be determined based on the data location. On-chip execution gives the processor a location in the on-chip code from which other on-chip locations can be calculated.

It was relatively easy to make the 80130 configuration independent, because (like most operating-system kernels) it contains only general-purpose functions. The off-chip code contains all the drivers for particular peripheral chips. The Interactive Configuration Utility integrates the drivers with the 80130.

The interface between the off-chip and on-chip codes remains stable across component steps. The stepping-independence interface (see Figure 4) resides on the chip, and is a map of the on-chip code. This interface gives the off-chip code indirect access to all on-chip "publics" (e.g., externally accessible routines, modules, and labels). It is also a chart that routes execution to the proper on-chip location. The off-chip code uses an index of this chart to specify which public should



**FIGURE 4.** All on-chip accesses are routed through the on-chip stepping-independence interface, which provides compatibility between on-chip and off-chip code. Because the interface structure stays constant, the external reference also stays constant, while the on-chip OFFSET changes to point to the new location of the on-chip code.

be accessed. The index of a given routine remains the same across component steps, even though the actual address (offset into the component) of the public has changed. For different versions of the on-chip and off-chip codes to work correctly, all access from outside the component must be routed through the stepping-independence interface.

**The 80150: CP/M-86\* in Silicon**

Intel's decision to implement CP/M-86 operating system in silicon (the 80150) raised a different design problem. With the 80130, Intel only had to deal with Intel-designed software. Code design, implementation, extensions, corrections, support, and the subsequent effect on the end user were all under Intel's control. The selection of an independent software system such as CP/M-86 (a product of Digital Research, Inc.) introduced new factors into the implementation.

**The CP/M-86 Architecture**

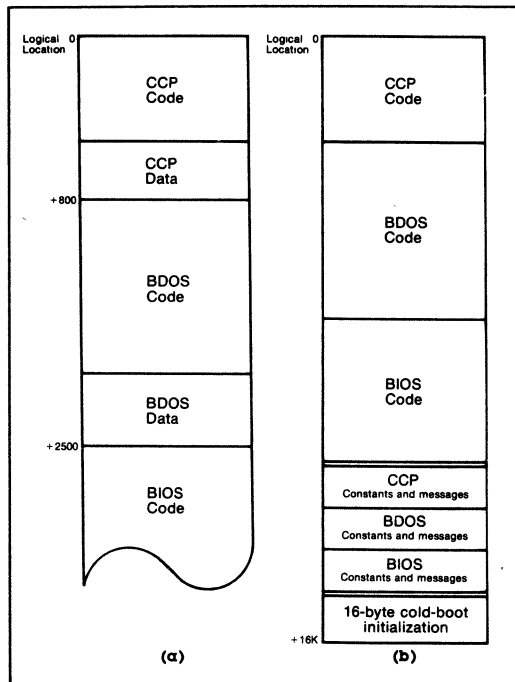
The CP/M-86 operating system consists of three modules. The Console Command Processor (CCP) handles command line processing, and executes built-in utilities. The Basic Disk Operating System (BDOS) performs logical disk I/O, including disk reading and writing, directory management, and sector allocation. The Basic Input/Output System (BIOS), which contains the configuration-dependent code and data, also provides I/O for specific peripheral chips.

CP/M-86 is a single-user, single-tasking operating system written in position-dependent code. The 80150 contains the entire CP/M-86 operating system; for many configurations, it requires no off-chip code. Intel's goal was to use the configuration-independent CCP and BDOS elements as a base, and add to them a BIOS that supported a variety of peripheral components but was still configuration independent.

The 80150 BIOS supports the following two functional configuration options:

1. A *preconfigured-mode system*, for which the system designer needs to do no operating-system code development or extension.
2. A *configurable-mode system*, for which the designer makes a selection from among the Intel drivers supplied, and makes changes as required to meet hardware needs.

The 80150 BIOS includes drivers for the following chips:



**FIGURE 5.** (a) The standard disk-based CP/M-86 module is one long structure containing both code and data. (b) Intel reorganized the basic CP/M-86 architecture to fit the operating system into the 80150 OS firmware component.

- 8251A Universal Asynchronous Receiver/Transmitter (UART)
- 8274 Multi-Protocol Serial Controller (MPSC)
- 8255A Programmable Parallel Interface (PPI)
- 8275 Floppy-Disk Controller
- 8237 Direct Memory Access (DMA) Controller

If the 80150 is used as a co-processor with the iAPX 186 or the 188, then the on-chip peripherals of these processors (DMA, timers, interrupt controller, chip-select logic) are also used.

Configuration independence is achieved via the Configuration Block (CB), with which whole BIOS drivers, data structures, and built-in utilities can be selected independently by the system integrator.

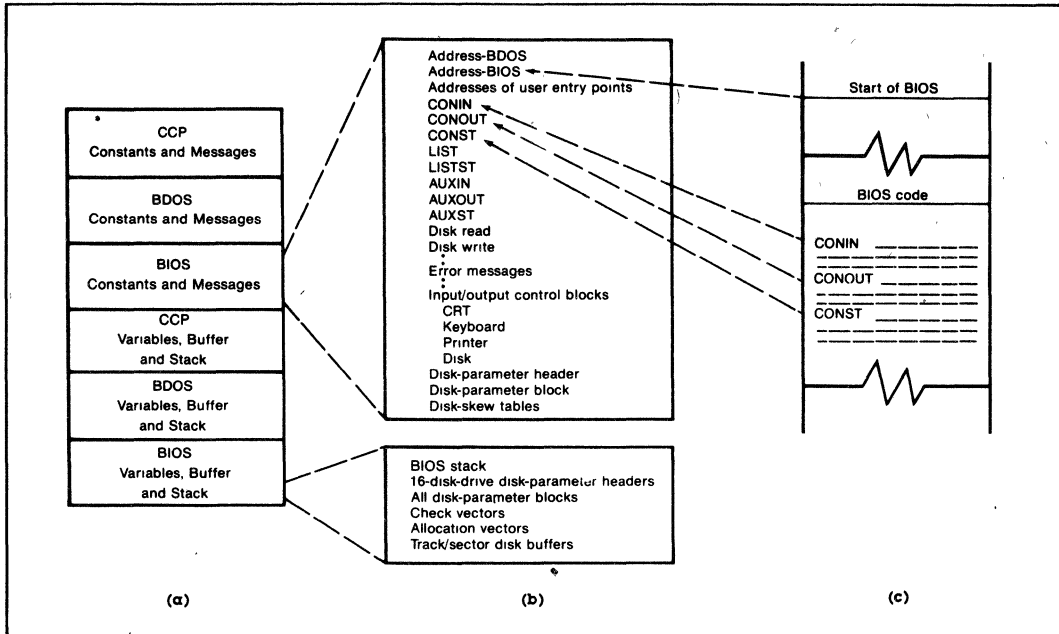
**CP/M-86 Transformations**

Intel and Digital Research together addressed the issues of position dependence and intermixed code, data, buffers, and stacks. The CCP and BDOS were reorganized to consolidate code and to use the 80150's ROM space efficiently.

CP/M-86 was originally developed using an 8080 model structure. The use of this structure implied that the code and data groups would overlap, as they do in the classical 8080-based CP/M design. Each module contained set-aside buffer areas, and included separate data stacks. Therefore, all variable areas

\*CP/M-86 is a trademark of Digital Research, Inc.





**FIGURE 6. The Configuration Block (CB) reconfigures the 80150 for specific hardware systems. a) The CB constants read down from the 80150, and variables used at run-time. b) The BIOS portion of the CB contains configuration-dependent data. c) These addresses provide access to the 80150 on-chip code, to alter execution paths for different configurations and steppings.**

and stack areas had to be removed from code that would reside in ROM.

Figure 5(a) shows the general structure of the original CCP and BDOS. Although a natural separation between code and data is clear, Digital Research did not distinguish between constants, literal messages, and pure scratch storage.

Intel's first step in the transformation of CP/M-86 was to group all variables within each module, including buffers and stacks. We then placed this data grouping at the end of the constants and literal messages for each of the CCP and BDOS modules.

The new structure (Figure 5(b)) includes all code, constants, and internal messages, as well as a 16-byte initial-program-load (IPL) boot resident in the 16K-byte OSF ROM. We removed all variables from the body of CP/M-86, and put them in an external RAM-based structure.

Second, the implementation of CP/M via the Intel 8086 "small model" (separate code and data segments) rather than via the 8080 model (intermixed code and data), meant that the necessary additional variable data space would be available at 80150 execution time. The segmented architecture of the iAPX 86 family made this implementation easy, because separate CPU registers were available for data and code addresses. As part of the BIOS initialization, we moved the constant data structures for the CCP, BDOS, and BIOS to the base of a RAM-resident Configuration Block (CB). An additional amount of RAM equivalent to the total variable space was also allocated and preset to zero. This 8086 "small-model" transformation not only made it easy to separate code and data, but also

made the code more efficient and eliminated approximately 2100 bytes.

We achieved configuration and stepping independence via the off-chip RAM-based Configuration Block. Figure 6(a) shows the overall structure of the CB as constructed during BIOS initialization. During initialization, the 80150 BIOS copies the CCP, BDOS, and BIOS constant and literal structures into the Configuration Block, and appends additional space for variable and scratch-pad storage. Even the location of the CB is alterable, based on the address stored in locations 0:3FE-3FF.

Figure 6(b) shows expanded portions of the CB. The data area contains pointers that can be changed to select custom off-chip code instead of the standard on-chip code. The entire BIOS can be replaced. (The BIOS code insert in Figure 6(c) and the various code labels are reflected back to the CB.) Complete I/O control block structures are provided for each CP/M logical device, including CRT, keyboard, list, auxiliary, and disk. The control block includes port addresses, protocol support, and other default data needed to detect and control the status of each peripheral. Figure 6(b) also expands the systems tables and buffers created for disk support.

The addresses in Figure 6(b) indicate how stepping independence is achieved. Any off-chip routines changed by the user can be selected by altering the address of the CB. If Intel updates an on-chip routine, the address in the CB is updated automatically when the 80150 copies its constant structures into the CB. As explained above, full stepping independence is maintained, because any ROM changes can also be imple-

mented off-chip by having the address in the CB point to an off-chip patch. (The CB contains BDOS entry points (shown in Figure 6(b)) that make this change possible.)

#### The Configuration-Independent Interface

Use of the predefined configuration requires that the 80150 be installed at the top of the 8086 memory address space (FC00:0). The 16-byte internal hardware boot is activated at all POWER ON and hardware resets, and passes control to the 80150. The 80150 initialization sequence uses this positioning to indicate the default hardware configuration (floppy disk, printer port, serial console, or auxiliary port). Each device has predefined port addresses, interrupt assignments, and protocols. The iAPX 186 or 188 CPU supports programmable chip-selection and the on-chip DMA drives the floppy disk controller.

If the configuration must be altered, or if the BIOS code needs revision, the 80150 can be installed on any 16K code boundary except at the very top or bottom of memory. A PROM that contains off-chip code and data for a user's particular configuration is also installed at the top of memory.

The 80150 initializes the default system hardware tables, then calls an EPROM to complete or revise the existing data in the off-chip CB RAM area. At this point, the CB contains the addresses that select either on-chip or off-chip code. When the configuration is complete, control is returned to the 80150. The 80150 completes the CP/M initialization, displaying the familiar CP/M "A" sign-on.

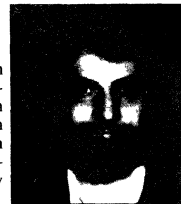
#### Conclusion

Converting software to silicon is not new. But redesigning software to consist of on-chip ROM code and configurable

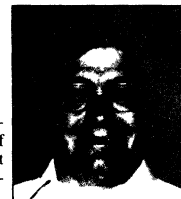
RAM data is somewhat more innovative. One silicon-related specter that haunts software designers is the fear of "committing code before its time." But software designers can *never* expect to produce bug-free code the first time. And system designers cannot always predict the capabilities or the implementation requirements of peripheral devices that have yet to be built. Nevertheless, software designers who use the general silicon-implementation strategies of position independence and configuration independence, and who provide for stepping independence, can create standard silicon hardware without fear of component obsolescence. □

#### About the Authors

**Ron Slamp** received the A.S. degree in software technology from Portland Community College, and gained much of his skill in electronics at Clark Community College in Vancouver, Washington. He has worked in Intel's OEM Module Operation in Hawthorne, Oregon since 1978 and is currently the project leader for component software.



**Jim Person** received the B.S. degree in mathematics in 1962 from the University of Arizona. He was the engineering project manager at Intel for the 80150 "CP/M-on-a-chip."



June 1983

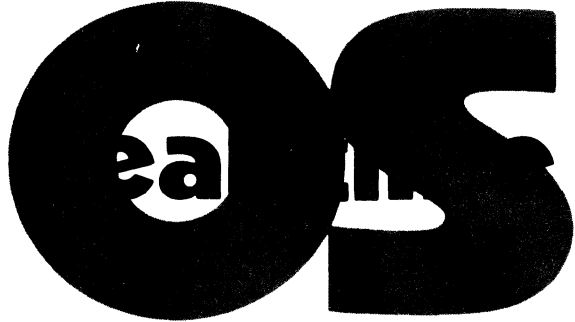
# Putting Real-Time Operating Systems to Work

Stephen Evanczuk  
Electronics Magazine

## SPECIAL REPORT

Punching in for real-time jobs in industry, R&D, and offices, operating systems use special software structures to squeeze better-than-ever performance out of 16-bit microprocessors

by Stephen Evanczuk, *Software Editor*



□ A special class of operating systems is hard at work in the 16-bit microsystem world. For controlling environmental processes, acquiring data at high speed, or even handling transactions at a commercial bank, these operating systems contain mechanisms that enable them to respond rapidly to external events and that differentiate them from the more familiar general-purpose operating systems.

In fact, all the operating systems for 16-bit microprocessors respond in a reasonable period of time. But the general-purpose, or developmental, operating systems like CP/M, Bell Laboratories' Unix, and MS-DOS are intended for standard programming activities like editing, compiling, and file management [*Electronics*, March 24, 1982, p. 113]. As such, they lack certain software structures needed for reliable control of processes producing data at a high speed.

Real-time operating systems tend to fall into two general categories—multipurpose and embedded, reflecting the type of hardware they run on. Multipurpose real-time systems are typically built around full-fledged microcomputer systems with terminal, keyboard, plenty of system memory, and mass storage. Furthermore, in process-control or data-acquisition applications, some special-purpose hardware is usually included in these systems to serve equipment or high-speed data input operations. Besides the familiar applications for research and development, transaction-processing environments are an example of situations needing multipurpose real-time systems.

No doubt the largest class in volume because of their growing use in consumer items, embedded systems are minimal hardware systems, often just one-chip microprocessors that control limited parts of a larger system. Programmers ordinarily employ a special development system to create the software, which is loaded into the target system for use and ideally is never seen again.

To meet the needs of these two classes of applications, real-time operating systems come in three flavors for 16-bit microprocessors. Serving multipurpose real-time systems, one type—discussed in the

first part of this report (see p. 106)—includes all the software development support found in their general-purpose counterparts. Furthermore, many can be stripped of the layers needed in the developmental environment and placed in programmable read-only memory for use in an embedded system.

For those who swear by Unix, the group of Unix-based operating systems discussed in the second part (see p. 111) may mean no need to swear at it in real-time applications. A growing number of vendors are starting to convert this admittedly non-real-time operating system into versions that can be used to handle external processes. Although the industry is cautious, if not downright skeptical, of real-time versions of Unix, the fact that C—the language of Unix—is so highly regarded for use in real-time applications may help swing this group into the forefront.

The potential for distributed-control systems based on embedded microprocessors hinges largely on the availability of high-performance real-time operating systems that can be plugged into the application with the same ease as an integrated circuit. Called silicon software, these operating systems discussed in the last part (see p. 114) have been designed to be stored in read-only memory. Providing a fixed set of system calls, they present programmers with a consistent set of high-level commands to perform the low-level functions usually built from scratch.

Building system-level software from scratch has long been the hallmark of real-time programmers, even a mark of honor. Fortunately, however, the increased acceptance of ready-made operating systems using well-understood algorithms (described in the first part) is helping to replace this software “random logic” with rather more standardized packages.

On still another level, the unique responsiveness and throughput demonstrated by real-time operating systems is a truly user-friendly feature. For this reason, these systems should find their way into less obvious real-time applications, such as transaction processing, word processing, and personal work stations for office automation.



# Algorithms star in multipurpose systems

□ Whatever environment it finds itself in, the function of an operating system is the efficient management of shared resources by a number of users, whether these are human beings accessing a computer through terminals or programs vying for a single central processing unit. In fact, the degree of sophistication of an operating system is reflected by the number and types of physical resources it manages and by the fineness of control it exercises in their management. And operating systems targeted for control of the external environment must wrestle with the most demanding resource of all—time. The degree of care with which such software is designed to manage time is what determines its suitability for the real-time environment.

## Schedulers and queues

Two critical aspects of the real-time environment are the random nature of physical events and the simultaneous occurrence of physical processes. Consequently, interrupt handling and multitasking are primary attributes of a real-time operating system. In fact, it might be

argued that the mechanism for handling multitasking—the scheduler—is the heart of the operating system. The rest of the operating system lies atop this kernel and serves the specific demands of the application environment.

In particular, the lists, or queues, and their managers that surround the scheduler are constructed to deal with the different physical resources supported by the operating system. Thus, one queue may contain those tasks (processes, or programs in the course of being run) that are ready to execute on the processor, another queue may be tasks waiting for access to input/output hardware, and another queue may contain tasks waiting for some specified event to occur.

In any multitasking operating system, the scheduler uses the queues as input. Its output, on the other hand, is a single task that has been activated and allowed to execute on the central processing unit. The scheduling algorithm in large part defines the operating system.

In one system, the scheduler may simply select a task on a first-come, first-served basis, allowing it to run until completion or until some specified period of time has elapsed. This type of relatively primitive algorithm was commonly used in mainframe computers running simple batch-oriented operating systems.

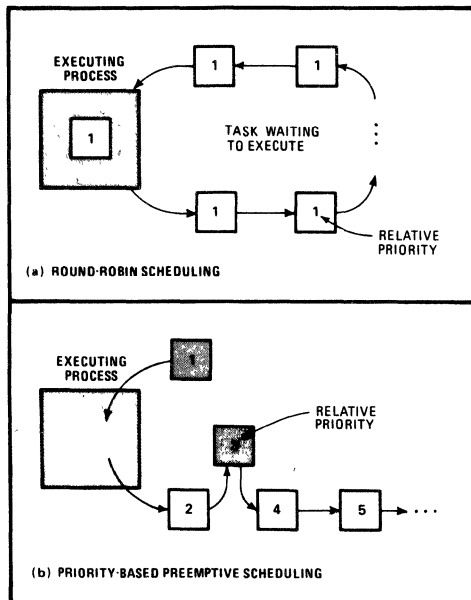
In a slightly more sophisticated operating system that can be used interactively through terminals, the scheduler may select tasks on a round-robin basis and permit each of them to run for a specified period of time (Fig. 1). Once the task exceeds its time slice, it is placed at the end of the queue and forced to wait until all other tasks have had a chance to execute.

Round-robin scheduling with equal time slices is adequate if every task is no more important than any other task. However, if some are considered to possess a higher priority, then a more sophisticated scheduling algorithm must be used—one that recognizes that some tasks are more important, but that no task should be excluded from using the CPU.

One solution is the use of several queues, where the length of the time slice is related to the priority of elements in the queue. In this case, the scheduler would allow all tasks in each queue of a different priority to execute on the CPU, but lower-priority tasks would be given less time.

A further refinement permits higher-priority tasks to suspend a running task. This technique, called preemptive scheduling, is an important feature for real-time environments, in which the delayed execution of a high-priority task could have disastrous results, rather than simply disappointing the user.

In scheduling algorithms, tasks may exist in a number of logical states, depending on their readiness to run. In the Versatile Real-Time Executive (VRTX) from Hunter



1. **Priorities.** In round-robin scheduling (a), tasks (or processes) take equal turns executing, while a higher-priority task will supersede a lower-priority one in priority-based preemptive scheduling (b). Most schedulers employ some combination of these techniques.

& Ready Inc., Palo Alto, Calif., for example, tasks are driven through four possible states by external events, by other tasks and system utilities, or by their own system calls (Fig. 2). For example, an executing task may delete itself—in which case it enters a dormant state—or may cause itself to be blocked either explicitly through a call to suspend itself or implicitly through a call to perform some I/O function. On the other hand, once suspended, a task may reschedule itself through a system call, or an external real-time event may bring the task back into the ready queue.

Recognizing the importance of scheduler design, at least one software vendor has made it easier for real-time users to build systems around a prepared kernel. United States Software of Portland, Ore., is offering a basic scheduler that assembles into less than 100 bytes of object code for the target microprocessor [*Electronics*, Nov. 17, 1982, p. 206]. Furthermore, in anticipation of real-time systems targeted for specific application areas, U. S. Software supplies a list of design notes detailing extensions to the basic kernel.

#### Another use for queues

In addition to having queues serving the scheduler directly, most systems use them as the preferred means of associating a task with a required resource. For example, one capability commonly found in real-time operating systems is the ability to suspend a task for a specified period of time. Typically, the operating system contains a special queue for this function. Each element in the queue is a task in a suspended state. Associated with each task is a counter that contains the number of clock ticks remaining until it should be reactivated.

For example, in iRMX-86 from Intel Corp., Santa Clara, Calif., the counters keep track of the incremental time remaining with respect to the previous element in the queue, rather than the total time remaining before

the task may be reactivated. Thus at each clock tick only the counter in the element at the head of the queue need be decremented, rather than every counter in every queue element. This method takes longer to insert new elements into the queue and so requires slightly higher overhead for insertion than when the total time is maintained by each counter; however, that overhead is more than offset by the time saved by updating only a single counter.

Real-time environments pose a special set of problems for resource allocation. Besides all the more familiar problems of scheduling, a real-time operating system must maintain reliable behavior under extremes of load when it is driven by a high rate of external stimuli. From the system user's point of view, the system must maintain a predictable level of response and throughput.

In an interactive environment, users sitting at terminals measure response as the time the system needs to react to a keystroke. In general, system response is the time that the system needs to detect and collect data from some external stimulus. Throughput, in an interactive environment, is seen as the number of users able to utilize the installation simultaneously. In a more general real-time environment, throughput is the rate at which the system is able to collect, process, and store data.

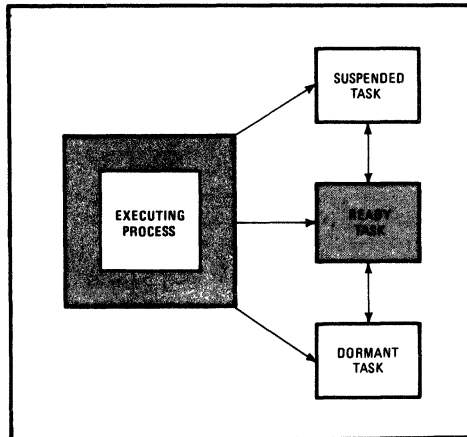
In fact, although response and throughput share some common software elements, operating-system designers will invariably find themselves forced to make choices that will tend to optimize one at the expense of the other. Often, the interrupt-handling requirements of a real-time operating system force this choice.

Interrupt processing is hardware and software integration at its most demanding (see "Handling hardware interrupts," p. 108). To handle interrupts, operating systems often place layers of software between the user and the microprocessor in order to allow different levels of performance and capability.

Intel's RMX-86 is a typical example of distinct levels of software used to perform basic interrupt processing. At the lowest level, an interrupt handler works intimately with the hardware to execute some operation, such as sending a message character by character to a printer. Code for interrupt handlers is kept compact and simple, since system interrupts are disabled during their operation. The higher level, called the interrupt task, works at a priority associated with the particular hardware it services. Interrupt tasks act as interfaces between application tasks, working with specific interrupt handlers to complete execution of operations dealing with external devices. RMX makes this interrupt-handling mechanism available to application programs through a special set of system calls.

#### Protection and communication

Once the interrupt software has completed its function, tasks that use the data are indistinguishable from any other task in the system as far as the operating system is concerned. Unless special care is taken, conflicts could still arise between two separate tasks that might need to use the same resource, such as the same location in memory. MP/M-86, for example, employs a special queue, called a mutual exclusion queue, that contains a unique message representing the shared resource. In or-



**2. Task states.** As one task (or process) runs, others may be in various states of readiness. In Hunter & Ready's VRTX, for example, tasks can be ready (able to run immediately), suspended (waiting for a resource), or dormant (deleted by a system call).



der to use the resource, a task must first capture the message, much as a node in a token-passing network must first obtain the token before being at liberty to transmit.

Per Brinch Hansen<sup>1</sup> identified such shared resources as key elements in multitasking systems. Sections of code that access critical resources are called critical regions. The simple expedient of ensuring that only one task at a time is allowed in a critical region guarantees that multiple tasks may share the same critical resource without fear that its integrity may be compromised when two of them attempt to access it simultaneously (Fig. 3).

This concept of the mutual exclusion of tasks from critical regions is implemented in a structure called a monitor, in which critical regions are gathered in one section of code and protected from use by more than one task at a time. The MSP operating system from Hemenway Corp. of Boston [*Electronics*, Jan. 27, 1983, p. 119] explicitly supports mutual exclusion through monitors in its internal structure.

Furthermore, user-written routines needing monitor protection are provided with four functions in MSP that are implemented using hardware traps for rapid access: Entermon, Exitmon, Wait, and Signal. Entermon and Exitmon serve as monitor entry and exit points, respectively, performing required housekeeping functions. Entermon disables system interrupts and preserves all registers, while Exitmon reverses these actions. Wait and Signal, on the other hand, work in tandem to control access to a critical resource. Wait queues up tasks needing an unavailable resource. Signal releases them from the queue when the resource becomes available.

Wait and Signal are examples of an intertask communication mechanism, called semaphores, found in most real-time operating systems. As noted, these commands simply queue up and release tasks needing a critical resource. Such a resource may be an I/O device, a memory location, or simply a go-ahead signal that synchronizes a pair of tasks. For example, task A may execute only after task B has completed. In this case, task A would begin with a Wait (flag) command, where the flag is used as an associated variable. Task B, on the other hand, would end with a Signal (flag) command. In this way, task A would be blocked until task B had executed its Signal command at the end of its processing. But exchanging simple go-no-go signals is not sufficient for many multitasking environments.

For longer messages, real-time operating systems offer extensive intertask communication facilities called mailboxes. Mailboxes are essentially semaphores with storage. As such, tasks needing data from another task will wait until the other has loaded the mailbox with the information. Intel's object-oriented RMX-86 transfers any of the defined objects in the system through mailboxes. Hemenway's MSP, on the other hand, provides a buffer of fixed size that may be used without restriction on its contents, as long as the 256-byte buffer is not exceeded. With its Multibus message exchange (IMMX) extension to RMX for

## Handling hardware interrupts

Underlying the special software of a real-time system is the assumption that the hardware itself can respond in a coordinated fashion to external events, or interrupts. In fact, microprocessors contain subsystems whose sole function is to deal with interrupts in a way that eases integration of the interrupt-handling software.

All modern computers integrate interrupt-handling hardware and software at a very low level of design. When a user accesses a microprocessor through a terminal, the same hardware interrupt facilities come into play as when, for example, an analog-to-digital converter sends data to the same type of microprocessor. The software response, on the other hand, depends on the type of operating system, but both real-time and general-purpose operating systems must take some action, like read in the data value or the character.

Examining the details of a simple keyboard task illustrates the complex nature of real-time processing. It also serves as a vehicle for introducing some of the basic vocabulary in this field.

A standard software subsystem in a microcomputer system, called the keyboard monitor, is responsible for working with the hardware interrupt system to detect a character, collect it, and effect some action based on the input character. When a key is struck on a terminal, the corresponding byte is converted into a serial stream of bits that are passed from the terminal to a universal asynchronous receiver-transmitter. Once it receives the full character, the UART generates a hardware signal, or interrupt, that notifies the processor. Since interrupt management is a common activity, processors contain special hardware to respond to this signal.

Although the details may vary from one particular microprocessor to the next, the result is the same for all. When its interrupt-request line is asserted, the processor ceases its current processing and places values from its internal registers into system memory. Typically, the processor status and instruction-address registers are saved in the system stack, a last-in, first-out buffer located in some portion of system memory. As the figure shows, the processor responds to the original interrupt-request signal by issuing a signal of its own, called an interrupt acknowledge.

The peripheral hardware that originated the interrupt detects the interrupt-acknowledge signal on the system bus and responds by returning the memory addresses of both the interrupt-handling subroutine and the new processor status. Typically, the new processor status will provide for disabling any further interrupts. This latter action is a simple precaution, preventing a single external stimulus from causing a continuous series of interrupts that will eventually result in an overflow of the system stack.

Such an interrupt mechanism, called a vectored interrupt, allows the speediest identification and reaction to an interrupt. (An alternative interrupt mechanism used by earlier processors, called a device-polling interrupt, simply forced the processor to switch to a defined address in memory containing software that polled each peripheral device until the device that generated the interrupt was discovered.) At

this point in the interrupt-handling task, all the activity was exclusively in hardware, but nevertheless resulted in extensive processor activity and bus traffic due to multiple accesses of system memory and the involved peripheral-device controller.

Consequently, it is not surprising that the time for hardware to set the processor to handle the interrupt—the hardware-interrupt latency—should be several processor cycle times in length. In general, hardware-interrupt latency is not a fixed number, but will lie within some range, since the processor will need a variable length of time to complete its current instruction and to initiate the interrupt-acknowledge signal. For example, if a processor is involved in a lengthy floating-point operation, several microseconds could elapse before the interrupt is acknowledged.

Once the processor has reached the interrupt-handling subroutine, the contents of only a minimal set of its internal registers have been preserved. However, before the real work of the subroutine may commence, the contents of other registers and variables shared by independent sections of the operating system must be preserved. The time needed to perform this action is called the context-switching time. Only after the software context is switched is the system ready to begin handling the special requirements of the device that originated the interrupt. The period of time between the occurrence of the external event and this state is the total interrupt-response latency.

In real-time operating systems, interrupt-response latency is usually a specified value—around 100 microseconds in very high-performance systems based on 16-bit microprocessors. Designers often bypass the constraints imposed by response latency by including special-purpose hardware to boost system response to external events.

Throughout all this time, system interrupts are still disabled. However, now that the context switch has taken place, the keyboard handler is free to transfer the character from the UART. Deciding where to put the character is important in terms of system throughput and overall efficiency. When it is put in some specified location in system memory, system interrupts must remain disabled; otherwise, if the handler attempted to service a subsequent interrupt, the new character would overwrite the character already in the location, but not yet fully processed.

In general, there are two methods for handling this problem. In the first method, the character is simply placed on the system stack and referenced through the relevant pointer. In an alternative method, the character is placed in a block of memory that has been reserved just for the handler and is called a context block. In this case, the character is referred to by using a specified offset from the base of the context block. Each time the keyboard handler is called in response to an interrupt, one of

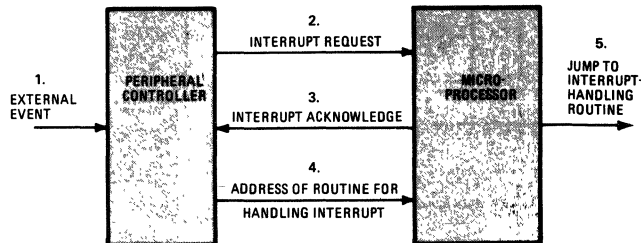
these context blocks is reserved from available system memory. Setting up a context block and switching the processor to it in a context switch accounts for a significant fraction of the time that is needed to respond to an interrupt.

Software code, such as the UART handler in this example, that does not contain any memory locations for variables is called reentrant because the processor may asynchronously enter it, be called away by an interrupt (even one that results in another call to the same piece of code), and return without loss of data or context. If the code is not already resident in system memory, another routine causes a copy of the code to be read from storage into memory. With reentrant code, only a single copy of the program or task need be resident at any time. Each context block, or logical copy of the task, is called an instance of the task.

Multiple instances of a task help explain some of the confusion associated with performance figures reported as a result of benchmarks. In examining benchmark figures, it should be clear just what the values are that are being reported. Total interrupt latency generally includes hardware interrupt latency, the time to create an instance of a task (plus the time to call in the task into memory if not already resident), the context-switch time, and an additional period needed to execute a variable amount of code that causes the data to be read from the peripheral registers. Creating a new task means either calling in a new task and creating a context block for an instance of it or just creating a new instance of a task already existing in memory.

Once the handler in the UART example reads the character from the receiver buffer, it will reenable interrupts. The time between entry to the interrupt routine, when interrupts were disabled, until the time when interrupts are reenabled is an important factor in determining the effective latency of system response.

This dead time must be minimized, or the system will remain deaf to external stimuli for unacceptably long periods of time. In fact, the length of time that system interrupts are disabled is one of the criteria for determining the usefulness of an operating system for real-time applications. The longest period during which interrupts are disabled is a good measure of the responsiveness of the system. Because of the weight of disabled interrupts on total system performance, modern microprocessors use a number of hardware-related levels of priority and enable interrupts at or below the priority level of the device originating the interrupt.







multiprocessor-based systems, Intel replaces the concept of a mailbox with that of a software port connecting different tasks, whether they exist on the same or different physical processor.

Unlike memory-intensive software development systems, real-time environments find less need to support a virtual address space. In fact, the increased system overhead is less than desirable, because the designer seeks to minimize response latency. A useful feature, however, that can be found in some real-time operating systems is a set of system calls responsible for dynamically allocating and deallocating memory.

For example, in the ZRTS system from Zilog Corp., which comes in different versions for the Cupertino, Calif., firm's segmented Z8001 and nonsegmented Z8002, a set of three system calls provides for dynamic allocation and deallocation, as well as information on the status of memory allocation. The system call for memory allocation allows application programs to specify the attributes of the memory block to be allocated and returns a name referring to the created structure.

Besides similar system calls, Intel's RMX adds some calls suited to its context-based architecture. In RMX, each task lies within the context of a job environment that bounds the scope of tasks within it (Fig. 4). As such, each task is allowed to draw from the memory pool of its job. In case more memory is required than that initially allocated to the job, a pair of system calls provides for querying the system on the size of the job memory pool and for dynamically changing it.

Dynamic memory allocation and deallocation is a relatively advanced concept that exacts some overhead during runtime. However, the alternative—static allocation before runtime based on expected requirements—may be less suitable for applications in which the real-time environment is relatively unpredictable.

In real-time operating systems, disk-file management is treated as just another asynchronous task possessing a particular set of critical resources—mass-storage devices. In real-time environments, file-management utilities have

to meet not only the requirements of general-purpose systems but some additional demands.

In terms of system response, a requirement of real-time operating systems in heavily loaded systems is the ability to conduct asynchronous I/O operations. In such an operation, the calling task simply queues up the I/O request, then immediately returns as if the task were completed in zero time. When the I/O request is fulfilled, the operating system switches the processor to a separate routine whose address is supplied when the original asynchronous request was made. This completion routine then may continue any processing that may be required following the I/O request.

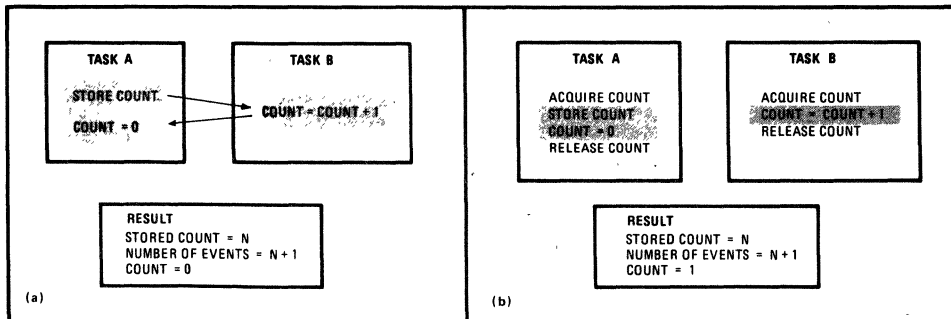
System throughput depends heavily on the efficiency and performance of the I/O subsystem. Peripheral controllers with direct memory access and the ability to move the disk's read-write head without necessarily performing data transfer can significantly reduce the overhead associated with data movement.

**Reducing overhead**

System software can also contribute to reduced overhead by providing a simple disk organization when high throughput is needed. One of the simplest structures is a file consisting of an unbroken access of disk sectors, such as the contiguous file in Hemenway's MSP or the physical file in Intel's RMX. By ensuring that the next block of data will be written to the next physical sector on a disk, the operating system can reduce the delay caused by head movement on the disk.

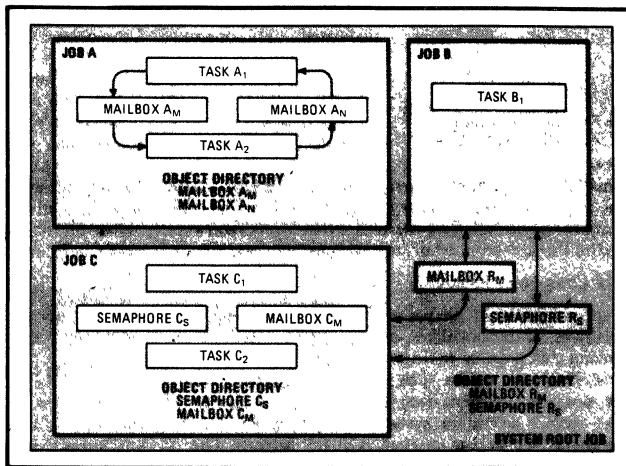
In their use of an I/O interface that is common to all system device drivers, MSP and RMX attack another important aspect of system design, though one not necessarily tied to their utility in real-time applications. In MSP, a basic I/O routine called Iohldr serves for all operations by accessing a special block of information in memory. RMX, on the other hand, uses a number of device-independent system calls to handle communication with peripheral devices.

Next to multiprocessor-based software systems, real-time software systems are the most difficult to debug. Again, the cause is the distinguishing feature of real-time operating systems—precise management of time. Standard debugging tools for single-user general-purpose op-



**3. Critical regions.** If two asynchronous tasks use a counter, events can be miscounted if task B interrupts task A before the counter is reset (a). Forcing the tasks to acquire a counter before using it (b) ensures synchronization through the critical regions (tinted).

**4. Job context.** In Intel's RMX, all jobs exist within the context of another job. A directory defines the objects that are known to other objects in the same context. For example, all three jobs may use mailbox  $R_M$  since it is in the system's root-job object directory



erating systems generally disable all system interrupts in various phases of the debugging routines. Since the object of a real-time software system is asynchronous involvement with the task under control, this effect makes standard debugging tools useless.

Ideally, debugging real-time software would use performance-analysis tools and troubleshooting aids built into the operating system itself. Unfortunately, the processing overhead and additional memory requirements imposed by such a technique make this an unpopular notion in the design of an operating system. However, some systems do provide some means for run-time error handling. The exception handlers in RMX, for example, are procedures that are associated with each task when it is created. If a task attempts to use a system call but encounters an error, called an exception, the operating system invokes the associated exception handler to allow some graceful recovery from the error.

Although the technique in VRTX is not true exception handling, Hunter & Ready's silicon-software system does include a mechanism to build run-time debugging software. A special location in the VRTX configuration table (see p. 115) causes a user-defined routine to be called whenever a context switch is performed. By recording information about the task as well as the processor, such

a routine can be used to create a list, called a trace, of the history of task execution.

Because real-time systems often include special-purpose hardware, the accepted technique for debugging user-written routines uses the classical approach of collecting data before and after passing through a suspect region, along with a logic analyzer to monitor timing of traffic through critical regions.

Intel offers some relief to this problem through the iRMX debugger. In particular, the debugger allows the user to work with individual tasks without interfering in the operation of other tasks, as well as to monitor the activity of the system as a whole without disturbing it. The debugger recognizes data structures in the RMX kernel, so the user may examine system objects. In addition, Intel's crash analyzer brings mainframe debugging power to microprocessor-based applications using RMX.

Zilog's ZRTS configuration language offers another level of support to the development of systems targeted to specific hardware complements. By defining the details of the hardware, a system designer can configure ZRTS to particular systems.

#### Reference

1. Per Brinch Hansen, "Operating System Principles," Prentice-Hall, Englewood Cliffs, N. J., 1973, p. 84



## Designers tune Unix for real-time use

□ With an eye on the growing momentum of Bell Laboratories' Unix, real-time system designers have endeavored to squeeze this complex operating system into the rigid confines imposed by the demands of real-time environments. Although Unix brought advanced system ca-

pability to mini- and microcomputers, the original intent was to provide a hospitable software-development environment, rather than to include the features considered necessary for real-time uses.

Until now, data-acquisition systems employing unmod-

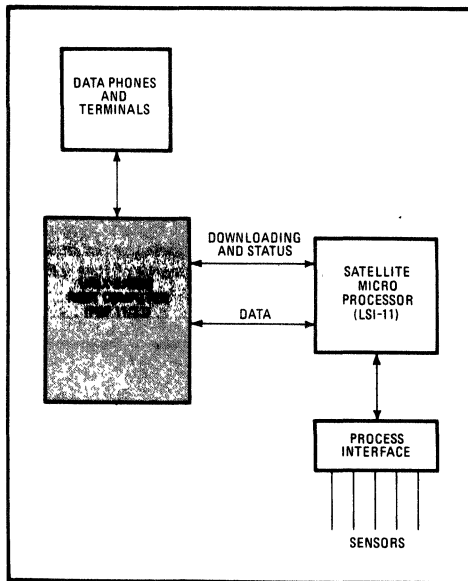


ified Unix typically used dedicated microprocessors to buffer a central computer from constant random activity caused by external events. For example, in the Concepts process-control system from Bell Laboratories, Murray Hill, N. J., a Unix-based host is linked with auxiliary microprocessors. In each microprocessor, software derived from Unix software handles the low-level details of real-time activity (Fig. 1).

### Unix goes real-time

Appearing in all shapes and sizes, Unix-compatible executives, Unix lookalikes, and new Unix versions are bringing this popular environment into real-time applications. However, unlike their colleagues creating totally new operating systems (see pp. 106-111), designers of these second-generation systems are constrained by the boundaries set by the original. Caught between Unix's complex organization and the high-speed needs of some real-time applications, they have opted for preserving the basic architecture. Still, for intensive data-acquisition applications, vendors like VenturCom, Cambridge, Mass., and Masscomp, Littleton, Mass., add on dedicated hardware like high-speed peripheral controllers to link devices into the main system without losing the generality of the Unix software architecture.

For microprocessor-based dedicated systems, memory-



1. **Satellite processing.** In Bell Labs' Concepts system, separate microprocessors handle low-level details of process control. Yet another processor—a host computer that runs the Unix operating system—is in charge of coordinating these satellite machines.

resident kernels like the C Executive bring a measure of Unix compatibility to even dedicated systems. Offered by JMI Software Consultants of Roslyn, Pa., the C Executive combines support of an extensive C-language run-time library with many of the features considered important in real-time applications. Although not directly supporting shared data in its multitasking architecture, the executive's intertask-communication facilities include data exchange through a queuing mechanism. As befits a real-time executive, the task-scheduling algorithm allows higher-priority tasks to preempt lower-priority ones. Because it is intended primarily for embedded systems—that is, dedicated microsystems that do not have disks—the C Executive is totally contained in system memory and does not support the extensive Unix file-management subsystem.

### Controlling real-time tasks

Full-blown Unix lookalikes, on the other hand, find themselves forced to deal with some of the very internal structures that aided Unix's rise in popularity. For applications like program development where regular scheduling is more important than instant response, scheduling is aided by Unix's manipulation of the priority levels of tasks (or processes, in Unix's preferred terminology). For real-time applications, however, the slight uncertainties this feature introduces could destroy the synchrony of timed events controlled by the system.

Consequently, one enhancement commonly found in the real-time offshoots is the addition of some mechanism to ensure more precise control of real-time tasks. A technique that sits well within Unix's task-oriented (that is, process-oriented) design is the definition of a real-time class of tasks (or processes). This class earns special rights in the operating system, such as a guarantee that each task will not be swapped out of memory, but remain locked in and ready to respond more rapidly to events.

VenturCom's Venix, for example, defines a real-time priority level. The scheduler allows tasks running at this level to maintain control of the processor for as long as necessary. In contrast, Regulus from Alcyon Corp. of San Diego, Calif., speeds response to real-time events through the use of 32 user-defined priority signals.

### Better I/O handling

In addition to its scheduling algorithm, Unix's method of handling input/output operations needs improvement to perform well in real-time applications. Aiding total system response, the asynchronous I/O procedure in Venix supplements the conventional synchronous procedure in Unix, in which the requesting task must be suspended until the I/O operation is completed (Fig. 2). By placing asynchronous requests at the head of the I/O request queue, Venix's manager lets real-time tasks issue a write request, for example, and immediately continue processing, assured that the request will be honored next.

Concentrating instead on improving what happens when I/O requests have been completed, Masscomp's enhanced version of Bell Labs' Unix System III adds a modified signal called an asynchronous signal trap. Similar to the concept of completion routines in other operat-

**Going Forth with alternatives**

Programmers evoke the feelings of dread experienced by a programmer who must alter code that has been developed by others. In this case, the programmer knows that the code is all right, but it is not what he wants. Fortunately, there are alternatives on the horizon that are appearing one of the commercially available real-time operating systems. As programmers now are dealing with a set of alternatives, they are being urged to look for the high ground, and that is being achieved by using someone else's system. At the developer trying to eliminate an error, the system is being used by the operating system, which is being used.

For real-time, or high-performance, process-control applications, the use of a finite-state machine as the controller is an easily implemented technique. A finite-state machine is a logic device that produces a defined output state based on its input state. For example, a microprocessor may read some input register, access a table in memory, using the input as the address, and send out the value contained in the accessed location. In such a system, a value could be created with a single indirect move instruction in a microprocessor using a memory-mapped input/output scheme. Clearly, using a microprocessor this way would allow only a relatively small number of states.

Besides the hardware approach, the software alternatives include the interpreters for high-level languages, such as Forth and concurrent versions of Pascal, that are appearing in the read-only memory of single-chip 8-bit microcomputers. For example, the GDP 1804F complementary

MOS single-chip microcomputer from RCA Corp.'s Solid State Division, Sunnyvale, N. J. [Electronics, Nov. 29, 1982, p. 127] contains a core interpreter for some Concurrent Pascal (mCP) from Evans & Int. of Laramie, Pa. Based on Per Erieh Hansen's Concurrent Pascal, mCP contains all the constructs necessary for real-time applications, such as shared data, monitors, interrupt handling, and task queuing and scheduling. RCA also provides a ROM that enables the core interpreter to include full multitasking support. Software for the microsystem is developed using an RCA microcomputer available on various host machines.

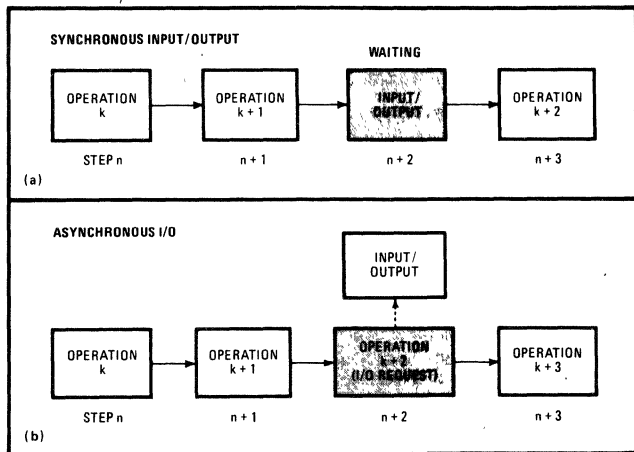
In parallel with the use of microcomputers, the development of the mCP Forth interpreter is one of the most advanced as single-chip microcomputers. The GDP 1804F, the RP1/12 from Rockwell International Corp., Milpitas, Calif. [Electronics, Nov. 29, 1982, p. 41] also has development in the area of real-time applications. It has gained a slow acceptance among system developers, and with the formation of Forth standards committees and the spread of interpreters into more systems, the state-oriented language is rapidly attracting the attention of other houses.

Forth is a compiled language in which basic procedure calls, or words, are used to build up more complex words. Because of its brevity, programs tend to be very compact. Once the programmer gets used to reverse-engineering notation, program development is simply a matter of building up the system dictionary with the words needed for a particular application.

ing systems, the AST mechanism allows tasks to perform operations that were contingent on the completion of a separate real-time operation. For example, by issuing an AST when it has completed its work, a read task is able to notify another task that a buffer has been filled. The other task is then free to initiate whatever calculation

may be needed to make use of this new data.

Besides such modifications improving Unix's response to asynchronous events, Masscomp upgraded the system's throughput by adding support for contiguous files to the file-management system. In this way, large amounts of data may be written at a high speed to



**2. No blocking.** In synchronous I/O, execution of a task blocks, or waits (tinted), until the data transfer is completed (a). Since I/O is handled independently, a task need only request an I/O operation (shaded) and continue on to the next operation.

consecutive disk sectors. Since other disk accesses are locked out in this mode, the disk head will be positioned correctly, thereby eliminating unnecessary and time-consuming movements.

In addition to these I/O add-ons, Masscomp boosted intertask communication capability by enlarging the Unix standard intertask communication mechanism, called pipes, to allow tasks to transfer buffers. In an alternative approach, Charles River Data Systems of Natick, Mass., allows tasks in its Unix-like Unos system to share data directly. A number of independently constructed software tasks may use a common set of locations in memory to transfer data between themselves or to perform some sequence of calculations. However, whenever asynchronous tasks share some common re-

source, their use of the resource could result in corrupted data—unless some mechanism coordinates their activities, such as the monitor concept described on page 108. Unos provides a mechanism called event counts to help avoid these conditions.

Event counts are integer values that are a nondecreasing count of the number of times some particular event has occurred. By using an event count associated with some task that produces shared data and another event count for a task that consumes the shared data, programmers may ensure the correct sequencing of asynchronous data-producing and -consuming tasks. Similarly, event counts serve as primitive operations for emulating the synchronization function that is provided by semaphores and the mutual exclusion that is furnished by monitors.



## Chips come to aid of embedded systems

□ Storing machine instructions in read-only memory is hardly a new concept in microprocessors. If supporting software totally breaks down, Digital Equipment Corp.'s LSI-11, for example, resorts to a basic keyboard monitor stored in a special ROM that is logically placed in the input/output address space. Using a primitive on-line debugging technique stored in the same ROM as the monitor, a software designer may read and alter memory locations and initiate a bootstrap loading operation from storage—a common provision in computer systems.

From these primitive beginnings, however, ROM-based software has evolved into complete operating systems in memory, engendering the term silicon software. Complementing hardware for distributed-processing architectures, such silicon-software systems signal a migration of application software into dedicated microcomputers previously considered unable to gain full systems capability. For developers of dedicated microcomputers embedded in some larger real-time system, silicon software spells the end of the need to reinvent the wheel to carry out the fundamental functions of a real-time operating system.

### Extending the microprocessor

Functionally, silicon operating systems extend the microprocessor's instruction set to include system-level instructions that perform operations on software structures, like queues and tables, rather than on hardware registers. Application-program developers are then presented with a virtual machine—one that is perceived by the programmer as different from the actual host processor. In these virtual operating-system machines, their instruction set includes a well-defined set of system calls as well as the basic machine instructions of the host microprocessor. For example, with systems like VRTX and RMX, the virtual microprocessor has a special set of instructions for handling interrupts (see Table 1).

For system developers, however, the problems in developing reliable silicon software extend beyond resource

protection, timing, and communication problems (see pp. 106-111). In fact, the development problems extend beyond the purely logistical exercise of maintaining a separate ROM-based instruction store and one for variables that need to be placed in system read-write memory. Treading a fine edge between the full function of a general operating system and the fine-tuned performance of special-purpose software, silicon systems need to balance the need for a wide range of system functions with the requirement that they squeeze into a minimal amount of ROM.

### Flexibility for expansion

Still, once a system meets a reasonable compromise between capability and size, it should not irrevocably lock the user into accepting its choices. For example, many real-time applications require some custom peripheral-device drivers and system-level functions. Consequently, the program should provide a mechanism for logically incorporating user-written extensions to the operating system, such as the user-defined pointers in the VRTX system from Hunter & Ready, Palo Alto, Calif.

In VRTX, a configuration table (Table 2) in system random-access memory allows specification of a custom routine that is to be executed whenever the system is initialized. For even more delicate control of system operations by custom software, a trio of pointers in the table specifies user-written routines to be accessed whenever a task is created or deleted or whenever a context switch is performed. Hunter & Ready also includes a location in this baseline configuration table for its anticipated file-management extensions to VRTX.

The 80130, an RMX-86 kernel in silicon from Intel Corp., Santa Clara, Calif., generalizes this approach through an index table containing pointers to system routines. If circumstances require the replacement of an existing system routine, the index-table pointer is merely altered to indicate the address of the new routine. In an

TABLE 1. SYSTEM CALLS FOR HANDLING INTERRUPTS

Versatile Real Time Executive (VRTX)	
UI POST	deposit message from interrupt handler
UI EXIT	exit from interrupt handler
UI TIMER	timer interrupt
UI RXCHR	receiver ready interrupt
UI TXRDY	transmitter ready interrupt
iRMX-86	
RQSETSINTERRUPT	assign interrupt handler
RQSRESETSINTERRUPT	deassign interrupt handler
RQSGETSLEVEL	return number of highest-priority interrupt level currently being processed
RQSSIGNALSINTERRUPT	signal from interrupt handler that event has occurred
RQSWAITSINTERRUPT	wait for occurrence of event
RQSEXITSINTERRUPT	relinquish control of the system
RQSENABLE	enable hardware to accept interrupts
RQSDISABLE	disable hardware from accepting interrupts

embedded system, this new routine could be placed in ROM along with application software.

Now that programs in ROM have matured into silicon systems, the development of software for embedded systems may now follow a more hospitable development cycle. The particular method used to create embedded systems will, in general, fall into one of two paths represented by the two major camps.

On one hand, kernels in silicon from systems such as RMX-86 or the MSP from Hemenway Corp., Boston, Mass., for the 68000 or Z8000 are self-contained subsets of the full operating system. Consequently, software programmers may use the full development version of the same operating system as that in the eventual target to create the application package. On the other hand, development of application programs around the ZRTS system from Zilog Corp., Cupertino, Calif., or Hunter & Ready's VRTX for the Z8002, iAPX-86 family, or 68000 relies on the use of a separate development system to create software for the target microprocessor, since this software does not have development versions.

### Two approaches

The significance of these two approaches as usual depends on the intended application. Hunter & Ready views VRTX as a set of processor-independent building blocks that programmers use to construct application packages for embedded systems. As such, the programmers employ the same development systems that they might use to build application code, but now with the benefit of a sophisticated set of ready-made system-soft-ware components.

In playing its part in Intel's systematic drive toward

TABLE 2. VRTX CONFIGURATION TABLE

Table Entry	Entry Description
sys RAM addr	system beginning address
sys RAM-size	system memory size
sys-stack-size	system stack size
user RAM-addr	starting address for available memory in initial partition
user-RAM-size	size of initial partition
user-block-size	size of memory block for dynamic allocation
user-stack-size	size of stack for user tasks
user-task-addr	address of first user task
user-task-count	maximum number of tasks
sys-init-addr	address of user-supplied initialization routine
sys-tcreate-addr	address of user-supplied routine accessed when a task is created
sys-tdelete-addr	address of user-supplied routine accessed when a task is deleted
sys-tswap-addr	address of user-supplied routine accessed when a context switch occurs
[RESERVED]	address of Hunter & Ready future extensions to VRTX

providing an integrated environment around the iAPX-86 family, the 80130 holds the anchor position in an interlocked set of components. Able to function independently of the upper layers of the operating system, it provides a hardware base for the rest of RMX-86. Serving as a viewport into this system-software base for the central processing unit, Intel's universal run-time and development interfaces offer the mechanism for software portability needed for the next stage in the company's plan to grow into higher-performance microprocessors, such as the 186, 286, and 386.

While interlocking with the software in this way, the 80130 also must play its role in the complementary relationships being established at the hardware level. As such, it includes on-chip hardware support for system-level functions, including timers, interrupt controller, bus control, and bus interface.

Meanwhile, Intel's plan for software-in-silicon becomes evident as it gathers the other pieces of the puzzle, such as the 82730 text-coprocessor chip, the 82586 local-network coprocessor, and the 82720 graphics processor chip. Similar to the 80130 software connection, the 82720 graphics part interlocks with the rest of the system at the software level through its support of another well-defined software interface—the virtual device interface. Yet to come are pieces for voice I/O support, as well as some level of hardware support for data-base access. □

June 1983

**Intel's Matchmaking Strategy:  
Marry iRMX<sup>™</sup> Operating  
System with Hardware**

**Chappell Brown**  
Software Editor  
Electronic Engineering Times

## Intel's Matchmaking Strategy: Marry iRMX™ Operating System With Hardware

*Intel's major software product, the iRMX™-86 16-bit operating system, which is now in its fifth release, represented a three-year development investment which most independent software vendors would have found a daunting prospect in 1978 when the project was conceived.*

*The investment was essential. By the mid-1970s, feedback from OEMs working with Intel's hardware revealed problems with system integration—the marriage of software with hardware. It consequently slowed sales, with the prospect of even greater problems at higher levels of circuit integration. Intel management, looking for ways of coping with the ballooning software requirements of the rapidly accelerating hardware program, began stepping up software development programs in the mid-1970s.*

“The RMX program illustrates a number of things one needs to keep in mind with developing a real-time operating system,” explained Bill Lattin, Intel's OEM microcomputer systems manager. “Foundations must be well laid so the system can grow and evolve over time. And there is a need for the system to be open to modification by typical OEM-specific applications.

“Although the RMX program has been around since 1978, it has only recently hit its stride, as processor technology has advanced to use the full range of its features,” Lattin said.

The fast-paced microcomputer market had created a new situation for systems designers in terms of a radical shift in the hardware/software cost ratio. Earlier hardware generations involved various expensive centralized facilities. Not only was software cheap in comparison, but the hardware environment changed slowly, so that it was also feasible to rewrite systems as needed.

But when the price of a computer drops to as low as \$5, the hardware environment becomes volatile and software turns into a major investment. Intel was finding that customers might invest as much as two-thirds of their development costs in software, only to see it eclipsed by evolving VLSI technology.

It became evident that merely supplying components would become increasingly counterproductive. Thus, the Intel “total solution” emerged—a consistent systems approach to hardware sales, which naturally depends heavily on a viable software program.

Object-oriented programming is a method which has worked best in creating a software program blending with the component approach. By hiding data representation within an object with its own object manager, changes in the hardware environment that affect the data can be accommodated without having to change the rest of the software.



A price is paid in terms of program size with this approach, however. And it was difficult at the time to justify this kind of liability with the existing onboard memories of the 8-bit generation.

Bill Stevens, iRMX-86 program manager for release five, explained the difficult decisions that had to be made at the outset of the program. “Every engineering decision involves a trade-off. We wanted to optimize program productivity and we had to have modularity. The consequence of this was large size. It turned out that a minimum configuration was 12 kbytes wide and the full configuration was 128 kbytes. At the time we did not have 64k dynamic RAMs and 64k EPROMs, so we didn't have the technology to realize the systems of initial specifications times. Bruce Schafer has to take credit for making that decision to go ahead anyway, early on. . . it was a gutsy decision, and it turned out to be absolutely right!”



Had Intel known of the difficulty it was about to encounter in producing its 64-kbyte RAM, Schafer may have had second thoughts.

Schafer joined Intel in 1976 and began working on iRMX-80. "It was a nice little system," Schafer said. "A miniature dispatcher had evolved to handle multiple asynchronous events and became a primitive OEM operating system. It was tempting to do an enlarged version of it, mainly because I was already working on it for the 16-bit generation."

Schafer soon found himself centrally involved in the task of heading off the 16-bit software crunch, laying groundwork for a system that could cover a wide range of applications, many of them unknown at the time, and a system which could also evolve with hardware advances.

"When you set out to design a system of that scope, you don't just sit down and start writing code. It's definitely a top-down process," explained Schafer. He discovered early in the project that the purely technical hurdles in writing software were minor compared to orchestrating a team of engineers on such a comprehensive project.

The iRMX-86 system is multi-layered, and the project had to be coordinated across these layers along with the sequence of planning, design and implementation. On top of that, a thorough testing program had to be coordinated with all phases.

"I had a difficult time convincing engineers on the project that documentation of their work was as important as the work itself. Specifications were absolutely crucial to the development phase," said Schafer.

Schafer began with a customer survey to discover the kind of problems OEMs were experiencing with system design. He wrote a production implementation plan, which was critiqued by marketing and engineering personnel. This was approved in June 1978 and formed the basis for engineering specifications. A critiquing process evolved as the organizing principle behind initial product design; engineers on the project would exchange documentation and then meet to evaluate the progress of the system.

The sessions were lively and the problems of coordinating implementation, testing and design along with the pressure of deadlines for the whole program generated quite a bit of excitement.

Development testing turned out to be a particularly thorny problem—the asynchronous interrupts and multiple-

processing aspects of real-time applications required a special test apparatus to simulate a real-world environment.

What they came up with is a nucleus executing directly on the 8086 and 8088 processors as the basic building block of the system. Together with the next layer—a basic I/O system—a minimal operating system can be configured, which has been found useful in many applications.

However, it was necessary to develop an application on the Series-III development system even though the target was going to be RMX. "We quickly realized that users want to be able to do development work on the machine they target on," said Schafer. "This is particularly important for field maintenance . . . you can't drag a Series-III out to an oil derrick." To realize this goal, Intel built higher layers around iRMX so that program development could be done without a Series-III. Higher layers involve extended I/O and human interface facilities. After this, customer-written software can be added in high-level languages.

A major objective has been to provide a stable base for independent software vendors; with its latest release, Intel also announced an ISV program initially involving three major vendors; Microsoft, Digital Research and Mark Williams Inc.

The first release of iRMX-86 came out in April 1980. Since then, the system has been refined and released four more times, with release five appearing last December. An Interactive Configuration Utility appeared for the first time with release five, a further attempt to aid OEMs in putting their systems together. The system designer runs the ICU program on a terminal and is quizzed on his requirements, after which the program generates the unique iRMX software for his application.

"It has been a successful product in its own right, apart from its role in the hardware program, but I doubt that anyone would have wanted to invest in a three-year development process before there was a chance at some return," observed Stevens, who has been most excited by the diverse applications he has seen. "I've really enjoyed the iRMX symposiums. There is always some new system demonstrated. In Tokyo, I just saw an 8086-based scientific system with really first-class graphics put together by Seiko. Another time I saw a blood analyzer based on the system. There are even RMX-based personal computers."

June 1983

**iRMX™ 86 Has Functionality,  
Configurability**

**Bruce W. Schafer**  
Software News

## iRMX™ 86 Has Functionality, Configurability

The iRMX™ 86 operating system provides a modular set of building blocks from which users can create a wide variety of applications. iRMX 86 features include: multitasking; interrupt support; multiprogramming support; device independence; tree-structured directories, file access control; and interactive debugging.

The iRMX 86 operating system combines the concepts of objects, types, and type extension to form a highly-functional and highly-configurable foundation for applications software. The operating system is designed for use with programs executing on the iAPX 86 and iAPX 88 processors. The 8087 numeric data processor is supported as an option.

### Execution Environment

The iRMX 86 Operating System can be used with a variety of hardware configurations. Interactive disk-based systems as well as ROM-resident systems can be constructed.

Any part of the operating system's code can reside in ROM/PROM memory. Alternatively, all or part of this code can be "bootstrapped" into RAM using a small, configurable bootstrap loader provided with the product. The application code can similarly be committed to PROM or bootstrapped into RAM.

The operating system divides the execution environment into **jobs** and **tasks**. A task is described by a set of processor registers, a stack, a priority, and a state. Jobs provide resources for tasks. A job can be viewed as a task environment. In the simplest case a job represents a memory pool. Tasks executing in the same job share the same pool of memory. When a job is deleted, all tasks within the job are also deleted and all memory allocated to these tasks is deallocated.

The iRMX 86 system is composed of several layers. The innermost layer is the Nucleus, which provides multitasking, interrupt control, and multiprogramming support. The first optional layer, the Basic I/O System, supports device-independence, directories, random access, and file access control.

"On top" of this layer, users may add the Extended I/O System (providing services such as automatic buffering) or the Application Loader (which supports loading both absolute code and locatable code). The Human Interface uses these inner layers to support user-defined commands in addition to a set of standard commands.

The design of the iRMX 86 Operating System is based on a set of *object types*. The operating system supports dynamic object creation. Each time an object is created, the operating system allocates the proper resources to the object and returns a 16-bit virtual address called the object's *token*. This token is subsequently used by the application to identify the specific object.

By implementing this object-oriented approach, the iRMX 86 Operating System hides implementation details from the application software. The iRMX 86 Nucleus also allows users to add custom object types without changing the Nucleus.

### I/O Devices

I/O devices can be manipulated in two ways. The first approach allows the application to receive interrupts directly from the I/O device. The second approach utilizes the iRMX 86 Basic I/O System. With this approach, a device driver must be written for initiation of I/O requests and for interrupt handling. The application software interfaces to these drivers through the Basic I/O System by making **read**, **write**, **seek**, and **special-function** requests.

The iRMX 86 Operating System currently includes device drivers for diskettes, Winchester disks, magnetic bubble storage devices, and Storage Module Device (SMD) interfaces.

The iRMX 86 Extended I/O System defines the concept of a *logical device*. Using this feature, each device is assigned a logical name. Application programs refer to logical devices without knowing which **physical device** is associated with each logical device. In this manner, the physical device can be changed without changing the application programs.

The Basic I/O System provides asynchronous I/O functions. Each asynchronous function is initiated by a procedure call that queues the request. The procedure call returns immediately with an indication of whether the request was successfully queued. When the request is actually completed, a response message is sent to the mailbox specified.

The Extended I/O System automatically synchronizes I/O requests. Again, a procedure call is used to initiate I/O. The procedure, however, does not return until the request is complete. To enhance efficiency when this automatic synchronization is used, the Extended I/O System permits read-ahead and write-behind.

The iRMX 86 Human Interface automatically parses input lines and invokes the appropriate program based on the first word in each line. A program executing under the iRMX 86 Human Interface can request command execution by providing the text for these commands to the command line interpreter.

The iRMX 86 Human Interface is supplied with a basic set of commands to manipulate files. These commands include **directory** display, **create** directory, **rename** file, **copy** file, **delete** file, and **submit** a set of commands. Users can add custom commands to this set.

The iRMX 86 Debugger provides the capability to debug one or more tasks while the rest of the system continues to execute. The Debugger allows a user to specify that a task be suspended when the task executes a particular instruction and when the task communicates with other tasks.

The most general communication mechanism provided by the iRMX 86 Nucleus is the **mailbox** object type. Each object of this type is described by two queues—a queue of messages waiting to be handled by tasks and a queue of tasks waiting for messages. An additional attribute of a mailbox is the specification of whether the queue of tasks is to be handled first-in, first-out or on a relative priority basis.

The iRMX 86 Operating System also provides a **semaphore** object type. Each semaphore is described by a queue of waiting tasks and a unit count. This unit count is equivalent to a count of empty messages at a mailbox, but, because no actual messages are involved, a semaphore is a more efficient mechanism than a mailbox. Since semaphores allow multiple units to be sent at the same time, semaphores are used to create deadlock allocation functions.

To provide additional efficiency, the iRMX 86 Nucleus also provides a special type of semaphore called a **region**.

Each iRMX 86 task has a **dynamic priority** attribute. This priority describes the relative importance of the Task's function with respect to other system functions. The iRMX 86 Nucleus always runs the highest priority ready task. When several tasks of the same priority are ready, the Nucleus arbitrarily chooses between them.

Scheduling requires changing the state of the task and placing the task in a queue of ready tasks. Whenever a task is scheduled or descheduled, the Nucleus checks the ready queue and allocates the processor to the highest priority ready task. In order to ensure event-driven scheduling, the iRMX 86 Nucleus is designed to place an absolute limit on the interval during which interrupts are masked.

One attribute of the **job** type is a **memory pool**. Each memory pool represents the memory resources available to the tasks executing within a job. All objects created by these tasks are allocated memory from the pool.

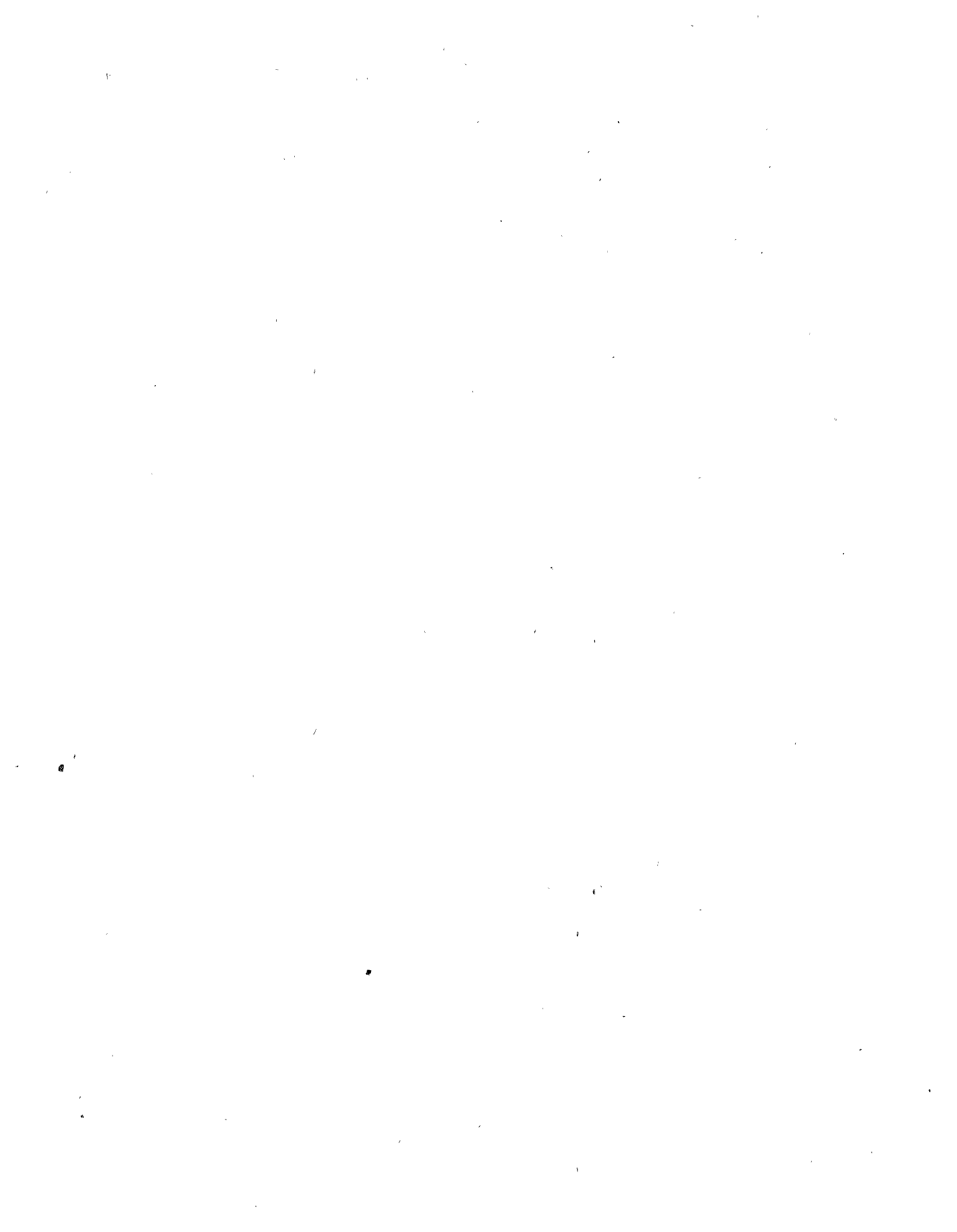
The iRMX 86 Operating System supports three **file types**. In all cases, application programs read and write data without knowing the device or the file type that is used. The following file types are supported:

- 1) **Physical**—A device accessed as a physical file is treated as a contiguous sequence of bytes.
- 2) **Stream**—Stream files do not exist on actual physical devices; rather, data is transmitted directly from one program to another.
- 3) **Named**—Named files represent the traditional notion of files. Named files are described by a path through a tree-structured network of directories.

The name of an iRMX 86 file is given as a path through a tree-structured network of directories. Each directory in this structure can point directly to data files and to other directories. One directory on each device is considered the **root** directory. All paths on a particular device begin in this directory.

The basic file functions for all three file types are: **open**, **close**, **read**, and **write**. When random file access is required, the **seek** system call is added to this set. For named files, additional functions are needed. These functions include the **rename** function, the **truncate** function, and the **change-access** function.

When a file is opened, the calling program specifies the type of file access required. For a data file, three types of access are permitted: **read**, **write**, and the **read/write** combination. The I/O system verifies that the specified access is available and grants the **open** request only if the requested access is available.



---

# **Translators and Utilities for Program Development**

---

**3**



## **TRANSLATORS AND UTILITIES FOR PROGRAM DEVELOPMENT**

Intel offers an extensive selection of program development tools for its microprocessor (8080, 8086, 8088, 80186, 80286) and microcontroller (8048, 8051, 8096 etc.) families. These tools include translators and programming utilities such as linkers, relocators, and library managers. These program development tools are high quality, time tested tools for the professional. Based on a set of well-defined standards, they provide an integrated development environment. The result is an extremely flexible and productive program development environment.

### **A LANGUAGE FOR EVERY NEED**

The iAPX-86 family has the most comprehensive set of translators available for a microprocessor. These include a macro assembler and compilers for PL/M, Pascal, FORTRAN, and C (see Table 1). The macro assembler produces the most optimum code. PL/M is the most popular 8086 language for systems programming and provides the best of both optimal code and high level language capabilities.

The main advantage of 'C' is portability across different target machines. Pascal and FORTRAN are used extensively for applications programming. To allow applications to be portable, Pascal and FORTRAN conform to ISO and ANSI77 standards respectively, with many useful extensions for microprocessor applications.

Intel's microcontroller family (8048, 8051, 8096 etc.) is similarly the best supported in the industry. PL/M-51 was the first high level language ever to be introduced for a microcontroller. The 8096 is similarly supported with PL/M-96. Every microcontroller in the family is supported with an assembler and linkage utilities.

### **USE A MIXTURE OF LANGUAGES FOR MAXIMUM FLEXIBILITY**

Programs are typically decomposed into modules to exploit the many benefits of modular programming. Intel's integrated programming technology allows different modules of the same program to be programmed in a variety of languages. For instance, the most performance-sensitive system modules may be coded in assembler or in PL/M. The application modules, on the other hand, can be written in Pascal to speed up programming. The system and application modules can then be linked into one program using the linker. Hence, the various modules of a program can each be coded in the most suitable programming language.

### **UTILITIES ENHANCE PROGRAMMING PRODUCTIVITY**

A set of utilities is provided to support modular and position independent programming. The linkers combine the constituent modules of a program into one system. A locator is provided to position the code in memory. This allows code to be placed in appropriate ROM and RAM locations. Also, coding can be done in a position-independent way. The librarian provides a structured way of organizing frequently used routines. The routines needed by a particular program can be linked in by the linker. The linker automatically selects only those modules from the library that are needed by the program. For the protected, virtual-memory, and multi-tasking processor iAPX 286, a sophisticated operating system configuration utility BUILD-286, is provided.

### **FULL RANGE OF DEBUG SUPPORT**

The programming tools are integrated with the debugging tools via the well-defined Intel object module format standard. iAPX-86 family programs may be debugged using any of the Intel 8086 debug tools. This includes PSCOPE which provides source level software debug, and the ICE products which provide in-target real-time debug. Microcontroller software is similarly supported by the various emulators and ICE units.

### **CHOOSE FROM A VARIETY OF HOST CONFIGURATIONS**

The programming tools are provided on a variety of development host environments to meet the needs of different project sizes and development budgets (see Table 1). The environments span personal development systems (iPDS), stand alone development systems (Series III, Series IV), network development systems (NDS-II) and even the VAX/VMS microcomputer. The programming tools work identically, no matter which of the available host configurations is chosen. This allows the user to grow his development environment, as his needs grow, without impacting previous investment in software.

\* VAX/VMS is a trademark of Digital Equipment Corporation.



**Table 1. Intel Translator/Host Summary**

Language	Component Family	Host Code
Macro Assembler + Utilities	2920	1,2
	MCS-85 Family	1
	MCS-48 Family	1
	MCS-51 Family	1
	iACX-96 Family	2
	iAPX-86 Family	1,2,3
	iAPX-286 (Protected Mode)	2,3*
PL/M	MCS-85 Family	1
	MCS-51 Family	1
	iACX-96 Family	2*
	iAPX-86 Family	1,2,3
	iAPX-286 (Protected Mode)	2,3*
PASCAL	MCS-85 Family	1
	iAPX-86 Family	2,3
	iAPX-286 (Protected Mode)	2,3*
FORTRAN	MCS-85 Family	1
	iAPX-86 Family	2
	iAPX-286 (Protected Mode)	3*
"C"	iAPX-86 Family	2,3*
	iAPX-286 (Protected Mode)	2*,3*
Ada	iAPX-86 Family	3*
	iAPX-286 (Protected Mode)	3*

NOTE: \* = Planned

**HOST CODES**

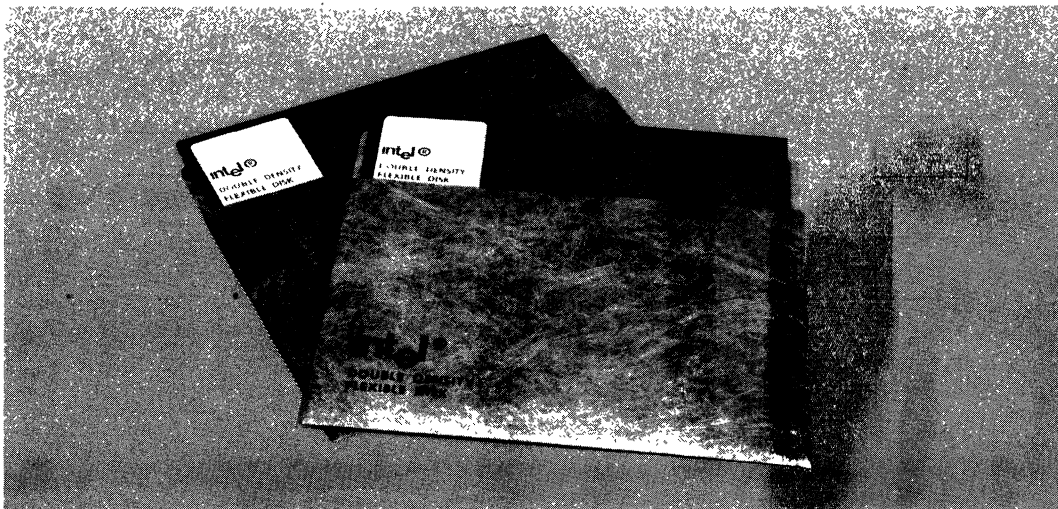
- 1 = Intel 8085 Based Development System (iPDS, MDX Series IIE)
- 2 = Intel iAPX-86 Based Development System (Series III, Series IV)
- 3 = VAX/VMS Minicomputer



## PL/M 80 HIGH LEVEL PROGRAMMING LANGUAGE

- Provides Resident Operation on Intellec® Microcomputer Development System and Intellec® Series II Microcomputer Development Systems
- Produces Relocatable and Linkable Object Code
- Sophisticated Code Optimization Reduces Application Memory Requirements
- Speeds Project Completion with Increased Programmer Productivity
- Cuts Software Development and Maintenance Costs
- Improves Product Reliability with Simplified Language and Consequent Error Reduction
- Eases Enhancement as System Capabilities Expand

The PL/M 80 High Level Programming Language Intellec Resident Compiler is an advanced, high level programming language for Intel 8080 and 8085 microprocessors, iSBC-80 OEM computer systems, and Intellec microcomputer development systems. PL/M has been substantially enhanced since its introduction in 1973 and has become one of the most effective and powerful microprocessor systems implementation tools available. It is easy to learn, facilitates rapid program development and debugging, and significantly reduces maintenance costs. PL/M is an algorithmic language in which program statements naturally express the algorithm to be programmed, thus freeing programmers to concentrate on system development rather than assembly language details (such as register allocation, meanings of assembler mnemonics, etc.). The PL/M compiler efficiently converts free-form PL/M programs into equivalent 8080/8085 instructions. Substantially fewer PL/M statements are necessary for a given application than would be using assembly language or machine code. Since PL/M programs are problem oriented and thus more compact, programming in PL/M results in a high degree of productivity during development efforts, resulting in significant cost reduction in software development and maintenance for the user.



## FUNCTIONAL DESCRIPTION

The PL/M compiler is an efficient multiphase compiler that accepts source programs, translates them into object code, and produces requested listings. After compilation, the object program may be first linked to other modules, then located to a specific area of memory, and finally executed. The diagram shown in Figure 1 illustrates a program development cycle where the program consists of three modules: PL/M, FORTRAN, and assembly language. A typical PL/M compiler procedure is shown in Table 1.

### Features

Major features of the Intel PL/M 80 compiler and programming language include:

**Resident Operation** — on Intellec microcomputer development systems eliminates the need for a large in-house computer or costly timesharing system.

**Object Code Generation** — of relocatable and linkable object codes permits PL/M program development and debugging in small modules, which may be easily linked with other modules and/or library routines to form a complete application.

**Extensive Code Optimization** — including compile time arithmetic, constant subscript resolution, and common subexpression elimination, results in generation of short, efficient CPU instruction sequences.

**Symbolic Debugging** — fully supported in the PL/M compiler and ICE-85 in-circuit emulators.

**Compile Time Options** — includes general listing format commands, symbol table listing, cross reference listing, and "innerlist" of generated assembly language instructions.

**Block Structure** — aids in utilization of structured programming techniques.

**Access** — provided by high level PL/M statements to hardware resources (interrupt systems, absolute addresses, CPU input/output ports).

**Data Definition** — enables complex data structures to be defined at a high level.

**Re-entrant Procedures** — may be specified as a user option.

### Benefits

PL/M is designed to be an efficient, cost-effective solution to the special requirements of microcomputer software development as illustrated by the following benefits of PL/M use:

**Low Learning Effort** — even for the novice programmer, because PL/M is easy to learn.

**Earlier Project Completion** — on critical projects, because PL/M substantially increases programmer productivity while reducing program development time.

**Lower Development Cost** — because increased programmer productivity requiring less programming resources for a given function translates into lower software development costs.

**Increased Reliability** — because of PL/M's use of simple statements in the program algorithm, which are easier to correct and thus substantially reduce the risk of costly errors in systems that have already reached full production status.

**Easier Enhancement and Maintenance** — because programs written in PL/M are easier to read and easier to understand than assembly language, and thus are easier to enhance and maintain as system capabilities expand and future products are developed.

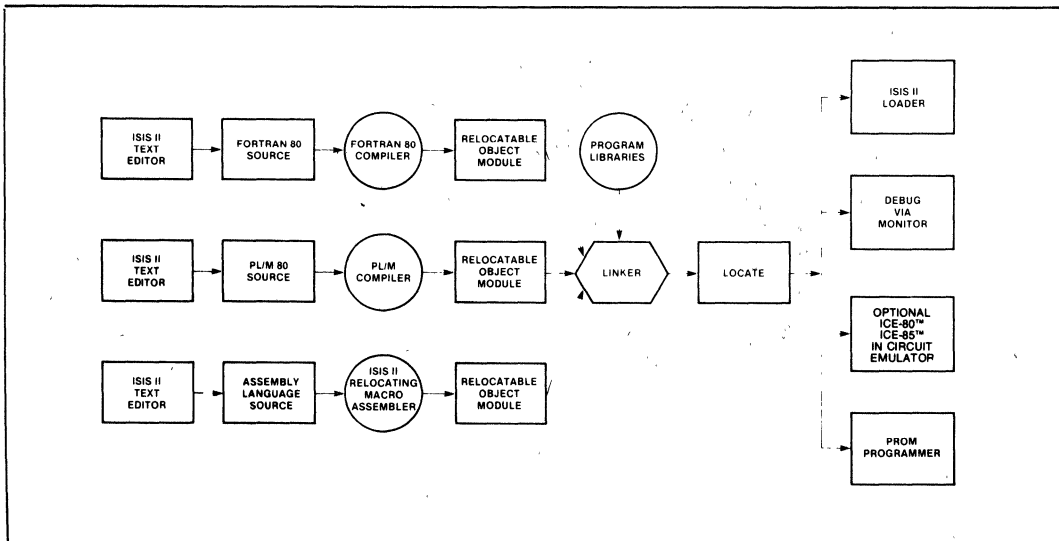


Figure 1. Program Development Cycle Block Diagram

**Simpler Project Development** — because the Intel microcomputer development system with resident PL/M 80 is all that is needed for developing and debug-

ging software for 8080 and 8085 microcomputers, and the use of expensive (and remote) timesharing or large computers is consequently not required.

**Table 1. PL/M-80 Compiler Sample Factorial Generator Procedure**

		\$OBJECT(F1:FACT.OB2)
		\$DEBUG
		\$XREF
		\$TITLE('FACTORIAL GENERATOR — PROCEDURE')
		\$PAGEWIDTH(80)
1		FACT:
		DO;
2	1	DECLARE NUMCH BYTE PUBLIC;
3	1	FACTORIAL: PROCEDURE (NUM,PTR) PUBLIC;
4	2	DECLARE NUM BYTE, PTR ADDRESS;
5	2	DECLARE DIGITS BASED PTR (161) BYTE;
6	2	DECLARE (I,C,M) BYTE;
7	2	NUMCH = 1; DIGITS(1) = 1;
9	2	DO M = 1 TO NUM;
10	3	C = 0;
11	3	DO I = 1 TO NUMCH;
12	4	DIGITS(I) = DIGITS(I)*M + C;
13	4	C = DIGITS(I)/10;
14	4	DIGITS(I) = DIGITS(I) — 10*C;
15	4	END;
16	3	IF C<>0 THEN
17	3	DO;
18	4	NUMCH = NUMCH + 1; DIGITS(NUMCH) = C;
20	4	C = DIGITS(NUMCH)/10;
21	4	DIGITS(NUMCH) = DIGITS(NUMCH) — 10*C;
22	4	END
		END;
24	2	END FACTORIAL;
25	1	END;

**SPECIFICATIONS**

**OPERATING ENVIRONMENT**

Intel Microcomputer Development Systems  
(Series II, Series III, Series IV)  
Intel Personal Development System

**DOCUMENTATION**

PL/M 80 Programming Manual  
ISIS-II PL/M 80 Compiler Operator's Manual

**ORDERING INFORMATION**

Product Code	Description
MDS*-PLM	PL/M 80 High Level Language Compiler. Needs Software License.

**SUPPORT:**

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

\*MDS is an ordering code only and is not used as a product or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.



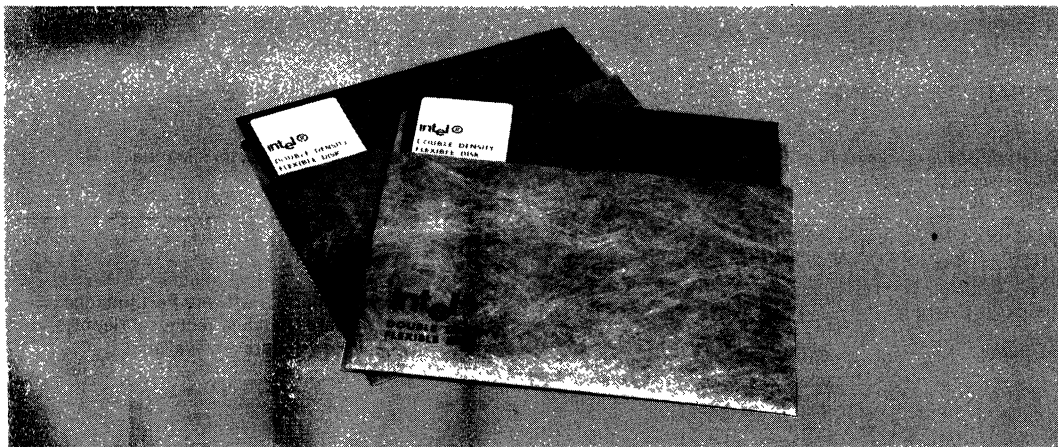
## FORTRAN 80 8080/8085 ANS FORTRAN 77 INTELLEC® RESIDENT COMPILER

- Meets ANS FORTRAN 77 Subset Language Specification plus adds Intel® microprocessor extensions
- Supports Intel Floating Point Standard with the FORTRAN 80 software routines, the iSBC-310™ High Speed Mathematics Board, or the iSBC-332™ math multimodule
- Executes on Intellec Microcomputer Development System, Intellec Series II Microcomputer Development System, and Personal Development System
- Supports full symbolic debugging with ICE-80™ and ICE-85™
- Produces relocatable and linkable object code compatible with resident PL/M 80 and 8080/8085 Macro Assembler
- Provides optional run-time library to execute in RMX-80™ environment
- Has well defined I/O interface for configuration with user-supplied drivers

FORTRAN 80 is a computer industry-standard, high-level programming language and compiler that translates FORTRAN statements into relocatable object modules. When the object modules are linked together and located into absolute program modules, they are suitable for execution on Intel 8080/8085 Microprocessors, iSBC-80 OEM Computer Systems, Intellec Microcomputer Development Systems and Personal Development Systems. FORTRAN 80 meets the ANS FORTRAN 77 Language Subset Specification<sup>1</sup>. In addition, extensions designed specifically for microprocessor applications are included. The compiler operates on the Intellec Microcomputer Development System and Personal Development System under the ISIS-II Disk Operating Systems and produces efficient relocatable object modules that are compatible for linkage with PL/M 80 and 8080/8085 Macro Assembler modules.

The ANS FORTRAN 77 language specification offers many powerful extensions to the FORTRAN language that are especially well suited to Intel 8080/8085 Microprocessor software development. Because FORTRAN 80 conforms to the ANS FORTRAN 77 standard, the user is assured of compatibility with existing FORTRAN software that meets the standard as well as a guarantee of upward compatibility to other computer systems supporting an ANS FORTRAN 77 Compiler.

<sup>1</sup>ANSI X3J3/90



## FORTRAN 80 LANGUAGE FEATURES

Major ANS FORTRAN 77 features supported by the Intel FORTRAN 80 Programming Language include:

- Structured Programming is supported with the IF ... THEN ... ELSE IF ... ELSE ... END IF constructs.
- CHARACTER data type permits alphanumeric data to be handled as strings rather than characters stored in array elements.
- Full I/O capabilities include:
  - Sequential and Direct Access files
  - Error handling facilities
  - Formatted, Free-formatted, and Unformatted data representation
  - Internal (in-memory) file units provide capability to format and reformat data in internal memory buffers
  - List Directed Formatting
- Supports arrays of up to seven dimensions.
- Supports logical operators
  - .EQV. — Logical equivalence
  - .NEQV. — Logical nonequivalence

Major extensions to FORTRAN 77 in Intel FORTRAN-80 include:

- Direct 8080/8085 port I/O supported by intrinsic subroutines.
- Binary and Hexadecimal integer constants.
- Well defined interface to FORTRAN-80 I/O statements (READ, OPEN, etc.), allowing easy use of user-supplied I/O drivers.
- User-defined INTEGER storage lengths of 1, 2 or 4 bytes.
- User-defined LOGICAL storage lengths of 1, 2 or 4 bytes.
- REAL STORAGE lengths of 4 bytes.
- Bitwise Boolean operations using logical operators on integer values.
- Hollerith data constants.
- Implicit extension of the length of an integer or logical expression to the length of the left-hand side in an assignment statement.
- A format descriptor to suppress carriage return on a terminal output device at the end of the record.

## FORTRAN 80 COMPILER FEATURES

- Supports multiple compilation units in single source file.
- Optional Assembly Language code listing.
- Comprehensive cross-reference, symbol attribute and error listing.
- Compiler controls and directives are compatible with other Intel language translators.
- Optional Reentrancy.
- User-defined default storage lengths.
- Optional FORTRAN 66 Do Loop semantics.
- Source files may be prepared in free format.

- The INCLUDE control permits specified source files to be combined into a compilation unit at compile time.
- Transparent interface for software and hardware floating point support, allowing either to be chosen at time of linking.

## FORTRAN 80 BENEFITS

FORTRAN 80 provides a means of developing application software for Intel MCS-80/85 products in a familiar, widely accepted, and computer industry-standardized programming language. FORTRAN 80 will greatly enhance the user's ability to provide cost-effective solutions to software development for Intel microprocessors as illustrated by the following:

- *Completely Complementary to Existing Intel Software Design Tools* — Object modules are linkable with new or existing Assembly Language and PL/M Modules.
- *Incremental Runtime Library Support* — Runtime overhead is limited only to facilities required by the program.
- *Low Learning Effort* — FORTRAN 80, like PL/M, is easy to learn and use. Existing FORTRAN software can be ported to FORTRAN 80, and programs developed in FORTRAN 80 can be run on any other computer with ANS FORTRAN 77.
- *Earlier Project Completion* — Critical projects are completed earlier than otherwise possible because FORTRAN 80 will substantially increase programmer productivity, and is complementary to PL/M Modules by providing comprehensive arithmetic, I/O formatting, and data management support in the language.
- *Lower Development Cost* — Increases in programmer productivity translates into lower software development costs because less programming resources are required for a given function.
- *Increased Reliability* — The nature of high-level languages, including FORTRAN 80, is that they lend themselves to simple statements of the program algorithm. This substantially reduces the risk of costly errors in systems that have already reached production status.
- *Easier Enhancements and Maintenance* — Like PL/M, program modules written in FORTRAN 80 are easier to read and understand than assembly language. This means it is easier to enhance and maintain FORTRAN 80 programs as system capabilities expand and future products are developed.
- *Comprehensive, Yet Simple Project Development* — The Intel Microcomputer Development System and Personal Development System, with the 8080/8085 Macro Assembler, PL/M 80 and FORTRAN 80 are the most comprehensive software design facilities available for the Intel MCS-80/85 Microprocessor family. This reduces development time and cost because expensive (and remote) timesharing or large computers are not required.

**SAMPLE FORTRAN-80 SOURCE PROGRAM  
LISTING**

```

*   ** THIS PROGRAM IS AN EXAMPLE OF ISIS-II FORTRAN-80 THAT
*   ** CONVERTS TEMPERATURE BETWEEN CELSIUS AND FARENHEIT

PROGRAM CONVRT

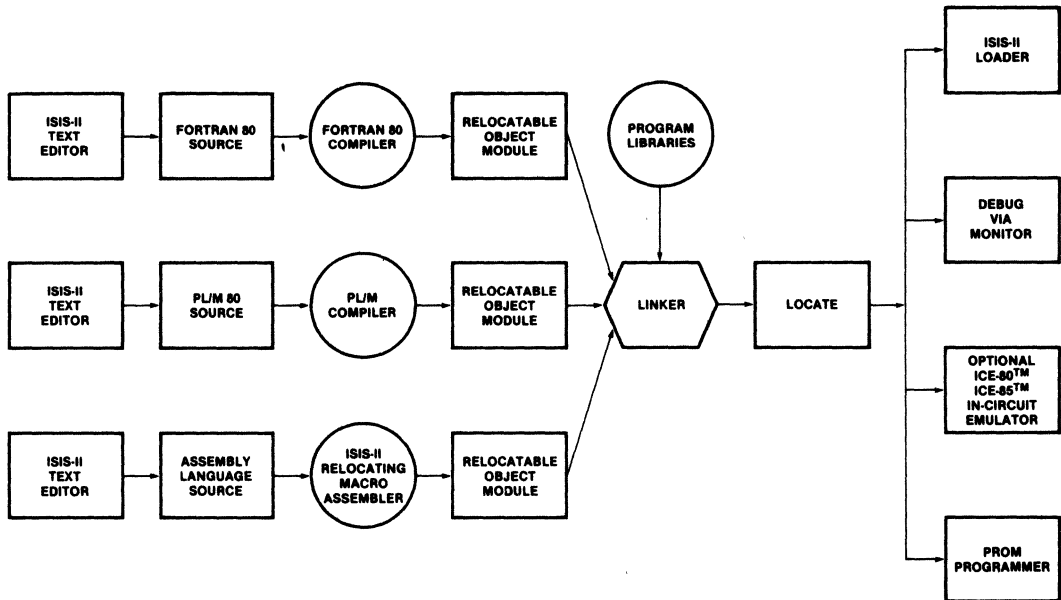
CHARACTER*1 CHOICE, SCALE

PRINT 100
*   ** ENTER CONVERSION SCALE (C OR F)
10  PRINT 200
    READ (5,300) SCALE

    IF (SCALE .EQ. 'C')
+     THEN
*       PRINT 400
*       ** ENTER THE NUMBER OF DEGREES FARENHEIT
        READ (5,*) DEGF
        DEGC = 5./9.*(DEGF-32)
*       ** PRINT THE ANSWER
        WRITE (6,500) DEGF,DEGC
*       ** RUN AGAIN?
20  PRINT 600
        READ (5,300) CHOICE
        IF (CHOICE .EQ. 'Y')
+         THEN
+           GOTO 10
+         ELSE IF (CHOICE .EQ. 'N')
+           THEN
+             CALL EXIT
+         ELSE
+           GOTO 20
+         END IF
    ELSE IF (SCALE .EQ. 'F')
+     THEN
*       ** CONVERT FROM FARENHEIT TO CELSIUS
        PRINT 700
        READ (5,*) DEGC
        DEGF = 9./5.*DEGC+32.
*       ** PRINT THE ANSWER
        WRITE (6,800) DEGC,DEGF
        GOTO 20
    ELSE
*       ** NOT A VALID ENTRY FOR THE SCALE
        WRITE (6,900) SCALE
        GOTO 10
    END IF
100  FORMAT(' TEMPERATURE CONVERSION PROGRAM',//,
+ ' TYPE C FOR FARENHEIT TO CELSIUS OR',/,
+ ' TYPE F FOR CELSIUS TO FARENHEIT',//)
200  FORMAT(/, ' CONVERSION? ', $)
300  FORMAT(A1)
400  FORMAT(/, 'ENTER DEGREES FARENHEIT: ', $)
500  FORMAT(/, F7.2, ' DEGREES FARENHEIT = ', F7.2, ' DEGREES CELSIUS.)
600  FORMAT(/, ' AGAIN (Y OR N)? ', $)
700  FORMAT(/, ' ENTER DEGREES CELSIUS: ', $)
800  FORMAT(/, F7.2, ' DEGREES CELSIUS = ', F7.2, ' DEGREES FARENHEIT', /)
900  FORMAT(/, 1H, A1, ' NOT A VALID CHOICE - TRY AGAIN!', /)
END

```

The FORTRAN 80 Compiler is an efficient, multiphase compiler that accepts source programs, translates them into relocatable object code, and produces requested listings. After compilation, the object program may be linked to other modules, located to a specific area of memory, then executed. The diagram shown below illustrates a program development cycle where the program consists of modules created by FORTRAN 80, PL/M 80 and the 8080/8085 Macro Assembler.



## SPECIFICATIONS

### OPERATING ENVIRONMENT

Required Hardware:

1. Intel Microcomputer Development Systems  
—MDS-800 and Series II  
or
2. Personal Development System

## DOCUMENTATION PACKAGE

FORTRAN-80 Programming Manual

ISIS-II FORTRAN-80 Compiler Operator's Manual

FORTRAN-80 Programming Reference Card

## ORDERING INFORMATION

PART NO.	DESCRIPTION
Model MDS-301	FORTRAN 80 Compiler for Intel Microcomputer Development Systems

Requires Software License.

## SUPPORT

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.





## MICROSOFT\*, INC. MACRO-80 UTILITY SOFTWARE PACKAGE

- Includes the MACRO-80 macro assembler, LINK-80 linking loader, and CREF-80 cross-reference facility
- Supports a complete, Intel-standard MACRO facility, including IRP, IRPC, REPEAT, local variables, and EXITM
- Supports conditional assembly, including testing of assembly pass, symbol definition, and parameters to MACROs
- Code is assembled in relocatable modules for easy manipulation by the LINK-80 linking loader
- Assembly rate of over 1000 lines per minute
- Provides "big computer" assembler features without sacrificing speed or memory space
- Provides a complete set of listing controls
- LINK-80 loads relocatable modules at user-specified locations
- CREF-80 cross-reference facility alphabetizes program variables and shows where each is defined and referenced

The Microsoft Utility Software Package is a complete system for developing assembly language programs, routines, and subroutines. The Utility Software Package includes the MACRO-80 macro assembler, the LINK-80 linking loader, and the CREF-80 cross-reference facility. The CP/M\* version also includes the LIB-80 Library Manager.

The Utility Software Package is supplied with all Microsoft compilers to provide assembly language subroutine support to main programs in the high-level programming languages. The LINK-80 linking loader is used by all Microsoft compilers for linking and loading compiled relocatable modules. Thus, LINK-80 allows the programmer to link together relocatable modules from different Microsoft languages.

---

### FEATURES

#### MACRO-80 Macro Assembler

MACRO-80 incorporates almost all "big computer" assembler features without sacrificing speed or memory space. The assembler supports a complete, Intel-standard macro facility, including IRP, IRPC, REPEAT, local variables, and EXITM. Macro names take precedence over instruction mnemonics and pseudo operations. Nesting of macros is limited only by memory. Code is assembled in relocatable modules that are easily manipulated with the flexible linking loader. Conditional assembly capability is greatly enhanced by an expanded set of conditional pseudo operations that include testing of assembly pass, symbol definition, and parameters to macros. Conditionals may be nested up to 255 levels.

More MACRO-80 features:

- Comment blocks
- Variable input radix from base 2 to base 16
- Octal or hex listings
- INCLUDE statement assembles an alternate source file into the current program
- PRINTX statement for printing assembly or diagnostic messages
- PHASE/DEPHASE statements allow code to reside in one area of memory but execute in another
- Complete set of listing controls



### LINK-80 Linking Loader

With LINK-80, any number of programs may be loaded with one command, relocatable modules may be loaded in user-specified locations, and external references between modules are resolved automatically by the loader. The loader also performs library searches for system subroutines and generates a memory load map showing the locations of the main program and subroutines.

### CREF-80 Cross-Reference Facility

The Cross-Reference Facility that is included with the Utility Software Package supplies a convenient alphabetic list of all program variable names, along with line numbers where they are referenced and defined.

## SPECIFICATIONS

### Operating Environment

MACRO-80 resides in approximately 19K bytes of memory. LINK-80 resides in approximately 14K bytes of memory. CREF-80 requires about 6K bytes. The MACRO-80 Utility Software Package is compatible with the CP/M\* operating system.

### Required Hardware

- Intellec® Microcomputer Development System
- iPDS (Personal Development System)
- minimum of 1 diskette drive

### Required Software

CP/M Operating System or MP/M-II\* Operating System.

### Documentation Package

One copy of each manual is supplied with the software package.

#### Description

Microsoft Utility Software Manual

## ORDERING INFORMATION

Order Code	Description
SD106CPM80F	Microsoft MACRO-80 Utility Software Package, CP/M version (iPDS Format)

### SUPPORT:

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required.  
 \*Microsoft is a trademark of Microsoft, Inc.  
 \*CP/M is a registered trademark of Digital Research, Inc.  
 \*MP/M-II is a trademark of Digital Research, Inc.



## MICROSOFT\*, INC. BASIC-80 INTERPRETER SOFTWARE PACKAGE

- Compatible with other Microsoft BASIC compilers and interpreters
- Sophisticated string handling and structured programming features for applications development
- Direct transfer of BASIC programs to the 8085, 8086 and 8088
- Random and sequential file manipulation where random file record length is user-definable
- Read or write memory location capabilities
- Meets the requirements for the ANSI subset standard for BASIC, and supports many enhancements
- Extensive text editing features built-in
- Automatic line number generation and renumbering
- Supports assembly language subroutine calls
- Trace facilities for easier debugging

BASIC Release 5.0 from Microsoft is an extensive implementation of BASIC. Microsoft BASIC gives users what they want from a BASIC—ease of use plus the features that are comparable to a minicomputer or large mainframe.

BASIC-80 meets the requirements for the ANSI subset standard for BASIC, as set forth in document BSRX3.60-1978. It supports many unique features rarely found in other BASICs.

---

### FEATURES

- Four variable types: Integer (–32768, +32767), String (up to 255 characters), Single-Precision Floating Point (7 digits), Double-Precision Floating Point (16 digits).
- Trace facilities (TRON/TROFF) for easier debugging.
- Error trapping using the ON ERROR GOTO statement.
- PEEK and POKE statements to read or write any memory location.
- Automatic line number generation and renumbering, including reference line numbers.
- Matrices with up to 255 dimensions.
- Boolean operators OR, AND, NOT, XOR, EQV, IMP.
- Formatted output using the PRINT USING facility, including asterisk fill, floating dollar sign, scientific notation, trailing sign, and comma insertion.
- Direct access to I/O ports with the INP and OUT functions.
- Extensive program editing facilities via EDIT command and EDIT mode subcommands.
- Assembly language subroutine calls (up to 10 per program) are supported.
- IF/THEN/ELSE and nested IF/THEN/ELSE constructs.
- Supports variable-length random and sequential disk files with a complete set of file manipulation statements: OPEN, CLOSE, GET, PUT, KILL, NAME, MERGE.



### BASIC-80 Commands, Statements, Functions

AUTO	RENUM	NAME
LIST	WIDTH	SAVE
NULL	CONT	EDIT
TROFF	MERGE	NEW
CLEAR	RUN	TRON
LOAD	DELETE	

### Program Statements

CALL	RANDOMIZE	RETURN
GOSUB	COMMON	WAIT
END	DEF FN	ON GOSUB
GOTO	ERROR	DIM
STOP	POKE	FOR/NEXT/
WHILE/	RESUME	STEP
WEND	SWAP	IF/THEN/
CHAIN	DEFDBL	ELSE
DEF USR	DEFSTR	ON ERROR
LET	DEFSNG	GOTO
REM	DEFINT	OPTION BASE

### Input/Output Statements and Functions

CLOSE	GET	NAME
KILL	POS	PUT
OUT	FIELD	EOF
RESTORE	LSET/RSET	SPC
READ	PRINT	INKEY\$
TAB	USING	INPUT
DATA	LOC	OPEN
LINE	MKIS	CVD
INPUT	MKS\$	CVI
PRINT	MKD\$	CVS
WRITE	LLIST	
LPRINT	LPOS	

### Arithmetic Functions

ABS	SIN	LOG
INT	CDBL	FIX
SGN	CSNG	COS
ATN	CINT	RND
EXP	SQR	TAN

### String Functions

ASC	STR\$	INSTR
LEN	HEX\$	RIGHT\$
STRING\$	OCT\$	MID\$
CHR\$	VAL	SPACE\$
LEFT\$		

### Operators

=	*	XOR
^	<=	NOT
<	+	EQV
>	< >	MOD
-	\	IMP
/	>=	OR
		AND

### Special Functions

ERL	ERR	VARPTR
USR	FRE	PEEK

## SPECIFICATIONS

### Operating Environment

The standard disk version of Microsoft BASIC-80 occupies 24K bytes of memory. Microsoft BASIC-80 Interpreter is compatible with Intel's ISIS operating system or CP/M\* operating system.

### Required Hardware

Intellec Microcomputer Development System  
—iPDS (Personal Development System)  
—minimum of 1 diskette drive

### Required Software

ISIS Operating System or CP/M Operating System.

### Documentation Package

One copy of each manual is supplied with the software package.

#### Description

BASIC-80 Reference Manual  
BASIC Reference Book



## **ORDERING INFORMATION**

<b>Order Code</b>	<b>Description</b>
SD102CPM80F	Microsoft BASIC-80 Interpreter Software Package, CP/M version (Double-Sided, Double Density 5¼" Floppy) iPDS format
SD102ISS80F	Microsoft BASIC-80 Interpreter Software Package, ISIS version (Double-Sided, Double Density 5¼" Floppy) iPDS format

---

## **SUPPORT**

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

---

An Intel Software License required.

\*Microsoft is a trademark of Microsoft, Inc.

\*CP/M is a registered trademark of Digital Research, Inc.

\*MP/M-II is a trademark of Digital Research, Inc.



## MICROSOFT\*, INC. PASCAL-80 SOFTWARE PACKAGE

- **Native code compiler with language extensions designed for system software implementation**
- **Meets current ISO draft standard**
- **Fully portable, with machine-independent front end**
- **Can replace assembly language for most system software programming tasks**
- **Assembly language routines callable from PASCAL programs**
- **Global optimizer produces compact, fast compiled code**
- **Three levels of implementation—standard, extended and system—to conform to different levels of standardization and levels of machine interface**

A critical need is emerging for more and more software "tools"—compilers, interpreters, operating systems, database managers, and dedicated application programs. The demand for system software with a short development time and a long lifespan is rapidly making assembly language impractical. As a result, the trend in system software is toward a new "tool maker"—a system-oriented language.

Microsoft Pascal-80 is a high-level language compiler specifically designed for microprocessor system software implementation. The language is ISO-standard Pascal, with the addition of many system-oriented extensions. The compiler includes a global optimizer and modular code generator identical to other Microsoft Pascal products.

With Pascal-80, system software is highly readable, modular, and transportable. Plus, Pascal's clean block structure and procedure orientation make system programming more efficient than with assembly language.

---

### FEATURES

Pascal-80 was designed to generate state-of-the-art compact system software. Many extensions have been made to the language that not only adapt it to system programming, but make it easier to use in any application.

- Expanded string support with variable-size LSTRING type
- UNITS and USES interfaces for clean separate compilation
- Machine-oriented WORD type and operators
- Dynamic and conformant arrays using SUPER array types
- Attributes for variables and procedures
- Machine address types and operators

All the enhancements to Pascal-80 are natural extensions of the existing language. Care has been taken to maintain the structured nature of Pascal while assimilating new features. Many low-level escapes have also been provided, such as direct access to memory locations, calls to assembly language sub-routines, and a RETYPE function.

### THE PASCAL-80 COMPILER DESCRIPTION

The compiler operates in three phases. The front-end phase translates the source into an intermediate form and does all error checking and listing generation. The global optimizer phase analyzes the intermediate code and does constant folding, common subexpression elimination, strength reduction, and other optimizations. The code generator phase translates the intermediate code into relocatable object code and does register allocation and peephole optimizations. All phases are written in Pascal.

The runtime system handles program initialization and termination, runtime error reporting, floating point arithmetic, string handling, set operations, dynamic variable allocation, input/output interfacing to the operating system, and low-level utility routines. Pascal-80 is designed to interface easily to operating system and floating point packages.

Pascal-80 uses Microsoft's Linking Loader which allows independent code and data segment location,

library searching, global memory maps, and combination of Pascal-80 output with the output of Microsoft's other compilers and assembler.

## Language Description

The Pascal-80 language is organized into three levels (Standard, Extended, and System) based on the degree of portability provided. Standard level includes all features in the pending ISO standard DP7185. Standard level also contains the meta-language, which controls error checking, listing format, and other options. Programs using only Standard level are portable across all Pascals that conform to the standard.

Microsoft's extensions to Pascal appear in the Extended level and the System level. The Extended level contains high-level features that are natural to Pascal and are machine independent. Programs using only Standard and Extended levels are portable across Microsoft Pascal target machines. The System level contains low-level extensions that provide an escape from Pascal restrictions or are machine dependent.

## Standard Level Features

### LISTING INFORMATION

—Object listing shows generated code, with line numbers.

### METALANGUAGE DIRECTIVES

—7 directives control specific runtime error checks.

## Extended Level Features

### DATA TYPES AND MODES

- Numeric constants in hexadecimal, octal, or binary
- BYTE/WORD types for 8/16-bit unsigned operations
- SUPER array types permit passing any size array to a procedure or allocating any size array on the heap
- LSTRING type provides variable-length array of characters; current length access with lstring.LEN
- STRING and LSTRING predeclared super array types
- CONST parameter type passes long constant by reference
- Functions can return arrays, records, or sets

—Attributes can be given to variables and procedures:

Variables: STATIC READONLY PORT  
Procedures: INTERRUPT PURE  
Either: PUBLIC EXTERN ORIGIN .

## OPERATORS AND INTRINSICS

- Bitwise AND, OR, and NOT on WORDS and INTEGERS
- String constants can be concatenated with "\*"
- String-oriented intrinsics:
  - STRING/LSTRING: POSITN SCANEQ SCANNE
  - LSTRING only: CONCAT INSERT DELETE
- Standard library includes 19 more routines
- FORTRAN library includes 17 more REAL functions
- Extended intrinsic procedures and functions:
  - LOWER, UPPER get bounds of array, set, subrange
  - ABORT invokes runtime error handler
  - SIZEOF returns size of variable in bytes

## CONTROL FLOW AND STRUCTURE

- CONST, TYPE, VAR, VALUE sections in any order
- MODULE source file for separate compilation

## INPUT/OUTPUT AND FILES

- READ enumerated, pointer, BOOLEAN, STRING, LSTRING
- WRITE enumerated, pointer, LSTRING
- READ and WRITE for hexadecimal, octal and binary
- Negative field width justifies left instead of right
- Temporary files, file name created automatically
- ASSIGN and CLOSE procedures
- FILEMODES type, access with file.MODE
- Error trapping, access with file.TRAP and file.ERRS

## SYSTEM LEVEL FEATURES

- Record types can give explicit byte offset to fields
- All file types identical to special FCBFQQ record type
- System intrinsic byte movers: MOVEL MOVER FILLC
- Address types permit low-level machine access
  - ADR OF type declares address (all ADRs compatible)
  - ADR prefix operator gets variable or constant address
  - adr. R gives WORD address value, adr gives data at address



### Utility Software Package

The PASCAL-80 package includes the Microsoft Utility Software Package. The Utility Software Package includes the MACRO-80 macro assembler, the

LINK-80 linking loader, and the CREF-80 Cross-Reference Facility. Refer to the description of the Microsoft Utility Software Package for full details.

## SPECIFICATIONS

### Operating Environment

The Pascal compiler running under CP/M resides in approximately 40K bytes of memory and requires an additional 8K byte minimum (or 48K bytes minimum) plus symbol table space when compiling.

### Required Software

CP/M\* Operating System or MP/M-II\* Operating System.

### Required Hardware

- Intellec Microcomputer Development System
- iPDS (Personal Development System)
- minimum of 1 diskette drive

### Documentation Package

One copy of each manual is provided with the software package.

#### Description

- PASCAL-80 Reference Manual
- Microsoft Utility Software Manual

## ORDERING INFORMATION

Order Code	Description
SD105CPM80F	Microsoft Pascal-80 Software Package, CP/M version (iPDS Format)

## SUPPORT

Intel offers several levels of support for this product, depending on the system configuration in which it is used. Please consult the price list for a detailed description of the support options available.

An Intel Software License required  
 \*Microsoft is a trademark of Microsoft, Inc.  
 \*CP/M is a registered trademark of Digital Research, Inc.  
 \*MP/M-II is a trademark of Digital Research, Inc





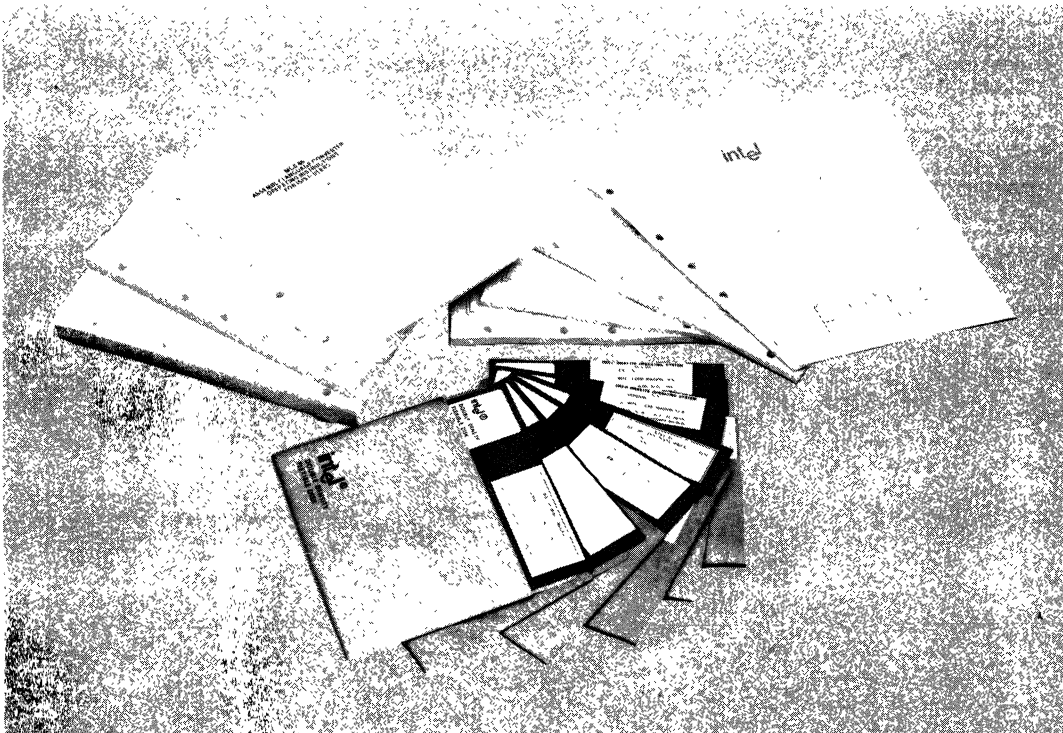
## **iAPX 86,88 SOFTWARE DEVELOPMENT PACKAGES FOR SERIES II/PDS**

- **PL/M 86/88 High Level Programming Language**
- **ASM 86/88 Macro Assembler for iAPX 86,88 Assembly Language Programming**
- **LINK 86/88 and LOC 86/88 Linkage and Relocation Utilities**
- **CONV 86/88 Converter for Conversion of 8080/8085 Assembly Language Source Code to iAPX 86, 88 Assembly Language Source Code**
- **OH 86/88 Object-to-Hexadecimal Converter**
- **LIB 86/88 Library Manager**

The iAPX 86,88 Software Development Packages for Series II provide a set of software development tools for the iAPX 86/88 CPUs and the iSBC 86/12A single board computer. The packages operate under the ISIS-II operating system on Intel Microcomputer Development Systems—Model 800, Series II or the Personal Development System (PDS)—thus minimizing requirements for additional hardware or training for Intel Microcomputer Development System users.

These packages permit 8080/8085 users to efficiently upgrade existing programs into iAPX 86/88 code from either 8080/8085 assembly language source code or PL/M 80 source code.

For the new Intel Microcomputer Development System user, the packages operating on a PDS or an Intellec Series II, such as a Model 235, provide total iAPX 86,88 software development capability.



## **PL/M 86/88 COMPILER FOR SERIES II/PDS**

- **Language is Upward Compatible from PL/M 80, Assuring MCS<sup>®</sup>-80/85 Design Portability**
- **Supports 16-bit Signed Integer and 32-bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-to-Learn, Block-Structured Language Encourages Program Modularity**
- **Produces Relocatable Object Code Which is Linkable to All Other 8086 Object Modules**
- **Supports Full Extended Addressing Features of the iAPX 86/10 and 88/10 Microprocessors (Up to 1 Mbyte)**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**

Like its counterpart for MCS-80/85 program development, PL/M 86/88 is an advanced, structured high-level programming language. The PL/M 86/88 compiler was created specifically for performing software development for the Intel iAPX 86,88 Microprocessors.

PL/M 86/88 has significant new capabilities over PL/M 80 that take advantage of the new facilities provided by the iAPX 86,88 microsystem, yet the PL/M 86/88 language remains compatible with PL/M 80.

With the exception of hardware-dependent modules, such as interrupt handlers, PL/M 80 applications may be recompiled with PL/M 86/88 with little need for modification. PL/M 86/88, like PL/M 80, is easy to learn, facilitates rapid program development, and reduces program maintenance costs.

PL/M is a powerful, structured, high-level system implementation language in which program statements can naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The PL/M 86/88 compiler efficiently converts free-form PL/M language statements into equivalent 86/88 machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user

---

### **FEATURES**

Major features of the Intel PL/M 86/88 compiler and programming language include:

#### **Block Structure**

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible by clearly defining the scope of user variables (local to a private procedure, global to a public module, for example).

The use of procedures to break down a large problem is paramount to productive software development. The PL/M 86/88 implementation of a block

structure allows the use of REENTRANT which is especially useful in system design.

#### **Language Compatibility**

PL/M 86/88 object modules are compatible with object modules generated by all other 86/88 translators. This means that PL/M programs may be linked to programs written in any other 86/88 language.

Object modules are compatible with ICE-88 and ICE-86 units; DEBUG compiler control provides the In-Circuit Emulators with symbolic debugging capabilities.

PL/M 86/88 Language is upward-compatible with PL/M 80, so that application programs may be easily ported to run on the iAPX 86 or 88.

## Supports Five Data Types

PL/M makes use of five data types for various applications. These data types range from one to four bytes, and facilitate various arithmetic, logic, and addressing functions:

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- Integer: 16-bit signed number
- Real: 32-bit floating point number
- Pointer: 16-bit or 32-bit memory address indicator

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

## Two Data Structuring Facilities

In addition to the five data types and based variables, PL/M supports two data structuring facilities. These add flexibility to the referencing of data stored in large groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of Each: Arrays of structures or structures of arrays

## 8087 Numerics Support

PL/M programs that use 32-bit REAL data may be executed using the Numeric Data Processor for improved performance. All floating-point operations supported by PL/M may be executed on the 8087 NDP, or the 8087 Emulator (a software module) provided with the package. Determination of use of the chip or emulator takes place at link-time, allowing compilations to be run-time independent.

## Built-In String Handling Facilities

The PL/M 86/88 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

## Interrupt Handling

PL/M has the facility for generating interrupts to the iAPX 86 or 88 via software. A procedure may be defined with the INTERRUPT attribute, and the compiler will automatically initialize an interrupt vector at the appropriate memory location. The compiler will also generate code to save and restore the processor status, for execution of the user-defined interrupt handler routine. The procedure SET\$INTERRUPT, the function returning an INTERRUPT\$PTR, and the PL/M statement CAUSE\$INTERRUPT all add flexibility to user programs involving interrupt handling.

## Segmentation Control

The PL/M 86/88 compiler takes full advantage of program addressing with the SMALL, COMPACT, MEDIUM, and LARGE segmentation controls. Programs with less than 64KB total code space can exploit the most efficient memory addressing schemes, which lowers total memory requirements. Larger programs can exploit the flexibility of extended one-megabyte addressing.

## Code Optimization

The PL/M 86/88 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions.
- "Strength reductions" (such as a shift left rather than multiply by 2); and elimination of common sub-expressions within the same block.
- Machine code optimizations; elimination of superfluous branches; re-use of duplicate code; removal of unreadable code.
- Byte comparisons (rather than 20-bit address calculations) for pointer variables; optimization of based-variable operations.

## Compiler Controls

The PL/M 86/88 compiler offers more than 25 controls that facilitate such features as:

- Conditional compilation
- Intra- and Inter-module cross reference
- Corresponding assembly language code in the listing file
- Setting overflow conditions for run-time handling

## **BENEFITS**

PL/M 86/88 is designed to be an efficient, cost-effective solution to the special requirements of iAPX 86 or 88 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

### **Low Learning Effort**

PL/M 86/88 is easy to learn and to use, even for the novice programmer.

### **Earlier Project Completion**

Critical projects are completed much earlier than otherwise possible because PL/M 86/88, a structured high-level language, increases programmer productivity.

### **Lower Development Cost**

Increases in programmer productivity translate immediately into lower software development costs

because less programming resources are required for a given programmed function.

### **Increased Reliability**

PL/M 86/88 is designed to aid in the development of reliable software (PL/M 86/88 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

### **Easier Enhancements and Maintenance**

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

---

## **iAPX 86,88 MACRO ASSEMBLER FOR SERIES II/PDS**

- **Powerful and Flexible Text Macro Facility with Three Macro Listing Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **“Strongly Typed” Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as “STRUCTURES” and “RECORDs”**
- **Over 120 Detailed and Fully Documented Error Messages**
- **Produces Relocatable and Linkable Object Code**

ASM 86/88 is the “high-level” macro assembler for the iAPX 86,88 assembly language. ASM 86/88 translates symbolic 86/10, 88/10 assembly language mnemonics into 86/10, 88/10 relocatable object code.

ASM 86/88 should be used where maximum code efficiency and hardware control is needed. The iAPX 86,88 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible 86/10, 88/10 machine instructions. ASM 86/88 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

ASM 86/88 offers many features normally found only in high-level languages. The iAPX 86,88 assembly language is strongly typed. The assembler performs extensive checks on the usage of variables and labels. The assembler uses the attributes which are derived explicitly when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be detected when the program is assembled, long before it is being debugged on hardware.

## FEATURES

Major features of the Intel iAPX 86,88 assembler and assembly language include:

### Powerful and Flexible Text Macro Facility

- Macro calls may appear anywhere
- Allows user to define the syntax of each macro
- Built-in functions
  - conditional assembly (IF-THEN-ELSE, WHILE)
  - repetition (REPEAT)
  - string processing functions (MATCH)
  - support of assembly time I/O to console (IN, OUT)
- Three Macro Listing Options include a GEN mode which provides a complete trace of all macro calls and expansions

### High-Level Data Structuring Capability

- STRUCTURES: Defined to be a template and then used to allocate storage. The familiar dot notation may be used to form instruction addresses with structure fields.
- ARRAYS: Indexed list of same type data elements.
- RECORDS: Allows bit-templates to be defined and used as instruction operands and/or to allocate storage.

### Fully Supports iAPX 86,88 Addressing Modes

- Provides for complex address expressions involving base and indexing registers and (structure) field offsets.
- Powerful EQU facility allows complicated expressions to be named and the name can be used as a synonym for the expression throughout the module.

### Powerful STRING MANIPULATION INSTRUCTIONS

- Permit direct transfers to or from memory or the accumulator.
- Can be prefixed with a repeat operator for repetitive execution with a count-down and a condition test.

### Over 120 Detailed Error Messages

- Appear both in regular list file and error print file.
- User documentation fully explains the occurrence of each error and suggests a method to correct it.

### Support for ICE-86 Emulation and Symbolic Debugging

- Debug options for inclusion of symbol table in object modules for In-Circuit Emulation with symbolic debugging.

### Generates Relocatable and Linkable Object Code—Fully Compatible with LINK 86/88, LOC 86/88 and LIB 86/88

- Permits ASM 86/88 programs to be developed and debugged in small modules. These modules can be easily linked with other ASM 86/88 or PL/M 86/88 object modules and/or library routines to form a complete application system.

## BENEFITS

The iAPX 86,88 macro assembler allows the extensive capabilities of the 86/88 CPU's to be fully exploited. In any application, time and space critical routines can be effectively written in ASM 86/88. The 86,88 assembler outputs relocatable and linkable object modules. These object modules may be easily combined with object modules written in PL/M 86/88—Intel's structured, high-level programming language. ASM 86/88 compliments PL/M 86/88 as the programmer may choose to write each module in the language most appropriate to the task and then combine the modules into the complete applications program using the iAPX 86,88 relocation and linkage utilities.

## **CONV 86/88 MCS<sup>®</sup>-80/85 to iAPX 86,88 ASSEMBLY LANGUAGE CONVERTER UTILITY PROGRAM**

- **Translates 8080/8085 Assembly Language Source Code to iAPX 86,88 Assembly Language Source Code**
- **Provides a Fast and Accurate Means to Convert 8080/8085 Programs to the iAPX 86/88 Facilitating Program Portability**
- **Automatically Generates Proper ASM 86/88 Directives to Set Up a "Virtual 8080" Environment that is Compatible with PL/M 86/88**

In support of Intel's commitment to software portability, CONV 86/88 is offered as a tool to move 8080/8085 programs to the iAPX 86/88. A comprehensive manual, "MCS-86 Assembly Language Converter Operating Instructions for ISIS-II Users," covers the entire conversion process. Detailed methodology of the conversion process is fully described therein.

- CONV 86/88 will accept as input an error-free 8080/8085 assembly-language source file and optional controls, and produce as output, optional PRINT and OUTPUT files.
- The PRINT file is a formatted copy of the 8080/8085 source and the 86/88 source file with embedded caution messages.
- The OUTPUT file is an 86/88 source file.
- CONV 86/88 issues a caution message when it detects a potential problem in the converted 86/88 code.
- A transliteration of the 8080/8085 programs occurs, with each 8080/8085 construct mapped to its exact 86/88 counterpart:

Registers  
Condition flags  
Instruction  
Operands  
Assembler directives  
Assembler control lines  
Macros

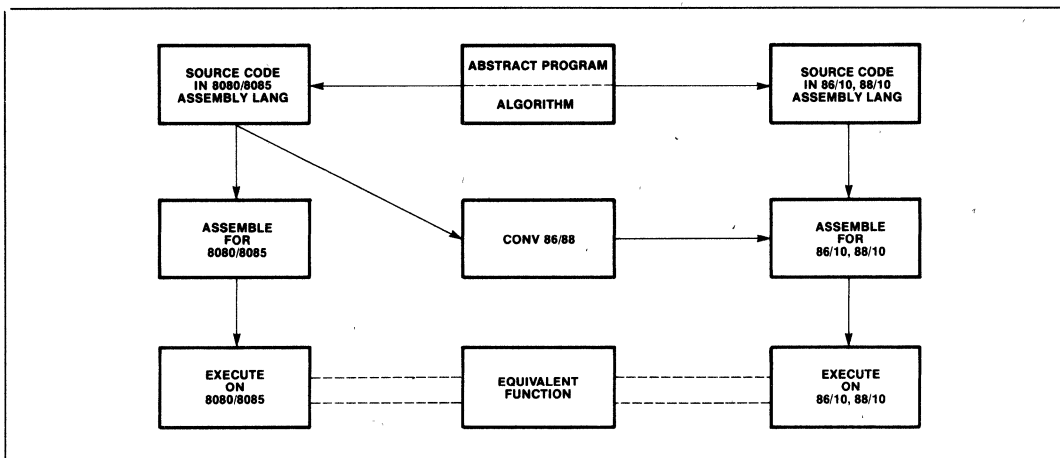
Because CONV 86/88 is a transliteration process, there is the possibility of as much as a 15%–20% code expansion over the 8080/8085 code. For compactness and efficiency it is recommended that critical portions of programs be re-coded in iAPX 86,88 assembly language.

Also, as a consequence of the transliteration, some manual editing may be required for converting instruction sequences dependent on:

- instruction length, timing, or encoding
- interrupt processing\*
- PL/M parameter passing conventions\*

\*Mechanical editing procedures for these are suggested in the converter manual.

The accompanying figure illustrates the flow of the conversion process. Initially, the abstract program may be represented in 8080/8085 or iAPX 86,88 assembly language to execute on that respective target machine. The conversion process is porting a source destined for the 8080/8085 to the 86/88 via CONV 86/88.



**Figure 1. Porting 8080/8085 Source Code to the iAPX 86/10 and 88/10**

## LINK 86/88

- Automatic Combination of Separately Compiled or Assembled iAPX 86, 88 Programs Into a Relocatable Module
- Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References
- Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively
- Automatic Generation of a Summary Map Giving Results of the LINK 86/88 Process
- Abbreviated Control Syntax
- Relocatable Modules may be Merged into a Single Module Suitable for Inclusion in a Library
- Supports "Incremental" Linking
- Supports Type Checking of Public and External Symbols

LINK 86/88 combines object modules specified in the LINK 86/88 input list into a single output module. LINK 86/88 combines segments from the input modules according to the order in which the modules are listed.

LINK 86/88 will accept libraries and object modules built from PL/M 86/88, ASM 86/88, or any other translator generating Intel's iAPX 86/88 Relocatable Object Modules.

Support for incremental linking is provided since an output module produced by LINK 86/88 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

LINK 86/88 supports type checking of PUBLIC and EXTERNAL symbols reporting an error if their types are not consistent.

LINK 86/88 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the LINK 86/88 process and to control the content of the output module.

LINK 86/88 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using LOC 86/88 and enter final testing with much of the work accomplished.

## **LIB 86/88**

- **LIB 86/88 is a Library Manager Program which Allows You to:  
Create Specially Formatted Files to Contain Libraries of Object Modules  
Maintain These Libraries by Adding or Deleting Modules  
Print a Listing of the Modules and Public Symbols in a Library File**
- **Libraries Can be Used as Input to LINK 86/88 Which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program LIB 86/88 creates and maintains files containing object modules. The operation of LIB 86/88 is controlled by commands to indicate which operation LIB 86/88 is to perform. The commands are:

CREATE: creates an empty library file  
ADD: adds object modules to a library file  
DELETE: deletes modules from a library file  
LIST: lists the module directory of library files  
EXIT: terminates the LIB 86 program and returns control to ISIS-II

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

---

## **LOC 86/88**

- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Lengths, and Debug Symbols and their Addresses**
- **Automatic and Independent Relocation of Segments. Segments May Be Relocated to Best Match Users Memory Configuration**
- **Extensive Capability to Manipulate the Order and Placement of Segments in iAPX 86/88 Memory**
- **Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**
- **Abbreviated Control Syntax**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

LOC 86/88 converts relative addresses in an input module to absolute addresses. LOC 86/88 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

LOC 86/88 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the LOC 86/88 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program on the Intel development system and then simply relocate the object code to suit your application.



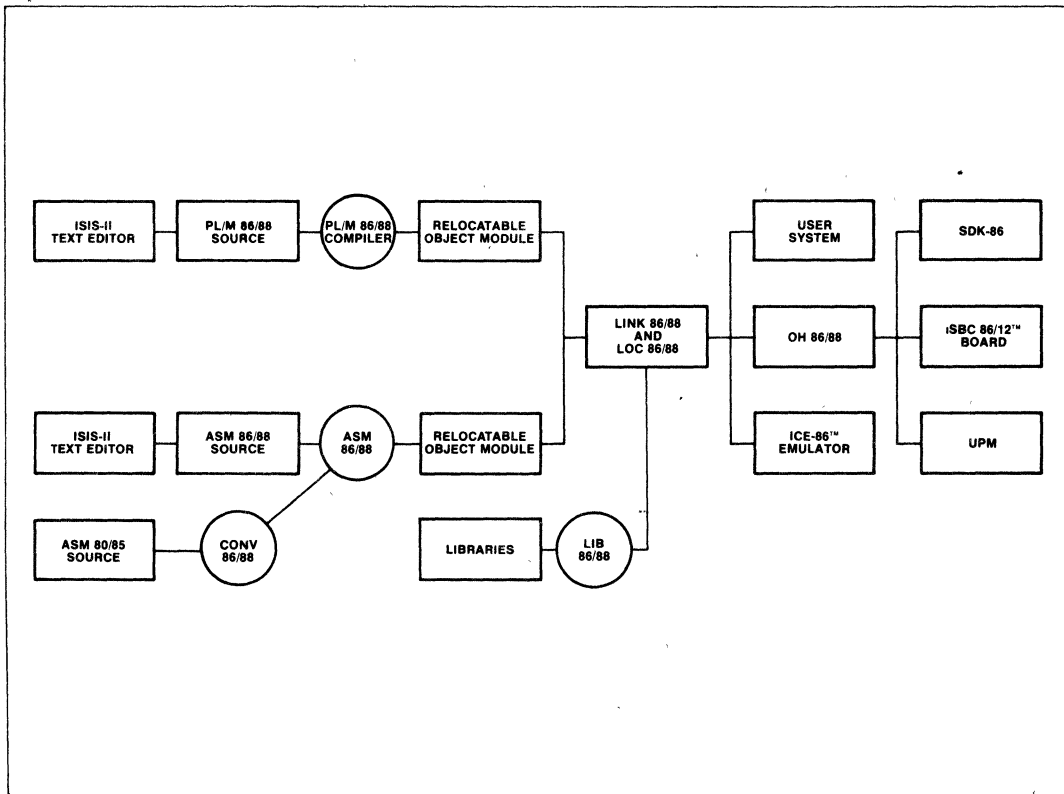
## OH 86/88

- Converts an iAPX 86/88 Absolute Object Module to Symbolic Hexadecimal Format

■ Facilitates Preparing a File for Later Loading by a Symbolic Hexadecimal Loader, such as the iSBC™ Monitor SDK-86 Loader, or Universal PROM Mapper
- Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging

The OH 86/88 utility converts an 86/88 absolute object module to the hexadecimal format. This conversion may be necessary to format a module for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or Universal PROM Mapper. The conversion may also be made to put the module in a more readable format than can be displayed or printed.

The module to be converted must be in absolute format; the output from LOC 86/88 is in absolute format.



**Figure 2. iAPX 86,88 Software Development Cycle**

**SPECIFICATIONS**

**Operating Environment**

Intel Microcomputer Development Systems  
Intel Personal Development System

**Documentation**

*PL/M-86 Programming Manual*

*ISIS-II PL/M-86 Compiler Operator's Manual*

*MCS-86 User's Manual*

*MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users*

*MCS-86 Macro Assembly Language Reference Manual*

*MCS-86 Macro Assembler Operating Instructions for ISIS-II Users*

*MCS-86 Assembly Language Converter Operating Instructions for ISIS-II Users*

*Universal PROM Programmer User's Manual*

**ORDERING INFORMATION**

**iAPX 86,88 Software Development Packages for Series II:**

<b>Documentation</b>	<b>Part No.</b>	<b>Description</b>
<i>PL/M-86 Programming Manual</i>		
<i>ISIS-II PL/M-86 Compiler Operator's Manual</i>	MDS-308*	Assembler and Utilities Package
<i>MCS-86 User's Manual</i>		
<i>MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users</i>		
<i>MCS-86 Macro Assembly Language Reference Manual</i>	MDS-309*	PL/M compiler and Utilities Package
<i>MCS-86 Macro Assembler Operating Instructions for ISIS-II Users</i>		
<i>MCS-86 Assembly Language Converter Operating Instructions for ISIS-II Users</i>	MDS-311*	PL/M compiler, Assembler, and Utilities Package
<i>Universal PROM Programmer User's Manual</i>		All Packages Require Software Licenses

---

**SUPPORT:**

Hotline Telephone Support, Software Performance Reports (SPR), Software Updates, Technical Reports, Monthly Newsletters are available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.



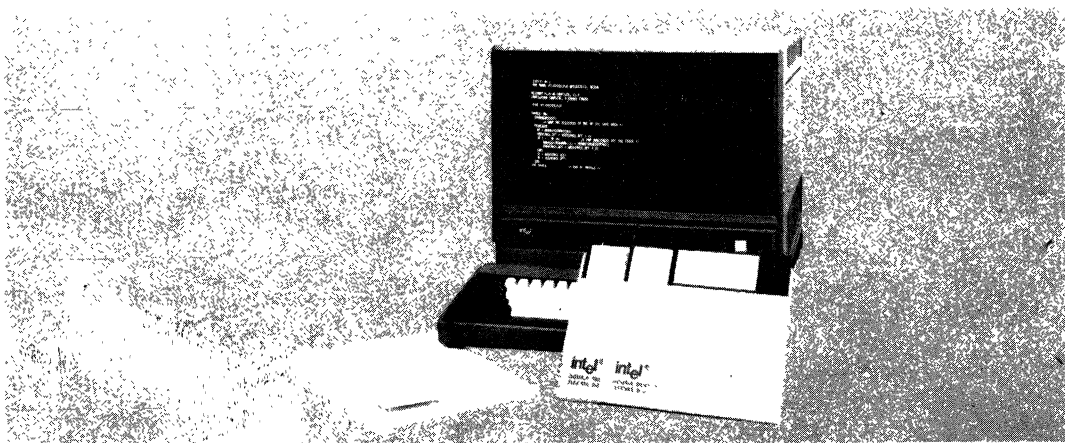
## PL/M 86/88/186/188 Software Package

- **Systems Programming Language for the iAPX 86/88/186/188 Processors**
- **Language Is Upward Compatible from PL/M 80, Assuring MCS<sup>®</sup>-80/85 Design Portability**
- **Advanced Structured System Implementation Language for Algorithm Development**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-to-Learn Block-Structured Language Encourages Program Modularity**
- **Improved Compiler Performance Now Supports More User Symbols and Faster Compilation Speeds**
- **Produces Relocatable Object Code Which Is Linkable to All Other 8086 Object Modules**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Resident on iAPX 86 Intel Microcomputer Development Systems**

PL/M 86 is an advanced, structured, high-level systems programming language. The PL/M 86 compiler was created specifically for performing software development for the Intel 8086, 8088, 80186 and 80188 Microprocessors. PL/M was designed so that program statements naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The PL/M 86 compiler efficiently converts free-form PL/M language statements into machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-up maintenance costs for the user.



NOTE: The Intel<sup>®</sup> Development System pictured here is not included with the PL/M 86/88 Software package but merely depicts a language in its operating environment. The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products. BXP, CREDIT, i, iCE, iCS, iM, Insite, Intel, INTEL, Intelevison, Intelink, Inteltec, iMMX, iOSP, iPDS, iRMX, iSBC, iSBX, Library Manager, MCS, MULTIMODULE, Megachassis, Micromainframe, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RUPi, RMX/80, System 2000, UPI, and the combination iCS, iRMX, iSBC, iSBX, iCE, i<sup>2</sup>ICE, MCS, or UPI and numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel product. No Other Patent Licenses are implied. © INTEL CORPORATION, 1983

MAY 1983

ORDER NUMBER:210689-002

## FEATURES

Major features of the Intel PL/M 86 compiler and programming language include:

### Block Structure

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible, by clearly defining the scope of user variables (local to a private procedure).

The use of procedures to break down a large problem is paramount to productive software development. The PL/M 86 implementation of a block structure allows the use of REENTRANT (recursive) procedures, which are especially useful in system design.

### Language Compatibility

PL/M 86 object modules are compatible with object modules generated by all other iAPX 86 translators. This means that PL/M programs may be linked to programs written in any other iAPX 86 language.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with symbolic debugging capabilities.

PL/M 86 Language is upward compatible with PL/M 80, so that application programs may be easily ported to run on the iAPX 86.

### Supports Seven Data Types

PL/M makes use of seven data types for various applications. These data types range from one to four bytes, and facilitate various arithmetic, logic, and addressing functions:

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- DWORD: 32-bit unsigned number
- Integer: 16-bit signed number
- Read: 32-bit floating point number
- Pointer: 16-bit or 32-bit memory address indicator
- Selector: 16-bit base portion of a pointer

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

### Two Data Structuring Facilities

In addition to the five data types and based variables, PL/M supports two data structuring facilities. These help the user to organize data into logical groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of Each: Arrays of structures or structures of arrays

### 8087 Numerics Support

PL/M programs that use 32-bit REAL data may be executed using the Numeric Data Processor for improved performance. All floating-point operations supported by PL/M may be executed on the iAPX 86/20 or 88/20 NDP, or the 8087 Emulator (a software module) provided with the package. Determination of use of the chip or Emulator takes place at linktime, allowing compilations to be run-time independent.

### Built-In String Handling Facilities

The PL/M 86 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

### Interrupt Handling

PL/M has the facility for handling interrupts. A procedure may be defined with the INTERRUPT attribute, and the compiler will automatically initialize an interrupt vector at the appropriate memory location. The compiler will also generate code to save and restore the processor status, for execution of the user-defined interrupt handler routine. The procedure SET\$INTERRUPT, the function returning an INTERRUPT\$PTR, and the PL/M statement CAUSE\$INTERRUPT all add flexibility to user programs involving interrupt and handling.

### Compiler Controls

Including several that have been mentioned, the PL/M 86 compiler offers more than 25 controls that facilitate such features as:

- Conditional compilation
- Including additional PL/M source files from disk
- Corresponding assembly language code in the listing file
- Setting overflow conditions for run-time handling

### Segmentation Control

The PL/M 86 compiler takes full advantage of program addressing with the SMALL, COMPACT, MEDIUM, and LARGE segmentation controls. Programs with less than 64KB total code space can exploit the most efficient memory addressing schemes, which lowers total memory requirements. Larger programs can exploit the flexibility of extended one-megabyte addressing.

### Code Optimization

The PL/M 86 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or “folding” of constant expressions; and short-circuit evaluation of Boolean expressions
- “Strength reductions” (such as a shift left rather than multiply by 2); and elimination of common sub-expressions within the same block
- Machine code optimizations; elimination of superfluous branches; re-use of duplicate code; removal of unreachable code
- Byte comparisons (rather than 20-bit address calculations) for pointer variables; optimization of based-variable operations

### Error Checking

The PL/M 86 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed set of programming and compilation errors is provided by the compiler.

```

M DO, /* Beginning of module */
SORTPROC PROCEDURE (PTR, COUNT, RECSIZE, KEYINDEX) PUBLIC;
DECLARE PTR POINTER, (COUNT, RECSIZE, KEYINDEX) INTEGER,
/* Parameters
PTR is pointer to first record
COUNT is number of records to be sorted
RECSIZE is number of bytes in each record—max is 128
KEYINDEX is byte position within each record of a BYTE scalar
to be used as sort key */
DECLARE (RECORD BASED PTR) (1) BYTE,
CURRENT (128) BYTE,
(I, J) INTEGER,
SORT DO J=1 TO COUNT-1,
CALL MOVB(@RECORD(J*RECSIZE), @CURRENT, RECSIZE),
I=J,
FIND DO WHILE I > 0
AND RECORD((I-1)*RECSIZE - KEYINDEX)
-CURRENT(KEYINDEX),
CALL MOVB(@RECORD((I-1)*RECSIZE),
@RECORD(I*RECSIZE),
RECSIZE),
I=I-1,
END FIND,
CALL (MOVB) @CURRENT, @RECORD(I*RECSIZE), RECSIZE),
END SORT,
END SORTPROC,
END M, /*End of module*/

```

PUBLIC and EXTERNAL attributes promote program modularity

“Based” Variables allow manipulation of external data by passing the base of the data structure (a pointer). This minimizes the STACK space used for parameter passing, and the execution time to perform many STACK operations

The “AT” operator returns the address of a variable, instead of its contents. This is very useful in passing pointers for based variables.

One of several PL/M built-in procedures for string manipulation

Figure 1. Sample PL/M 86 Program.

**BENEFITS**

PL/M 86 is designed to be an efficient, cost-effective solution to the special requirements of iAPX 86 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

**Cost-Effective Alternative to Assembly Language**

PL/M 86 programs are code efficient. PL/M 86 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the iAPX 86 architecture. Consequently, for the development of systems software, PL/M 86 is the cost-effective alternative to assembly language programming.

**Low Learning Effort**

PL/M is easy to learn and to use, even for the novice programmer.

**Earlier Project Completion**

Critical projects are completed much earlier than otherwise possible because PL/M 86, a structured high-level language, increases programmer productivity.

**Lower Development Cost**

Increases in programmer productivity translate immediately into lower software development costs because fewer programming resources are required for a given programmed function.

**Increased Reliability**

PL/M 86 is designed to aid in the development of reliable software (PL/M 86 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

**Easier Enhancements and Maintenance**

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

---

**SPECIFICATIONS****Operating Environment****REQUIRED HARDWARE:**

Intel Microcomputer Development Systems (Series III/Series IV)

**Documentation Package**

*PL/M-86 User's Guide for 8086-based Development Systems (121636)*

**SUPPORT:**

Hotline Telephone Support, Software Performance Reporting (SPR), Software Updates, Technical Reports, Monthly Newsletter available.

---

**ORDERING INFORMATION**

<b>Part Number</b>	<b>Description</b>
MDS-313*	PL/M 86 Software Package

Requires Software License

\*MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.



## PASCAL 86/88 SOFTWARE PACKAGE

- Resident on iAPX 86 Based Intel Microcomputer Development Systems
- Object Compatible and Linkable with PL/M 86/88, ASM 86/88 and FORTRAN 86/88
- ICE™ Symbolic Debugging Fully Supported
- PSCOPE Source Level Debugging Fully Supported
- Implements REALMATH for Consistent and Reliable Results
- Unlimited User Program Symbols
- Supports iAPX86/20, 88/20 Numeric Data Processors
- Strict Implementation of ISO Standard Pascal
- Useful Extensions Essential for Microcomputer Applications
- Separate Compilation with Type-Checking Enforced Between Pascal Modules
- Compiler Option to Support Full Run-Time Range-Checking

PASCAL 86/88 conforms to and implements the ISO Draft Proposed Pascal standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The PASCAL 86/88 compiler runs on Series III and Series IV Microcomputer Development Systems. A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs as an alternate to the development system environment. Program modules compiled under PASCAL 86/88 are compatible and linkable with modules written in PL/M 86/88, ASM 86/88 or FORTRAN 86/88. With a complete family of compatible programming languages for the iAPX 86, 88, 186, 188 one can implement each module in the language most appropriate to the task at hand.

PASCAL 86/88 object modules contain symbol and type information for program debugging using ICE™ emulators and PSCOPE source language debugger. For final production version, the compiler can remove this extra information and code.



## FEATURES

Includes all the language features of Jensen & Wirth Pascal as defined in the ISO Draft Proposed Pascal Standard.

Supports required extensions for microcomputer applications.

- Interrupt handling
- Direct port I/O

Separate compilation extensions allow:

- Modular decomposition of large programs
- Linkage with other Pascal modules as well as PL/M 86/88/186/188, ASM 86/88/186/188 and FORTRAN 86/88.
- Enforcement of type-checking at LINK-time

Supports numerous compiler options to control the compilation process, to INCLUDE files, flag non-standard Pascal statements and others to control program listings and object modules.

Utilizes the IEEE standard for Floating-Point Arithmetic (the Intel REALMATH standard) for arithmetic operations.

Well-defined and documented run-time operating system interfaces allow the user to execute the applications under user-designed operating systems.

Predefined type extensions allow:

- Create precision in read, integer, and unsigned calculations.
- Means to check 8087 errors
- Circumvention of rigid type checking on calls to non-Pascal routines

## BENEFITS

Provides a standard Pascal for iAPX 86, 88, 186, 188 based applications.

- Pascal has gained wide acceptance as the portable application language for microcomputer applications
- It is being taught in many colleges and universities around the world
- It is easy to learn, originally intended as a vehicle for teaching computer programming
- Improves maintainability: Type mechanism is both strictly enforced and user extendable
- Few machine specific language constructs

Strict implementation of the proposed ISO standard for Pascal aids portability of application programs. A compile time option checks conformance to the standard making it easy to write conforming programs.

PASCAL 86/88 extensions via predefined procedures for interrupt handling and direct port I/O make it possible to code an entire application in Pascal without compromising portability.

Standard Intel REALMATH is easy to use and provides reliable results, consistent with other Intel languages and other implementations of the IEEE proposed Floating-Point standard.

Provides run-time support for co-processors. All real-type arithmetic is performed on the 86/20 numeric data processor unit or software emulator. Run-time library routines, common between Pascal and other Intel languages (such as FORTRAN), permit efficient and consistently accurate results.

Extended relocation and linkage support allows the user to link Pascal program modules with routines written in other languages for certain parts of the program. For example, real-time or hardware dependent routines written in ASM 86/88/186/188 or PL/M 86/88/186/188 can be linked to Pascal routines, further extending the user's ability to write structured and modular programs.

PASCAL 86/88 programs "talk" to the resident operating system using Intel's standard interface for translated programs. This allows users to replace the development operating system by their own operating systems in the final application.

PASCAL 86/88 takes full advantage of iAPX 86, 88, 186, 188 high level language architecture to generate efficient machine code.

Compiler options can be used to control the program listings and object modules. While debugging, the user may generate additional information such as the symbol record information required and useful for debugging using PSCOPE or ICE emulation. After debugging, the production version may be streamlined by removing this additional information.



**SPECIFICATIONS**

**Operating Environment**

**Documentation Package**

**REQUIRED HARDWARE**

*PASCAL 86 User's Guide*

Intel Microcomputer Development Systems (Series III, Series IV)

---

**ORDERING INFORMATION**

**Part Number Description**

MDS\*-314 PASCAL 86/88 Software Package

Requires software license.

\* MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Science.

---

**SUPPORT:**

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.



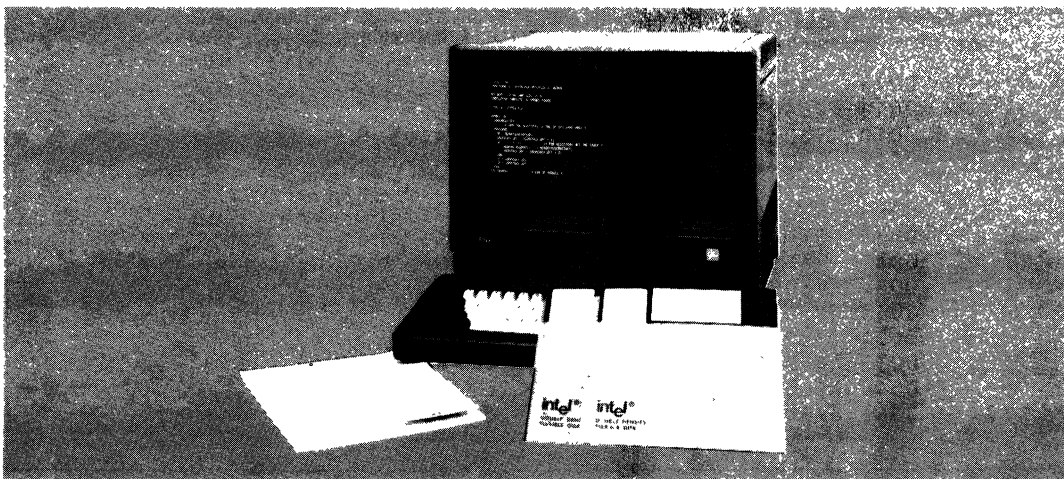
## FORTRAN 86/88 SOFTWARE PACKAGE

- Features high-level language support for floating-point calculations, transcendentals, interrupt procedures, and run-time exception handling
- Meets ANS FORTRAN 77 Subset Language Specifications
- Supports iAPX 86/20, 88/20 Numeric Data Processor for fast and efficient execution of numeric instructions
- Uses REALMATH Floating-Point Standard for consistent and reliable results
- Supports Arrays Larger Than 64K
- Unlimited User Program Symbols
- Offers powerful extensions tailored to microprocessor applications
- Offers upward compatibility with FORTRAN 80
- Provides FORTRAN run-time support for iAPX 86,88,186,188-based design
- Provides users ability to do formatted and unformatted I/O with sequential or direct access methods
- ICE™ Symbolic Debugging Fully Supported
- PSCOPE Source Level Debugging Fully Supported

FORTRAN 86/88 meets the ANS FORTRAN 77 Language Subset Specification and includes many features of the full standard. Therefore, the user is assured of portability of most existing ANS FORTRAN programs and of full portability from other computer systems with an ANS FORTRAN 77 Compiler.

FORTRAN 86/88 programs developed and debugged on the Intel Microcomputer Development Systems may be tested with the prototype using ICE symbolic debugging, and executed on an RMX-86 operating system, or on a user's iAPX 86,88,186,188-based operating system.

FORTRAN 86/88 is one of a complete family of compatible programming languages for iAPX 86,88,186,188 development: PL/M, Pascal, FORTRAN, and Assembler. Therefore, users may choose the language best suited for a specific problem solution.



**FEATURES**

**Extensive High-Level Language Numeric Processing Support**

Single (32-bit), double (64-bit), and double extended precision (80-bit) floating-point data types

REALMATH Proposed IEEE Floating-Point Standard) for consistent and reliable results

Full support for all other data types: integer, logical, character

Ability to use hardware (iAPX 86/20, 88/20 Numeric Data Processor) or software (simulator) floating-point support chosen at link time

ANS FORTRAN 77 Standard

**Intel® Microprocessor Support**

FORTRAN 86/88 language features support of iAPX 86/20, 88/20 Numeric Data Processor

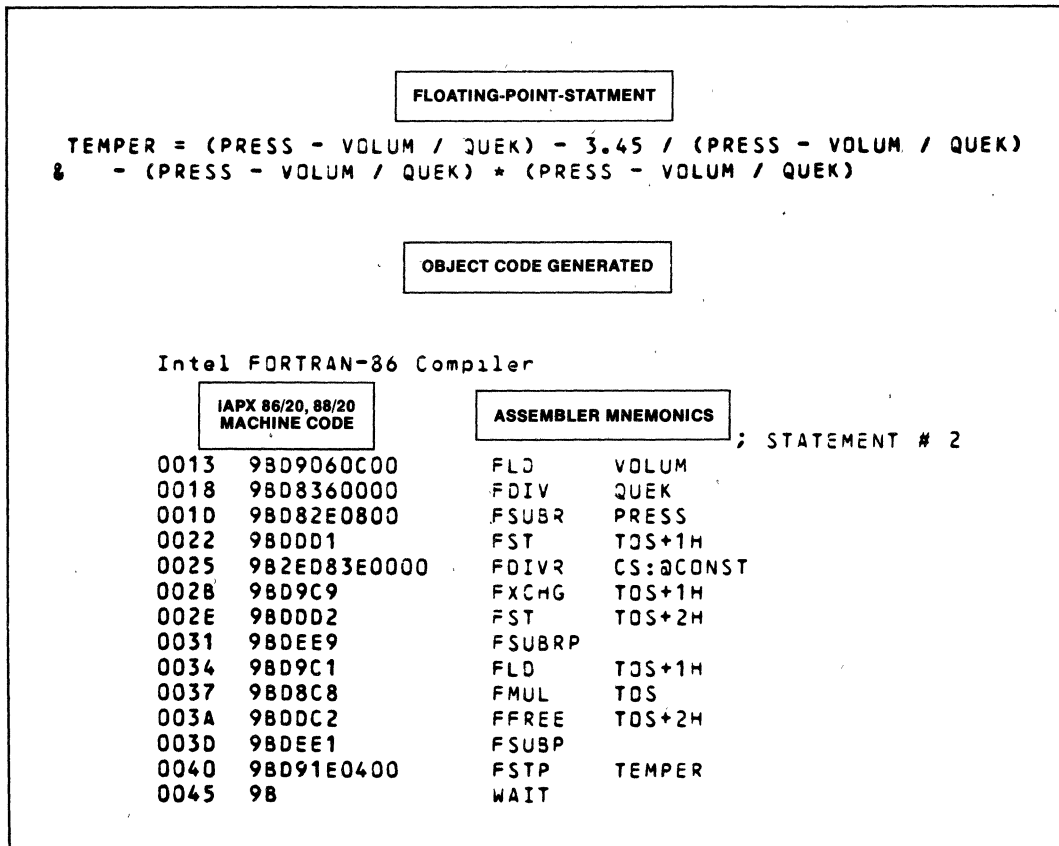
Compiler generates in-line iAPX 86/20, 88/20 Numeric Data Processor object code for floating-point arithmetic (See Figure 1)

Intrinsics allow user to control iAPX 86/20, 88/20 Numeric Data Processor

iAPX 86,88,186,188 architectural advantages used for indexing and character-string handling

Symbolic debugging of application using ICE emulators

Source level debugging using PSCOPE.



**Figure 1. Object Code Generated by FORTRAN 86/88 for a Floating-Point Calculation Using iAPX 86/20, 88/20 Numeric Processor**

**Microprocessor Application Support**

- Direct byte- or word-oriented port I/O
- Reentrant procedures
- Interrupt procedures

**Flexible Run-Time Support**

Application object code may be executed in iAPX 86, 88, 186, 188-based environment of user's choice:

- a Series III or Series IV Intel Development System
- an iAPX 86,88,186,188-based system with iRMX-86 Operating System
- an iAPX 86,88,186,188-based system with user-designed Operating System

Run-time exception handling for fixed-point numerics, floating-point numerics, and I/O errors

Relocatable object libraries for complete run-time support of I/O and arithmetic functions. In-line code execution is generated for iAPX 86/20, 88/20 Numeric Data Processor

**BENEFITS**

FORTRAN 86/88 provides a means of developing application software for the Intel iAPX 86,88,186,188 products lines in a familiar, widely accepted, and industry-standard programming language. FORTRAN 86,88 will greatly enhance the user's ability to provide cost-effective software development for Intel microprocessors as illustrated by the following:

**Early Project Completion**

FORTRAN is an industry-standard, high-level numerics processing language. FORTRAN programmers can use FORTRAN 86/88 on microprocessor projects with little retraining. Existing FORTRAN software can be compiled with FORTRAN 86/88 and programs developed in FORTRAN 86/88 can run on other computers with ANS FORTRAN 77 with little or no change. Libraries of mathematical programs using ANS 77 standards may be compiled with FORTRAN 86/88.

**Application Object Code Portability for a Processor Family**

FORTRAN 86/88 modules "talk" to the resident Intel development operating system using Intel's standard interface for all development-system software. This allows an application developed under the ISIS-II operating system to execute on iRMX/86, or a user-supplied operating system by linking in the iRMX/86 or other appropriate interface library. A standard logical-record interface enables communication with non-standard I/O devices.

**Comprehensive, Reliable and Efficient Numeric Processing**

The unique combination of FORTRAN 86/88, iAPX 86/20, 88/20 Numeric Data Processor, and REALMATH (Proposed IEEE Floating-Point Standard) provide universal consistency in results of numeric computations and efficient object code generation.

---

**SPECIFICATIONS****Operating Environment**

Intel Microcomputer Development Systems (Series III/Series IV)

**Documentation Package**

*FORTRAN 86/88 User's Guide*



**ORDERING INFORMATION**

**Part Number Description**

MDS\*-315 FORTRAN 86/88 Software Package

Requires Software License

---

**SUPPORT**

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.



## C-86 C COMPILER FOR THE 8086

- Implements full C Language
- Produces high density code rivaling assembler
- Supports Intel Object Module Format (OMF)
- Runs under the Intel UDI on Intel Development Systems and iRMX™ 86
- Available for the VAX/VMS\* Operating System
- Supports both small and large models of computation
- Supports ICE™ 86 and DEBUG- 86/88
- Supports IEEE Floating Point Math with 8087 coprocessor
- Supports Bit Fields
- Supports full standard I/O Library (STDIO)
- Written in C

The C Programming Language was originally designed in 1972 and has become increasingly popular as a systems development language. C is not a "very high level" language and is not tied to any specific application area. Although it is used for writing operating systems, it has been used equally well to write numerical, text-processing and data base programs. C combines the flexibility and programming speed of a higher level language with the efficiency and control of assembly language.

Intel C-86 brings the full power of the C programming language to 8086 and 8088 based microprocessor systems.

Intel C-86 supports the full C language as described in the Kernighan and Ritchie book, "The C Programming Language," (Prentice-Hall, 1978). Also included are the latest enhancements to the C language: structure assignments, functions taking structure arguments and returning structures, and the "void" and "enum" data types.

C is rapidly becoming the standard microprocessor system implementation language because it provides:

1. the ability to manipulate the fundamental objects of the machine (including machine addresses) as easily as assembly language.
2. the power and speed of a structured language supporting a large number of data types, storage classes, expressions and statements,
3. processor independence (most programs developed for other processors can be easily transported to the 8086), and
4. code that rivals assembly language in efficiency

---

### INTEL C-86 COMPILER DESCRIPTION

The C-86 compiler operates in four phases: preprocessor, parser, code generator, and optimizer. The preprocessor phase interprets directives in C source code, including conditional compilations (#define). The parser phase converts the C program into an intermediate free form and does all syntactic and

semantic error checking. The code generator phase converts the parser's output into an efficient intermediate binary code, performs constant folding, and features an extremely efficient register allocator, ensuring high quality code. The optimizer phase converts the output of the code generator into

relocatable Intel Object Module Format (OMF) code, without creating an intermediate assembly file. Optionally, the C-86 compiler can produce a symbolic assembly like file. The C-86 optimizer eliminates common code, eliminates redundant loads and stores, and resolves span dependencies (shortens branches) within a program.

The C-86 runtime library consists of a number of functions which the C programmer can call. The runtime system includes the standard I/O library

(STDIO), conversion routines, routines for manipulating strings, special routines to perform functions not available on the 8086 (32-bit arithmetic and emulated floating point), and (where appropriate) routines for interfacing with the operating system.

C-86 uses Intel's linker and locator and generates debug records for symbols and lines on request, permitting access to Intel's ICE-86 and DEBUG-86 to aid in program testing.

## FEATURES

### Support for Small and Large Models

Intel C-86 supports both the SMALL and LARGE modes of segmentation. A SMALL model program can have up to 64K bytes of code and 64K bytes of data, with all pointers occupying two bytes. Because two byte pointers permit the generation of highly compact and efficient code, this model is recommended for programs that can meet the size restrictions. The LARGE segmentation model is used by programs that require access to the full addressing space of the 8086/8088 processors. In this model, each source file generates a distinct pair of code and data segments of up to 64k bytes in length. All pointers are four bytes long.

### Preprocessor Directives

**#define**—defines a macro

**#include**—includes code outside of the program source file

**#if**—conditionally includes or excludes code

Other preprocessor directives include **#undef**, **#ifdef**, **#ifndef**, **#else**, **#endif**, and **#line**.

### Statements

The C language supports a variety of statements:

Conditionals: IF, IF-ELSE

Loops: WHILE, DO-WHILE, FOR

Selection of cases: SWITCH, CASE, DEFAULT

Exit from a function: RETURN

Loop control: CONTINUE, BREAK

Branching: GOTO

### Expressions and Operators

The C language includes a rich set of expressions and operators.

Primary expression: invoke functions, select ele-

ments from arrays, and extract fields from structures or unions

Arithmetic operators: add, subtract, multiply, divide, modulus

Relational operators: greater than, greater than or equal, less than, less than or equal, not equal

Unary operators: indirect through a pointer, compute an address, logical negation, ones complement, provide the size in bytes of an operand.

Logical operators: AND, OR

Bitwise operators: AND, exclusive OR, inclusive OR, bitwise complement

### Data Types and Storage Classes

Data in C is described by its type and storage class. The type determines its representation and use, and the storage class determines its lifetime, scope, and storage allocation. The following data types are fully supported by C-86:

#### **char**

an 8 bit signed integer

#### **int**

a 16 bit signed integer

#### **short**

same as int (on the 8086)

#### **long**

a 32 bit signed integer

#### **unsigned**

a modifier for integer data types (char, int, short, and long) which doubles the positive range of values

#### **float**

a 32 bit floating point number which utilizes the 8087 or a software floating point library

#### **double**

a 64 bit floating point number

**void**

a special type that cannot be used as an operand in expressions; normally used for functions called only for effect (to prevent their use in contexts where a value is required).

**enum**

an enumerated data type

These fundamental data types may be used to create other data types including: arrays, functions, structures, pointers, and unions.

The storage classes available in C-86 include:

**register**

suggests that a variable be kept in a machine register, often enhancing code density and speed

**extern**

a variable defined outside of the function where it is declared; retaining its value throughout the entire program and accessible to other modules

**auto**

a local variable, created when a block of code is entered and discarded when the block is exited

**static**

a local variable that retains its value until the termination of the entire program

**typedef**

defines a new data type name from existing data types

---

## BENEFITS

### Faster Compilation

Intel C-86 compiles C programs substantially faster than standard C compilers because it produces Intel OMF code directly, eliminating the traditional intermediate process of generating an assembly file.

### Generates High Quality Code

For typical programs using the SMALL model, the Intel C-86 Compiler produces code that is 14–16% smaller than on a Digital Equipment PDP-11.

### Portability of Code

Because Intel C-86 supports the .STDIO and produces Intel OMF code, programs developed on a

variety of machines can easily be transported to the 8086.

### Rapid Program Development

Intel C-86 provides the programmer with detailed error messages and access to ICE-86 and DEBUG-86 to speed program development.

### Full Manipulation of the 8086

Intel C-86 enables the programmer to utilize features of the C language to control bit fields, pointers, addresses and register allocation, taking full advantage of the fundamental concepts of the 8086.

---

## SPECIFICATIONS

### Operating Environment

The C-86 compiler runs host resident on both the Intel Series III Microcomputer Development System under ISIS-II and on the System 86/330 under the iRMX 86 operating system. C-86 can also run as a cross compiler on a VAX 11/780 computer under the VMS operating system. 96 KBytes of User Memory is required on all versions. Specify desired version when ordering.

### Required Hardware

Development System Version

—Intellic<sup>®</sup> Microcomputer Development System; Series III or Series IV

—Dual Diskette Drives, Single or Double Density  
—System Console; CRT or Hardcopy Interactive Device

iRMX 86 version:

—Any iAPX 86/88, iSBC<sup>®</sup> 86/88, iTPS 86/XXX, or SYS 86/3XX based system capable of running the iRMX 86 Operating System

VAX version:

—Digital Equipment Corporation VAX 11/780 or compatible computer





**Optional Hardware**

ISIS-II version:

—ICE-86

iRMX 86 version:

—Numeric Data Processors for support of the REALMATH standard

VAX version:

—None

**Required Software**

ISIS-II version:

- ISIS-II Diskette Operating System
- Series III Operating System

iRMX 86 version:

- iRMX 86 Realtime Multiprogramming Operating System
- iRMX 860 Utilities Package

VAX version:

—VMS Operating System

**Optional Software**

Development System Version:

—None

iRMX 86 version:

—None

VAX version:

- MDS\*-384 Kit-Mainframe Link for distributed development, or iMDX-394 Asynchronous Communications Link.
- VAX-IAPX 86/88/186 MACRO Assembler and utilities package (iMDX-341VX)

**Documentation Package**

*The C Programming Language* by Kernighan and Ritchie (1978 Prentice-Hall)

*C-86 User Manual*

**Shipping Media**

Development System Version:

—One single and one double density ISIS-II format 8" diskette, one S-25" Series IV Format

iRMX 86 version:

- Double Density iRMX 86 format 8" diskette
- Double Density iRMX 86 format 5 1/4" diskette

VAX version:

- 1600 bpi, 9 track Magnetic tape
- One single and one double density ISIS-II format 8" diskette

**ORDERING INFORMATION**

Order Code	Description
iMDX-317	C-86 Compiler for ISIS-II
iRMX-866	C-86 Compiler for iRMX 86
iMDX-347	C-86 Cross Compiler for VAX/VMS

Intel Software License required.

**SUPPORT**

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.



## 8087 SOFTWARE SUPPORT PACKAGE

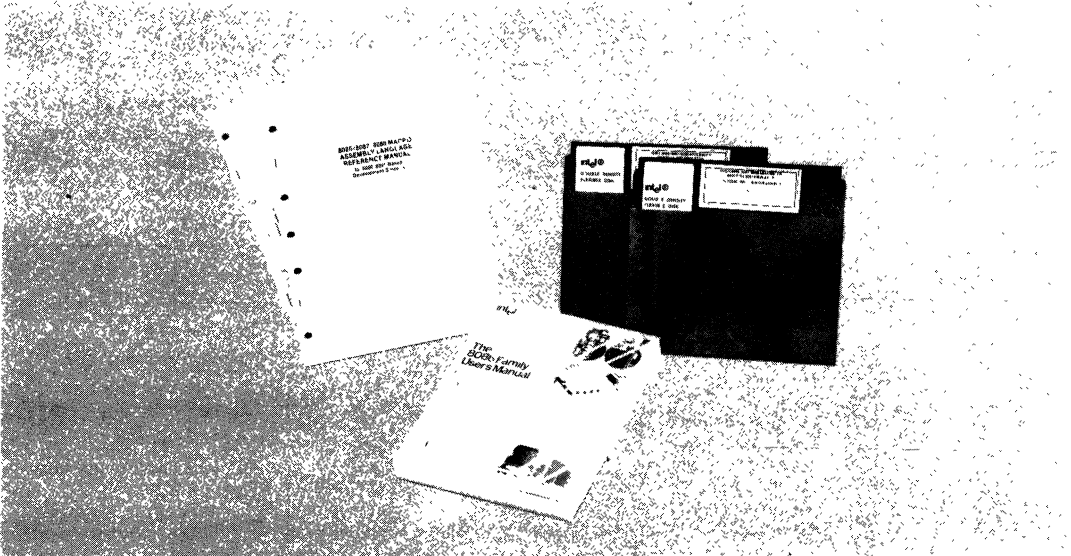
- **Program Generation for the 8087 Numeric Data Processor on 8080/8085 Based Intel Microcomputer Development Systems**
- **Consists of: 8086/8087/8088 Macro Assembler, 8087 Software Emulator**
- **Macro Assembler Generates Code for 8087 Processor or Emulator, While Also Supporting the 8086/8088 Instruction Set**
- **8087 Emulator Duplicates Each 8087 Floating-Point Instruction in Software, for Evaluation of Prototyping, or for Use in an End Product**
- **Macro Assembler and 8087 Emulator are Fully Compatible with Other 8086/8088 Development Software**
- **Implementation of the IEEE Proposed Floating-Point Standard (the Intel® Realmath Standard)**

The 8087 Software Support Package is an optional extension of Intel's 8086/8088 Software Development Package that runs under ISIS-II.

The 8087 Software Support Package consists of the 8086/8087/8088 Macro Assembler, and the Full 8087 Emulator. The assembler is a functional superset of the 8086/8088 Macro Assembler, and includes instructions for over sixty new floating-point operations, plus new data types supported by the 8087.

The 8087 Emulator is an 8086/8088 object module that simulates the environment of the 8087, and executes each floating-point operation using software algorithms. This emulator functionally duplicates the operation of the 8087 Numeric Data Processor.

Also included in this package are interface libraries to link with 8086/8087/8088 object modules, which are used for specifying whether the 8087 Processor or the 8087 Emulator is to be used. This enables the run-time environment to be invisible to the programmer at assembly time.



The following are trademarks of Intel Corporation and may be used only to identify Intel products: BXP, CREDIT, Intelec, Multibus, i, iSBC, Multimodule, iCE, iSBX, PROMPT, iRMX, iCS, Library Manager, Promware, insite, MCS, RMX, Intel, Megachassis, UPI, Intelevison, Micromap,  $\mu$ Scope and the combination of iCE, iCS, iSBC, iSBX, MCS, or RMX and a numerical suffix.

© INTEL CORPORATION, 1983

## FUNCTIONAL DESCRIPTION

### 8086/8087/8088 Macro Assembler

The 8086/8087/8088 Macro Assembler translates symbolic macro assembly language instructions into appropriate machine instructions. It is an extended version of the 8086/8088 Macro Assembler, and therefore supports all of the same features and functions, such as limited type checking, conditional assembly, data structures, macros, etc. The extensions are the new instructions and data types to support floating-point operations. Realmath floating-point instructions (see Table 1) generate code capable of being converted to either 8087 instructions or interrupts for the 8087 Emulator. The Processor/Emulator selection is made via interface libraries at LINK-time. In addition to the new

floating-point instructions, the macro assembler also introduces two new 8087 data types: QWORD (8 bytes) and TBYTE (ten bytes). These support the highest precision of data processed by the 8087.

### Full 8087 Emulator

The Full 8087 Emulator is a 16-kilobyte object module that is linked to the application program for floating-point operations. Its functionality is identical to the 8087 chip, and is ideal for prototyping and debugging floating-point applications. The Emulator is an alternative to the use of the 8087 chip, although the latter executes floating-point applications up to 100 times faster than an 8086 with the 8087 Emulator. Furthermore, since the 8087 is a "co-processor," use of the chip will allow many operations to be performed in parallel with the 8086.

**Table 1. 8087 Instructions**

#### Arithmetic Instructions

Addition	
FADD FADDP FIADD	Add real Add real and pop Integer add
Subtraction	
FSUB FSUBP FISUB FSUBR FSUBRP FISUBR	Subtract real Subtract real and pop Integer subtract Subtract real reversed Subtract real reversed and pop Integer subtract reversed
Multiplication	
FMUL FMULP FIMUL	Multiply real Multiply real and pop Integer multiply
Division	
FDIV FDIVP FIDIV FDIVR FDIVRP FIDIVR	Divide real Divide real and pop Integer divide Divide real reversed Divide real reversed and pop Integer divide reversed
Other Operations	
FSQRT FSCALE FPREM FRNDINT FEXTRACT  FABS FCHS	Square root Scale Partial remainder Round to integer Extract exponent and significand Absolute value Change sign

#### Processor Control Instructions

FINIT/FNINIT	Initialize processor
FDISI/FNDISI	Disable interrupts
FENI/FNENI	Enable interrupts
FLDCW	Load control word
FSTCW/FNSTCW	Store control word
FSTSW/FNSTSW	Store status word
FCLEX/FNCLEX	Clear exceptions
FSTENV/FNSTENV	Store environment
FLDENV	Load environment
FSAVE/FNSAVE	Save state
FRSTOR	Restore state
FINCSTP	Increment stack pointer
FDECSTP	Decrement stack pointer
FFREE	Free register
FNOP	No operation
FWAIT	CPU wait

#### Comparison Instructions

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Integer compare
FICOMP	Integer compare and pop
FTST	Test
FXAM	Examine

**Table 1. 8087 Instructions (cont'd)**
**Transcendental Instructions**

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	$2^x - 1$
FYL2X	$Y \cdot \log_2 X$
FYL2XP1	$Y \cdot \log_2(X + 1)$

**Constant Instructions**

FLDZ	Load +0.0
FLD1	Load +1.0
FLDPI	Load $\pi$
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

**Data Transfer Instructions**

Real Transfers	
FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers
Integer Transfers	
FILD	Integer load
FIST	Integer store
FISTP	Integer store and pop
Packed Decimal Transfers	
FBLD	Packed decimal (BCD) load
FBSTP	Packed decimal (BCD) store and pop

**SPECIFICATIONS**
**Operating Environment**
**REQUIRED HARDWARE**

- Intel Microcomputer Development Systems
- Model 800
- Series II
- Personal Development System
- Series IV

**REQUIRED SOFTWARE**

8086/8088 Software Development Package

**Documentation Package**

*8086/8087/8088 Macro Assembly Language Reference Manual for 8080/8085-Based Development Systems*

*8086/8087/8088 Macro Assembler Operating Instructions for 8080/8085-Based Development Systems*

*The 8086 Family Users Manual Supplement for the 8087 Numeric Data Processor*

**ORDERING INFORMATION**
**Part Number Description**

MDS\*-387 8087 Software Support Package

Requires Software License

**SUPPORT**

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

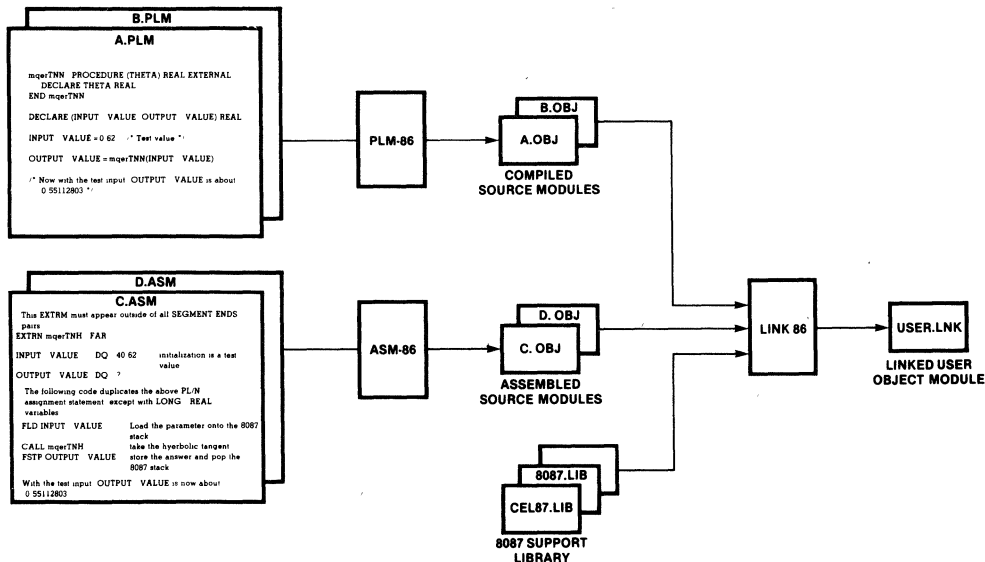


# 8087 SUPPORT LIBRARY

- Library to support floating point arithmetic in PL/M-86 and ASM-86
- Full 8087 Software Emulator for software debugging without the 8087 component
- Common elementary function library provides trigonometric, logarithmic and other useful functions
- Accurate, verified and efficient implementation of algorithms for functions
- Decimal conversion module supports binary-decimal conversions
- Supports proposed IEEE Floating Point Standard for high accuracy and software portability
- Error-handler module simplifies floating point error recovery

The 8087 Support Library provides PL/M-86 and ASM-86 users with the equivalent numeric data processing capability of Fortran-86. With the Library, it is easy for PL/M-86 and ASM-86 programs to do floating point arithmetic. Programs can link in modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs. The 8087 Support Library implements Intel's REALMATH standard and also supports the proposed IEEE Floating Point Standard. Consequently, by using this Library, the PL/M-86 user not only saves software development time, but is guaranteed that the numeric software meets industry standards and is portable—his software investment is maintained.

The 8087 Support Library consists of the common elementary function library, the decimal conversion module, the error handler module, the full 8087 Software emulator and interface libraries to the 8087 and to the 8087 emulator.



## CEL87.LIB

# THE COMMON ELEMENTARY FUNCTION LIBRARY

CEL87.LIB contains commonly used floating point functions. It is used along with the 8087 numeric coprocessor or the 8087 emulator and it provides a complete package of elementary functions, giving valid results for all appropriate inputs. This library provides PL/M-86 and ASM-86 users all the math functions supported intrinsically by the Fortran-86. Following is a summary of CEL87 functions, grouped by functionality.

### Rounding and Truncation Functions:

mqrEX, mqrE2, and mqrE4 round a real number to the nearest integer; to the even integer if there is a tie. The answer returned is real, a 16-bit integer or a 32-bit integer respectively.

mqrAX, mqrA2, mqrA4 round a real number to the nearest integer, to the integer away from zero if there is a tie; the answer returned is real, a 16-bit integer or a 32-bit integer, respectively.

mqrCX, mqrC2, mqrC4 truncate the fractional part of a real input; the answer is real, a 16-bit integer or a 32-bit integer, respectively.

### Logarithmic and Exponential Functions:

mqrLGD computes decimal (base 10) logarithms.

mqrLGE computes natural (base e) logarithms.

mqrEXP computes exponentials to the base e.

mqrY2X computes exponentials to any base.

mqrYI2 raises an input real to a 16-bit integer power.

mqrYI4 is as mqrYI2, except to a 32-bit integer power.

mqrYIS is as mqrYI2, but it accommodates PL/M-86 users.

### Trigonometric and Hyperbolic Functions:

mqrSIN, mqrCOS, mqrTAN compute sine, cosine, and tangent.

mqrASN, mqrACS, mqrATN compute the corresponding inverse functions.

mqrSNH, mqrCSH, mqrTNH compute the corresponding hyperbolic functions.

mqrAT2 is a special version of the arc tangent function that accepts rectangular coordinate inputs.

### Other Functions:

mqrDIM is FORTRAN's positive difference function.

mqrMAX returns the maximum of two real inputs.

mqrMIN returns the minimum of two real inputs.

mqrSGH combines the sign of one input with the magnitude of the other input.

mqrMOD computes a modulus, retaining the sign of the dividend.

mqrRMD computes a modulus, giving the value closest to zero.

---

## DCON87.LIB

# THE DECIMAL CONVERSION LIBRARY

DCON87.LIB is a library of procedures which convert binary representations of floating point numbers and ASCII-encoded string of digits.

The binary-to-decimal procedure mqcBIN DECLOW accepts a binary number in any of the formats used for the representation of floating point numbers in the 8087. Because there are so many output formats for floating point numbers, mqcBIN\_DECLOW does not attempt to provide a finished, formatted text string. Instead, it provides the "building blocks" for you to use to construct the output string which meets your exact format specification.

The decimal-to-binary procedure `mqcDEC_BIN` accepts a text string which consists of a decimal number with optional sign, decimal point, and/or power-of-ten exponent. It translates the string into the caller's choice of binary formats.

Decimal-to-binary procedure `mqcDECLOW_BIN` is provided for callers who have already broken the decimal number into its constituent parts.

The procedures `mqcLONG_TEMP`, `mqcSHORT_TEMP`, `mqcTEMP_LONG`, and `mqcTEMP_SHORT` convert floating point numbers between the longest binary format, `TEMP_REAL`, and the shorter formats.

---

## **EH87.LIB THE ERROR HANDLER MODULE**

**EH87.LIB** is a library of five utility procedures which a user can utilize for writing trap handlers. Trap handlers are called when an unmasked 8087 error occurs.

The 8087 error reporting mechanism can be used not only to report error conditions, but also to let software implement IEEE standard options not directly supported by the chip. The three such extensions to the 8087 are: normalizing mode, non-trapping not-a-number (NaN), and non-ordered comparison. The utility procedures support these extra features.

**DECODE** is called near the beginning of the trap handler. It preserves the complete state of the 8087, and also identifies what function called the trap handler, and returns available arguments and/or results. **DECODE** eliminates much of the effort needed to determine what error caused the trap handler to be called.

**NORMAL** provides the "normalizing mode" capability for handling the "D" exception. By calling **NORMAL** in your trap handler, you eliminate the need to write code in your application program which tests for non-normal inputs.

**SIEVE** provides two capabilities for handling the "I" exception. It implements non-trapping NaN's and non-ordered comparisons. These two IEEE standard features are useful for diagnostic work.

**ENCODE** is called near the end of the trap handler. It restores the state of the 8087 saved by **DECODE**, and performs a choice of concluding actions, by either retrying the offending function or returning a specified result.

**FILTER** calls each of the above four procedures. If your error handler does nothing more than detect fatal errors and implement the features supported by **SIEVE** and **NORMAL**, then your interface to **EH87.LIB** can be accomplished with a single call to **FILTER**.

---

## **E8087 THE FULL 8087 EMULATOR**

**E8087** is an object module that functionally emulates the 8087 coprocessor chip. It is ideal for use during prototyping and debugging floating point programs. However, the target system should use the 8087 component because it executes 1000 times faster and uses significantly less memory.

## **E8087.LIB, 8087.LIB, 87NULL.LIB INTERFACE LIBRARIES**

E8087.LIB, 8087.LIB and 87NULL.LIB libraries configure a user's application program for his run-time environment: running with the emulator, with the 8087 component or without floating point arithmetic, respectively.

---

### **SPECIFICATIONS**

#### **TARGET ENVIRONMENT**

8086/8088 Based Microcomputer System

#### **DEVELOPMENT ENVIRONMENT**

##### **Required Hardware**

All Intel Microcomputer Development Systems (Series II, Series III/Series IV)

##### **Required Software**

For Series II:

8086/8088 Software Development Package

##### **Documentation Package**

Numeric Support Library Manual

---

\*Recommended

### **ORDERING INFORMATION**

<b>Part Number</b>	<b>Description</b>
MDS*-319	8087 Support Library

Requires Software License

---

### **SUPPORT**

Intel offers several levels of support for this product which are explained in detail in the price list. Please consult the price list for a description of the support options available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.





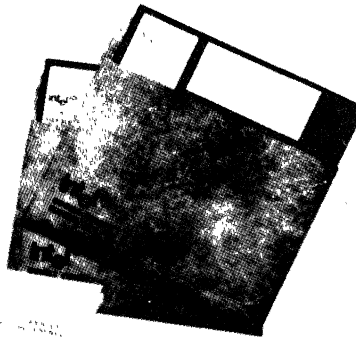
## 8089 IOP SOFTWARE SUPPORT PACKAGE #407200

- Program Generation for the 8089 I/O Processor on the Intellec® Microcomputer Development System
- Contains 8089 Macro Assembler, plus Relocation and Linkage Utilities
- Relocatable Object Module Compatible with All iAPX 86 and iAPX 88 Object Modules
- Fully Supports Symbolic Debugging with the RBF-89 Software Debugger
- Supports 8089-Based Addressing Modes with a Structure Facility that Enables Easy Access to Based Data
- Powerful Macro Capabilities
- Provides Timing Information in Assembly Listing
- Fully Detailed Set of Error Messages

The IOP Software Support Package extends Intellec Microcomputer Development System support to the 8089 I/O Processor. The macro assembler translates symbolic 8089 macro assembly language instructions into relocatable machine code. The relocation and linkage utilities provide compatibility with iAPX 86, iAPX 88, and 8089 modules, and make structured, modular programming easier.

The macro assembler also provides symbolic debugging capability when used with the RBF-89 software debugger. 8089 program modularity is supported with inter-segment jumps and calls. The macro assembler also provides instruction cycle counts in the listing file, for giving the programmer execution timing information. The programs in the 8089 Software Support Package run on any Intellec Series II or Model 800 with 64K bytes of memory.

8089 MACRO ASSEMBLER  
USER'S GUIDE



The following are trademarks of Intel Corporation and may be used only to identify Intel products: BXP, CREDIT, Intellec, Multibus, iSBX, Multimodule, ICE, iSBX, PROMPT, iCS, iRMX, Library Manager, Promware, Instec, MCS, RMX, Intel, Megachassis, UPI, Intelevison, Micromap, μScope and the combination of iCE, iSBC, iSBX, MCS, or RMX and a numerical suffix.

MAY 1983

© INTEL CORPORATION 1983

ORDER NUMBER:210853-002

## FUNCTIONAL DESCRIPTION

The IOP Software Support Package contains:

- ASM89 —The 8089 Macro Assembler.
- LINK86 —Resolves control transfer references between 8089 object modules, and data references in 8086, 8088, and 8089 modules.
- LOC86 —Assigns absolute memory addresses to 8089 object modules.
- OH86 —Converts absolute object modules to hexadecimal format.
- UPM —The Universal PROM Mapper, which supports PROM programming in all iAPX 86/11 and iAPX 88/11 applications.

ASM89 translates symbolic 8089 macro assembly language instructions into the appropriate machine codes. The ability to refer to both program and data addresses with symbolic names makes it easier to develop and modify programs, and avoids the errors of hand translation.

The powerful macro facility allows frequently used code sequences to be referred to by a single name,

so that any changes to that sequence need to be made in only one place in the program. Common code sequences that differ only slightly can also be referred to with a macro call, and the differences can be substituted with macro parameters.

ASM89 provides symbolic debugging information in the object file. The RBF-89 debugger makes use of this information, so the programmer can symbolically debug 8089 programs. ASM89 also provides cycle counts for each instruction in the assembly listing file (see Table 1). These cycle counts help the programmer determine how long a particular routine or code sequence will take to execute on the 8089.

ASM89 provides relocatable object module compatibility with the 8086 and 8088 microprocessors. This object module compatibility, along with the 8086/8088 relocation and linkage utilities, facilitates the designing of iAPX 86/11 and iAPX 88/11 systems.

ASM89 fully supports the based addressing modes of the 8089. A structure facility allows the user to define a template that enables accessing of based data symbolically.

## SPECIFICATIONS

### Operating Environment

Intel Microcomputer Development Systems (Model 800, Series II, Series III, Series IV)

### Support

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

### Documentation Package

- 8089 Macro Assembler User's Guide (9800938)*
- 8089 Macro Assembler Pocket Reference (9800936)*
- MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users (9800639)*
- Universal PROM Programmer User's Manual (9800819)*

## Shipping Media

—Single and Double Density Diskettes

## ORDERING INFORMATION

### Part Number Description

MDS*-312	8089 IOP Software Support Package
Requires Software License	

\*MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation

### Table 1. Sample Program Listing

```

I8089 MACRO ASSEMBLER

I8089-11 8089 MACRO ASSEMBLER X105 ASSEMBLY OF MODULE TASK
OBJECT MODULE PLACED IN FI TASK OBJ
ASSEMBLER INVOKED BY 15499 FI task 099 gen macro debug page,width(132) print:(fi:taskx.lst)

LOC. OBJECT CODE          TIMING INC MAC LINE SOURCE
1 *****
2 *
3 *          8089 TASK PROGRAM
4 *
5 *****
6
7 name TASK
8 TASK segment
9
10
11      In the first part of this      nple program data is moved from
12      part, the data is moved f     cal to the 8089 IOP. In the second
13      also in the 8089 I/O spac     the local memory to a data port
14
15 data@port@0251 equ    0C000h      ;0251 DP on 8089 local bus
16 command@port@0251 equ 0C001h      ;0251 CP on 8089 local bus
17 buffer@0889 equ      0200h       ;RAM buffer in 8089 I/O space
18
19 extrn  buffer@0889      ;RAM buffer in 8086 system memory
20 extrn  y                ;location of the buffer count
21
22 %define (macro_1)
23     movl  gb.buffer@0889      ; Move buffer address into GB
24     lpd:  gc:y                ; Load pointer to count into GC
25     movb  bc:[gc]            ; Move byte count into BC
26
27 %define (macro_2(param_1, param_2)) local loop
28     inc  %param_1             ; Increment pointer into source
29     dec  %param_2             ; Decrement byte count
30     jnz  %param_2,xloop      ; Loop back if byte count > 0
31
32 ONE:  lpd:  ga.buffer@0886    ; Load register GA with address
33         f                    ; of 8086 buffer
34 %macro_1
35     movl  gb.buffer@0889      ; Move buffer address into GB
36     lpd:  gc:y                ; Load pointer to count into GC
37     movb  bc:[gc]            ; Move byte count into BC
38
39 loop00: movb  [gb].[gc]      ; Move byte from 8086 to 8089 buffer
40         inc  ga                ; Increment pointer into 8086 buffer
41 %macro_2(gb,gc)
42     inc  %PARAM_1             ; Increment pointer into source
43     gb  %PARAM_2             ; Decrement pointer into source
44     dec  %PARAM_2             ; Decrement byte count
45     gc  %PARAM_2             ; Decrement byte count
46     jnz  %PARAM_2            ; Loop back if byte count > 0
47     gc,xLOOP
48     LOOP00                   ; Loop back if byte count > 0
49
50 TWO:  movl  ga.data@port@0251 ; load GA with address of 0251 DP
51     movl  ga.command@port@0251 ; load GC with address of 0251 CP
52 %macro_1
53     movl  gb.buffer@0889      ; Move buffer address into GB
54     lpd:  gc:y                ; Load pointer to count into GC
55     movb  bc:[gc]            ; Move byte count into BC
56
57 loop01: inlt  [gc].0,loop01   ; loop until 0251 transmit ready
58     movb  [gc].[gb]          ; move message into buffer
59 %macro_2(gb,gc)
60     nc  %PARAM_1             ; Increment pointer into source
61     gb  %PARAM_2             ; Decrement pointer into source
62     dec  %PARAM_2             ; Decrement byte count
63     gc  %PARAM_2             ; Decrement byte count
64     jnz  %PARAM_2            ; Loop back if byte count > 0
65     gc %LOOP
66     LOOP01                   ; Loop back if byte count > 0
67
68     hit  TASK ends
69 TASK ends
70 END

```

ASSEMBLY COMPLETE, NO ERRORS FOUND



## IAPX 286 SOFTWARE DEVELOPMENT PACKAGE

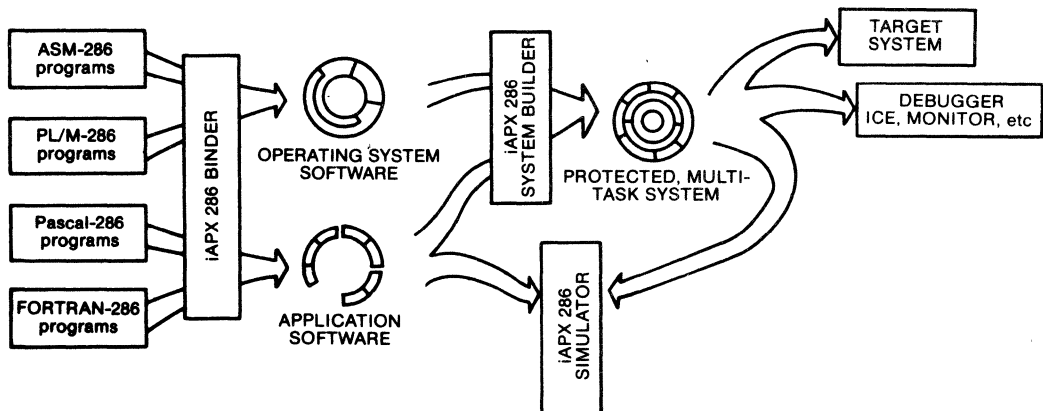
- **Complete System Development Capability for High-Performance IAPX 286 Applications.**
- **Allows creation of Multi-User, Virtual Memory, and Memory-Protected Systems.**
- **Macro Assembler for Machine-Level Programming.**
- **System Utilities for Program Linkage and System Building.**
- **Software Simulator for Execution and Symbolic Debugging on Intel Development System.**
- **Package Supports Program Development with PL/M-286, Pascal-286, and FORTRAN 286.**
- **Extends Existing Intellec® Development Systems to Provide Broad Support for the iAPX 286 Micro-processor.**

The iAPX 286 is a 16-bit microprocessor system with 32-bit virtual addressing, integrated memory protection, and instruction pipelining for high performance. The iAPX 286 Software Development Package is a cohesive set of software design aids for programming the iAPX 286 microprocessor system. The package enables system programmers to design protected, multi-user and multi-tasking operating system software, and enables application programmers to develop tasks to run on a protected operating system.

The iAPX 286 Software Development package contains a macro assembler, a program binder (for linking separately compiled modules together), a system builder (for configuring protected multiple-task systems), and a software simulator (for execution and symbolic debugging).

The memory protection features of the iAPX 286 architecture are invisible to application programmers, who use language translators and the program binder. System programmers may use special memory protection features in ASM-286 or PL/M 286, and use the system builder for initializing and managing protection features. The Simulator duplicates the operation of the 80286 CPU, as well as the floating point operations of the 80287.

All the utilities in the Software Development Package run on the Intel Microcomputer Development Systems (Series III/Series IV).



The iAPX 286 Software Development Package keeps the protection mechanism invisible to the application programmer, yet easy to configure for the system programmer.

## **iAPX 286 MACRO ASSEMBLER**

- **Instruction Set and Assembler Mnemonics Are Upward Compatible with ASM-86/88.**
- **Powerful and Flexible Text Macro Facility.**
- **Type-Checking at Assembly Time Helps Reduce Errors at Run-Time.**
- **Structures and RECORDS Provide Powerful Data Representation.**
- **“High-Level” Assembler Mnemonics Simplify the Language.**
- **Supports Full Instruction Set of the iAPX 286/20, Including Memory Protection and Numerics.**

ASM-286 is the “high-level” macro assembler for the iAPX 286 assembly language. ASM-286 translates symbolic assembly language mnemonics into relocatable object code. The assembler mnemonics are a superset of ASM-86/88 mnemonics; new ones have also been added to support the new iAPX 286 instructions. The segmentation directives have been greatly simplified.

The iAPX 286 assembly language includes approximately 150 instruction mnemonics. From these few mnemonics the assembler can generate over 4,000 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 150 mnemonics to generate all possible machine instructions. ASM-286 will generate the shortest machine instruction possible (given explicit information as to the characteristics of any forward referenced symbols).

The powerful macro facility in ASM-286 saves development and maintenance time by coding common program sequences only once. A macro substitution is made each time the sequence is to be used. This facility also allows for conditional assembly of certain program sequences.

ASM-286 offers many features normally found only in high-level languages. The assembly language is strongly typed, which means it performs extensive checks on the usage of variables and labels. This means that many programming errors will be detected when the program is assembled, long before it is being debugged.

ASM-286 object modules conform to a thorough, well-defined format used by all 286 high-level languages and utilities. This makes it easy to call (and be called from) HLL object modules.

### **Key Benefit:**

For programmers who wish to use assembly language, ASM-286 provides many powerful “high-level” capabilities that simplify program development and maintenance.

## iAPX 286 BINDER

- **Links Separately Compiled Program Modules Into an Executable Task.**
- **Makes the iAPX 286 Protection Mechanism Invisible to Application Programmers.**
- **Works with PL/M-286, Pascal-286, FORTRAN-286 and ASM-286 Object Modules.**
- **Performs Incremental Linking with Output of Binder and Builder.**
- **Resolves PUBLIC/EXTERNAL Code and Data References, and Performs Intermodule Type-Checking.**
- **Provides Print File Showing Segment Map, Errors and Warnings.**
- **Assigns Virtual Addresses to Tasks in the 2<sup>32</sup> Address Space.**
- **Generates Linkable or Loadable Module for Debugging.**

BND-286 is a utility that combines iAPX 286 object modules into executable tasks. In creating a task, the Binder resolves Public and External symbol references, combines segments, and performs address fix-ups on symbolic code and data.

The Binder takes object modules written in ASM-286, PL/M-286, Pascal-286 or FORTRAN-286, and generates a loadable module (for execution or debugging), or a linkable module (to be re-input to the Binder later; this is called incremental binding). The binder accepts library modules as well, linking only those modules required to resolve external references. BND-286 generates a print file displaying a segment map, and error messages.

The Binder will be used by system programmers and application programmers. Since application programmers need to develop software independent of any system architecture, the 286 memory protection mechanism is "hidden" from users of the Binder. This allows application tasks to be fully debugged before becoming part of a protected system. (A protected system may be debugged, as well.) System protection features are specified later in the development cycle, using the 286 System Builder. It is possible to link operating system services required by a task using either the Binder or the Builder. This flexibility adds to the ease of use of the 286 utilities.

### Key Benefit:

The Binder is the only utility an application programmer needs to develop and debug an individual task. Users of the Binder need not be concerned with the architecture of the target machine, making application program development for the 286 very simple.

---

## iAPX 286 MAPPER

- **Flexible Utility to Display Object File Information.**
- **MAP-286 Selectively Purges Symbols from a Load Module.**
- **Provides Inter-Module Cross-Referencing for Modules Written in All Languages.**
- **Mapper Allows Users to Display:**

Protection Information:	Debug Information:
SEGMENT TABLES	MODULE NAMES
GATE TABLES	PROGRAM SYMBOLS
PUBLIC ADDRESSES	LINE NUMBERS

### Key Benefit:

A cross-reference map showing references *between* modules simplifies debugging; the map also lists and controls all symbolic information in one easy-to-read place.

## iAPX 286 LIBRARIAN

- **Fast, Easy Management of iAPX 286 Object Module Libraries.**
- **Only Required Modules Are Linked, When Using the Binder or Builder.**
- **Librarian Allows Users to:**
  - Create Libraries
  - Add Modules
  - Replace Modules
  - Delete Modules
  - Copy Modules from Another Library
  - Save Library Module to Object File
  - Create Backup
  - Display Module Information  
(creation date, publics, segments)

### Key Benefit:

Program libraries improve management of program modules, and reduce software administrative overhead.

## iAPX 286 SYSTEM BUILDER

- **Supports Complete Creation of Protected, Multi-task Systems.**
- **Resolves PUBLIC/EXTERNAL Definitions (between protection levels).**
- **Supports Memory Protection by Building System Tables, Initializing Tasks, and Assigning Protection Rights to Segments.**
- **Creates a Memory Image of a 286 System for Cold-start Execution.**
- **Target System may be Boot-loadable, Programmed into ROM, or Loaded From Mass-store.**
- **Generates Print File with Command Listing and System Map.**

BLD-286 is the utility that lets system programmers configure multi-tasking, protected systems from an operating system and discrete tasks. The Builder generates a cold-start execution module, suitable for ROM-based or disk-based systems.

The Builder accepts input modules from iAPX 286 translators or the iAPX 286 Binder. It also accepts a "Build File" containing definitions and initial values for the 286 protection mechanism—descriptor tables, gates, segments, and tasks. BLD-286 generates a Loadable or bootloadable output module, as well as a print file with a detailed map of the memory-protected system.

Using the Builder command Language, system programmers may perform the following functions:

- Assign physical addresses to segments; also set segment access rights and limits.
- Create Call, Trap, and Interrupt "Gates" (entry-points) for inter-level program transfers.
- Make gates available to tasks; this is an easier way to define program interfaces than using interface libraries.
- Create Global (GDT), Interrupt (IDT), and any Local (LDT) Descriptor Tables.
- Create Task State Segments and Task Gates for multi-task applications.
- Resolve inter-module and inter-level references, and perform type-checking.
- Automatically select required modules from libraries.
- Configure the memory image into partitions in the address space.
- Selectively generate an object file and various sections of the print file.

### Key Benefit:

Allows a system programmer to define the configuration of a protected system in *one* place, with one easy-to-use Utility. This specification may then be adopted by all project members, using either the Builder *or just the Binder*. The flexibility simplifies program development for all users.



## iAPX 286 SIMULATOR

- **Supports Symbolic Debugging of Complete, Protected 286 Systems.**
- **Allows 286 Program Execution and Debugging in Absence of iAPX 286 Hardware Execution Vehicle.**
- **Functionally Duplicates the Operation of the iAPX 286 Microprocessor, Including Memory Protection.**
- **Executes Full Instruction Set, Including 80287 Numerics.**
- **Symbolic Access to Program Variables as well as Descriptor Tables.**
- **Two Execution Timers for Program Benchmarking and Interrupt Simulation.**
- **UDI File System Support for User Program.**

SIM-286 is an 8086-resident program designed to support development of iAPX 286 O.S. kernels, systems, and applications. All of these may be developed and debugged without the use of a 286 hardware execution vehicle.

The Simulator consists of a human interface layer, and software executors for the 80286 CPU and 80287 Numeric Data Processor. The human interface receives commands with symbolic names, and passes control to the executor as though it were a 286-resident monitor.

SIM-286 lets designers manipulate a 286 program using the symbolic names given for code and data. It also lets users symbolically examine and modify the protection features (such as system tables, access rights, etc.), if it is desired.

SIM-286 contains two instruction timers. One may be set and incremented during execution; this allows program sequences to be benchmarked in clock cycles and microseconds. The second, an interval timer, may be set to generate interrupts every  $\eta$  clock cycles, to simulate event-driven processing. These timers are extremely useful for developing system kernels.

For programs that make operating system calls for file I/O, SIM-286 provides access to these services through the Universal Development Interface.

### Key Benefit:

Symbolic system debugging (for protected 286 software) may be performed in the absence of a 286-based target.

## SPECIFICATIONS

### OPERATING ENVIRONMENT

Intel Microcomputer Development Systems  
(Series III/Series IV)

### DOCUMENTATION

ASM 286 Language Reference Manual  
ASM 286 Macro Assembler Operating Instructions  
iAPX 286 Utilities User's Guide

iAPX 286 System Builder User's Guide

iAPX 286 Simulator User's Guide

Pocket Reference for all the above:

ASM 286

Utilities

SIM 286

### SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.

## ORDERING INFORMATION

Product Code	Description
IMDX-321	iAPX 286 Software Development Package





# iAPX 286 EVALUATION PACKAGE

- Provides elementary program development and debug capability for the iAPX 286 microprocessor:
  - Assembly
  - System/task build
  - Symbolic debug
- Easy evaluation of all microprocessor dependencies: architecture, execution speed, program benchmarks
- Provides a programmatic understanding of the iAPX 286 architecture:
  - Instruction set
  - Memory protection
  - Segmentation
  - Program timing
- Includes an iAPX 286 demonstration program that exploits and illustrates architectural features of the 286

The Intel iAPX 286 Evaluation Package is an integrated set of software tools that aids the programmer in understanding how to use the iAPX 286 microprocessor. The package runs on an Intel Microcomputer Development System (Series III or Intel Equivalent).

The Evaluation Package will allow a programmer to create, build, execute, and debug an assembly-level iAPX 286 task. It will also show how a programmer can take advantage of iAPX 286 architecture features.

The software tools contained in the package are a macro assembler, a task builder, run-time support procedures, an iAPX 286 simulator, and a demonstration program. The simulator has a built-in timer for benchmarking code sequences and programs. It also provides symbolic debugging capability, in addition to iAPX 286 program execution.

The benefits of using the iAPX 286 Evaluation Package are two-fold. System designers may now learn the iAPX 286 architecture (and determine its applicability) in the quickest manner possible. In addition, software may be developed now for a future 286 application, thereby getting a head start on a very time-consuming phase of design.

```

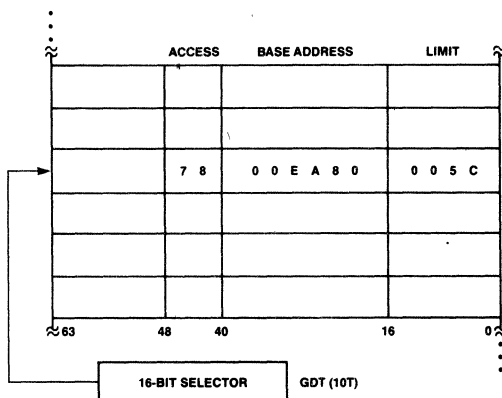
-RUN :F1:SM286E
SERIES III iAPX 286 Evaluation Package Simulator, V1.0
? LOAD DM286E *the demo program
Program Size = 10432      Total Memory = 60128
? GO

Interrupt 3 at 0118:0D72
? FR *contents of the task register
0250 .RPL=0 .TI=0 .INDEX=000A
? TSS *current task state segment
LINK=0000 SP0=0000 SS0=0260 SPI=0000 SS1=0139 SP2=0000 SS2=0142 IP=0D73
FL=0204 AX=0004 CX=0021 DX=0000 BX=0684 SP=FFA0 BP=FFA0 SI=003F DI=FFB5
ES=0260 CS=0118 SS=0260 DS=0118 RLDI=0018
? GO

Interrupt 13 at 0118:0D73 General Protection Ecode = 0000
?
? LDT
LDT (1T) DSEG BASE=000430 LIMIT=0107 P=1 DPL=0 ED=0 W=1 A=0 SR=0000
LDT (2T) DSEG BASE=00E100 LIMIT=0005 P=1 DPL=3 ED=0 W=1 A=1 SR=0000
? GDT(10T) *the 10th global descriptor table entry
GDT (10T) TSS BASE=00EA80 LIMIT=005C B=0 P=1 DPL=0 SR=0058
? EXIT

```

Sample Simulator Session



Segment Descriptor Table

---

**FUNCTIONAL DESCRIPTION****iAPX 286 Evaluation Macro Assembler**

The Evaluation Assembler (AS286E) accepts a source module written in the 286 Macro Assembly Language, and generates an object module and a listing file. The assembler is based upon ASM-86, and therefore performs type-checking on operands, supports complex data structures, and utilizes the same macro processor.

**iAPX 286 Evaluation Builder**

The Evaluation Builder (BD286E) accepts a single assembler object module, and generates a single-tasking executable load module. The Evaluation Builder performs the following functions:

- Assigns attributes to 286 segments: Privilege level, Access Rights, Base Address, Segment Length.
- Creates descriptor table entries (GDT & LDT) from segments.
- Initializes Segment Registers.
- Allows call gates, interrupt gates, and trap gates to be explicitly created, via the Interrupt Descriptor Table.
- Automatically creates call gates for X286E run-time procedures.
- Creates the Task State Segment for a one-task program.
- Produces a map showing all segments, gates, and public symbols.
- Binds segments to absolute addresses.

**iAPX 286 Evaluation Simulator**

The Evaluation Simulator (SM286E) loads and executes a 286 object module created by the Builder. Program execution functionally duplicates iAPX 286 processor operation; data protection, gates, processor traps and interrupts, and segmentation access are all supported in the same way as the iAPX 286. Compatibility mode and numerics are not included.

The symbolic debugger portion of the simulator supports two simultaneous code break-points and single-step execution, as well as modification of variables, registers, descriptor tables, and the task state segment. Code disassembly is also provided.

The Simulator has a built-in instruction timer to aid in benchmarking iAPX 286 programs. Another timer, which also counts clock cycles, can be used to generate interrupts at specific time intervals.

**Run-Time Support Procedures**

The Evaluation Package contains a set of run-time procedures (X286E) that may be "linked" to a user program at build-time to perform several software functions. These functions include creating and modifying segments, descriptors, and tables, creating new tasks, and dynamically allocating free memory for segments.

**The Demonstration Program**

The Demo Program (DM286E) is an application package that uses the Evaluation Tools (AS286E, BD286E, SM286E, X286E) to teach users how to program the iAPX 286.

It not only guides a programmer through the use of these tools, but the demo program itself illustrates how software can exploit the architecture of the 286. The following features are illustrated:

- Memory Protection using object descriptors.
- Gate creation and manipulation.
- Task switching, procedure entry and exit.
- Interrupt handling.
- Dynamic task creation.
- Inter-task communication.

The demonstration consists of a nucleus, a real-time clock interface, a time-of-day clock, a CPU-utilization spy, and a command interpreter. The user may execute these simultaneously on the simulator, and gain an understanding of how the 286 handles the functions listed above.

---

**SPECIFICATIONS****OPERATING ENVIRONMENT**

Intel Microcomputer Development Systems  
(Series III/Series IV)

**DOCUMENTATION**

iAPX 286 Evaluation Macro Assembly Language  
Reference Manual  
iAPX 286 Evaluation Macro Assembler Operating  
Instructions  
iAPX 286 Evaluation Simulator Operating  
Instructions  
iAPX 286 Evaluation Builder's user's Guide

---

**ORDERING INFORMATION**

Product Code	Description
MDS*-322	iAPX 286 Evaluation Package (Requires Software License)

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.

**SUPPORT:**

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.



## PL/M 286 SOFTWARE PACKAGE

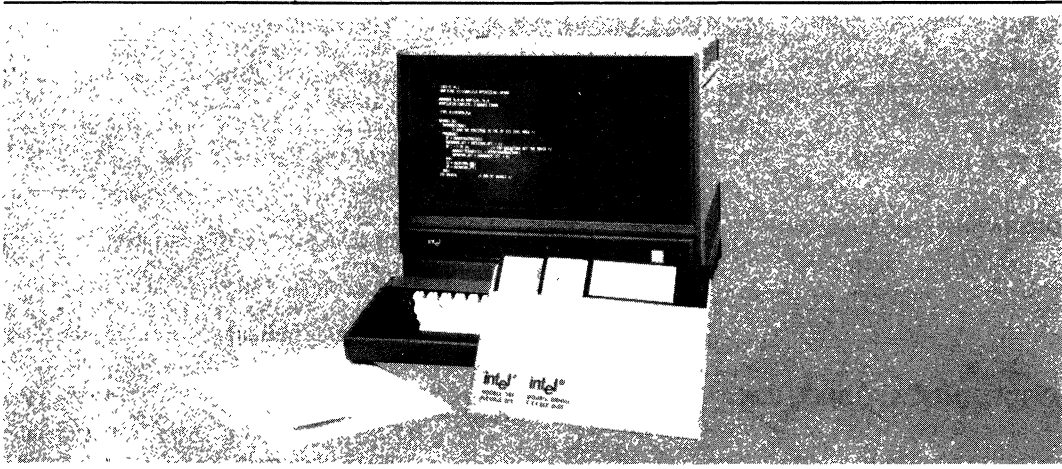
- **Systems programming language for the protected virtual address mode iAPX 286.**
- **Upward compatible with PL/M 86 and PL/M 80 assuring software portability**
- **Enhanced to support design of protected, multi-user, multi-tasking, virtual memory operating system software**
- **Advanced, structured system implementation language for algorithm development**
- **Produces relocatable object code which is linkable to object modules generated by all other iAPX 286 language translators**
- **Multiple levels of optimization**
- **Resident on Intel microcomputer development systems (Series III, IV)**

PL/M 286 is a powerful, structured, high-level system implementation language for the development of system software for the protected virtual address mode iAPX 286. PL/M 286 has been enhanced to utilize iAPX 286 features—memory management and protection—for the implementation of multi-user, multi-tasking virtual memory operating systems.

PL/M 286 is upward compatible with PL/M 86 and PL/M 80. Existing systems software can be re-compiled with PL/M 286 to execute in protected virtual address mode on the iAPX 286.

PL/M 286 is the high-level alternative to assembly language programming on the iAPX 286. For the majority of iAPX 286 system programs, PL/M 286 provides the features needed to access and to control efficiently the underlying iAPX 286 hardware and consequently it is the cost-effective approach to develop reliable, maintainable system software.

The PL/M 286 compiler has been designed to efficiently support all phases of software development. Features such as a built-in syntax checker, multiple levels of optimization, virtual symbol table and four models of program size and memory usage for efficient code generation provide the total program development support needed.



## FEATURES

Major features of the Intel PL/M 286 compiler and programming language include:

### Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible by clearly defining the scope of user variables (local to a private procedure, for example).

The use of modules and procedures to break down a large problem leads to productive software development. The PL/M 286 implementation of block structure allows the use of REENTRANT procedures, which are especially useful in system design.

### Language Compatibility

PL/M 286 object modules are compatible with object modules generated by all other 286 translators. This means that PL/M programs may be linked to programs written in any other 286 language.

Object modules are compatible with In-Circuit Emulators; DEBUG compiler control provides the In-Circuit Emulators with full symbolic debugging capabilities.

PL/M 286 language is upward compatible with PL/M 86 and PL/M 80 so that application programs may be easily ported to run on the protected mode iAPX 286.

### Supports Seven Data Types

PL/M makes use of seven data types for various applications. These data types range from one to four bytes and facilitate various arithmetic, logic, and addressing functions:

- Byte: 8-bit unsigned number
- Word: 16-bit unsigned number
- Dword: 32-bit unsigned number
- Integer: 16-bit signed number
- Real: 32-bit floating-point number
- Pointer: 16-bit or 32-bit memory address indicator
- Selector: 16-bit pointer base.

Another powerful facility allows the use of BASED variables which permit run-time mapping of var-

iables to memory locations. This is especially useful for passing parameters, relative and absolute addressing, and dynamic memory allocation.

### Two Data Structuring Facilities

In addition to the seven data types and based variables, PL/M supports two powerful data structuring facilities. These help the user to organize data into logical groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of both: Arrays of structures or structures of arrays.

### Numerics Support

PL/M programs that use 32-bit REAL data are executed using the 80287 Numeric Data Processor for high performance. All floating-point operations supported by PL/M are executed on the 80287 according to the IEEE floating-point standard. PL/M 286 programs can use built-in functions and predefined procedures—INIT\$REAL\$MATH\$UNIT, SET\$REAL\$MODE, GET\$REAL\$ERROR, SAVE\$REAL\$STATUS, RESTORE\$REAL\$STATUS—to control the operation of the 80287 within the scope of the language.

### Built-In String Handling Facilities

The PL/M 286 language contains built-in functions for string manipulation. These byte and word functions perform the following operations on character strings: MOVE, COMPARE, TRANSLATE, SEARCH, SKIP, and SET.

### Built-In Port I/O

PL/M 286 directly supports input and output from the iAPX 286 ports for single BYTE and WORD transfers. For BLOCK transfers, PL/M 286 programs can make calls to predefined procedures.

### Interrupt Handling

PL/M 286 has the facility for generating and handling interrupts on the iAPX 286. A procedure may be defined as an interrupt handler through use of the INTERRUPT attribute. The compiler will then generate code to save and restore the processor status on each execution of the user-defined

interrupt handler routine. The PL/M statement CAUSE\$INTERRUPT allows the user to trigger a software interrupt from within the program.

## Protection Model

PL/M 286 supports the implementation of protected operating system software by providing built-in procedures and variables to access the protection mechanism of the iAPX 286. Predefined variables—TASK\$REGISTER, LOCAL\$TABLE, MACHINE\$STATUS, etc.—allow direct access and modification of the protection system. Untyped procedures and functions—SAVE\$GLOBAL\$TABLE, RESTORE\$GLOBAL\$TABLE, SAVE\$INTERRUPT\$TABLE, RESTORE\$INTERRUPT\$TABLE, CLEAR\$TASK\$SWITCHED\$FLAG, GET\$ACCESS\$RIGHTS, GET\$SEGMENT\$LIMIT, SEGMENT\$READABLE, SEGMENT\$WRITABLE, ADJUST\$RPL—provide all the facilities needed to implement efficient operating system software.

## Compiler Controls

The PL/M 286 compiler offers controls that facilitate such features as:

- Optimization
- Conditional compilation
- The inclusion of additional PL/M source files from disk
- Cross-reference of symbols
- Optional assembly language code in the listing file
- The setting of overflow conditions for run-time handling.

## Addressing Control

The PL/M 286 compiler uses the SMALL, COMPACT, MEDIUM, and LARGE controls to generate optimum addressing instructions for programs. Programs of any size can be easily modularized into "subsystems" to exploit the most efficient memory addressing schemes. This lowers total memory requirements and improves run-time execution of programs.

## Code Optimization

The PL/M 286 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or "folding" of constant expressions; and short-circuit evaluation of Boolean expressions

- "Strength reductions": a shift left rather than multiply by 2; and elimination of common sub-expressions within the same block
- Machine code optimizations; elimination of superfluous branches; reuse of duplicate code; removal of unreachable code
- Optimization of based-variable operations and cross-statement load/store.

## Error Checking

The PL/M 286 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed and helpful set of programming and compilation error messages is provided by the compiler and user's guide.

## BENEFITS

PL/M 286 is designed to be an efficient, cost-effective solution to the special requirements of protected mode iAPX 286 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

### Low Learning Effort

PL/M 286 is easy to learn and use, even for the novice programmer.

### Earlier Project Completion

Critical projects are completed much earlier than otherwise possible because PL/M 286, a structured high-level language, increases programmer productivity.

### Lower Development Cost

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

### Increased Reliability

PL/M 286 is designed to aid in the development of reliable software (PL/M 286 programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in

systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

### **Easier Enhancements and Maintenance**

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

### **Cost-Effective Alternative to Assembly Language**

PL/M 286 programs are code efficient. PL/M 286 combines all of the benefits of a high-level language (ease of use, high productivity) with the ability to access the iAPX 286 architecture. This includes language features for control of the iAPX 286 protection mechanism. Consequently, for the development of systems software, PL/M 286 is the cost-effective alternative to assembly language programming.

---

## **SPECIFICATIONS**

### **Operating Environment**

Intel Microcomputer Development System (Series III/Series IV)

### **Documentation Package**

PL/M 286 User's Guide

---

## **ORDERING INFORMATION**

<b>Part Number</b>	<b>Description</b>
iMDX 323	PL/M 286 Software Package

Requires Software License

---

## **SUPPORT:**

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and Monthly Technical Newsletters are available.



## **VAX\*/VMS\* RESIDENT iAPX-86/88/186 SOFTWARE DEVELOPMENT PACKAGES**

- **Executes on DEC VAX\* Minicomputer under VMS\* Operating System to translate PL/M-86, Pascal-86 and ASM-86 Programs for iAPX-86, 88 and 186 Microprocessors.**
- **Packages include Pascal-86; PL/M-86; ASM-86; Link and Relocation Utilities; OH-86 Absolute Object Module to Hexadecimal Format Converter; and Library Manager Program.**
- **Output linkable with Code Generated on Intellec® Development Systems.**

The VAX/VMS Resident Software Development Packages contain software development tools for the iAPX-86, 88, and 186 microprocessors. The package lets the user develop, compile, maintain libraries, and link and locate programs on a VAX running the VMS operating system. The translator output is object module compatible with programs translated by the corresponding version of the translator on an Intellec Development System.

Three packages are available:

1. An ASM-86 Assembler Package which includes the Assembler, the Link Utility, the Locate Utility, the absolute object to hexadecimal format conversion utility and the Library Manager Program.
2. A PL/M-86 Compiler Package which contains the PL/M-86 Compiler and Runtime Support Libraries.
3. A Pascal-86 Compiler Package which contains the Pascal-86 Compiler and Runtime Support Libraries.

The VAX/VMS resident development packages and the Intellec Development System development packages are built from the same technology base. Therefore, the VAX/VMS resident development packages and the Intellec Development System development packages are very similar.

Version numbers can be used to identify features correspondence. The VAX/VMS resident development packages will have the same features as the Intellec Development System product with the same version number.

Support for the iAPX-186 processor will be provided as an update to the iAPX-86, 88 software.

The object modules produced by the translators contain symbol and type information for programming debugging using ICE™ translators and/or the PSCOPE debugger. For final production version, the compiler can remove this extra information and code.

\*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation

---

## **VAX\*-PL/M-86/88/186 SOFTWARE PACKAGE**

- **Executes on VAX\* Minicomputer Under the VMS\* Operating System**
- **Supports 16-Bit Signed Integer and 32-Bit Floating Point Arithmetic in Accordance with IEEE Proposed Standard**
- **Easy-To-Learn Block-Structured Language Encourages Program Modularity**
- **Produces Relocatable Object Code Which is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX\* or Intellec® Development Systems**
- **Code Optimization Assures Efficient Code Generation and Minimum Application Memory Utilization**
- **Built-In Syntax Checker Doubles Performance for Compiling Programs Containing Errors**
- **Source Input/Object Output Compatible with PL/M-86 Hosted on an Intellec® Development System**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**

Like its counterpart for MCS®-80/85 program development, and Intellec® hosted iAPX-86 program development, VAX-PL/M-86 is an advanced, structured high-level programming language. The VAX-PL/M-86 compiler was created specifically for performing software development for the Intel iAPX-86, 88, and 186 Microprocessors.

PL/M is a powerful, structured, high-level system implementation language in which program statements can naturally express the program algorithm. This frees the programmer to concentrate on the logic of the program without concern for burdensome details of machine or assembly language programming (such as register allocation, meanings of assembler mnemonics, etc.).

The VAX-PL/M-86 compiler efficiently converts free-form PL/M language statements into equivalent iAPX-86/88/186 machine instructions. Substantially fewer PL/M statements are necessary for a given application than if it were programmed at the assembly language or machine code level.

The use of PL/M high-level language for system programming, instead of assembly language, results in a high degree of engineering productivity during project development. This translates into significant reductions in initial software development and follow-on maintenance costs for the user.



---

## **VAX\*-PASCAL-86/88 SOFTWARE PACKAGE**

- **Executes on VAX\* Minicomputer Under the VMS\* Operating System**
- **Produces Relocatable Object Code Which Is Linkable to All Other Intel 8086 Object Modules, Generated on Either a VAX\* or Intellec® Development Systems**
- **ICE™, PSCOPE Symbolic Debugging Fully Supported**
- **Implements REALMATH for Consistent and Reliable Results**
- **Supports iAPX-86/20, 88/20 Numeric Data Processors**
- **Strict Implementation of ISO Standard Pascal**
- **Useful Extensions Essential for Micro-computer Applications**
- **Separate Compilation with Type-Checking Enforced Between Pascal Modules**
- **Compiler Option to Support Full Run-Time Range-Checking**
- **Source Input/Object Output Compatible with Pascal-86 Hosted on a Intellec Development System**

VAX-PASCAL-86 conforms to and implements the ISO Pascal standard. The language is enhanced to support microcomputer applications with special features, such as separate compilation, interrupt handling and direct port I/O. Other extensions include additional data types not required by the standard and miscellaneous enhancements such as an allowed underscore in names, an OTHERWISE clause in CASE construction and so forth. To assist the development of portable software, the compiler can be directed to flag all non-standard features.

The VAX-PASCAL-86 compiler runs on the Digital Equipment Corporation VAX under the VMS Operating System. A well-defined I/O interface is provided for run-time support. This allows a user-written operating system to support application programs on the target system as an alternate to the development system environment. Program modules compiled under PASCAL-86 are compatible and linkable with modules written in PL/M-86, and ASM-86. With a complete family of compatible programming languages for the iAPX-86, 88, and 186 one can implement each module in the language most appropriate to the task at hand.

---

## **VAX\*-iAPX-86/88/186 MACRO ASSEMBLER**

- **Executes on VAX\* Minicomputer Under The VMS\* Operating System**
- **Produces Relocatable Object Code Which Is Linkable to All Other Intel iAPX-86/88/186 Object Modules, Generated on Either a VAX\* or Intellec® Development Systems**
- **Powerful and Flexible Text Macro Facility with Three Macro Listing Options to Aid Debugging**
- **Highly Mnemonic and Compact Language, Most Mnemonics Represent Several Distinct Machine Instructions**
- **“Strongly Typed” Assembler Helps Detect Errors at Assembly Time**
- **High-Level Data Structuring Facilities Such as “STRUCTURES” and “RECORDS”**
- **Over 120 Detailed and Fully Documented Error Messages**
- **Produces Relocatable and Linkable Object Code**
- **Source Input/Object Output Compatible with ASM-86 hosted on an Intellec Development System**

VAX-ASM-86 is the “high-level” macro assembler for the iAPX-86/88/186 assembly language. VAX-ASM-86 translates symbolic iAPX-86/88/186 assembly language mnemonics into iAPX-86/88/186 relocatable object code.

VAX-ASM-86 should be used where maximum code efficiency and hardware control is needed. The iAPX-86/88/186 assembly language includes approximately 100 instruction mnemonics. From these few mnemonics the assembler can generate over 3,800 distinct machine instructions. Therefore, the software development task is simplified, as the programmer need know only 100 mnemonics to generate all possible iAPX-86/88/186 machine instructions. VAX-ASM-86 will generate the shortest machine instruction possible given no forward referencing or given explicit information as to the characteristics of forward referenced symbols.

VAX-ASM-86 offers many features normally found only in high-level languages. The iAPX-86/88/186 assembly language is strongly typed. The assembler performs extensive checks on the usage of variable and labels. The assembler uses the attributes which are derived explicitly when a variable or label is first defined, then makes sure that each use of the symbol in later instructions conforms to the usage defined for that symbol. This means that many programming errors will be detected when the program is assembled, long before it is being debugged on hardware.

---

## **VAX\*-LIB-86**

- **Executes on VAX\* Minicomputer Under the VMS\* Operating System**
- **VAX\*-LIB-86 is a Library Manager Program which Allows You to:  
Create Specifically Formatted Files to Contain Libraries of Object Modules  
Maintain These Libraries by Adding or Deleting Modules  
Print a Listing of the Modules and Public Symbols in a Library File**
- **Libraries Can be Used as Input to VAX\*-LINK-86 Which Will Automatically Link Modules from the Library that Satisfy External References in the Modules Being Linked**
- **Abbreviated Control Syntax**

Libraries aid in the job of building programs. The library manager program VAX-LIB-86 creates and maintains files containing object modules. The operation of VAX-LIB-86 is controlled by commands to indicate which operation VAX-LIB-86 is to perform. The commands are:

**CREATE:** creates an empty library file  
**ADD:** adds object modules to a library file  
**DELETE:** deletes modules from a library file  
**LIST:** lists the module directory of library files  
**EXIT:** terminates the LIB-86 program and returns control to VMS

When using object libraries, the linker will call only those object modules that are required to satisfy external references, thus saving memory space.

## **VAX-OH-86**

- **Executes on VAX\* Minicomputer Under the VMS\* Operating System**
- **Converts an iAPX 86/88/186 Absolute Object Module to Symbolic Hexadecimal Format**
- **Facilitates Preparing a file for Loading by Symbolic Hexadecimal Loader (e.g. iSBC™ Monitor SDK-86 Loader), or Universal PROM Mapper**
- **Converts an Absolute Module to a More Readable Format that can be Displayed on a CRT or Printed for Debugging**

The VAX-OH-86 utility converts an 86/88 absolute object module to the hexadecimal format. This conversion may be necessary for later loading by a hexadecimal loader such as the iSBC 86/12 monitor or the Universal PROM Mapper. The conversion may also be made to put the module in a more readable format that can be displayed or printed.

The module to be converted must be in absolute form; the output from VAX-LOC-86 is in absolute format.

## VAX\*-LINK-86

- Executes on VAX\* Minicomputer Under the VMS\* Operating System
- Automatic Combination of Separately Compiled or Assembled 86/88/186 Programs Into a Relocatable Module, Generated on Either a VAX or an Intellec® Development System
- Automatic Selection of Required Modules from Specified Libraries to Satisfy Symbolic References
- Extensive Debug Symbol Manipulation, allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively
- Automatic Generation of a Summary Map Giving Results of the LINK-86 Process
- Abbreviated Control Syntax
- Relocatable modules may be Merged into a Single Module Suitable for Inclusion in a Library
- Supports "Incremental" Linking
- Supports Type Checking of Public and External Symbols

VAX-LINK-86 combines object modules specified in the VAX-LINK-86 input list into a single output module. VAX-LINK-86 combines segments from the input modules according to the order in which the modules are listed.

VAX-LINK-86 will accept libraries and object modules built from VAX-PL/M-86, VAX-PASCAL-86, VAX-ASM-86, or any other Intel translator generating 8086 Relocatable Object Modules, such as the Series III resident translators.

Support for incremental linking is provided since an output module produced by VAX-LINK-86 can be an input to another link. At each stage in the incremental linking process, unneeded public symbols may be purged.

VAX-LINK-86 supports type checking of PUBLIC and EXTERNAL symbols reporting a warning if their types are not consistent.

VAX-LINK-86 will link any valid set of input modules without any controls. However, controls are available to control the output of diagnostic information in the VAX-LINK-86 process and to control the content of the output module.

VAX-LINK-86 allows the user to create a large program as the combination of several smaller, separately compiled modules. After development and debugging of these component modules the user can link them together, locate them using VAX-LOC-86 and enter final testing with much of the work accomplished.

---

## **VAX\*-LOC-86**

- **Executes on the VAX\* Minicomputer Under the VMS\* Operating System**
- **Automatic Generation of a Summary Map Giving Starting Address, Segment Addresses and Length, and Debug Symbols and their Addresses**
- **Extensive Capability to Manipulate the Order and Placement of Segments in 8086/8088 Memory**
- **Abbreviated Control Syntax**
- **Automatic and Independent Relocation of Independent Relocation of Segments. Segments May be Relocated to Best Match Users Memory Configuration**
- **Extensive Debug Symbol Manipulation, Allowing Line Numbers, Local Symbols, and Public Symbols to be Purged and Listed Selectively**

Relocatability allows the programmer to code programs or sections of programs without having to know the final arrangement of the object code in memory.

VAX-LOC-86 converts relative addresses in an input module in iAPX-86/88/186 object module format to absolute addresses. VAX-LOC-86 orders the segments in the input module and assigns absolute addresses to the segments. The sequence in which the segments in the input module are assigned absolute addresses is determined by their order in the input module and the controls supplied with the command.

VAX-LOC-86 will relocate any valid input module without any controls. However, controls are available to control the output of diagnostic information in the VAX-LOC-86 process, to control the content of the output module, or both.

The program you are developing will almost certainly use some mix of random access memory (RAM), read-only memory (ROM), and/or programmable read-only memory (PROM). Therefore, the location of your program affects both cost and performance in your application. The relocation feature allows you to develop your program and then simply relocate the object code to suit your application.

---

### **SPECIFICATIONS**

#### **Operating Environment**

#### **Required Hardware**

VAX\* 11/780, 11/782, 11/750, or 11/730  
9 Track Magnetic Tape Drive, 1600 BPI

#### **Required Software**

VMS Operating System V3.0 or Later. All of the development packages are delivered as unlinked VAX object code which can be linked to VMS as designed for the system where the development package is to be used. VMS command files to perform the link are provided.

#### **Documentation Package**

iAPX-86, 88 Development Software Installation Manual and User's Guide for VAX/VMS, Order number 121950-001

#### **Shipping Media**

9 Track Magnetic Tape 1600 bpi

### **ORDERING INFORMATION**

#### **Part Number Description**

iMDX-341VX	VAX-ASM-86, VAX-LINK-86, VAX-LOC-86, VAX-LIB-86, VAX-OH-86, Package
iMDX-343VX	VAX-PLM-86 Package
IMDX-344VX	VAX-PASCAL-86 Package

**REQUIRES SOFTWARE LICENSE**

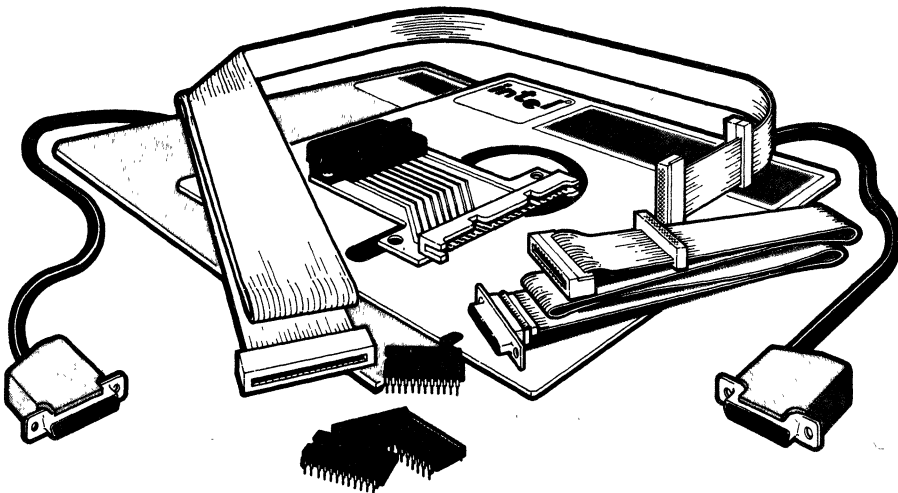
\*VAX, DEC, and VMS are trademarks of Digital Equipment Corporation



## iSDM™ 86 SYSTEM DEBUG MONITOR

- Supports target system debugging for iSBC® /iAPX 86, 88, 186 and 188-based applications
- Provides interactive debugging commands including single-step code execution and symbolic displays of results
- Supports 8087 Numeric Processor Extension (NPX) for high-speed math applications
- Allows building of custom commands through the Command Extension Interface (CEI)
- Supports application access to ISIS-II files
- Provides program load capability from an Intellec® Development System
- Contains configuration facilities which allow an applications bootstrap from iRMX™ 86 and 88 file compatible peripherals
- Modular to allow use from an Intellec Development System or from a stand-alone terminal

The Intel iSDM™ 86 System Debug Monitor package contains the necessary hardware, software, cables, EPROMs and documentation required to interface, through a serial or parallel connection, an iSBC® 86/05, 86/12A, 86/14, 86/30, 88/25, 88/40, 88/45, 186/03, 186/51, 188/48, or iAPX 86, 88, 186 or 188 target system to an MDS 800, Series II or Series III Intellec® Microcomputer Development System for execution and interactive debugging of applications software on the target system. The Monitor can: load programs into the target system; execute the programs instruction by instruction or at full speed; set breakpoints; and examine/modify CPU registers, memory content, and other crucial environmental details. Additional custom commands can be built using the Command Extension Interface (CEI). The Monitor supports the OEM's choice of the iRMX™ 86 Operating System, the iRMX 88 Real-Time Multitasking Executive or a custom system for the target application system. OEM's may utilize any iRMX 86, 88 supported target system peripheral for a bootstrap of the application system or have full access to the ISIS-II files of the Intellec System.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iMMX, iRMX, iSBC, iSBX, iSXM, MULTIBUS, Multichannel and MULTIMODULE. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied. Information contained herein supercedes previously published specifications on these devices from Intel.

## FUNCTIONAL DESCRIPTION

### Overview

The iSDM 86 Monitor extends the software development capabilities of the Intellec system so the user can effectively develop applications to ensure timely product availability.

The iSDM 86 package consists of four parts:

- The loader program
- The iSDM 86 Monitor
- The Command Extension Interface (CEI)
- The ISIS-II Interface

The user can use the iSDM 86 package to load programs into the target system from the development system, execute programs in an instruction-by-instruction manner, and add custom commands through the command extension interface. The user also has the option of using just the iSDM 86 Monitor and the CEI in a stand-alone application, without the use of an Intellec development system.

### Powerful Debugging Commands

The iSDM 86 Monitor contains a powerful set of commands to support the debugging process. Some of the

features included are: bootstrap of application software; selective execution of program modules based on break-points or single stepping requests; examination, modification and movement of memory contents; examination and modification of CPU registers, including NPX registers. All results are displayed in clearly understandable formats. Refer to Table 1 for a more detailed list of the iSDM 86 monitor commands.

### Numeric Data Processor Support

Arithmetic applications utilizing the 8087 Numeric Processor Extension (NPX) are fully supported by the iSDM 86 Monitor. In addition to executing applications with the full NPX performance, users may examine and modify the NPX's registers using decimal and real number format.

This feature allows the user to feel confident that correct and meaningful numbers are entered for the application without having to encode and decode complex real, integer, and BCD hexadecimal formats.

### Command Extension Interface (CEI)

The Command Extension Interface (CEI) allows the addition of custom commands to the iSDM 86 Monitor commands. The CEI consists of various procedures that can be used to generate custom commands. Up to three custom commands (or sets of commands) can be added.

Table 1. Monitor Commands

Command	Function
B	<b>Bootstrap</b> application program from target systems peripheral device
C	<b>Compare</b> two memory blocks
D	<b>Display</b> contents of memory block
E*	<b>Exit</b> from loader program to ISIS-II Interface
F	<b>Find</b> specified constant in a memory block
G	<b>Execute</b> application program
I	<b>Input</b> and display data obtained from input port
L*	<b>Load</b> absolute Intellec® object file into target system memory
M	<b>Move</b> contents of memory block to another location
N	<b>Display and execute</b> single instruction
O	<b>Output</b> data to output port
P	<b>Print</b> values of literals
R*	<b>Load and execute</b> absolute Intellec® object file in target system memory
S	<b>Display and (optionally) modify</b> contents of memory
T*	<b>Transfer</b> block of memory to an Intellec® file
U,V,W	<b>User</b> defined custom commands extensions
X	<b>Examine and (optionally) modify</b> CPU and NPX registers

\* Commands require an attached Series II/Series III.

to the monitor without programming new EPROMs or changing the monitor's source code.

### ISIS-II interface

The ISIS-II interface consists of libraries which contain interfaces to ISIS-II I/O calls. A program running on an iAPX 86, 88, 186 or 188-based system can use the ISIS-II interface and access the individual ISIS-II I/O calls. The interface allows the inclusion of these calls into the program; however, most of the calls require a Series II/ Series III system. Table 2 contains a summary of the major I/O calls and parameters.

### Program Load Capability

The iSDM 86 loader allows the loading of iAPX 86, 88, 186 or 188-based programs into the target system. It executes on a Intellec Microcomputer Development System and communicates with the target system through a serial or a parallel load interface. If a Series II/ Series III system containing an Intel I/O expansion board is being used, the board can be used as a fast parallel load interface, freeing up the UPP port for application use.

### Configuration Facility

The monitor contains a full set of configuration facilities which allow it to be carefully tailored to the requirements

of the target system. Pre-configured EPROM-resident monitors are supplied by Intel for the iSBC 86/05, 86/12A, 86/14, 86/30, 88/45, 186/03, 186/51, and 188/48 boards. The monitor must be configured by the user for the iSBC 88/25, 88/40 boards and for other iAPX 86, 88, 186, 188 applications. iRMX 86 and iRMX 88 system users may use the configuration facilities to include the iAPX 86, 88 Bootstrap Loader (V5.0 or newer) in the monitor.

### Variety of Connections Available

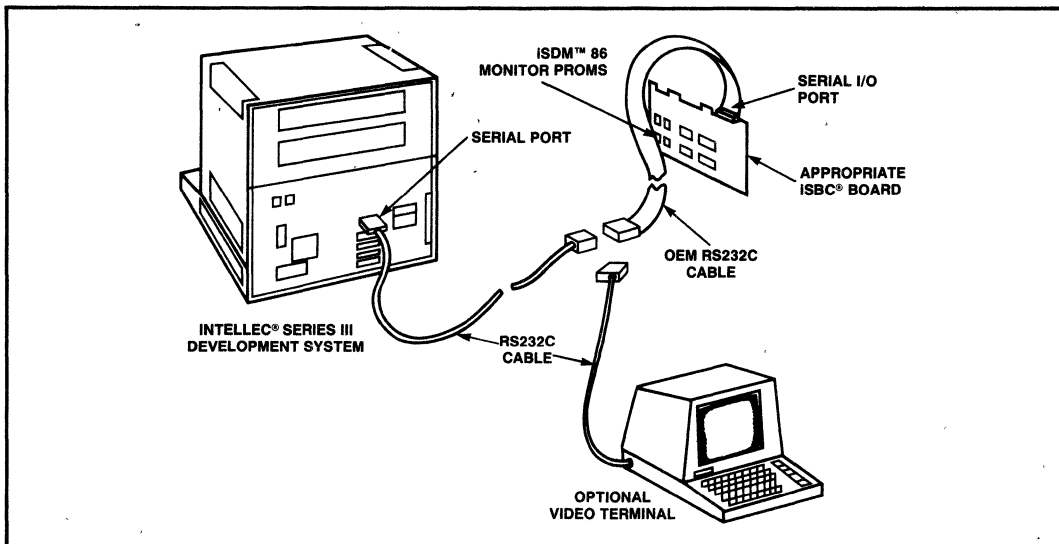
The physical interface between the Intellec Microcomputer Development System and the target system can be established in one of three ways. The systems can be connected via a serial link, a parallel link or a fast parallel link. The fast parallel link requires the use of an iSBC 108(A), 116(A), 517 or 519 I/O expansion board in the Intellec system and is only available for connections with the Series II/ Series III systems. The cabling arrangement is different depending upon the development system being used. Figure 1 displays the cable connections needed between an Intellec Series III system and a target system for a serial interface.

The iSDM 86 Monitor does not require the use of a development system. The monitor can be used by simply attaching a stand-alone terminal to the target system. Figure 1 also displays the cable connections needed for this arrangement.

**Table 2. Routines for ISIS-II Services Available to Target System Applications**

Routine	Target System Function
ATTRIB	Changes to ISIS-II file <b>attribute</b>
CI	Returns a character <b>input</b> from the <b>console</b>
CO	Transfers a character for <b>console output</b>
CLOSE	<b>Closes</b> an opened ISIS-II file
DELETE	<b>Deletes</b> the specified ISIS-II file
DQ\$CFG	Returns information about monitor's communication link and type
ERROR	Displays an <b>error</b> message on the Intellec® console
EXIT	<b>Exits</b> to the target system monitor
LOAD	<b>Loads</b> target system memory with ISIS-II object code file
OPEN	<b>Opens</b> an ISIS-II file for access
READ	<b>Reads</b> up to 4096 bytes from an ISIS-II file to memory
RENAME	<b>Renames</b> an ISIS-II disk file
SEEK	<b>Seeks</b> to the specified ISIS-II file location
WRITE	<b>Writes</b> up to 4096 bytes from memory to an ISIS-II file





**Figure 1. Typical iSDM™ 86 Serial Connection Environment**

## SPECIFICATIONS

### Development System Environment

The Intellec Microcomputer Development System may be utilized for application program development and, if used, requires the following to support the iSDM 86 package:

- 48 Kbytes memory
- Double density or single density diskette subsystem
- ISIS-II Operating System and associated language translators

### iAPX 86, 88, 186, 188 TARGET SYSTEM ENVIRONMENT

To support the iSDM 86 package, the target system must contain the following:

- 2K read-write memory beginning at location 0H
- 16K read-only memory beginning at location FC000H
- For Parallel link:
  - 8255A Programmable Peripheral Interface

- For Serial link:
  - 8251A USART or 8274 Multiprotocol Serial Controller, and 8253/4 or 80130 or iAPX 186/188 timer, or
  - 82530 Serial Communications Controller, including 82530 timer

### Hardware

- Supported ISBC Microcomputers:
 

iSBC 86/05	Single Board Computer
iSBC 86/12A	Single Board Computer
iSBC 86/14	Single Board Computer
iSBC 86/30	Single Board Computer
iSBC 88/25	Single Board Computer
iSBC 88/40	Single Board Computer
iSBC 88/45	Single Board Computer
iSBC 186/03	Single Board Computer
iSBC 186/51	Single Board Computer
iSBC 188/48	Single Board Computer
- Supported ISBX MULTIMODULE™ Boards:
 

ISBX 350 Parallel I/O MULTIMODULE Board
ISBX 351 Serial I/O MULTIMODULE Board

**iSDM™ 86 Package Contents**
**Cables:**

- 1 — Parallel I/O Cable (upload/download)
- 2 — RS232 Cables

**Adaptors:**

- 1 — Parallel Status Adaptor
- 1 — Parallel Adaptor

**I/O Drivers and Terminators:**

- 4 — Pull-up Resistor Packs
- 4 — Pull-up/down Resistor Packs
- 4 — Line Driver Packs

**Interface and Execution Software Diskettes:**

- 1 — Single Density, ISIS Compatible
- 1 — Double Density, ISIS Compatible

**System Monitor EPROMs:**

Microcomputer	EPROM
iSBC® 86/05 iSBC® 86/12A iSBC® 86/14 iSBC® 86/30	Four 2732A EPROMs
iSBC® 88/45	Two 2764 EPROMs
iSBC® 186/03 iSBC® 186/51	Two 2764 EPROMs
iSBC® 188/48	Two 2764 EPROMs

**Reference Manual (Supplied):**

**146165-001** — iSDM 86 System Debug Monitor Reference Manual

**ORDERING INFORMATION**
**Part Number Description**

**iSDM 86** Intellec to target system interface and target system monitor, suitable for use on iSBC 86, 88, 186, 188 computers, or other iAPX 86, 88, 186, 188 microcomputers. Package includes cables, EPROMs, software and operator manual.

The iSDM 86 package includes SPR Service for 90 days after shipment.

As with all Intel Software, purchase of any of these options requires execution of a standard Intel Master Software License.

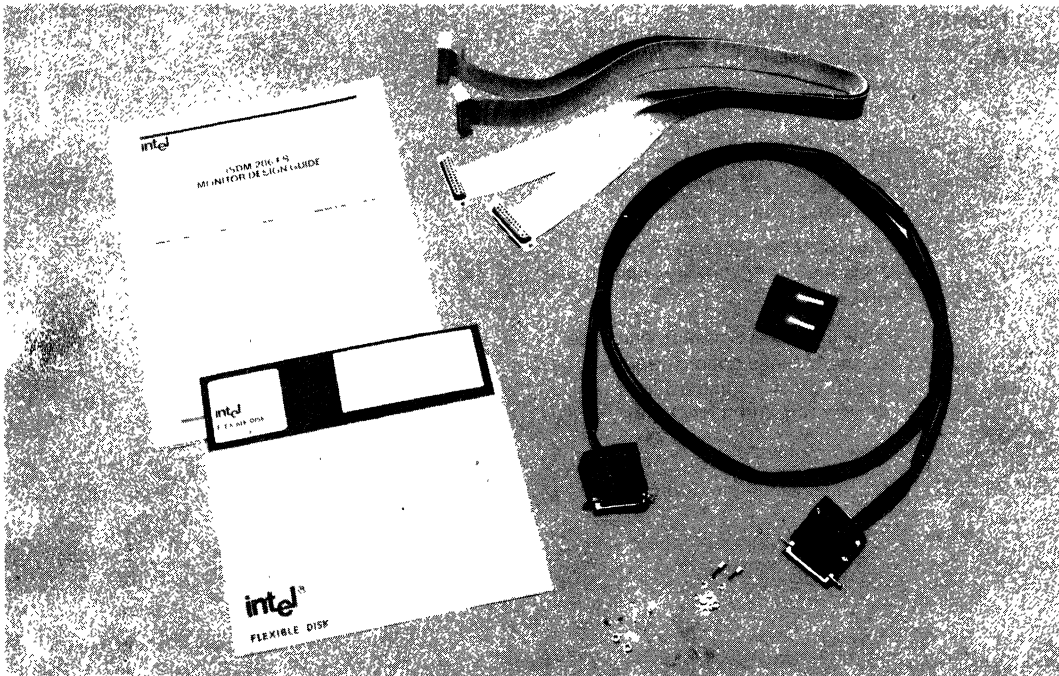
**iSDM 86 RO** Object Software

**iSDM 86 BSR** Machine Readable Source

## iSDM™ 286 iAPX 286 SYSTEM DEBUG MONITOR

- Development support of iSBC® 286-and iAPX 286-based applications
- Real Address Mode (RAM) and Protected Virtual Address Mode (PVAM) support
- Universal Development Interface (UDI) support via development system connection
- Underlying debugging tool for iRMX™ 286R applications
- Supports 80287 Numeric Processor Extension (NPX) for high-speed math applications
- Program load capability from Intellec® Series III Development Systems
- Bootstrap Loader for iRMX™ 286R, 86, and 88 file compatible peripherals
- iAPX 286 single step operation allowed

The Intel iSDM™ 286 System Debug Monitor package contains the necessary hardware, software, cables, EPROMs, and documentation required to interface in iSBC® 286 board or iAPX 286 component applications to an Intellec® Series III through a high-speed link. The System Debug Monitor supports an OEM's choice of the iRMX™ 286R Real-Time Multitasking Operating System or custom operating system, with debugging tools to examine CPU registers, memory content, CPU descriptor tables, and other crucial environmental details. The Monitor also allows programs to access files on the development system via the internal UDI support and the serial communication link.



## FUNCTIONAL DESCRIPTION

### Overview

The iSDM 286 System Debug Monitor provides programmers of iAPX 286-based applications with the debugging tools needed to test new applications ranging from single-user systems to complex operating systems. Programmers are given direct access to both the Real Address (ram) and Protected Virtual Address (pvam) Modes of the CPU via a simple terminal interface, or via an Intellec Series III Development System.

### Universal Development Interface

Any iRMX 86, Series III, or other UDI-based application can be supported by the iSDM 286 Monitor. The Monitor emulates many of the UDI calls (ram or pvam), and passes all requests for a file system to the host development station. UDI applications such as compilers and other programs available from Independent Software Vendors can be tested in the target iAPX 286 environment immediately.

### Powerful Debugging Commands

A powerful set of user functions includes commands to:

- Examine and Modify CPU Registers
- Examine, Modify, and Move memory locations
- Symbolic reference to variable names
- Find and compare memory contents
- Set program breakpoints
- Bootstrap load application software

Single-step CPU operation

Change between Real Address Mode and Protected Virtual Address Mode

### Formatted Displays

The iSDM 286 Monitor formats all iAPX 286 pre-defined data structures into clearly understandable displays. This display gives programmers a formatted view of CPU registers such as LDTs, GDTs, IDTs, Segment Selectors, and Task State Segments — not just a series of unconnected digits.

### Numeric Data Processor Support

In addition to executing 80287 Numeric Processor Extension (NPX) applications with full NPX performance, programmers may examine and modify NPX registers using decimal and real number format. Any location in memory known to contain numeric values in standard real format (IEEE P754) may be examined or modified using normal decimal notation. In this manner programmers may feel confident that correct and meaningful numbers are available to applications without having to encode and decode complex real, integer, and BCD hexadecimal formats.

### High-Speed Serial Connection

Target application hardware is connected to the development system via a serial link capable of 19.2K baud. All control operations and UDI file manipulations occur over this link through the cables supplied. As shown in Figure 1, the serial link is supported by the iSBC 86 USART port of the development system.

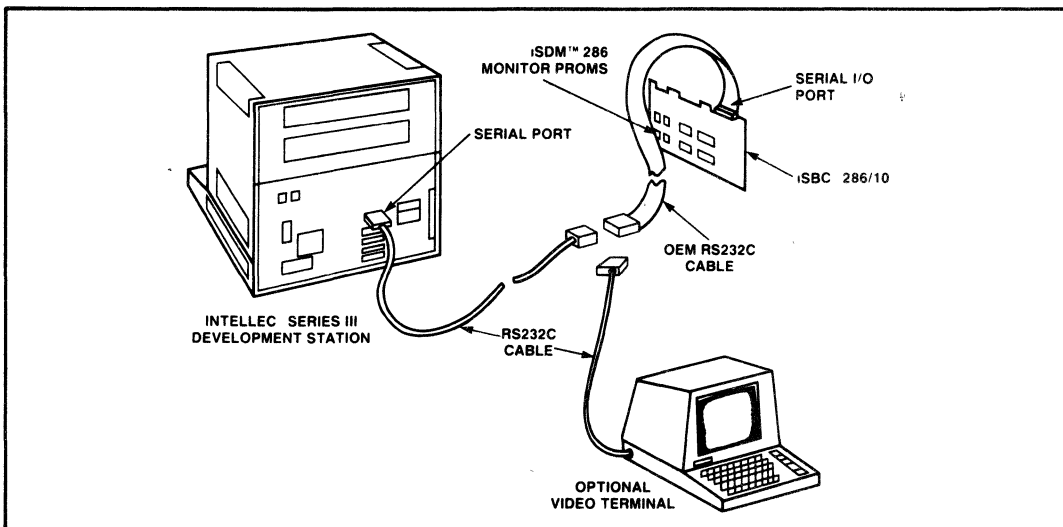


Figure 1. Typical iSDM™ 286 Environment

## SPECIFICATIONS

### Development System Environment

Intellec MDS Series III with 64 KBytes.

### Target System Environment

Any iAPX 286 system with 8274 (non-vector mode) serial link and 8254 timer, such as the iSBC 286/10 Single Board Computer. The 8259A interrupt controller is optional.

EPROMs are supplied for locations 0FF8000H through 0FFFFFFH.

---

## ORDERING INFORMATION

### Part Number Description

SDM 286 iSBC 286 and iAPX 286 System Debug Monitor package including cables, EPROMs, software, and operator manual.

Also available with the iSBC 286/10 ES Kit or with the iRMX 286R Kit.

A Software License Agreement must be or have been executed.

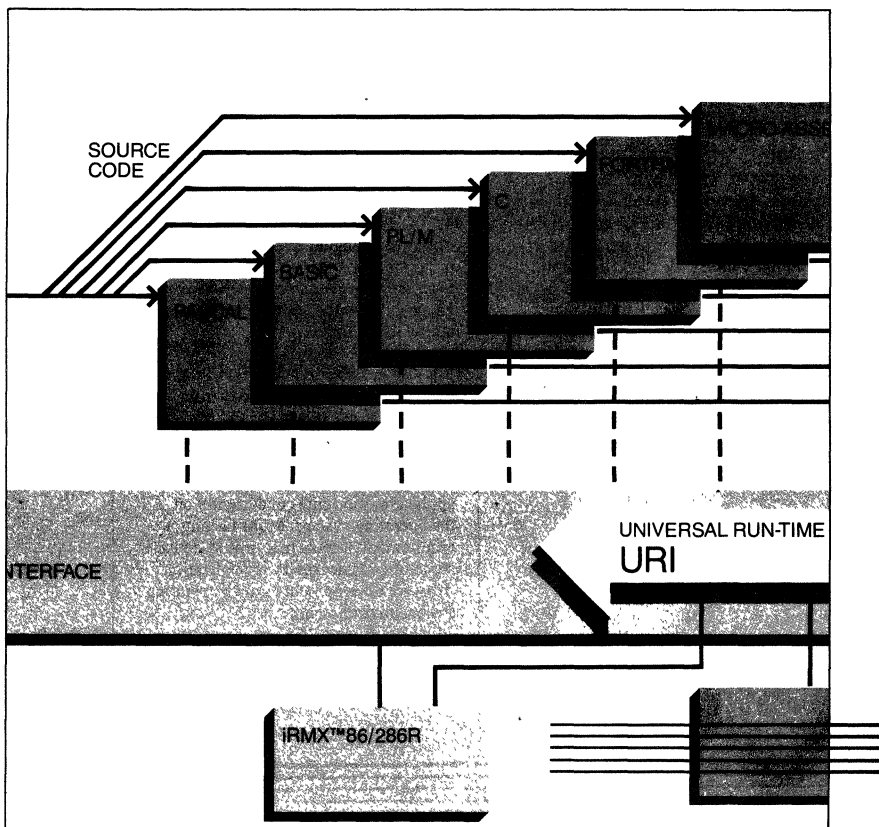
SDM 286 BSR Machine Readable source for SDM 286.

Special source code license agreement is required in addition to Software License Agreement.



## iRMX™ LANGUAGES

- Industry-standard languages and utilities for developing applications on iRMX-based systems. Includes FORTRAN, Pascal, C, BASIC, PL/M, assembler, text editor
- Complete set of utilities to create and manage object modules
- Mix languages on single application system with UDI standard
- Intel 8087 math coprocessor support
- Worldwide post-sales service and support organization



### Full Language Support for iRMX™ -Based Systems

Intel's iRMX™ 86 and 286R-based systems are completely supported by a wide variety of popular languages and utilities with which to build fast, real-time, multi-tasking applications. Included are the latest versions of FORTRAN, Pascal, BASIC, C, PL/M and Assembler for Intel's iAPX 86 and iAPX 286 processors. Previously developed applications using any of these languages port easily to iRMX-based systems with minimal source code modifications.

In addition to the wealth of languages available, iRMX-based systems are complemented by utilities with which to create and manage object modules. This latitude in configurability allows programmers to team their efforts in order to achieve a shorter development time than would otherwise be possible.

Because the high-level languages are actually resident on the iRMX-based system, OEMs can pass application software directly on to end users. End users may then tailor the OEM's system to better meet application needs by writing programs using the same languages.

### Language-Independent Application Development

Intel's Universal Run-time Interface (URI) and Object Module Format (OMF) enable several users to write different modules of an application, in different languages, then link them together.

The OMF provides users with the ability to mix languages on a single application system, affording the luxury of choosing exactly the right language tools for specific pieces of the application, rather than compromising specialized tasks for the sake of one, project-wide language.

iRMX languages are fully compatible with the Intel Series III Development System, should the user choose to develop applications on a specialized development system. Applications are easily moved to the final target system for test, debug and minor redevelopment.

### Fast, Lean Programs For Rapid Processing

The iRMX language products enable programmers to write the smallest, fastest programs available in high-level languages, due to the compiler's superior ability to optimize code.

It is also possible to make iRMX operating system calls directly from FORTRAN, PASCAL and PL/M. This means that application developers can take full advantage of the iRMX multi-tasking capability, whereby multiple applications execute concurrently on the operating system. Multi-tasking, a requirement of most real-time systems, is sometimes as necessary in application software development as in an operating system environment.

### Standardized REALMATH Support

All the iRMX languages (except BASIC and C) support the REALMATH floating point standard. This ensures universal consistency in numeric computation results and enables the user to take advantage of the Intel iAPX 86/20 and iAPX 88/20 Numeric Data Processor or iSBC® 337 MULTIMODULE™ boards, which boost performance two to four times over that possible on a mini-computer.

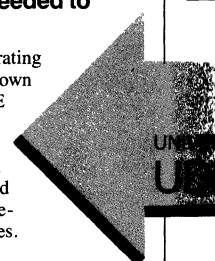
### All the Utilities Needed to Link Languages

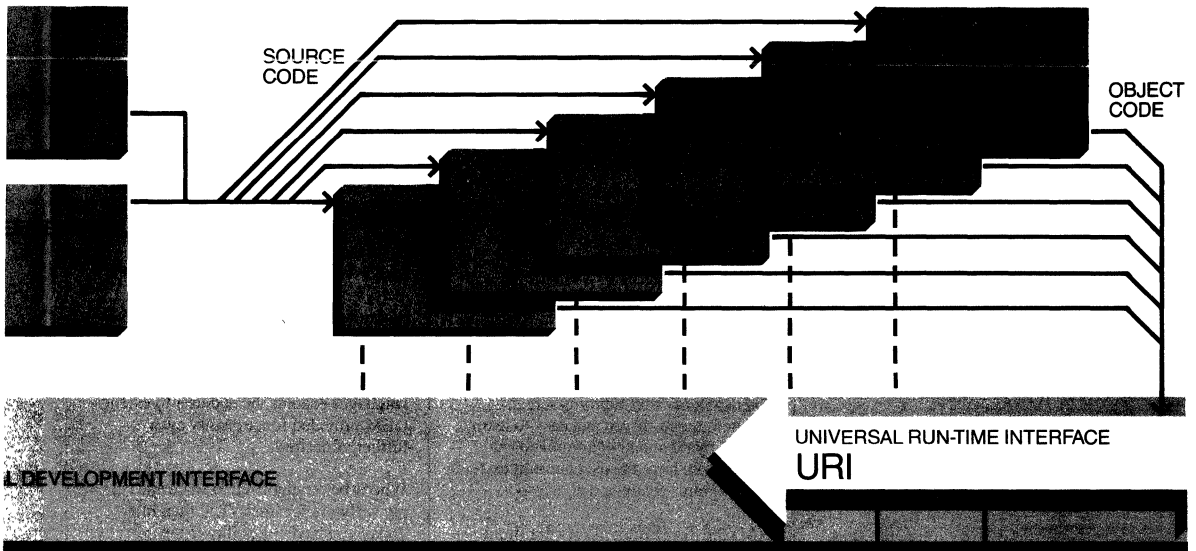
Utilities for iRMX operating systems include Intel's own EDIT, LINK, LOCATE and LIBRARIAN. The iRMX EDIT program meets the needs of both novice and sophisticated users with powerful line-oriented editing facilities.

Using the iRMX LINK program, users may link individually compiled object modules to form a single, relocatable object module. This provides the ability to merge work from several programmers into one cohesive application system.

The iRMX LOCATE utility maps relocatable object code into the processor memory segments, allowing user definition of module/memory type allocation. For example, often-used portions of an application may be mapped to (P)ROM.

The LIBRARIAN object code library manager affords easy creation, collection and maintenance of related object code to reduce the overhead of separately maintained modules.





Finally, the iRMX Assembler for the iAPX 86 and iAPX 286 processors generate extremely efficient code and invoke 8086/8087 machine instructions.

### iRMX™ 86 Pascal

iRMX Pascal meets the proposed ISO language standard and implements several microcomputer extensions. A compile-time option checks conformance to the standard, making it easy to write uniform code. Industry-standard specifications contribute to portability of application programs and provide greater reliability.

iRMX 86 Pascal supports extensions, such as an interrupt-handler and direct

port I/O extension, that allow programs to be written specifically for microcomputers. Separate module compilation allows linkage of Pascal modules with modules written in other high-level languages.

For more information on iRMX 86 Pascal see the Pascal 86 Software Package data sheet (Intel order number 400670).

### iRMX™ 86 FORTRAN

The iRMX 86 FORTRAN compiler provides total compatibility with FORTRAN

66 language standards, plus most new features provided by the FORTRAN 77 language standard (the only significant exception is complex numbers). iRMX 86 FORTRAN includes extensions specifically for microcomputer application development. Programming is simplified by relocatable object libraries, which provide run-time support for execution time activities.

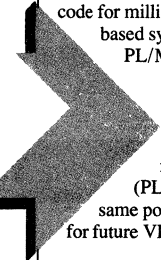
iRMX 86 FORTRAN supports the 8087 math coprocessor for the most powerful



microcomputer solution available in number-intensive applications. For more information on iRMX 86 FORTRAN see the FORTRAN 86 Software Package data sheet (Intel order number 400630).

### **iRMX™ 86 PL/M**

PL/M offers full access to microcomputer architecture while simultaneously offering all the benefits of a high-level language. Invented by Intel in 1976, PL/M 80 was the first microcomputer-specific, block-structured, high-level language available. Since then, thousands of users have generated code for millions of microcomputer-based systems using PL/M 80 and PL/M 86.



Software written for 8-bit processors (PL/M 80) are easily ported to the more powerful 16-bit (PL/M 86) environment. The same portability will be available for future VLSI.

For more information about iRMX 86 PL/M see the PL/M 86/88 Software Package data sheet (Intel order number 210689).

### **iRMX™ 86 BASIC**

Intel's offering of Microsoft BASIC is a standardized version of the most popular high-level language in the world. Existing BASIC programs are easily ported to iRMX-based systems. BASIC is an excellent pass-through language by which an OEM can offer customers the ability to write and modify their own applications.

### **iRMX™ 86 C Compiler**

The popular new programming language, C (Mark William's Company version), is fully supported on iRMX-based systems. iRMX 86 C offers both small and large

segmentation models, enabling applications to be written efficiently. The iRMX 86 C compiler combines assembly language efficiency with high-level language convenience; it can manipulate on a machine-address level while maintaining the power and speed of a structured language.

The iRMX 86 C compiler affords easy portability of existing C programs to iRMX-based systems. For more information on the iRMX C compiler see the iRMX 86 C Software Package data sheet (Intel order number 210768).

### **iRMX™ 86 Text Editor**

The iRMX 86 Text Editor is screen-oriented, menu-driven and easy to learn. Guided by the menu of commands always before him, the user can edit text and programs easily and efficiently.

iRMX 86 Text Editor allows the simultaneous edit of two files. This allows easy transferral of text between files and use of existing material in the creation of new files. Creating macros, strings of frequently-used commands, is also very simple. The editor 'remembers' the selected commands and allows the user to re-use them repeatedly.

### **Worldwide Service and Support**

All iRMX systems are completely supported by Intel's worldwide staff of trained hardware and software engineers. iRMX Language customers receive a warranty that includes Hotline Support, Software Updates, and Subscription Service.

Complete documentation is provided for all operating system and application software languages, as well as for system hardware components. An Intel system is not a collection of hardware and software pieces as much as a cohesive whole that is supported and serviced as such.

## **Intel Has Total Solutions for Real-Time Systems**

iRMX 86 and 286R are the fastest, most powerful operating systems available for multi-tasking, multi-user, real-time applications. Complemented by a wide range of industry-standard languages and utilities, the iRMX-based systems are highly flexible and configurable.

Application development for iRMX-based systems is possible at the board or the system level. OEMs can integrate functionality at the most profitable level of product design, using one system for both development and target use. Intel's choice of industry standard high-level languages enables the end user to extend OEM-provided functionality even further, if desired.

Who is better qualified to write and supply software for Intel VLSI than Intel? Today you have the ability to tap into hundreds of available application software packages, languages and utilities, peripherals and controllers and MULTIBUS® boards.

Tomorrow, and ten years down the road, you will be able to tap into the latest, high-performance VLSI—without losing today's software investment.



## Specifications

<p><b>Required Hardware</b></p> <ul style="list-style-type: none"> <li>• Any iAPX 86/286 based or iSBC 86/286 based system including Intel's System 86/300 and 286/300 family. In addition, object code from the compilers will run on iAPX 88 based systems.</li> <li>• 140KB of memory</li> <li>• Two iRMX 86 compatible floppy disks or one hard disk</li> <li>• One 8" double density or 5.25" double-density floppy disk drive for distribution of software</li> <li>• System console device</li> </ul>	<p><b>Required Software</b></p> <p>The iRMX 86 Operating System Release 5 or later including the nucleus, basic I/O system, extended I/O system and human interface</p> <p>— or —</p> <p>The iRMX 286 Operating System Release 2 or later including the nucleus, basic I/O system, extended I/O system and human interface</p> <p>Purchase of any RMX language requires signing of Intel's OEM License Agreement (OLA).</p>	
--	---	--

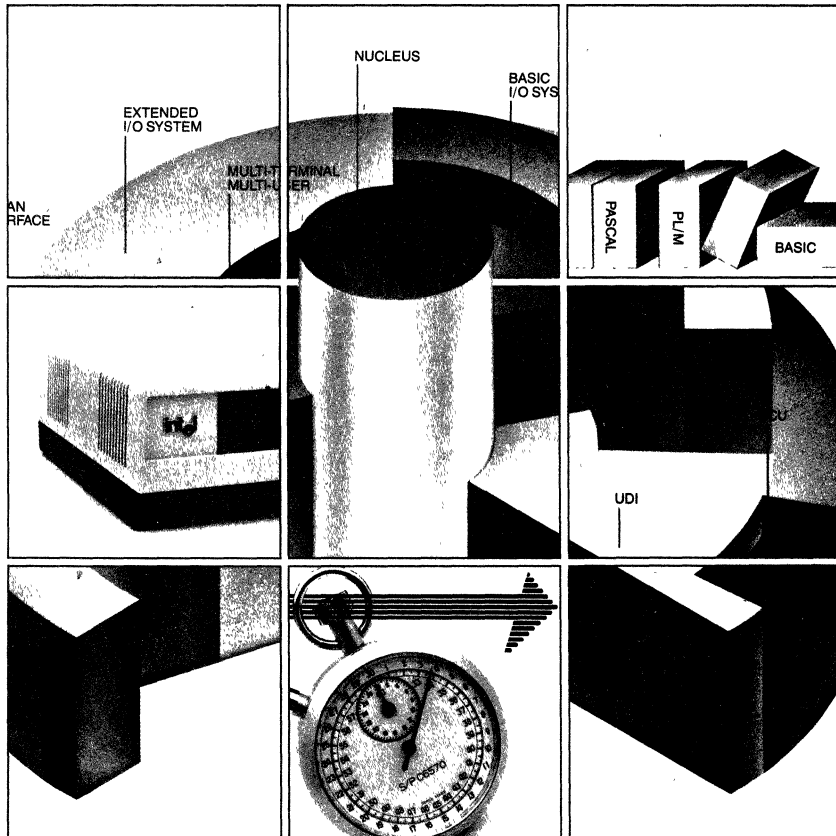
## Ordering Information

<b>Language</b>	<b>Order Code</b>	<b>Product Contents</b>	<b>Warranty</b>
ASM 86, Utilities	RMX 860	Two 8" disk and two 5.25" diskettes Edit Reference Manual—143587 iAPX 86/88 Family Utilities User's Guide—121616 Macro Assembler Operating Instructions—121628 ASM 86 Language Reference Manual—121703 8087 Support Library Reference Manual—121725	<b>90 days:</b> Software Updates, Subscription Service, Hotline Support
Pascal	RMX 861	Two 8" diskettes and two 5.25" diskettes Pascal 86 User's Guide—121539	<b>90 days:</b> Software Updates, Subscription Service, Hotline Support
FORTRAN	RMX 862	Two 8" diskettes and two 5.25" diskettes FORTRAN 86 User's Guide—121570	<b>90 days:</b> Software Updates, Subscription Service, Hotline Support
PL/M	RMX 863	One 8" diskette and one 5.25" diskette PL/M 86 User's Guide—121636	<b>90 days:</b> Software Updates, Subscription Service, Hotline Support
TX Editor	RMX 864	One 8" diskette and one 5.25" diskette TX Screen Echter User's Guide—145410	<b>90 days:</b> Software Updates, Subscription Service
BASIC	RMX 865	One 8" diskette and one 5.25" diskette BASIC Reference Manual—121806 BASIC 86 User's Guide—121986 One 8" diskette and one 5.25" diskette	<b>90 days:</b> Software Updates, Subscription Service
C	RMX 866	One 8" diskette and one 5.25" diskette C <i>Programming Language</i> by Kernighan and Ritchie (Prentice Hall) C 86 Compiler User's Guide—122085	<b>90 days:</b> Software Updates, Subscription Service, Hotline Support



## iRMX™ OPERATING SYSTEMS

- High performance, real-time, multi-tasking operating system for Intel's 86/300 and 286/300 microcomputer systems.
- Highly configurable, modular structure for easy system expansion.
- Wealth of design facilities and industry-standard languages to support fast, easy development.
- Application software portable to next generation of Intel VLSI.
- Supported by Intel's post-sales software support organization.



## The Total Solution for the Real-Time Application OEM

Intel's iRMX™ 86 and iRMX 286R Operating Systems are real-time, multi-tasking, multiuser, multiprogramming operating systems designed to support high performance, time-critical applications such as factory automation, industrial control and communications networks. The iRMX operating systems serve as optimized event-driven executives for managing and extending the resources of Intel's 86/300 and 286/300 systems in real-time applications where high speed and low interrupt latency are required. Added performance for demanding numeric-intensive tasks comes from support of Intel's floating point math coprocessors.

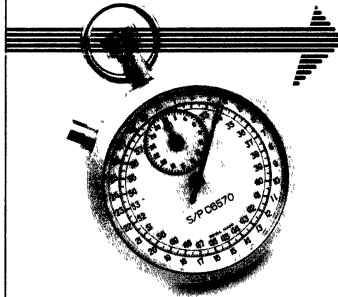
Comprised of modular layers, Intel's iRMX operating systems are highly configurable, allowing the OEM to easily customize the system to meet the needs of target applications. In addition to application customization, the iRMX operating systems provide OEMs with complete development capabilities. They have systems debuggers, crash analyzers, screen editors, utilities, and an Interactive Configuration Utility (ICU) — everything the development engineer needs to design and configure efficiently.

A complete set of industry-standard languages enables OEMs to take advantage of existing application software which further reduces development time. Shaving months off development time is a key advantage to the competitive OEM.

### Speed, the Name of the Real-Time Game

In a real-time system the computer must respond to interrupts instantly; time is always at a premium. Intel's iRMX operating systems deliver superior real-time performance, thanks to ultra-fast context switching, task synchronization and memory-based message passing.

The iRMX 286R Operating System manages the resources of the 286/300 systems in real-address mode. iRMX 286R makes



possible the utilization of the high-performance capabilities of Intel's iAPX 286 microprocessor for those demanding high-speed applications.

Further accelerating processing power in number-crunching and floating point math applications is iRMX operating system's support of Intel's math coprocessors.

Our 8087 numeric data processor in our iRMX 86-based systems can perform floating point operations four times faster than competitive minicomputers with hardware math processors. For even greater performance, OEMs can select the iAPX 286 and the 80287 coprocessor working in tandem in iRMX 286R-based systems.

The superior price/performance ratio that results from combining Intel's iRMX operating systems and the System 300 family makes the choice clear: a more competitive Intel micro-based system over a more expensive minicomputer-based system.

### Add More Processors for More Power, More Speed

Need still more micro-muscle in your application? In an iRMX-based system, additional intelligent boards can be added to enhance system throughput.

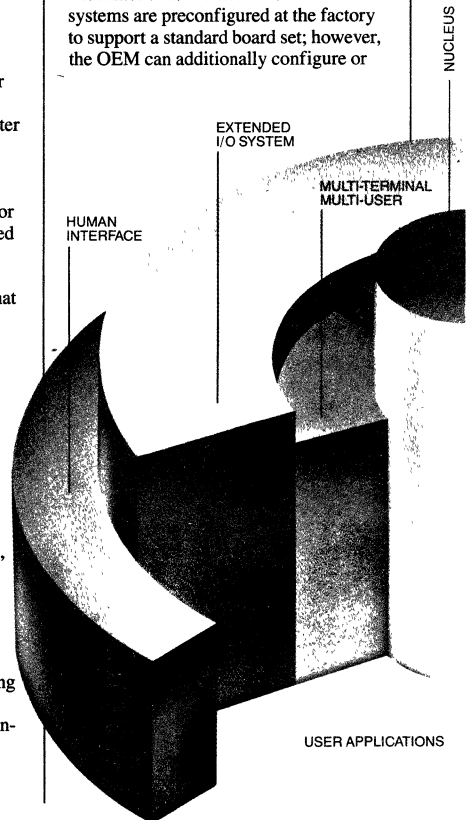
With the iMMX™ 800 (MULTIBUS® Message Exchange) software package, the iRMX 86 and iRMX 286R Operating Systems support a loosely-coupled multiprocessing environment. Tasks running on one board may communicate

with tasks running on other boards, even if they operate under different Intel operating systems or microprocessors.

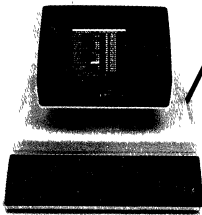
Multiprocessing is possible due to the hardware capabilities of Intel's System 300 MULTIBUS System Bus and the software support provided by iMMX 800. Overall system performance and flexibility can be greatly enhanced by off-loading the main CPU with such intelligent I/O boards as Intel's quad serial communication controller, digital controller or Ethernet communications controller.

### Modular Software for Versatile, Easy Configuration

The iRMX operating systems shipped with Intel's 86/300 and 286/300 hardware systems are preconfigured at the factory to support a standard board set; however, the OEM can additionally configure or



extend the operating system to meet specific needs.



Intel's iRMX operating systems are configurable by system layer and by system call within each layer. Such flexibility gives designers the ability to choose software features that best suit their application's size and functional requirements. The iRMX operating systems also include I/O drivers for many of Intel's MULTIBUS boards and industry-

standard peripherals. You simply select the ones you need.

The Interactive Configuration Utility (ICU) is a built-in facility for assisting the OEM in the configuration process. The ICU prompts the user for system parameters and requirements, then builds a command file to compile, assemble, link, and locate necessary files.

The net results for the OEM: fast, easy system configuration with quick time-to-market benefits.

For customizing and extending your iRMX system, Intel has provided all the "hooks" necessary to make the job easy. The iRMX 86 and iRMX 286R Operating Systems contain extendability features that enable the OEM to add custom operating system calls, custom features, and custom functionality to his application — at any time in the application's life. The ability to add functions late in a product's life is key to an OEM's competitive edge in a fast-changing market.

### **iRMX™ Operating System has All the Fundamentals Too!**

In addition to multiprocessing, Intel's iRMX operating systems have all the basics you would expect to find in a minicomputer operating system... capabilities such as multitasking, multiprogramming, and multiterminal support.

Multitasking requires a method of managing the different processes of an application and for allowing these processes to communicate with each other. The iRMX Nucleus provides these facilities plus task scheduling. The Basic I/O System provides users with the system calls for direct management of I/O devices needed for real-time applications. The Extended I/O System adds a number of I/O management capabilities to simplify access to files, such as automatic buffering and synchronization of I/O requests.

The Human Interface function give users and applications simple access to the file and system management capabilities. Using the multiterminal support provided by the Basic I/O system, the Human Interface can support several simultaneous users. For example, multi-terminal support allows one person to be using the iRMX Editor, while another compiles a FORTRAN or Pascal program, while several others load and access applications.

### **On-Target Development: One System Does It All**

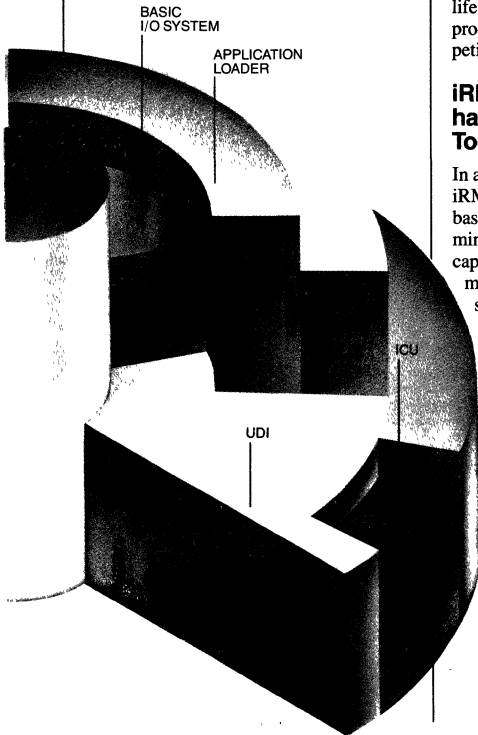
The beauty of Intel systems lies in their flexibility. Engineers developing an iRMX-based target system can use the same iRMX-based system in the development process; the development and target systems are one in the same. The bottom-line benefit is low entry-level costs for the OEM.

On-target development contributes immeasurably to a shorter development curve and decreased time-to-market, since it isn't necessary to purchase and learn separate development systems. With Intel's iRMX-based system, one system does it all.

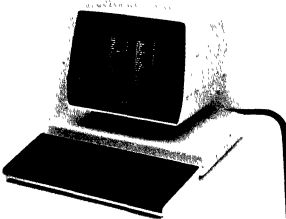


### **Tap into a Wide Range of Languages and Utilities**

An Intel iRMX-based system supports many industry-standard and widely available languages: FORTRAN 77, Pascal (ISO Draft Standard) and PL/M compilers; Intel Assemblers, and popular independent vendor products, such as Microsoft's BASIC and Mark Williams' C compiler.



iRMX operating systems also have a menu-driven, screen oriented text editor and a variety of utilities for manipulating object code to facilitate the development process.



Multiple-language support is made possible by a set of systems calls known as the Universal Development Interface (UDI) which enables the iRMX systems to interface with many compilers and language translators. UDI ensures that users will be able to transport applications to future releases of iRMX operating systems as well as use language and utilities of other software vendors that support UDI. (For more information on Intel iRMX languages, see the iRMX Language Fact Sheet)

### Intel's Open Systems Approach Means Freedom to Grow

At Intel, we believe that systems need to expand in order to meet the needs of a changing market; and that is how we design our products.

Standards are the key to systems that are open to future expansion, future technology and future markets.

Intel's iRMX operating systems are built from the inside-out with industry standards: UDI (Universal Development Interface), RTI (Runtime Interface), MULTIBUS System Bus (IEEE 796), iMMX 800 Package (MULTIBUS multi processing), Ethernet (IEEE 802.3), extended math format (IEEE P754), and industry-standard peripheral device interfaces.

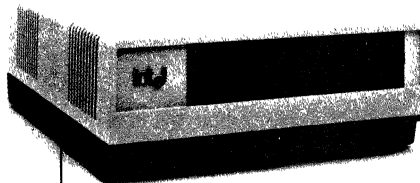
An OEM who builds his product around one of Intel's RMX-board systems is assured of multi vendor hardware/software alternatives and a future upgrade path. In today's highly competitive markets, that is the only kind of system to build.

Today, you'll have the ability to tap into readily available application software packages, languages, and utilities, MULTIBUS boards, and peripherals. Tomorrow, you will be able to tap into the latest, high-performance VLSI without sacrificing today's software investment. Applications written on iRMX 86 (for Intel's iAPX 86) are completely portable to iRMX 286R running on Intel's iAPX 286-based systems.

Not to be forgotten are the advantages of starting from the systems level to begin with. Intel has invested hundreds of man-years in software and hardware development for its systems products. For the OEM trying to meet a market window, time-to-market is much faster when starting with a system instead of boards or components. It makes good business sense to let Intel provide the "micro-engine"; so you can concentrate on your area of expertise and get to market sooner!

### Worldwide Service and Support

The iRMX 86 Operating System is a mature proven product with thousands of installations at the component, board and systems levels. Post-sales software support is available to Intel iRMX 86 and iRMX 286R Operating Systems OEMs in the form of software updates and routine systems software maintenance. Software support is extendable in one-



year increments after the initial 90-day warranty. Hotline service is available separately to customers needing quick response software support. All software is completely documented, and users receive monthly technical reports, newsletters and access to the iRMX users group and software libraries.

iRMX users can also take advantage of Intel's worldwide staff of trained hardware and software engineers for application design assistance. We offer complete training for operating system software and associated system hardware, bringing OEM's up to speed and helping get their products to market quickly.

### Intel, the Technology Leader ... With the Total Solution

Intel started the microprocessor revolution with the 4004 and has been the market leader with every generation of advanced microprocessor VLSI since. We not only invented the microprocessor but MULTIBUS single board computers, as well.

Intel's technology leadership has, by necessity, extended from micro-processors into operating system software. iRMX is recognized as the industry standard real-time VLSI operating system.

It has evolved since 1978 utilizing the experience of thousands of installations to contribute to enhancing the performance and quality of the product.

OEMs can enhance their product's marketability by leveraging their value-added on top of the solid foundation of an iRMX-based Intel 300 microcomputer system. Intel's solution offers the most price/performance with the least risk to progressive OEMs... because we know the real-time game from the inside out.

# iRMX OPERATING SYSTEMS



## Specifications

### Supported Software Products

iRMX 860	iRMX 86 Development Utilities Package including the iAPX 86 and 88 Linker, Locator, Macro Assembler, Librarian, and the iRMX 86 Editor
iRMX 861	Pascal 86/88 Compiler
iRMX 862	FORTRAN 86/88 Compiler
iRMX 863	PL/M 86/88 Compiler
iRMX 864	TX-Screen-Oriented Editor
iRMX 865	BASIC Interpreter
iRMX 866	C Compiler
iMMX 800	MULTIBUS® Message Exchange software package for iRMX 80, 86, 88, and 286 application systems

### Supported Hardware Products

#### ISBC® MULTIBUS® Products

iSBC 86/12A, 86/05, 86/14, 86/30, 88/25, 88/40, and 286/10	Single Board Computers
iSBC 204	Flexible Disk Controller
iSBC 206	Hard Disk Controller
iSBC 208	Flexible Disk Controller
iSBC 215	Winchester Disk Controller
iSBC 220	SMD Disk Controller
iSBC 251	Bubble Memory System (iRMX 286R only)

iSBC 254	Bubble Memory System
iSBC 534	4-Channel Terminal Interface
iSBC 544	Intelligent 4-Channel Terminal Interface and Controller
iSBX 218	Flexible Disk Controller
iSBX 350	Parallel Port (Centronix-type Printer Interface)
iSBX 351	Serial Communications Port
iSBX 270	CRT, Light Pen and Keyboard Interface

### Available Literature

iRMX 286R Operating System Installation and Configuration Guide for Release 1 (145556-001)

All of the manuals listed below are supplied with iRMX 86 Release 5 and are available separately under the order numbers shown.

Introduction to the iRMX 86 Operating System (9803124-04)

iRMX 86 Operator's Manual (144523-001)

Master Index for iRMX 86 Release 5 Documentation (145015-001)

Getting Started With The Release 5 iRMX 86 System (145073-001)

iRMX 86 Installation Guide (9803125-05)

iRMX Configuration Guide (9803126-05)

iRMX 86 Nucleus Reference Manual (9803122-04)
iRMX 86 Terminal Handler Reference Manual (143324-002)
iRMX 86 Debugger Reference Manual (143323-002)
iRMX 86 Basic I/O System Reference Manual (9803123-05)
iRMX 86 Loader Reference Manual (143318-002)
iRMX 86 Extended I/O System Reference Manual (143308-002)
iRMX 86 Human Interface Reference Manual (9803202-003)
Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems (142926-004)
iRMX 86 Programming Techniques (142982-003)
User's Guide for the iSBC 957B, iAPX 86, 88 Interface and Execution Package (143979-002)
iRMX 86 Disk Verification Utility Reference Manual (144133-002)
Runtime Support Manual for iAPX 86, 88 Applications (1211776-002)
iRMX 86 Crash Analyzer Reference Manual (144522-001)

### iRMX™ 86/286R Configuration Size Chart

System Layer	Min. ROMable Size	Max. Size	Data Size
Bootstrap Loader	0.5K	1.5K	6K*
Nucleus	10.5K	24K	2K
BIOS	26K	78K	1K
Application Loader	4K	10K	2K
EIOS	10.5K	12.5K	1K
Human Interface	22K	22K	15K
UDI	11K	11K	0
Terminal Handler	3K	3K	0.3K
Debugger	28.5K	28.5K	1K
Human Interface Commands			116K
Interactive Configuration Utility			308K
System 86/300 Memory:	348KB		
Maximum Addressable Memory:	1MB		
Minimum Memory Required with ICU Loaded:	448KB		

\*Usable by System after Bootloading



## Ordering Information

Each iRMX operating system includes a preconfigured version supporting Intel's System 300 standard hardware, a configurable iRMX operating system, iRMX 860 (Assembler, Linker, Locator, Libraries, Editor, Utilities), iRMX 863 (PL/M Language), iRMX System Software License and are prepaid incorporation Fee. Also included: Software Problem Reporting Service (SPR), and a 90 day System Software Subscription (new s/w release updates). Also includes System Software documentation.

NOTE: iRMX operating systems for Intel's System 300 microcomputers are available kitted with System 300 hardware only.

Refer to Intel's OEM price list, OEM Microcomputer System section, for ordering information.





## High-level Language Support for XENIX-Based Systems

Intel's Xenix operating system, available for component, board, or system-level integration, is a multi-user operating system well suited for both technical and commercial interactive applications. Typical applications include small business systems, software development/engineering workstations, distributed data processing and graphics.

For OEM and end-user application development on Xenix, Intel has provided two industry-standard, high-level languages — FORTRAN and COBOL — with which to build microcomputer-based solutions for systems products or component and board-level applications. Xenix FORTRAN and COBOL accommodate easy porting of existing mainframe and mini-based applications to the micro environment.

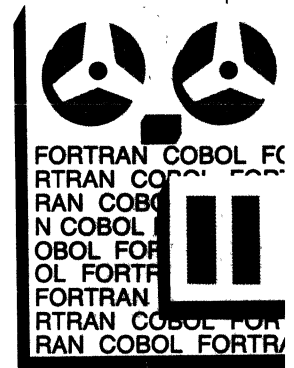
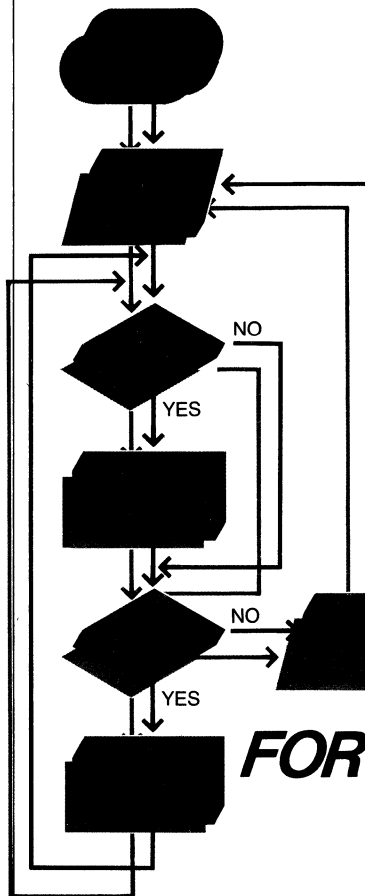
### XENIX FORTRAN for Scientific and Technical Applications

FORTTRAN is the most popular programming language for scientific and numerical applications. There are thousands of existing FORTRAN programs and subroutines written in mainframe and minicomputer environments, most of which can be ported to a micro environment via Intel's offering of Microsoft FORTRAN.

Compliance with the X3.9 1978 ANSI standard for FORTRAN at the subset level ensures portability with minimal source code modifications. By moving to a microcomputer-based system, you lose none of your mainframe and mini-developed software investment.

### Speed and Accuracy Where They're Needed

Scientific, math-oriented applications usually require fast, highly accurate processing. Xenix FORTRAN delivers accuracy with double-precision arithmetic



which handles numbers containing 14 significant digits.

High speed results from Xenix FORTRAN support of the Intel 8087 floating point coprocessor, as well as from an extensive subroutine library, which includes subroutines for 16- and 32-bit integer arithmetic and 32- and 64-bit floating-point arithmetic. Because of Xenix FORTRAN's 8087 math coprocessor support, some programs written in Xenix FORTRAN will execute from two to four times faster than their minicomputer counterparts.

Calls to "C" and ASM 86 are possible, making it easy to interface non-standard peripherals to Xenix FORTRAN programs.

# FORTRAN

### XENIX COBOL for the Micro Environment

Intel's offering of Microfocus COBOL is a mainframe-caliber compiler for ANSI 1974 COBOL programs, enabling Xenix-based systems to compile and run existing COBOL programs with minimal source code modification. Xenix COBOL also contains features specifically aimed at facilitating the interactive

**F**ORTRAN **C**OBOL **F**  
**O**RTRAN **C**OBOL **F**O  
**R**TRAN **C**OBOL **F**OR  
**T**RAN **C**OBOL **F**OR  
**R**AN **C**OBOL **F**ORTR  
**A**N **C**OBOL **F**ORTR

program development of new applications in a microcomputer environment.

These features include a facility for dynamically loading sub-programs from disk as required which effectively removes limits on the size of the application code that can be run. Xenix COBOL augments the functionality of the ANSI standard with additional compiler features, such as interactive screen-handling, that further increase convenience and programmer productivity.

Users can license a separate run-time support package. This enables OEMs to pass COBOL applications onto customers at a much lower cost than that involved in transferring full COBOL packages.

Xenix COBOL is one of only eleven COBOL compilers in existence—and the only one for microcomputers—that has been GSA-certified as error-free at the High Level. A special ANSI-



defined communications module provides the user with a standard mechanism for program-to-program message-passing in multi-user networks such as those found in an "office of the future" setting.

### Forms-2™ Support for Screen-Painting

Xenix COBOL supports FORMS-2, a powerful visual programming tool that speeds the creation of programs involving interactive screen-handling. In an extremely user-friendly environment, the user "paints" a form on the screen, and FORMS-2 generates the COBOL source code to support it. FORMS-2 results in greatly improved programmer productivity in a microcomputer, screen-building environment.

### Worldwide Service and Support

All Xenix systems are fully supported by Intel's worldwide staff of trained hardware and software engineers. Complete documentation is provided for all operating systems and application software languages, as well as for system

hardware components. The Xenix and Xenix Languages warranty includes Hotline support, Software Updates, and Subscription Service.

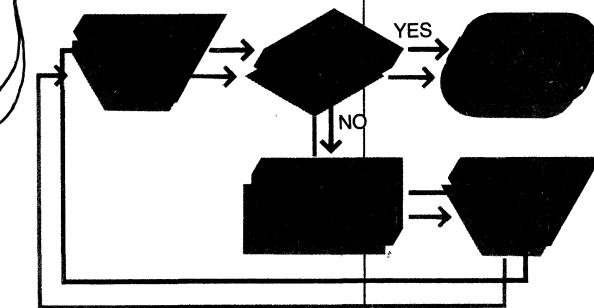
### Total Solutions for Interactive, Multi-User Applications

Intel's Xenix-based systems offer the most complete solutions for interactive, multi-user applications requiring fast, accurate throughput and a friendly programming environment. Xenix is complemented by industry-standard, high-level languages with which OEMs can create flexible and open end-user systems.

Xenix languages are completely portable—from one level of integration to another (chip to board to system).

Intel is paving the way into the future of VLSI and pioneering VLSI-based systems. We are committed to providing customers with smooth, uninterrupted application development on the latest VLSI-based systems - today and tomorrow.

## COBOL



**XENIX LANGUAGES**



## Specifications

<p><b>Required Hardware:</b></p> <ul style="list-style-type: none"> <li>• Any iAPX 86/286 based or iSBC® 86/286 based system including Intel's System 86/300, 286/300 family and iDIS systems</li> <li>• 128 KB memory</li> <li>• Two floppy disks or one hard disk</li> <li>• One 8" double-density or 5.25" double-density floppy disk drive for distribution of media</li> </ul>	<p><b>Required Software:</b></p> <ul style="list-style-type: none"> <li>• Intel's Xenix 86 or Xenix 286* Operating System</li> <li>• Purchase of any Xenix Language requires signing of Intel's OEM License Agreement (OLA)</li> </ul> <p><small>*The first release of FORTRAN will support only Xenix 86</small></p>	
---	---	--

## Ordering Information

Language	Order Code	Product Contents	Warranty
COBOL	XNX 867	One 8" diskette and one 5.25" diskette Level II COBOL Language Reference Manual—122158 Level II COBOL Operating Guide—122159 Forms II Utility Manual—122160 Level II COBOL Pocket Guide—122161	<b>90 days:</b> Software Updates, Subscription Service
	XNX 868	Incorporation Fee for passing through the COBOL Runtime System	
FORTRAN	XNX 862	One 8" diskette and one 5.25" diskette Fortran Reference Manual Fortran User's Guide	<b>90 days:</b> Software Updates, Subscription Service

FORMS-2 is a trademark of Micro Focus



## 2920 SOFTWARE SUPPORT PACKAGE

- Complete software design and development support for the 2920
- Extends Intellect® Microcomputer Development System to support 2920 software development

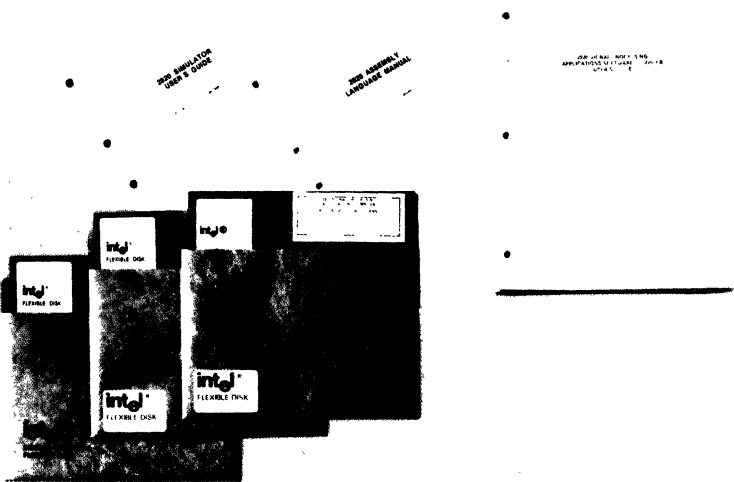
The 2920 Software Support Package furnishes a 2920 Signal Processing Applications Software/Compiler, 2920 Assembler, and 2920 Software Simulator. These three software design and development tools run on the Intellect® Microcomputer Development System.

The 2920 Signal Processing Application Software/Compiler is an interactive tool for designing software to be executed on the 2920 Signal Processor. The compiler accepts English-like statements from the user and generates 2920 assembly language code.

The assembler translates symbolic 2920 assembly language programs into the machine operation code. The user can load the code into the simulator for 2920 simulation or to the Universal PROM Programmer for 2920 EPROM programming.

The simulator, operating entirely in software, allows the user to test and symbolically debug 2920 programs. The user can specify input signals, simulate program execution, set up breakpoints, display input and output, and display and alter the contents of the 2920 registers and memory locations. The simulator can also stop or trace the program and constructively give the user access to the key elements inside a 2920 for analyzing his program.

The compiler, assembler, and simulator enable the designer to develop and test an entire program without a complete prototype design. The 2920 designer works on the Intellect® Microcomputer Development System rather than on a breadboard. The development system can program, store and recall programs or routines and aid in 2920 program design.



### 2920 Software Support Package

The following are trademarks of Intel Corporation and may be used only to identify Intel products: BXP, Intellect, Multibus, iSBX, Multimodule, ICE, iSBX, PROMPT, iCS, Library Manager, Promware, Insite, MCS, RMX, Intel, Megachassis, UPI, Intelevison, Microamp, μScope and the combination of ICE, iCS, iSBX, MCS, or RMX and a numerical suffix.

## 2920 SIGNAL PROCESSING APPLICATIONS SOFTWARE/COMPILER

- **Compiler generates 2920 Assembly Language Code**
- **Extensive command set for designing electrical filters**
- **Graphics capability enhances analysis of filter response or piecewise linear function approximations**
- **Powerful MACRO capability for executing frequently used routines**
- **Interactive software support tool for 2920 Signal Processor**
- **Extends Intellec® Microcomputer Development System support of the 2920**
- **Contains MACRO library for several standard filters and signal processing functions**

The 2920 Signal Processing Applications Software/Compiler (SPAS20) is an interactive tool for designing software to execute on the 2920 Signal Processor

The SPAS20 package can be visualized as being comprised of four inter-related sections: A compiler section, a filter design section, a curve fitting section, and a MACRO section.

Among the abilities of SPAS20 are: ability to generate 2920 assembly language code directly from specifications of signal processing building blocks such as filters and waveform generators; ability to generate 2920 assembly language code for several classes of algebraic equations such as  $Y = C * X$ ,  $Y = C * Y$ , and  $Y = C * X + Y$  where  $X$ ,  $Y$  are variables and  $C$  is a constant; ability to generate 2920 assembly language code for one variable function  $Y(X) = F(X)$ ; ability to examine time and frequency responses of filter sections specified by continuous or sampled poles and zeroes; ability to examine piecewise linear approximation of specific function; ability for users to implement more complex commands by grouping sets of commonly used commands into a MACRO.

The SPAS20 package runs under ISIS-II on any Intellec® Microcomputer Development System with 64K RAM. The output of SPAS20 can be assembled with the 2920 assembler, tested with the 2920 Simulator, and programmed into the 2920 chip with the Universal PROM Programmer for prototyping.

**FUNCTIONAL DESCRIPTION**

The 2920 Signal Processing Applications Software/ Compiler gives the analog designer a "high level language" for his 2920 applications—it decreases the need to code 2920 assembly language. Furthermore, the compiler is interactive. This feature enables the designer to define a filter, or transfer function, graph their response, and change their parameters many times, without having to program and test in an actual 2920 implementation.

Once a filter is realized by moving poles and zeros in the continuous and sampled planes, the filter may be coded and written onto an ISIS file. Similarly, after a function  $Y = F(X)$  has been defined, the code for a piecewise linear approximation can be stored onto an ISIS file. Several other file commands are available to store and retrieve command sequences for SPAS20 sessions.

**SPAS20 Command Language**

- DEFINE** This command defines a pole or zero by associating it with a number (i.e., POLE 3), and with real and imaginary coordinates in the continuous or sampled plane.  
  
This command also defines a symbol by associating a name with a numeric value, or a MACRO by providing a pointer to a specified command sequence.
- GRAPH/ OGRAPH** This command graphically displays the values of object(s) specified. For example, GRAPH GAIN and GRAPH PHASE are used to display filter response. The OGRAPH command will "overgraph" the new response over the old response, after any changes have been made. (You may also graph Group Delay, Step, and Impulse.)
- MOVE** Allows the definition of a pole or zero to be changed—its coordinates, its plane, or both.
- REMOVE** Deletes the definition of a pole, zero, symbol, or macro.
- HELP** Types an explanatory message on the console, pertaining to a command or its attributes.
- FIT** This command performs curve fitting, i.e. it approximates an arbitrary user supplied function with a piecewise linear function.

- DATA** This command allows for specification of a set of vertices (i.e. X – Y coordinate pairs) which determine a piecewise linear approximation of some defined function, filter response characteristics, etc.
- HOLD** Command to correct attenuation due to sample-and-hold distortion: if ON, it corrects absolute gain by  $\sin(x)/x$  and phase by adding  $x$ , where  $x = TS * \text{FREQ} * \pi$ . It corrects group delay by subtracting  $\pi * TS$ .
- EVALUATE** Gives the decimal numeric value of any expression.
- CODE** Creates 2920 assembly language code for given poles, and zeros, equations, and user defined functions.

The SPAS20 compiler also recognizes the following commands for file handling:

- PUT/ APPEND** Writes out objects (commands) to a specified file, either creating a new one or appending an existing one. This enables the user to store all or part of a SPAS20 session on a diskette to be brought back later with the INCLUDE command.
- DISPLAY** Copies the contents of a file to the console.
- INCLUDE** Executes a sequence of instructions from a diskette file as if they were typed in from the console.
- LIST** Creates a file containing all console interactions.

In addition to naming macros for specific command sequences, compound and conditional commands may be formed using all of the above statements. These compound commands are:

- IF** Establishes conditional flow of control within a block of commands.
- REPEAT** Used for repetition of a block of commands; executes indefinitely or until a condition is met (using WHILE, UNTIL, and END statements).
- COUNT** Establishes the number of times a command sequence is to be executed, in a looping fashion.

## SPAS20 MACRO Facility

A macro is a sequence of commands that is stored on a temporary diskette file. The command sequence is executed when the macro name is entered as a command. This saves repetitive entry of the sequence, and permits algorithms to be saved on diskette for future use. This SPAS20 facility allows you to do the following:

- Display the text of any macro.
- Define a macro, specifying its name and any parameters that are to be used by the block. This definition is followed by the contents of the macro (commands) and the EM statement to end its definition.
- Invoke a macro by entering its name and appropriate values for any parameters.
- List the names of all defined macros.
- Remove any or all macros.

Intel also supplies several MACRO library files containing the following commonly needed MACROS:

- Filter design MACROS
  - Butterworth filter
  - Chebyshev filter
  - Bilinear transform
  - Evaluate gain or phase of digital filter in parallel form
  - Time response simulation
- Function design MACROS
  - Code and error optimization
  - Calculate interstitial error
- MACROS for generation of 2920 code
  - Code for all-POLE filter
  - Input and A/D conversion
  - Multiplication
  - Division
  - Logarithm functions
  - Square-root functions
  - Sinewave oscillator

## SAMPLE SPAS20 FILTER DESIGN SESSION

```

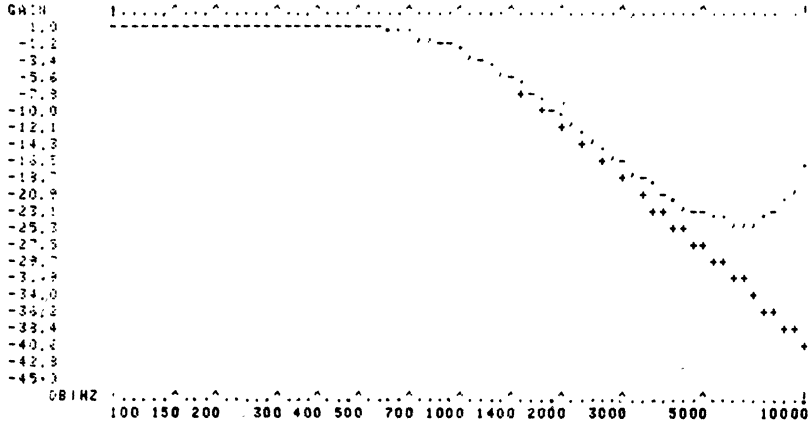
--PI : SPAS20 . SFT
ISIS-II 2920 SIGNAL PROCESSING APPLICATIONS COMPILER. V2.0
*
*DEFINE POLE 1 = -707,707      ; CREATE A POLE IN CONTINUOUS S-PLANE
*
*PI      ; LIST ALL POLES AND ZEROS
POLE 1 = -707 00000,707 00000,CONTINUOUS
*
*FSCALE = 100,10000          ; ESTABLISHES FREQUENCY RANGE OF INTEREST
*
*YSCALE = -45,1              ; ESTABLISHES MAGNITUDE RESPONSE RANGE OF INTEREST
*
*GRAPH GAIN                  ; PLOT MAGNITUDE RESPONSE OF POLE PAIR

GAIN
1.0
-1.2
-3.4
-5.6
-7.8
-10.0
-12.1
-14.3
-16.5
-18.7
-20.9
-23.1
-25.3
-27.5
-29.7
-31.9
-34.1
-36.3
-38.5
-40.7
-42.9
-45.0
DB1HZ
100 150 200 300 400 500 700 1000 1400 2000 3000 5000 10000
**
* . THE UNITS USED IN GRAPHING GAIN ARE SHOWN IN THE LOWER LEFT CORNER
* . GAIN IN DECIBELS IS GRAPHED VERSUS FREQUENCY IN HERTZ
*
* . PREPARE TO MOVE TO THE DIGITAL DOMAIN
* . SAMPLE RATE MUST BE SPECIFIED
*
*TS = 1/13020      ; RATE FOR 192 INSTRUCTION PROGRAM AND 10MHZ CLOCK
TS = 7 6805004/10**5
    
```

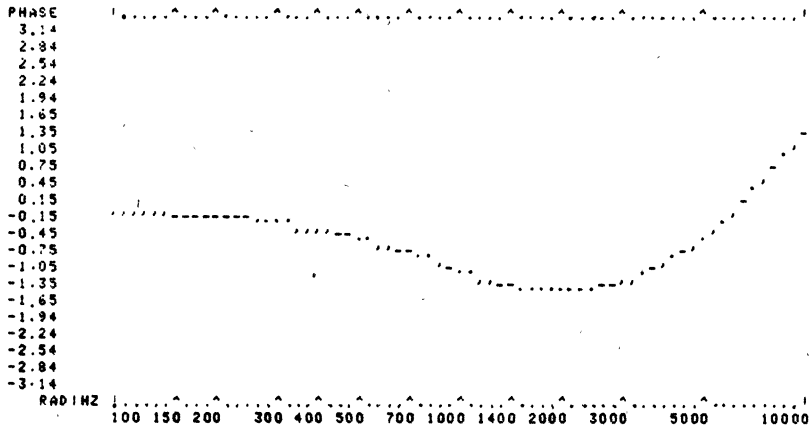


SAMPLE SPAS20 FILTER DESIGN SESSION (Cont'd.)

```
*MOVE POLE TO Z      ; CONVERT FILTER TO DIGITAL VIA MATCHED-Z TRANSFORMATION
1 POLES/ZEROES MOVED
*
*P      : LIST TRANSFORMED POLE
POLE 1 = 0.71092836,0.34118369,2
*
*: COMPARE RESPONSES OF THE ANALOG AND DIGITAL FILTERS BY GRAPHING THE
* NEW RESPONSE OVER THE OLD
*
*GRAPH GAIN
```



```
G*
*
* PLUS SIGNS INDICATE OLD CURVE
* NOTE THAT THE DIGITAL FILTER RESPONSE BEGINS TO INCREASE AGAIN
* AT HALF THE SAMPLE RATE ( 6510 HZ ).
*
*: THE PHASE CHARACTERISTICS OF THIS FILTER CAN BE EXAMINED
*
*YSCALE = -PI,PI      ; ESTABLISHES RANGE OF INTEREST
*
*GRAPH PHASE
```



```
P*
*
*PUT :F1:POLE P2      ; SAVE THE POLE LOCATION IN A DISK FILE BACKUP
*
*LOAD POLE 1 INSTK11  ; GENERATE 2920 ASSEMBLY CODE FOR THIS FILTER
B=.1 33989*90 B2=-0.50541914
```

## SAMPLE SPAS20 FILTER DESIGN SESSION (Cont'd.)

OPTIMIZED 2920 CODE IS NOW GENERATED TO SAVE SPACE. SOME OF THE SCREEN OUTPUT HAS BEEN DELETED NORMALLY ALL ATTEMPTS BY THE COMPILER TO GENERATE CODE ARE ECHOED ON THE SCREEN )

```

INST=10
POLE 1 = 0 71089458.0 34116779.Z
BEST: PERFOR = 3 3795874/10**5.1 58846567/10**5

; NOTE: MAKE SURE SIGNAL IS <0 74635571
LDA OUT2_P1.OUT1_P1.R00
; OUT2_P1=1 00000000*OUT1_P1
LDA OUT1_P1.OUT0_P1.R00
; OUT1_P1=1 00000000*OUT0_P1
SUB OUT0_P1.OUT1_P1.R05
; OUT0_P1=1 00000000*OUT0_P1-0 031250000*OUT1_P1
ADD OUT0_P1.OUT0_P1.R03
; OUT0_P1=1 12500000*OUT0_P1-0 035156250*OUT1_P1
ADD OUT0_P1.OUT1_P1.R02
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1
SUB OUT0_P1.OUT2_P1.R01
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1-0 50000000*OUT2_P1
SUB OUT0_P1.OUT2_P1.R08
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1-0 50390625*OUT2_P1
ADD OUT0_P1.OUT2_P1.R11
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1-0 50341796*OUT2_P1
SUB OUT0_P1.OUT2_P1.R09
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1-0 50537109*OUT2_P1
ADD OUT0_P1.IN0_P1.R00
; OUT0_P1=1 12500000*OUT0_P1+0 21484375*OUT1_P1-0 50537109*OUT2_P1+1 00000000*IN0_P1
*
*; THE CODE COMMAND SPECIFIED THAT THE POLE PAIR BE CODED IN LESS THAN 11
*; INSTRUCTIONS, SO 10 INSTRUCTIONS WERE GENERATED, WITH COMMENTS
*; THE FINAL ERROR IN RADIUS AND ANGLE FOR THE POLE PAIR WAS OF THE
*; ORDER OF 1/10**5 AS INDICATED ABOVE IN PERFOR
*; THIS OPTIMIZED 2920 ASSEMBLY CODE CAN NOW BE APPENDED TO A FILE
*; WHICH MAY CONTAIN OTHER CODED FUNCTIONAL BLOCKS OF A 2920 PROGRAM
*
*EXIT

```

## SAMPLE SPAS20 CURVE FITTING SESSION

```

-; DEMONSTRATION OF THE SPAS20 CURVE-FITTING PACKAGE
-
-SPAS20.SFT

ISIS-II 2920 SIGNAL PROCESSING APPLICATIONS SOFTWARE/COMPILER, V2.0
*LIST XCUBED.829
*
*
*; THE CURVE FITTING COMMANDS IN SPAS20 WILL GENERATE 2920 CODE TO CALCULATE
*; SOME FUNCTION SUCH AS Y**3. Y**3 COULD BE COMPUTED ON THE 2920 CHIP
*; WITH TWO MULTIPLIES USING ABOUT 18 INSTRUCTIONS AND THE DAR. HOWEVER IT
*; WOULD TIE UP THE DAR TOO LONG. THE CODE GENERATED BY THE CURVE FITTING
*; COMMANDS DOES NOT USE THE DAR.
*
*CODE FIT YXCUBED(X) = Y**3 ERROR<.05 ;ERROR BOUND OF .05
*
*CODE ; HERE IS THE CODE GENERATED.
LDA TEMP,X,R00
; TEMP=1.00000000*X
LDA XCUBED,X,R01
; XCUBED=0.50000000*X
ADD XCUBED,X,R06
; XCUBED=0.51562500*X
ADD TEMP,X,R01
; TEMP=0.50000000*X+1.00000000*TEMP
ADD XCUBED,TEMP,R05
; XCUBED=1.00000000*XCUBED+0.03125000*TEMP
SUB XCUBED,TEMP,R02
; XCUBED=1.00000000*XCUBED-0.21875000*TEMP
ADD TEMP,X,R00
; TEMP=1.00000000*X+1.00000000*TEMP
ADD XCUBED,TEMP,R08
; XCUBED=1.00000000*XCUBED+0.0039062500*TEMP
SUB XCUBED,TEMP,R04
; XCUBED=1.00000000*XCUBED-0.058593750*TEMP
LDA XCUBED,XCUBED,L02
; XCUBED=4.00000000*XCUBED-0.23437500*TEMP
*
*INST ; THE FUNCTION WAS CODED IN THIS MANY INSTRUCTIONS;
INST = 10.00000000
*

```



# 2920 SOFTWARE SUPPORT PACKAGE

```

*ERROR ; THE CODE APPROXIMATES X**3 WITHIN THIS ERROR;
ERROR = 0.046875000
*
*DATA 0 THRU 1 ; EXAMINE THE PIECEWISE LINEAR FUNCTIONS.
DATA 0.00000000 THRU 1.00000000 = 0.00000000 AT 0.00000000,&
0.065625012 AT 0.40000000,&
0.26562500 AT 0.66666669,&
0.95312500 AT 1.00000000
*
*GRAPH DATA(X) ; THE DATA ARRAY APPROXIMATES THE FUNCTION AND CAN BE GRAPHED.
FUNCTION !.....!
0.95
0.91
0.86
0.82
0.77
0.73
0.68
0.64
0.59
0.54
0.50
0.45
0.41
0.36
0.32
0.27
0.23
0.19
0.14
0.09
0.05
0.00
!.....!
*
*GRAPH X**3 ; THE DIFFERENCE BETWEEN THE CODED AND THE ACTUAL APPEARS AS "+".
FUNCTION !.....!
1.00
0.95
0.90
0.86
0.81
0.76
0.71
0.67
0.62
0.57
0.52
0.48
0.43
0.38
0.33
0.29
0.24
0.19
0.14
0.10
0.05
0.00
!.....!
*
*GRA (X**3)-DATA(X) ; THE ERROR WILL BE GRAPHED.
FUNCTION !.....!
0.047
0.043
0.039
0.036
0.032
0.028
0.025
0.021
0.017
0.014
0.010
0.006
0.003
-0.001
-0.005
-0.008
-0.012
-0.016
-0.020
-0.023
-0.027
-0.031
!.....!
*
*EXIT ; THAT'S ALL FOLKS

```



## 2920 ASSEMBLER

2920 program development on Intellec®  
Microcomputer Development Systems

Produces Assembly Listing, Object Code  
File, and Error Diagnostics

Translates symbolic assembly language  
instructions into 2920 machine code

Output used for 2920 programming with  
the Intellec PROM Programmer or the  
2920 Simulator for program debug

The 2920 Assembler translates symbolic 2920 Assembly Language instructions into the appropriate machine operation codes. Through this facility, the programmer is able to symbolically program 2920 hardware operations. Compared to machine code, these symbolic references provide faster programming, easier debugging, and greater reliability.

The Assembler produces an object code file (executable machine code), a complete assembly listing, and error diagnostics. The object code output from the Assembler may be loaded directly into the Intel Universal PROM Programmer for programming the 2920 EPROM. The object code may also be loaded to the 2920 Simulator for 2920 system design and debug.

The 2920 Assembler runs under the ISIS-II Operating System on the Intellec Microcomputer Development Systems.

### Sample 2920 Assembly Listing

```

ISIS-II 2920 ASSEMBLER X102                                PAGE 1
ASSEMBLER INVOKED BY: AS2920 SAM ASM DEBUG
SAWTOOTH WAVE GENERATOR

LINE  LOC  OBJECT SOURCE STATEMENT
   1                $TITLE('SAWTOOTH WAVE GENERATOR')
   2                ;
   3                ;
   4    0 0000EF      INO           ; SAMPLE INPUT CHANNEL 0
   5    1 0000EF      INO
   6    2 0000EF      INO
   7    3 008AEB      SUB Y,KP1,INO ; SIMULTANEOUSLY CALCULATE SAWTOOTH
   8    4 008A0A      SUB Y,KP1,R1,INO ; BY SUBTRACTING 3/16 FROM Y
   9    5 0044EF      LDA DAR,Y,INO ; ALSO CHECK SIGN BIT OF Y
  10    6 7ABAED      ADD Y,KP7,CNDS ; IF Y NEGATIVE START NEXT TOOTH
  11    7 6000EF      CVTS          ; CONVERT SAMPLED INPUT TO DIGITAL (SIGN BIT)
  12    8 7082EF      LDA Y,KP0,CNDS ; SUPPRESS SAWTOOTH IF INPUT WAS < 0
  13    9 4044EF      LDA DAR,Y     ; PREPARE TO OUTPUT SAWTOOTH
  14   10 4000EF      NOP           ; ANALOG LEVEL MUST SETTLE
  15   11 4000EF      NOP
  16   12 4000EF      NOP
  17   13 8000EF      OUTO          ; OUTPUT SAWTOOTH
  18   14 8000EF      OUTO
  19   15 8000EF      OUTO
  20   16 5000EF      EOP           ; PROGRAM WILL END IN THREE MORE INSTRUCTIONS
  21   17 8000EF      OUTO
  22   18 8000EF      OUTO
  23   19 8000EF      OUTO
  24                END
  25

SYMBOL:                                VALUE:
Y                                           0

ASSEMBLY COMPLETE
ERRORS   = 0
WARNINGS = 0
RAMSIZE  = 1
ROMSIZE  = 20

```

## 2920 SIMULATOR

**Speeds test and debug of 2920 programs**

**Simulates 2920 internal operation**

**Operates on Intellec® Microcomputer Development Systems**

**Allows users to specify 2920 input signals, and display or alter ROM, RAM, and system variables**

**Output and internal data can be saved on disk for further analysis.**

**Provides ability to set breakpoints and to collect trace information**

**Easy-to-learn commands**

The 2920 Simulator is a software facility that provides testing and symbolic debugging of 2920 programs in an Intellec Microcomputer Development Systems environment. The 2920 designers have the capability to specify the 2920 input signals, to set breakpoints, to collect and display 2920 input, output, system variables, and ROM and RAM data values during simulation. The 2920 Simulator accepts the hex format object files produced by the 2920 assembler. Output values and internal trace data may be saved on ISIS-II disk files for further analysis.

### Functional Description

#### 2920 Input Signal Specification

The four analog signal inputs to the 2920 processor can be specified as algebraic combinations of basic functions of time. The basic functions are SIN, COS, EXP, LOG, SQR, SAW, SQW, ABS.

#### 2920 Simulation

The simulation of 2920 machine instructions is performed in software. All 2920 internal registers, memory, input values, output values, and other system variables can be examined and modified. The internal processing of the 2920 is simulated. Time constants for the sample and hold capacitors are assumed to be zero. Calculation of input signals is performed in single precision floating point. The speed of simulation varies with the complexity of the input signal, breakpoint setting, and trace condition. Exclusive of I/O time requirements, 2920 instructions will be simulated at a rate of approximately several hundred instructions per second.

#### Breakpoint Capabilities

After each instruction is simulated, the breakpoint is evaluated to determine whether to stop or continue simulation. Conditional breakpoints are also provided for debugging purposes. Simulation can be manually stopped at any time by pressing the ESC key on the Intellec console.

#### Trace Capabilities

Based on the qualifier's condition, trace data records can be collected during simulation. The trace data

records are stored in Intellec resident memory and are optionally written to the console for display or to a disk file for record.

#### Symbolic Debugging Capabilities

The 2920 Simulator allows the user to refer to program addresses symbolically. The user can load or save the symbols generated from the hex format object files or created during the debugging session. 2920 program memory in ROM can be disassembled, or filled with assembled instructions.

The 2920 Simulator is designed to provide users with powerful, easy-to-use commands. The user interfaces to the Simulator by entering commands to the Intellec console. The commands consist of one command line, terminated by one of the two line terminators — carriage return or line feed.

The 2920 Simulator offers two types of commands:

#### Simulation and Control Commands

Command	Operation
Simulate	Starts simulation of the input signals and the 2920 program in simulated ROM memory. Initial setting is "FOREVER."
Trace	Controls the trace selection. Initial setting is "TIME."
Qualifier	Sets qualifier condition during trace. Initial setting is "ALWAYS."
Breakpoint	Sets breakpoint condition during simulation. Initial setting is "NEVER."

### Interrogation and Utility Commands

Command	Operation
Display	Displays the values of symbols, RAM, ROM, input, output, registers and system variables.
Change	Alters the values of symbols, RAM, ROM, input, register and system variables.
Base	Establishes the mode of display for output data.
Suffix	Establishes the mode of display for input data.
Load	Fetches user symbol table and object code from input device.
Save	Sends user symbol table and object code to output device.
Define	Enters symbol name and value to user symbol table.
Console	Controls the console on/off display
List	Defines list device.
Exit	Returns program control to ISIS-II.
Evaluate	Converts expression to equivalent values in binary, decimal, and hex
Remove	Deletes symbols from symbol table.
Help	Provides a brief summary of the syntax for the command languages.
Graphics On/Off	Switches output mode between list and graphics.
X Size	Enters horizontal display size.

### Keyword References

The 2920 Simulator provides users with keyword references to gain access to all of the numeric valued system variables including simulated 2920's memory, register, status flags and input/output. These keyword references can function as the evaluation command, display command, and change command.

#### • 2920 Processor Keyword References

IN0	Analog input 0 in volts
IN1	Analog input 1 in volts
IN2	Analog input 2 in volts
IN3	Analog input 3 in volts
OUT0	Analog output 0 in volts (read only)
OUT1	Analog output 1 in volts (read only)
OUT2	Analog output 2 in volts (read only)
OUT3	Analog output 3 in volts (read only)
OUT4	Analog output 4 in volts (read only)
OUT5	Analog output 5 in volts (read only)
OUT6	Analog output 6 in volts (read only)
OUT7	Analog output 7 in volts (read only)
IN	Sampled and held analog input signal in volts
DAR	Digital to analog register (RAM location 40)
PC	Program counter (integer 1 to 192)
CY	Carry (integer 0 or 1)
OVF	Overflow (integer 0 or 1, read only)
OVE	Overflow enable (integer 0 or 1)

#### • Software Simulator Keyword References

TIME	Elapsed simulated time in seconds (read only)
TQUAL	Time when the qualifier last matched in seconds (read only)
COUNT	Number of instructions simulated since last SIMULATE command (integer, read only)
BUFFERSIZE	Number of trace data records (integer, read only)
TINST	Time between successive instructions in seconds (read only)
SIZE	Number of instructions in program disregarding actual EOP placement
TPROG	Time between successive program passes in seconds
VREF	Reference analog level voltage in volts

The above keyword references are designed to aid 2920 program debugging.

### ISIS Compatibilities

The 2920 software simulator runs under the ISIS "submit" facility. The 2920 software simulator uses the ISIS-II line editing capabilities to correct errors in an input line on the Intellec console.

## Sample 2920 Simulation Session

```

-
-SM2920.SFT

ISIS-II 2920 SIMULATOR, V1.1
*
*; THIS IS THE SIMULATION OF THE 'SAWTOOTH GENERATOR'
*
*LIST SRG.LOG ; LISTS THE SIMULATION SESSION TO AN ISIS F
*LOAD SRG.HEX ; LOAD THE OBJECT CODE INTO THE 2920 SIMUL
*ROM 0 TO 5 ; DISPLAY SRG PROGRAM
ROM 000 = LDA .K,KP5,R00,NOP
ROM 001 = ADD .K,KP1,R05,NOP
ROM 002 = LDA .K,.K,R02,NOP
ROM 003 = SUB .OSC,.K,R00,NOP
ROM 004 = LDA DAR,.OSC,R00,NOP
ROM 005 = ADD .OSC,KP4,L01,CNDS
*TRPROG=1/10000 ; SET THE SAMPLE RATE
*TRA=PC,RAM .K ; SET THE ITEMS TO BE TRACED
*BASE=B ; DISPLAY THE RESULTS IN BINARY
*SIMULATE FROM 0 TILL COUNT=3 ; SIMULATE THREE INSTRUCTIONS
TO VERIFY CONSTANT

PC RAM 0
SIMULATION BEGUN
1.00000000000000000000000000000000 0.10100000000000000000000000000000
2.0000000E+0 0.10100001000000000000000000000000
3.0000000E+0 0.00101000010000000000000000000000
SIMULATION TERMINATED
*QUALIFIER=PC=0 ; TRACE EVERY PROGRAM PASS
*TRACE=T,DAR,RAM .OSC ; SET THE ITEMS TO BE TRACED
*RAM .OSC=ONE ; INITIALIZE THE RAM LOCATION
*BREAKPOINT=T>.00132 ; SIMULATE FOR TWO CYCLES
*BASE=D ; SET THE BASE TO DECIMAL
*SIMULATE FROM 0 ; BEGIN SIMULATION
T DAR RAM 1

```



```

SIMULATION BEGUN
0.00010000      0.83984375      0.84277334
0.00020000      0.68359375      0.68554683
0.00030000      0.52734375      0.52832026
0.00040000      0.36718750      0.37109370
0.00050000      0.21093750      0.21386714
0.00060000      0.05468750      0.05664056
0.00070000     -0.10156250      0.89941396
0.00080000      0.73828125      0.74218745
0.00090000      0.58203125      0.58496089
0.00100000      0.42578125      0.42773733
0.00110000      0.26953125      0.27050776
0.00120000      0.10937500      0.11328119
0.00130000     -0.04687500      0.95605459
SIMULATION TERMINATED
*GRAPH ON          ; SWITCHES THE DISPLAY MODE TO GRAPHICS
*TRACE=T,0,DAR,RAM :OSC,-1,-1,1,1 ; SETS ITEMS TO BE TRACED
*RAM :OSC=ONE      ; INITIALIZE THE RAM LOCATION
*SIMULATE FROM 0
  T          0          DAR          RAM 1          -1
SIMULATION BEGUN
->*          1          *          *
0 *          1          *          *
. *          1          *          *
0 *          1          *          *
0 *          1          *          *
0 *          1 *          *          *
1 *          2 1          *          3 *
0 *          1          *          *
0 *          1          *          *
0 *          1          *          *
0 *          1          *          *
*          1          *          *
*          1          *          *
*          2 1          *          3 *
SIMULATION TERMINATED
*EXIT
-

```

### SPECIFICATIONS

#### Operating Equipment

#### Required Hardware

Intellec® Microcomputer Development System  
RUNNING ISIS

#### Required Software

ISIS-II Diskette Operating System

#### Optional Hardware

Line Printer  
Universal PROM Programmer

#### Optional Software

FORTRAN-80 (Product Code MDS-301)

### Documentation Package

2920 Assembly User's Guide (9800987)  
2920 Simulator User's Guide (9800988)  
2920 Signal Processing Application Compiler  
User's Guide (121529)

### Shipping Media

Flexible Diskettes

## ORDERING INFORMATION

#### Product Code Description

MCI-20-SPS 2920 Software Support Package  
Includes 2920 Signal Processing  
Application Software/Compiler and 2920  
Assembler/Simulator Software



## MCS<sup>®</sup>-48 DISKETTE-BASED SOFTWARE SUPPORT PACKAGE

- Extends Intellec microcomputer development system to support MCS-48 development
- MCS-48 assembler provides conditional assembly and macro capability
- Takes advantage of powerful ISIS-II file handling and storage capabilities
- Provides assembler output in standard Intel hex format

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes, and provides both conditional and macroassembler programming. Output may be loaded either to an ICE-49 module for debugging or into the iUP Universal PROM Programmer for 8748 PROM programming. The MCS-48 assembler operates under the ISIS-II operating system on Intel Development systems.





## FUNCTIONAL DESCRIPTION

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes. The ability to refer to program addresses with symbolic names eliminates the errors of hand translation and makes it easier to modify programs when adding or deleting instructions. Conditional assembly permits the programmer to specify which portions of the master source document should be included or deleted in variations on a basic system design, such as the code required to handle optional external devices. Macro capability allows the programmer use of a single label to define a routine. The MCS-48 assembler will assemble the code required by the reserved routine whenever the macro label is inserted in the text. Output from the assembler is in standard Intel hex format. It may be either loaded directly to an in-circuit emulator (ICE-49) module for integrated hardware/software debugging, or loaded into the iUP Universal PROM Programmer for 8748 PROM programming. A sample assembly listing is shown in Table 1.

The MCS 48 assembler supports the 8048, 8049, 8050, 8020, 8021, 8022, 8041 and 8042. The MCS 48 assembler can also support CMOS versions of the 8048 family.

Table 1. Sample MCS-48 Diskette-Based

LOC	OBJ	SEQ	SOURCE STATEMENT
ISIS II 8048 MACROASSEMBLER, V1 0 PAGE 1			
		1	DECIMAL ADDITION ROUTINE ADD BCD NUMBER
		2	AT LOCATION 'BETA' TO BCD NUMBER AT 'ALPHA' WITH
		3	RESULT IN 'ALPHA' LENGTH OF NUMBER IS 'COUNT' DIGIT
		4	'PAIRS' (ASSUME BOTH BETA AND ALPHA ARE SAME LENGTH
		5	AND HAVE EVEN NUMBER OF DIGITS OR MSD IS 0 IF
		6	ODD)
		7	INIT MACRO AUGND,ADDND,CNT
		8	MOV RO,#AUGND
	L1	9	MOV R1,#ADDND
		10	MOV R2,#CNT
		11	ENDM
		12	
0001E		13	ALPHA EQU 30
0028		14	BETA EQU 40
002E		15	COUNT EQU 5
0100		16	ORG 100H
		17	INIT ALPHA,BETA,COUNT
0100 881E		18+	MOV RO,#ALPHA
0102 B928		19+L1	MOV R1,#BETA
0104 BA32		20+	MOV R2,#COUNT
0106 97		21	CLR C
0107 F0		22 LP	MOV A,@RO
0108 71		23	ADDC A,@R1
0109 57		24	DA A
010A A1		25	MOV @RO,A
010B 18		26	INC RO
010C 19		27	INC R1
010D EA07		28	DJNZ R2,LP
			END
USER SYMBOLS			
ALPHA	0001E	BETA	0028
COUNT	002E		
L1	0102	COUNT	0005
		LP	0107
ASSEMBLY COMPLETE NO ERRORS			
ISIS II ASSEMBLER SYMBOL CROSS REFERENCE V1 0 PAGE 1			
SYMBOL CROSS REFERENCE			
ALPHA	13E	17	
BETA	144	17	
COUNT	156	17	
INIT	7A	17	
L1	196		
LP	22A	28	

## SPECIFICATIONS

### Operating Environment

- (All) Intel Microcomputer Development Systems (Series II, Series III/Series IV)
- Intel Personal Development System

### Documentation Package

- Titles of: User Guides
- Operating Instructions
- Reference Manuals

### Ordering Information

Part Number	Description
MDS-D48*	MCS-48 Disk Based Assembler
	Requires Software License

### SUPPORT:

Hotline Telephone Support, Software Performance Reports (SPR), Software Updates, Technical Reports, Monthly Newsletters are available.

\*MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation.



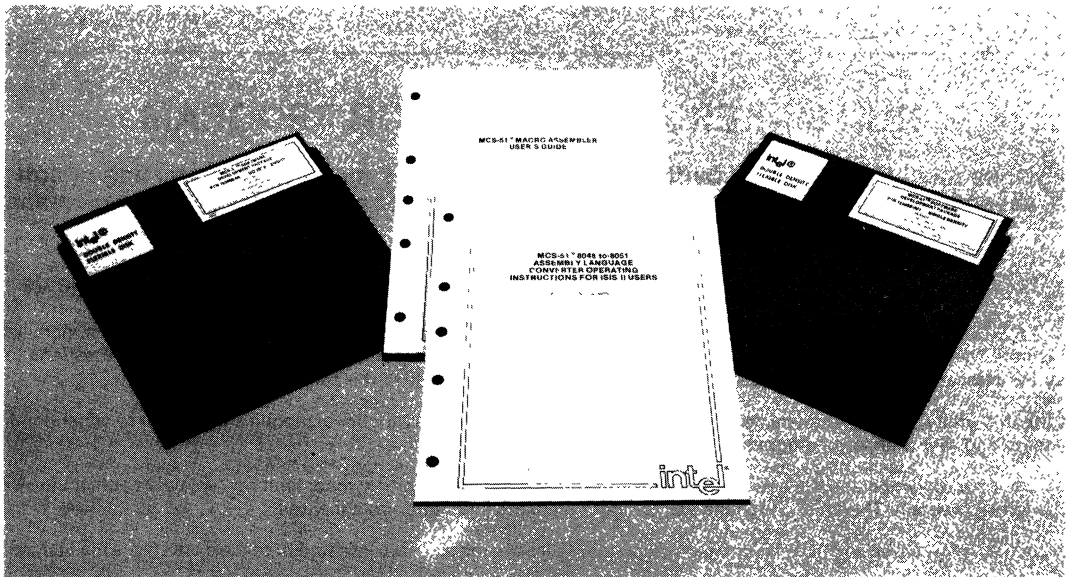
## 8051 SOFTWARE DEVELOPMENT PACKAGE

- Symbolic relocatable assembly language programming for 8051 microcontrollers
- Extends intellec® Microcomputer Development System to support 8051 program development
- Produces Relocatable Object Code which is linkable to other 8051 Object Modules
- Encourage modular program design for maintainability and reliability
- Macro Assembler features conditional assembly and macro capabilities
- CONV51 Converter for translation of 8048 assembly language source code to 8051 assembly language source code
- Provides upward compatibility from the MCS-48™ family of single-chip microcontrollers

The 8051 software development package provides development system support for the powerful 8051 family of single chip microcomputers. The package contains a symbolic macro assembler and MCS-48 source code converter.

The assembler produces relocatable object modules from 8051 macro assembly language instructions. The object code modules can be linked and located to absolute memory locations. This absolute object code may be used to program the 8751 EPROM version of the chip. The assembler output may also be debugged using the ICE-51™ in-circuit emulator.

The converter translates 8048 assembly language instructions into 8051 source instructions to provide software compatibility between the two families of microcontrollers.



## 8051 MACRO ASSEMBLER

- Supports 8051 family program development on Intellec® Microcomputer Development Systems
- Gives symbolic access to powerful 8051 hardware features
- Produces object file, listing file and error diagnostics
- Object files are linkable and locatable
- Provides software support for many addressing and data allocation capabilities
- Symbolic Assembler supports symbol table, cross-reference, macro capabilities, and conditional assembly

The 8051 Macro Assembler (ASM51) translates symbolic 8051 macro assembly language modules into linkable and locatable object code modules. Assembly language mnemonics are easier to program and are more readable than binary or hexadecimal machine instructions. By allowing the programmer to give symbolic names to memory locations rather than absolute addresses, software design and debug are performed more quickly and reliably. Furthermore, since modules are linkable and relocatable, the programmer can do his software in modular fashion. This makes programs easy to understand, maintainable and reliable.

The assembler supports macro definitions and calls. This is a convenient way to program a frequently used code sequence only once. The assembler also provides conditional assembly capabilities.

Cross referencing is provided in the symbol table listing, showing the user the lines in which each symbol was defined and referenced.

ASM51 provides symbolic access to the many useful addressing features of the 8051 architecture. These features include referencing for bit and byte locations, and for providing 4-bit operations for BCD arithmetic. The assembler also provides symbolic access to hardware registers, I/O ports, control bits, and RAM addresses. ASM51 can support all members of the 8051 family.

Math routines are enhanced by the MULtipl and DIVide instructions.

If an 8051 program contains errors, the assembler provides a comprehensive set of error diagnostics, which are included in the assembly listing or on another file. Program testing may be performed by using the iUP Universal Programmer and iUP F87/51 personality module to program the 8751 EPROM version of the chip.

ICE51 and EMV51 are available for program debugging.

---

## RL51 LINKER AND RELOCATOR PROGRAM

- Links modules generated by the assembler
- Enables modular programming of software for efficient program development
- Locates the linked object to absolute memory locations
- Modular programs are easy to understand, maintainable and reliable

The 8051 linker and relocater (RL51) is a utility which enables 8051 programmers to develop software in a modular fashion. The linker resolves all references between modules and the relocater assigns absolute memory locations to all the relocatable segments, combining relocatable partial segments with the same name.

With this utility, software can be developed more quickly because small functional modules are easier to understand, design and test than large programs.

The number of symbols in the software is very large because the assembler symbol limit applies only per module not the entire program. Therefore programs can be more readable and better documented.

Modules can be saved and used on different programs. Therefore the software investment of the customer is maintained.

RL51 produces two files. The absolute object module file can be directly executed by the 8051 family. The listing file shows the results of the link/locate process.



# CONV51 8048 TO 8051 ASSEMBLY LANGUAGE CONVERTER UTILITY PROGRAM.

- Enables software written for the MCS-48 family to be upgraded to run on the 8051
- Maps each 8048 instruction to a corresponding 8051 instruction
- Preserves comments; translates 8048 macro definitions and calls
- Provides diagnostic information and warning messages embedded in the output listing

The 8048 to 8051 Assembly Language Converter is a utility to help users of the MCS-48 family of microcomputers upgrade their designs with the high performance 8051 architecture. By converting 8048 source code to 8051 source code, the software investment developed for the 8048 is maintained when the system is upgraded.

The goal of the converter (CONV51) is to attain functional equivalence with the 8048 code by mapping each 8048 instruction to a corresponding 8051 instruction. In some cases a different instruction is produced because of the enhanced instruction set (e.g., bit CLR instead of ANL).

Although CONV51 tries to attain functional equivalence with each instruction, certain 8048 code sequences cannot be automatically converted. For example, a delay routine which depends on 8048 execution speed would require manual adjustment. A few instructions, in fact, have no 8051 equivalent (such as those involving P4-P7). Finally, there are a few areas of possible intervention such as PSW manipulation and interrupt processing, which at least require the user to confirm proper translation. The converter always warns the user when it cannot guarantee complete conversion.

CONV51 produces two files. The output file contains the ASM51 source program produced from the 8048 instructions. The listing file produces correlated listings of the input and output files, with warning messages in the output file to point out areas that may require users' intervention in the conversion.

## SPECIFICATIONS

### OPERATING ENVIRONMENT

All Intel Microcomputer Development Systems or Intel Personal Development System

### Documentation Package:

- MCS-51 Macro Assembler User's Guide
- MCS-51 Utilities User's Guide for 8080/8085 Based Development System
- MCS-51 8048-to-8051 Assembly Language Converter Operating Instructions for ISIS-II Users

## ORDERING INFORMATION

Part Number	Description
MCI-51-ASM	8051 Software Development Package

\*Requires Software License

## SUPPORT:

Hotline Telephone Support, Software Performance Reporting (SPR), Software Updates, Technical Reports, Monthly Newsletter available.



## PL/M 51 SOFTWARE PACKAGE

- High-level programming language for the Intel MCS<sup>®</sup>-51 single-chip microcomputer family
- Compatible with PL/M 80 assuring MCS<sup>®</sup>-80/85 design portability
- Enhanced to support boolean processing
- Tailored to provide an optimum balance among on-chip RAM usage, code size and code execution time
- Allows programmer to have complete control of microcomputer resources
- Produces relocatable object code which is linkable to object modules generated by all other 8051 translators
- Extends high-level language programming advantages to microcontroller software development
- Improved reliability, lower maintenance costs, increased programmer productivity and software portability
- Includes the linking and relocating utility and the library manager
- Supports all members of the Intel MCS<sup>®</sup>-51 architecture

PL/M 51 is a structured, high-level programming language for the Intel MCS-51 family of microcomputers. The PL/M 51 language and compiler have been designed to support the unique software development requirements of the single-chip microcomputer environment. The PL/M language has been enhanced to support Boolean processing and efficient access to the microcomputer functions. New compiler controls allow the programmer complete control over what microcomputer resources are used by PL/M programs.

PL/M 51 is largely compatible with PL/M 80 and PL/M 86. A significant proportion of existing PL/M software can be ported to the MCS-51 with modifications to support the MCS-51 architecture. Existing PL/M programmers can start programming for the MCS-51 with a small relearning effort.

PL/M 51 is the high-level alternative to assembly language programming for the MCS-51. When code size and code execution speed are not critical factors, PL/M 51 is the cost-effective approach to developing reliable, maintainable software.

The PL/M 51 compiler has been designed to support efficiently all phases of software implementation with features like a syntax checker, multiple levels of optimization, cross-reference generation and debug record generation.

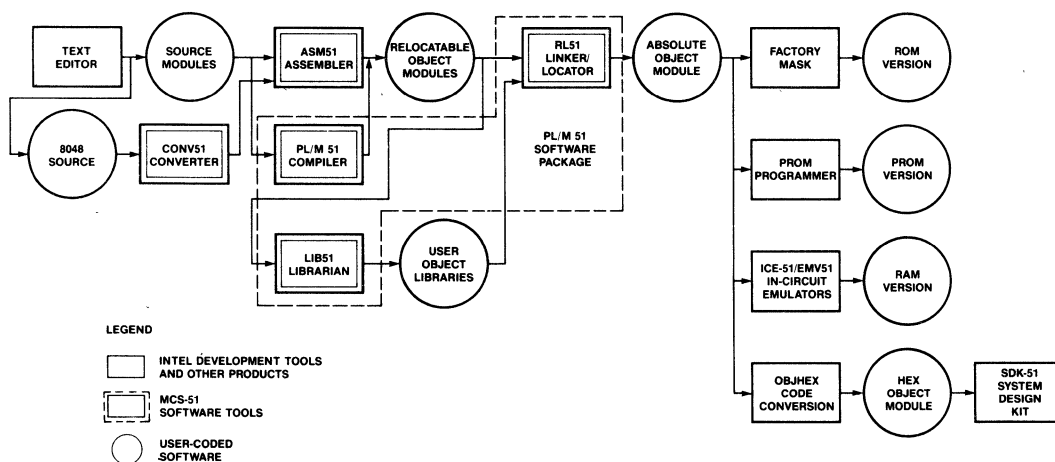


Figure 1. MCS<sup>®</sup>-51 Program Development Process

## PL/M 51 Compiler

### FEATURES

Major features of the Intel PL/M 51 compiler and programming language include:

#### Structured Programming

PL/M source code is developed in a series of modules, procedures, and blocks. Encouraging program modularity in this manner makes programs more readable, and easier to maintain and debug. The language becomes more flexible, by clearly defining the scope of user variables (local to a private procedure, for example).

#### Language Compatibility

PL/M 51 object modules are compatible with object modules generated by all other MCS-51 translators. This means that PL/M programs may be linked to programs written in any other MCS-51 language.

Object modules are compatible with In-Circuit Emulators and Emulation Vehicles for MCS-51 processors; the DEBUG compiler control provides these tools with symbolic debugging capabilities.

#### Supports Three Data Types

PL/M makes use of three data types for various applications. These data types range from one to sixteen bits and facilitate various arithmetic, logic, and address functions:

- Bit: a binary digit
- Byte: 8-bit unsigned number or,
- Word: 16-bit unsigned number.

Another powerful facility allows the use of BASED variables that map more than one variable to the same memory location. This is especially useful for passing parameters, relative and absolute addressing, and memory allocation.

#### Two Data Structuring Facilities

PL/M 51 supports two data structuring facilities. These add flexibility to the referencing of data stored in large groups.

- Array: Indexed list of same type data elements
- Structure: Named collection of same or different type data elements
- Combinations of Both: Arrays of structures or structures of arrays.

#### Interrupt Handling

A procedure may be defined with the INTERRUPT attribute. The compiler will generate code to save and restore the processor status, for execution of the user-defined interrupt handler routines.

#### Compiler Controls

The PL/M 51 compiler offers controls that facilitate such features as:

- Including additional PL/M 51 source files from disk
- Cross-reference
- Corresponding assembly language code in the listing file

#### Program Addressing Control

The PL/M 51 compiler takes full advantage of program addressing with the ROM (SMALL/MEDIUM/LARGE) control. Programs with less than 2 KB code space can use the SMALL or MEDIUM option to generate optimum addressing instructions. Larger programs can address over the full 64 KB range.

#### Code Optimization

The PL/M 51 compiler offers four levels of optimization for significantly reducing overall program size.

- Combination or “folding” of constant expressions; “Strength reductions” (a shift left rather than multiply by 2)
- Machine code optimizations; elimination of superfluous branches
- Automatic overlaying of on-chip RAM variables
- Register history: an off-chip variable will not be reloaded if its value is available in a register.

#### Error Checking

The PL/M 51 compiler has a very powerful feature to speed up compilations. If a syntax or program error is detected, the compiler will skip the code generation and optimization passes. This usually yields a 2X performance increase for compilation of programs with errors.

A fully detailed set of programming and compilation error messages is provided by the compiler and user's guide.

**BENEFITS**

PL/M 51 is designed to be an efficient, cost-effective solution to the special requirements of MCS-51 Microsystem Software Development, as illustrated by the following benefits of PL/M use:

**Low Learning Effort**

PL/M 51 is easy to learn and to use, even for the novice programmer.

**Earlier Project Completion**

Critical projects are completed much earlier than otherwise possible because PL/M 51, a structured high-level language, increases programmer productivity.

**Lower Development Cost**

Increases in programmer productivity translate immediately into lower software development costs because less programming resources are required for a given programmed function.

**Increased Reliability**

PL/M 51 is designed to aid in the development of reliable software (PL/M programs are simple statements of the program algorithm). This substantially reduces the risk of costly correction of errors in systems that have already reached full production status, as the more simply stated the program is, the more likely it is to perform its intended function.

**Easier Enhancements and Maintenance**

Programs written in PL/M tend to be self-documenting, thus easier to read and understand. This means it is easier to enhance and maintain PL/M programs as the system capabilities expand and future products are developed.

---

**RL51 Linker and Relocator**

- **Links modules generated by the assembler and the PL/M compiler**
- **Locates the linked object to absolute memory locations**
- **Enables modular programming of software-efficient program development**
- **Modular programs are easy to understand, maintainable and reliable**

The MCS-51 linker and relocater (RL51) is a utility which enables MCS-51 programmers to develop software in a modular fashion. The utility resolves all references between modules and assigns absolute memory locations to all the relocatable segments, combining relocatable partial segments with the same name.

With this utility, software can be developed more quickly because small functional modules are easier to understand, design and test than large programs.

The total number of allowed symbols in user-developed software is very large because the assembler number of symbols' limit applies only per module, not to the entire program. Therefore programs can be more readable and better documented.

Modules can be saved and used on different programs. Therefore the software investment of the customer is maintained.

RL51 produces two files. The absolute object module file can be directly executed by the MCS-51 family. The listing file shows the results of the link/locate process.

## LIB51 Librarian

The LIB51 utility enables MCS-51 programmers to create and maintain libraries of software object modules. With this utility, the customer can develop standard software modules and place them in libraries, which programs can access through a standard interface. When using object libraries, the linker will

call only object modules that are required to satisfy external references.

Consequently, the librarian enables the customer to port and reuse software on different projects —thereby maintaining the customer's software investment.

---

### SPECIFICATIONS

#### Operating Environment

All Intel Microcomputer Development Systems or Intel Personal Development Systems

#### Documentation Package

PL/M 51 User's Guide  
MCS-51 Utilities User's Guide

---

### ORDERING INFORMATION

#### Part Number

iMDX 352  
Requires Software License

#### Description

PL/M 51 Software  
Package

### SUPPORT:

Hotline Telephone Support, Software Performance Report (SPR), Software Updates, Technical Reports, and monthly Technical Newsletters are available.





## MCS<sup>®</sup>-96 SOFTWARE SUPPORT PACKAGE

- Symbolic relocatable assembly language programming for the 8096 microcontroller family
- System Utilities for Program Linking and Relocation
- Extends Intel<sup>®</sup> Microcomputer Development System to support MCS-96 program development
- Encourages modular program design for maintainability and reliability

The MCS-96 Software Support Package provides development system support for the MCS-96 family of 16 bit single chip microcomputers. The support package includes a macro assembler and system utilities.

The assembler produces relocatable object modules from MCS-96 macro assembly language instructions. The object modules then are linked and located to absolute memory locations.

The assembler and utilities run on the Intel<sup>®</sup> Series III or equivalent Microcomputer Development System.

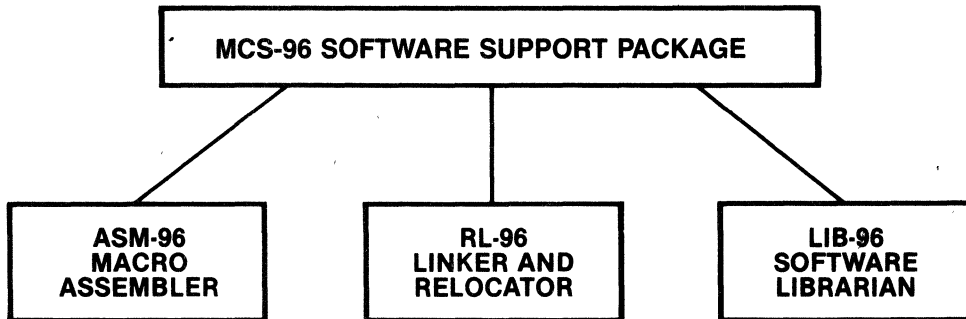


Figure 1. MCS-96 Software Support Package

## 8096 MACRO ASSEMBLER

- **Supports 8096 family program development on Intellec® Microcomputer Development System**
- **Gives symbolic access to powerful 8096 hardware features**
- **Object files are linkable and locatable**
- **Symbolic Assembler supports macro capabilities, cross reference, symbol table and conditional assembly**

ASM-96 is the macro assembler for the iACX family of microcontrollers. ASM-96 translates symbolic assembly language mnemonics into relocatable object code. Since the object modules are linkable and locatable, ASM-96 encourages modular programming practices.

The macro facility in ASM-96 allows programmers to save development and maintenance time since common code sequences only have to be done once. The assembler also provides conditional assembly capabilities.

ASM-96 supports symbolic access to the many features of the 8096 architecture. An "Include" file is provided with all of the 8096 hardware registers defined. Alternatively, the user can define any subset of the 8096 hardware register set.

Math routines are supported with mnemonics for 16 x 16-bit multiply or 32/16-bit divide instructions.

The assembler runs on a Series III/Series IV Intellec Development Systems for high performance.

---

## RL96 LINKER AND RELOCATOR PROGRAM

- **Links modules generated by the assembler**
- **Locates the linked object module to absolute memory locations**
- **Encourages modular programming for faster program development**
- **Automated selection of required modules from Libraries to satisfy symbolic references**

RL96 is a utility that performs two functions useful in MCS-96 software development:

- The link function which combines a number of MCS-96 object modules into a single program.
- The locate functions which assigns an absolute address to all relocatable addresses in the MCS-96 object module.

RL96 resolves all external symbol references between modules and will select object modules from library files if necessary.

RL96 creates two files:

- The program or absolute object module file that can be executed by the targeted member of the MCS-96 family.
- The listing file that shows the results of link/locate, including a memory map symbol table and an optional cross reference listing.

The relocater allows programmers to concentrate on software functionality and not worry about the absolute addresses of the object code. RL96 promotes modular programming. The application can be broken down into separate modules that are easier to design, test and maintain. Standard modules can be developed and used in different applications thus saving software development time.

---

## **LIB 96**

The LIB 96 utility creates and maintains libraries of software object modules. The customer can develop standard modules and place them in libraries. Application programs can then call these modules using predefined interfaces.

LIB 96 uses the following set of commands:

- CREATE: Creates an empty library file.
- ADD: Adds object modules to a library file.
- DELETE: Deletes object modules from a library file.
- LIST: Lists the modules in the library file.
- EXIT: Terminates LIB 96

When using object libraries, RL96 will include only those object modules that are required to satisfy external references, thus saving memory space.

---

### **SPECIFICATIONS**

#### **Operating Environment**

**REQUIRED HARDWARE:**  
Intellic Microcomputer Development System  
—Series III/Series IV

#### **Documentation Package:**

MCS-96 MACRO ASSEMBLER USER'S GUIDE  
MCS-96 UTILITIES USER'S GUIDE  
MCS-96 ASSEMBLER AND UTILITIES POCKET  
REFERENCE CARD

---

### **ORDERING INFORMATION**

#### **Part Number**

#### **Description**

iMDX-355

MCS-96 SOFTWARE SUPPORT PACKAGE

Requires Software License

---

---

# **Productivity Tools and Communications Software**

---

**4**



# PRODUCTIVITY TOOLS AND COMMUNICATIONS SOFTWARE

## INTRODUCTION

Improving an engineering team's productivity is a never ending task in today's competitive environments. Intel offers software tools and communication systems that optimize the usage of expensive engineering personnel and capital equipment. Software tools boost a programming team's productivity, thereby lowering development costs and shortening product development times. Communication software provides further productivity gains by linking multi-computer engineering environments into highly effective networks.

One software tool that substantially increases software productivity is PSCOPE, a source level symbolic debugger. The PSCOPE debugger allows the high-level language programmer to completely debug his code at the same level at which it was written. Breakpointing, tracing, and patching are all done in a faster and less error-prone manner than through obsolete machine-level debuggers. As software testing and maintenance consume a greater portion of development life-cycle time and cost, PSCOPE debugging can significantly improve programming efficiency.

Another set of valuable software tools are Intel's Program Management Tools (PMTs), which provide the essential ingredients to manage large software development projects. PMTs decrease the time spent on tracking program changes and manually generating new systems, thereby giving engineers more time for software design, development, and testing. PMTs consist of a Software Version Control System (SVCS), and an automated software generation facility (MAKE). Together these tools control, examine, and automate the management of a software system that may contain many versions consisting of numerous modules.

Intel's software toolboxes are collections of utilities that perform a variety of productivity-oriented functions. The ISIS-II Software Toolbox offers conditional submit file control tools, source management tools, and other tools that operate at the ISIS-II command level. The 8086 Software Toolbox is a collection of 16-bit software tools that are valuable for text formatting and preparation, software testing and performance analysis, 286/287 software development, and a multitude of other applications.

Intel also offers AEDIT, an advanced editor that significantly improves programmer productivity. AEDIT was designed with the programmer in mind, and offers full screen editing, the ability to edit two files at once, features for manipulating large blocks of text, and dynamic macro command definition. In addition, Intel continues to offer and support CREDIT, an 8-bit CRT-based editor.

The trend toward distributed data processing is well supported by Intel's complete line of communication software and systems. Mainframe Link integrates user mainframe computers with Inteltec Development systems by emulating the operation of an IBM 2780 or 3780 Remote Job Entry terminal. The Asynchronous Communication Link enables one or more Intel Microcomputer Development Systems to communicate with a Digital Equipment Corporation VAX computer. The iNA 950 and iNA 960 are ready-to-use communication software building blocks for OEM suppliers of networked systems for both technical and commercial applications. Finally, NDS-II Electronic Mail enables users to send and receive messages and files between any nodes on the NDS-II network.



# PSCOPE HIGH-LEVEL PROGRAM DEBUGGER

- Source-Level Debugging for High Productivity
- Breakpoint, Single-Step and Execution Trace by Statement Numbers, Procedure Names and Labels
- High-Level Code Patching
- Compatible with Intel's i<sup>2</sup>ICE™ Integrated Instrumentation and In-Circuit Emulation System for Target System Debugging
- Native CPU Execution for iAPX 88 and 86 Architectures
- Supports PL/M, Pascal, and FORTRAN Program Debugging

PSCOPE is an interactive, symbolic debugger for high-level language programs. It allows users to scrutinize program execution at the source level, using high-level statement numbers, procedure and variable names and labels. This is typically a more productive way of debugging high-level language (HLL) programs than at the machine level.

Source-level debugging means that traditional functions, such as setting breakpoints or tracing execution flow, are more powerful in PSCOPE. For example, tracing procedure entry (or exit) points conveys much more information than tracing machine instructions. Single-step execution is more powerful, using statements and procedures, as well.

The productivity improvement from debugging in a high-level language is analogous to programming in a high-level language, when compared to assembly-level programming and debugging.

PSCOPE users may define high-level code patches, which are "compiled" and patched into the user's program. Code patches may be stored on disk, so they may be later incorporated into the program source file.

PSCOPE is an integral part of the advanced i<sup>2</sup>ICE Integrated Instrumentation and In-Circuit Emulation System. This allows a smooth migration from *program* debugging to *target system* debugging.

PSCOPE's symbol capacity is virtually unlimited. Symbols are paged to disk when necessary.

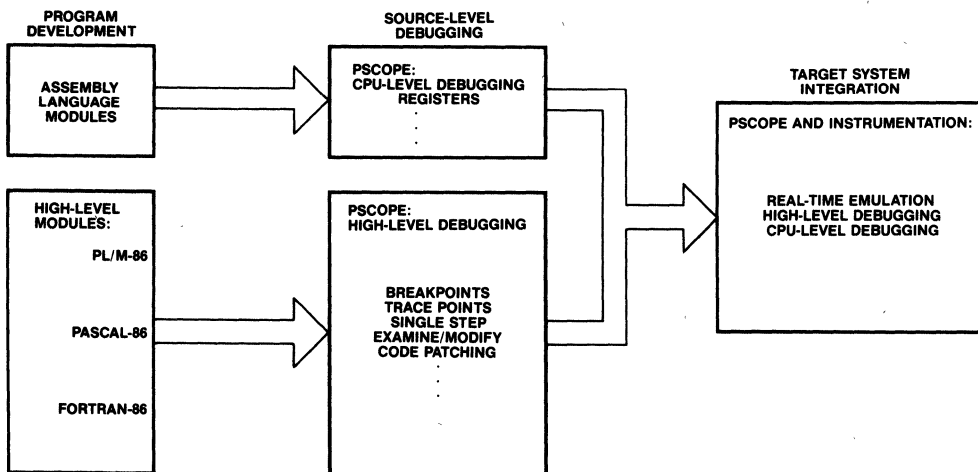


Figure 1. Debugging Methodology with PSCOPE

### SAMPLE DEBUGGER SESSION

```

SERIES-III Pascal-86, V1.1

Source File: :F2:MAXMIN.PAS
Object File: :F2:MAXMIN.OBJ
Controls Specified: DEBUG.

STMT LINE NESTING          SOURCE TEXT: :F2:MAXMIN.PAS
  1     1  0  0          program calc(input,output);
  2     2  0  0          var a,b:integer;

  3     4  0  0          procedure sum(x,y:integer);
  4     5  1  0          var z:integer;
  5     6  1  0          begin
  5     7  1  1          z:=x*y;
  6     8  1  1          writeln('The sum is ',z);
  7     9  1  1          end;

  8    11  0  0          procedure difference(x,y:integer);
  9    12  1  0          var z:integer;
 10    13  1  0          begin
 10    14  1  1          z:=abs(x-y);
 11    15  1  1          writeln('The difference is ',z);
 12    16  1  1          end;

 13    18  0  0          procedure maxmin(x,y:integer);
 14    19  1  0          begin
 14    20  1  1          if x<y then writeln('The maximum is ',y,
                        ' The minimum is ',x);
 16    21  1  1          if y<x then writeln('The maximum is ',x,
                        ' The minimum is ',y);
 18    22  1  1          if x=y then writeln('The two inputs are equivalent ');
 20    23  1  1          end;

 21    25  0  0          begin
 21    26  0  1          repeat (*forever*)
 21    27  0  2          write('Input two integers ');
 22    28  0  2          readln(a,b);

 23    30  0  2          sum(a,b);
 24    31  0  2          difference(a,b);
 25    32  0  2          maxmin(a,b);
 26    33  0  2          until l<0
 27    34  0  2          end.

```

The program listing for the sample PSCOPE session illustrates the high-level nature of PSCOPE debugging. The program consists of the module CALC, the procedures SUM, DIFFERENCE, and MAXMIN, plus global and local variables. Users exercise and manipulate the program using these symbols. Code

patches, stepping, tracing, etc. are all done on line numbers, procedures, labels, and symbolic names. To debug a program, just PSCOPE and a listing are required—no linkage maps, core dumps, locate maps, etc. are necessary. This is how high-level debugging relates to high-level programming.



## FEATURES

### Unlimited Breakpoints

Breakpoints may be set on statement numbers, procedure names, or program labels. Any number of breakpoint registers may be defined.

### High-Level Trace Points

Execution trace points are defined the same way as program breaks. Any number of trace points may be defined. A trace message is displayed when execution reaches a trace point.

### Conditional Break and Trace

Any break or trace point may be defined to automatically call a *debugger procedure*, which will execute PSCOPE commands and/or evaluate predefined conditions. The operations will be performed, and the condition will determine if the break or trace will be done.

### GO

The GO command initiates program execution from any starting point. A set of stopping points may be specified ("GO TIL"), and break/trace registers may be used ("GO USING").

### Source-Level Stepping

A program may be executed, one high-level statement at a time, using the LSTEP command. Also, entire procedures may be treated as single statements during stepping (PSTEP); the procedures will be executed, but not stepped through.

### Examine/Modify Data

PSCOPE allows users to symbolically examine (and change the value of) program variables and data structures. All PL/M and Pascal types are supported, including numerics, dynamic and stack variables, arrays, and fields within structures.

### Virtual Symbol Table

All user-program symbols are stored in a virtual symbol table. This means symbols will be paged to disk, if necessary.

### Help File

Many PSCOPE commands, facilities, and error messages have help information describing their use. The HELP command is used for learning the

PSCOPE command language, for quick reference of command syntax, and for learning the cause of command errors.

### Debugger Procedures

PSCOPE has the facility for defining procedures in its command language. This block-structured command language allows users to *extend* the capability of the program under debug. Like macros with parameters, these procedures may also be used for generating compound and conditional debugger commands.

### Code Patching

Program patches may be written in the debugger command language to augment or replace current program statements. These high-level code patches are much closer to actual program changes than machine-level patches, and are easy to use.

### Built-in Editor

A menu-driven, CRT-oriented editor is built into PSCOPE. This is used for creating and editing program patches, debugger procedures, and command lines. One key is used to invoke the editor to alter the last command entered, or any debugger definition (literally, trace register, patch, etc.) may be edited selectively.

### Debugger Command Language

GO/LSTEP/PSTEP—For controlling program execution.

DEFINE/DISPLAY/MODIFY/REMOVE—For manipulating debugger objects (such as break registers, patches, and procedures), or program objects (variables and data structures).

CALL/RETURN—For executing debugger procedures.

WRITE/CI—For console input and output.

DO/END—For defining command blocks.

REPEAT/COUNT—For repetition of commands or blocks.

IF/THEN/ELSE—For conditional execution of commands or blocks.

INCLUDE/PUT/APPEND—For saving/restoring commands and definitions to and from disk.

## **BENEFITS**

### **Shortened Development Cycle**

The ability to define debugger procedures and make code patches is very useful. It actually allows users to extend the capability of the program under debug. After debug sessions, users typically make program changes or enhancements. This involves the use of an editor, compiler and linkage tools that create a "new" load module for debugging. Since PSCOPE allows these changes and enhancements to be made *in the debugger*, the number of Edit/Compile/Link iterations is lowered. More confidence can be placed on a program during debugging, because its capabilities have been more fully exercised.

### **Improved Debugging Productivity**

PSCOPE provides users with the same conceptual interface to program debugging that was used in program design. This includes the high-level language constructs such as statements, procedures, labels and symbolic variables and data structures. Functions such as program trace and single-step execution are more meaningful with statements and procedures than machine instructions; therefore the improvement in debugging productivity is analogous to the programming productivity using high-level languages.

### **More Reliable Software**

Debugger procedures may be used to automate the software testing process. The procedure may repeatedly generate test values, execute the program with the input values, and record the results. Running more comprehensive tests, plus being able to "batch" the tests, yields more reliable software.

### **Easy to Learn and Use**

An extensive command language, which is similar to block-structured languages such as PL/M and Pascal, is very easy to use in an interactive debug session. The HELP facility makes learning to use PSCOPE extremely fast as well. The "Literally" facility and debugger procedures also allow users to extend and tailor the command language to suit individual needs.

### **Improved Software Management**

The use of debugger procedures allows parts of a software system to be debugged independently. Procedures can be substituted for program stubs, allowing programmers to debug different pieces of the system separately. This results in improved project management.

---

## **SPECIFICATIONS**

Supports Intel's standard 86/88 languages:

- PL/M 86/88
- Pascal 86/88
- FORTRAN 86/88

PSCOPE runs on an Intellec® Series III or Series IV Microcomputer Development System, either stand-alone or in an NDS-II network configuration. A 512K application memory space is recommended for most applications.

---

## **ORDERING INFORMATION**

<b>Order Code</b>	<b>Description</b>
iMDX-333	PSCOPE Program Debugger (for Series III and Series IV)
III-951A	PSCOPE Program Debugger and I <sup>2</sup> ICE Base Software for Series III with 8" single density disk drive
III-951B	PSCOPE Program Debugger and I <sup>2</sup> ICE Base Software for Series III with 8" double density disk drive
III-951C	PSCOPE Program Debugger and I <sup>2</sup> ICE Base Software for Series IV with 5¼" double density disk drive

---

```

-run :fl:pscope
SERIES-III PSCOPE-86, V1.0
*
*
*define literally d = 'define'
*d literally l = 'literally'
*d l br = 'brkreg'
*d l tr = 'trcreg'
*
*
*load :tl:maxmin.86
*dir
DIR of :CALC
PQ_OUTPUT . . . . . TEXT (file)
PQ_INPUT . . . . . TEXT (file)
B . . . . . integer
A . . . . . integer
SUM . . . . . procedure
X . . . . . integer
Y . . . . . integer
Z . . . . . integer
DIFFERENCE . . . . . procedure
X . . . . . integer
Y . . . . . integer
Z . . . . . integer
MAXMIN . . . . . procedure
X . . . . . integer
Y . . . . . integer
*
*
*ptest
[Step at :CALC#21]
*ptest
      INPUT TWO INTEGERS:
[Step at :CALC#22]
*ptest
      (input)  19  4
[Step at :CALC#23]
*ptest
      THE SUM IS 76
[Step at :CALC#24]
*ptest
      THE DIFFERENCE IS 15
[Step at :CALC#25]
*ptest
      THE MAXIMUM IS 19
      THE MINIMUM IS  4
[Step at :CALC#21]
*
*
*define patch #5 t1l #6 = z=x+y
*
*go t1l #21
      INPUT TWO INTEGERS:
      (input)  19  4
      THE SUM IS 23
      THE DIFFERENCE IS 15
      THE MAXIMUM IS 19
      THE MINIMUM IS  4
[Break at #21]
*
*
*define proc PR1 = do
.*write 'the numbers and product are: ',a,b,a*b
.*write using ('0,>') 'break ? '
.*if CI == 'y' then return true
.* else return false endif
.*end
*
*d br B3 = #21 call PR1
*go using b3
      INPUT TWO INTEGERS:
      (input)  23  24
      THE SUM IS 47
      THE DIFFERENCE IS 1
      THE MAXIMUM IS 24
      THE MINIMUM IS 24
the numbers and the product are: +23 +24 +552
break ? y
[Break at #21]
*
*
*exit
PSCOPE terminated
-

```

The Literally facility allows users to abbreviate, redefine and extend the command language to suit individual needs.

Any PL/M-86, Pascal-86 or FORTRAN-86 program may be loaded. All symbolic names may be displayed, in total or by type. Symbols defined at debug time may be displayed as well. All program types are supported, including numerics, user-defined types, and records. The symbols' types are displayed by the DIR command as well.

Several flavors of stepping are offered. This example illustrates PSTEP, a line-by-line step where procedures are executed as a single step. This program contains five steps in the main body, with three being procedure calls.

There appears to be a bug in the program, as the sum is displayed incorrectly. Looking at the program, we notice that X and Y were multiplied instead of added, at line #5. A code patch is defined, and the program executes correctly.

This illustrates the facility where a *debug procedure* (PR1) is called when reaching a breakpoint at line #21. Here, some values are displayed, and a condition is evaluated (in this case, a query to the user). Had the condition been false, program execution would continue with no break. The high-level constructs in the command language make this a very powerful facility.



## PROGRAM MANAGEMENT TOOLS

- **Increase Software Engineering Productivity**
- **Decrease Software Administration Overhead**
- **Allow Users to Control, Automate and Examine the Evolution of a Software Project**
- **Enhance the Capability of Networked (NDS-II) and Standalone Development Systems**
- **SVCS Simplifies Administration of Software Modules and Systems**
- **MAKE Automatically Generates New Releases of Software Systems**
- **Both Tools Easily Incorporated Into Existing Software Development Methodologies**

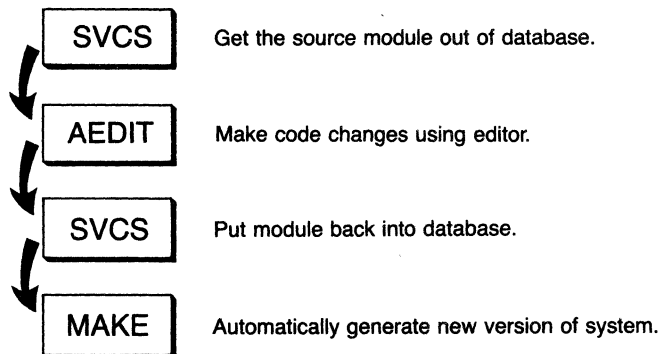
Intel's Program Management Tools (PMTs) provide the essential ingredients to manage large software development projects. PMTs decrease the time spent on tracking program changes and manually generating new systems, thereby giving engineers more time for software design, development, and testing.

PMTs consist of a "Software Version Control System" (SVCS), and an automated software generation facility (MAKE). Together these tools control, examine, and automate the management of a software system that may contain many versions consisting of numerous modules.

SVCS controls and documents software changes for all file types. SVCS handles storage and retrieval of different versions of a given module, controls update privileges, prevents different users from making changes independently, and requires all changes be thoroughly documented by recording who made what changes, when and why.

MAKE produces the specification of a "minimum-work" job required to generate a new system. This job (i.e. submit file) typically includes compiles and links of the latest versions of specified source and object modules. If a newer source module exists for any specified object module, MAKE will specify a compile of this module, replacing the older module in the completed program. Unnecessary links and compiles, however, are eliminated. MAKE does the minimum work required to ensure consistent, up-to-date software, thus saving many hours of compiles and links.

Incorporating PMTs into an existing project is easy. PMTs work with existing operating systems and software tools (editors, compilers, utilities) and require very little relearning. New users can quickly gain expertise in using PMTs by working through the examples contained in the PMT Tutorial Manual and Diskette, which are included with every PMT software package. Program Management Tools are ideal in a networked (NDS-II) environment, where multi-version software control is critical. PMTs are also extremely valuable on standalone systems (with Winchester disk) as well.



### OPTIMAL CONTROL OF A SOFTWARE PROJECT.

## SOFTWARE VERSION CONTROL SYSTEM (SVCS)

- **Simplifies Administration of Software Modules and Systems**
- **Maintains Change History Information on Every Module**
- **Prevents Users From Accidentally Deleting System Software or Making Simultaneous Module Changes**
- **Offers an Effective Software Version Generation and Control Mechanism**

Intel's Software Version Control System (SVCS) is a utility that greatly simplifies software system housekeeping. SVCS automatically controls and documents software modules in a large project, eliminating costly manual administration by a project leader or librarian.

SVCS maintains a system database of software modules called units. Each unit is divided into four classes: Source, which contains the unit's source code; Object, which contains the unit's object code; History, which contains the unit's history file; and Composition, which can be arbitrarily used by the user.

Users interact with the database by using SVCS administrative and access commands. Project managers use administrative commands to create new system databases, add and delete database units, set unit access rights, and create and name new system variants. Programmers use SVCS access commands to check out and return database modules when making system changes. For every change made, SVCS records *what* changed, *who* changed it, *when* it was changed, and *why*.

SVCS variant generation and control enable project administrators to effectively create and identify new versions of software systems. Stable versions may be write protected and placed in the public domain, working versions may be identified and accessible only to programming personnel, and special versions may be created for customized releases. In addition, version control can minimize software archival, maintenance, and support administrative overhead.

---

## AUTOMATED SOFTWARE GENERATION (MAKE)

- **Automatically Creates New Software Systems, Using the Latest Versions of Source Modules**
- **Automatically Determines Which Source Modules Need Recompiling**
- **Eliminates Unnecessary Compiles and Links**
- **Works Closely with SVCS for Generating Complete, Up-To-Date Systems**
- **Easily Adopted into Existing Development Methodologies**
- **Offers Many Powerful Macro Constructs**

MAKE is a utility that greatly simplifies the generation of software systems. MAKE produces a "minimum-work" submit file that can generate a complete, up-to-date system without any unnecessary compiles and links. MAKE can reduce system generation times from hours to minutes while concurrently minimizing administrative overhead.

MAKE accepts a text input file that instructs it how to generate a new software system. The input file specifies all modules required to generate the new system and includes a description of system dependencies. It also specifies specific system operations, such as compiles, links, SVCS operations, line-printer spoolings, and other system commands. MAKE uses this input file in conjunction with the time and date stamps on each module to determine the optimum system generation procedure that eliminates all unnecessary compiles and links.

Typically a MAKE input file is created once at the start of a project. Very occasionally during the life of the project it may need modification. A powerful set of macros makes the creation and subsequent modification of a generation procedure an easy task. Overall, the management of the MAKE input file is negligible compared to maintaining numerous submit files for system generations.

The close relationship between SVCS and MAKE help simplify the overall job of software control at all levels. For example, the very latest version of a source module may not be stable enough to be included in a generation. A less functional, but more reliable version may exist. Since SVCS keeps unique versions distinct, an SVCS-module containing the more reliable version may be specified in the MAKE input file.

---

## BENEFITS: SVCS AND MAKE

Intel's Program Management Tools eliminate common problems such as:

"We've modified module F00, which has introduced a new set of problems. Now we can't restore it back to the earlier version."

"Module F002 has been modified; no one seems to know who changed it, or why."

"We often have several programmers making changes to the same modules. Trying to avoid simultaneous changes is a lot of effort, and we waste time synthesizing two sets of changes into one module."

"To ensure that we release up-to-date, correct software, we periodically go into "release mode" for a few days. Everyone stops work completely while we find the latest versions, and then start the generation from the ground up. It literally takes days, when we could be making productive changes."

SVCS and MAKE together provide a service that fits easily into your existing design methodology, and solves administrative problems such as those described above.

## SPECIFICATIONS

### Networked, Multi-User Software Control

NDS-II with at least one Intel Microcomputer Development System  
iNDX, ISIS-III(N) System Software

### Standalone Use

Intel Series III with Model 750 Winchester Disk or Intel Series IV  
SVCS and MAKE will not operate on ISIS-II local floppies or Model 740 Hard Disks.  
SVCS and MAKE may be exported from any workstation in an NDS-II configuration.

### Documentation

"A User's Guide to Program Management Tools" (121958)

## SOFTWARE SUPPORT

This product includes a 90-day initial support consisting of new software releases, updates, subscription services (software performance reports and technical reports), and telephone hotline support. Additional software support services are available separately.

## ORDERING INFORMATION

Part Number	Description
iMDX-332	Intel Program Management Tools



## FUNCTIONAL DESCRIPTION

### Submit File Execution Control

**IF/ELSE/ENDIF**—conditional submit file execution based on file existence, program errors, pattern matching, plus several other conditions

**GOTO**—causes submit execution to resume at a specified label

**RETURN**—causes execution to return to the “submitter” (calling file)

**EXIT**—halts submit file execution

**LOOP**—forces execution to resume at the beginning of the submit file

**RESCAN**—allows submit execution to begin anywhere in file

**NOTE**—allows “progress report” notes to be placed in submit files

**WAIT**—displays a message, and waits for user input to continue or abort

**STOPIF**—halts submit file execution if specified listing contains errors

### Source Management

**XLATE2**—submit-like tool with intelligent parameter substitution (for version control)

**MRKOBJ**—“marks” object modules with source version information

**CHKLOD**—lists source version data put in load modules by MRKOBJ

**CLEAN**—deletes all old versions off a specified disk

**LATEST**—displays latest version numbers of specified files

### Operating System Functions

**CONSOL**—reassigns console input and console output as directed

**DSORT\***—alphabetically sorts floppy disk and hard disk directories

**RELAB\***—changes disk name to any other specified name

### Program Development and Debugging

**ERRS**—fast display of program errors in PL/M 80, PL/M 86, and ASM 86 listings

**MERG80**—merges debug data from locate maps into PL/M 80 listings

**MERG86**—merges debug data from symbol maps into PL/M 86 and Pascal 86 listings

**GENPEX**—produces include file for PL/M external declarations (source level)

**PASSIF**—general purpose assertion checking, testing, and reporting tool

### Text Processing

**COMPAR**—performs line-oriented text file comparison (shows source changes)

**UPPER**—changes all letters in an ASCII text file to uppercase

**LOWER**—changes all letters in an ASCII text file to lowercase

**LAST**—displays the last 512 bytes of a file

**SORT**—sophisticated line-oriented text file sorting tool

### Disk Backup and File Processing

**DCOPY**—fast track-by-track diskette copying

**HDBACK**—sophisticated hard disk to floppy disk backup program

**PACK**—compacts text files by removing strings of blanks

**UNPACK**—reconstitutes “packed” files

### Disk Recovery

**GANEF\***—interactively reads and writes floppy or hard disk data blocks

### Program Identification

**WHICH**—displays version number of Software Toolbox Programs

\*These programs will not operate on NDS-II remote directories.

This product includes a 90-day initial support consisting of new software releases, updates, subscription services (software performance reports and technical reports), and telephone hotline support. Additional software support services are available separately.

## ORDERING INFORMATION

Product Code	Description
MDS-363 <sup>†</sup>	ISIS-II SOFTWARE TOOLBOX

Requires software license.

<sup>†</sup>MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Science.





# 8086 SOFTWARE TOOLBOX

- **Collection of Tools That Speed Software Development**
- **MPL, a Standalone Macro Processor, is Ideal for Debugging Macros**
- **SCRIBL and SPELL Assist Text Preparation**
- **OMC 286 and 287 EMULATOR Aid 80286 and 80287 Software Development**
- **Many Other Valuable 16-Bit Software Tools Are Included**
- **Runs on Series III and Series IV Microcomputer Development Systems**

The 8086 Software Toolbox is a collection of 16-bit software tools that can significantly improve programmer productivity. These tools are valuable for text formatting and preparation, software testing and performance analysis, 286/287 software development, and a multitude of other applications.

Text processing tools ease document formatting and preparation. SCRIBL is a text formatting program that uses commands embedded in text to do paging, centering, left and right margins, subscripts, etc. SPELL finds misspelled words in a text file and comes with a user expandable dictionary. COMP compares two text or source files and displays their differences.

Test and performance analysis tools aid software testing and performance evaluation. PERF, a performance analysis tool for 8086 software, is ideal for isolating code "hot spots." PASSIF is a general-purpose assertion checking and reporting tool perfect for running test suites.

Software development for 286/287 components is assisted by two software tools: OMC 286, an 8086 to 80286 object module convertor, and 287 EMULATOR, an 80287 emulator that runs on the 80286.

Additional tools are included that aid 16-bit software development efforts. All tools run on Series III and Series IV Microcomputer Development Systems.

TEXT PROCESSING
SCRIBL
SPELL
MPL
WSORT
COMP

PERFORMANCE MEASUREMENT & TESTING
PERF
GRAPHIT
PASSIF

286/287 DEVELOPMENT
OMC 286
287 EMULATOR

MISCELLANEOUS TOOLS
FUNC
XREF
DC
HSORT

## 8086 SOFTWARE TOOLBOX TOOLS

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are implied

## FUNCTIONAL DESCRIPTION

### Text Processing

**SCRIBL**—text formatting program that does paging, centering, left and right margins, justification, page headers and footers, underlines, boldface type, subscripts and superscripts, upper and lower case, and much more. Formatting commands are embedded in text.

**MPL**—standalone macro processor that processes the macro language used in 8086, 80286, 8089, and 8051 assemblers. Can be used interactively which makes it ideal for debugging macros, MPL can be used to preprocess any text file.

**SPELL**—finds misspelled words in a text file. Dictionary of correctly spelled words is user expandable.

**WSORT**—utility for creating the SPELL dictionary.

**COMP**—performs line-oriented text file comparison (shows source changes). Also understands 8086 object module formats for comparing 8086 object files.

### Performance Measurement and Testing

**PASSIF**—general-purpose assertion checking, testing, and reporting tool. Helps automate the software testing process.

**PERF**—performance analysis tool for 8086 software. Monitors references in the code segment; segment monitored is user defined. Works with small or compact bound loadable modules. Ideal for isolating code "hot spots." Will only run on the Series III.

**GRAFIT**—graphing utility for use with PERF.

### Miscellaneous Tools

**OMC286**—object module convertor that converts 8086 object modules into 80286 object modules.

**287 EMULATOR**—an 80287 emulator that runs on the 80286.

**FUNC**—allows user to redefine the keys on a Series III keyboard and define function keys. Requires the iMDX 511 firmware.

**XREF**—produces cross-reference tables from translator list files. Cross-references all symbols—variables, labels, literals, and quoted strings.

**DC**—floating point desk calculator program; allows variable definitions.

**HSORT**—general heap sort utility.

## SPECIFICATIONS

### Operating Environment

#### Required Hardware

Series III or Series IV Microcomputer Development System

#### Required Software

ISIS-II (W), ISIS-III (N), or iNDX Operating System

### Documentaton

"8086 Software Toolbox."  
(122203)

### Software Support

This product includes a 90-day initial support consisting of new software releases, updates, subscription services (software performance reports and technical reports), and telephone hotline support. Additional software support services are available separately.

## ORDERING INFORMATION

Product Code	Description
iMDX-364	8086 Software Toolbox



## AEDIT TEXT EDITOR

- **AEDIT -80 Operates on any Intellec® Series II, Model 800, or IPDS Development System**
- **AEDIT-86 Operates on any Intellec® Series III or Series IV system**
- **Full Screen Editing Capabilities**
- **Menu-Driven, Easy-to-Use**
- **Designed for the Programmer**
- **Dual File Editing**
- **Easy Handling of Large Blocks of Text**

AEDIT is a programmer-oriented screen editor for use on any Intellec Development System. It is designed to be easy to learn and easy to use. The user is guided by a menu which is used not only to select commands, but also to select options to commands—thus the user is guided at all times.

AEDIT provides full screen editing capabilities. In addition, AEDIT offers features to easily handle (move, copy, delete) large blocks of text. Additional commands are available to find and selectively replace text.

AEDIT allows commands to be prefixed with a count, so commands can be repetitively applied to large portions of the file being edited. To facilitate command entry, the last command and last text strings (for FIND and REPLACE) are retained for convenient re-use.

AEDIT has been designed with the programmer in mind. Two files can be edited during one session. The user can easily switch between files and transfer text between the files. AEDIT has options to automatically indent text to help with the entry of high-level language source. This can considerably shorten the programmer's editing task.

AEDIT can optionally use blanks instead of tabs to indent text. This means that compiler-produced listing files will have the same indentation as the programmer-created source file.

Many other features make AEDIT the editor of the programmer's choice. AEDIT can edit files of any size and optionally creates back-up copies of the file being edited. The user need not bother with complex and incomprehensible command macro's—with AEDIT a macro is created simply by executing it. AEDIT remembers the user's actions for re-use, and stores them on file if requested.

```

This is a demonstration screen of the AEDIT TEXT EDITOR, running
on an INTELLEC SERIES III.

The editor is SIMPLE TO LEARN AND USE, BEING MENU-DRIVEN AND OFFERING
FULL SCREEN EDITING. The menu is always displayed on the bottom
line of the screen.

Many other unique and useful features make AEDIT
a valuable addition to the developer's tool kit:
- easy manipulation of large blocks of text
- editing of two files during the same session
- no limit on file size
- automatic indentation for block-structured source editing

AND MANY MORE!

--- SERIES-III AEDIT V1.0
Again Block Delete Execute Find -find Get --more--
```

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: BXP, CREDIT, i, ICE, ICS, Im, Insite, Int, INTEL, Intelevison, Intellec, iMMX, iOSP, iPDS, iRMX, iSBC, iSBX, Library manager, MCS, MULTIMODULE, Megachassis, Micromainframe, Micromap, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RMX/80, System 2000, UPI, and the combination of ICS, iRMX, iSBC, iSBX, ICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied.  
© INTEL CORPORATION, 1983

# AEDIT TEXT EDITOR

## MANUALS

AEDIT is supplied with a user manual documenting all the aspects of the editor, and a pocket reference card. The manual includes an introductory tutorial.

## HOST SYSTEM

AEDIT-80 is an 8080/8085-based utility and can be run on any Intellec Development System, Series IIE, Series II, Model 800, or iPDS, as well as on ISIS Cluster workstations.

The higher-performance AEDIT-86 is an 8086-based utility that can be run on any Intellec Series IIIE, Series III, or Series IV Development system. Any Series IIE, Series II or Model 800 system can be upgraded to Series III functionality.

AEDIT can be configured to run with non-Intel terminals. Tested configurations are available for the following popular terminals:

ADDS Regent 200, Viewpoint 3A +  
Beehive Mini-Bee  
DEC VT52, VT100  
Hazeltine 1510  
Lear-Seigler ADM-3A  
Zentec ZMS-35

*Regent 200 is a trademark of ADDS  
Mini-Bee is a trademark of Beehive  
DEC designated Digital Equipment Corporation  
ADM-3A is a trademark of Lear-Seigler*

---

## ORDERING INFORMATION

IMDX-335 AEDIT-80 Text Editor.  
Includes 8" single and double density diskettes for Series IIE, Series II, or Model 800, and a 5¼" diskette for iPDS.

IMDX-334 AEDIT-86 Text Editor  
Includes 8" single and double density diskettes for Series III.



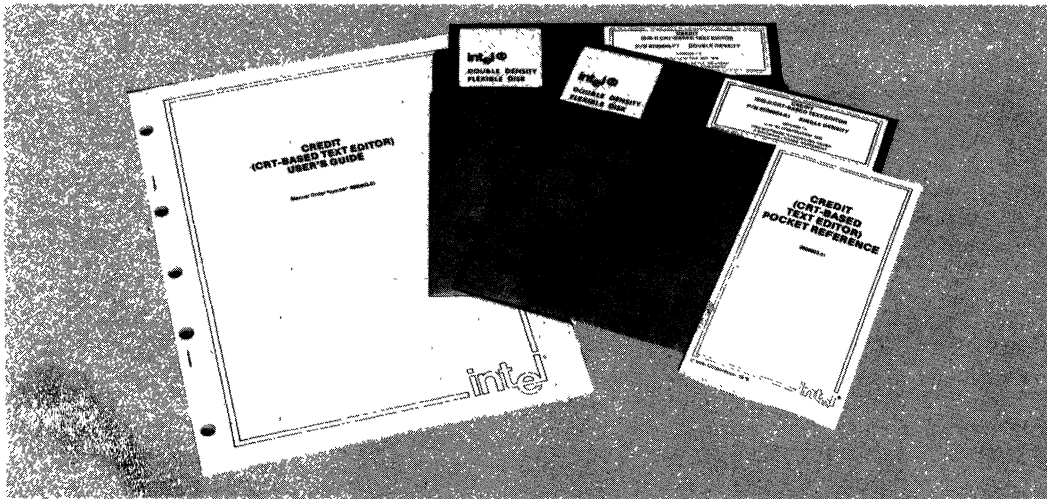
# CREDIT™ CRT-BASED TEXT EDITOR MICROCOMPUTER DEVELOPMENT SYSTEMS

- Provides Interactive Editing of ASCII Text Files
- Displays Full Page of Text
- CRT Screen Display with Cursor-Based Editing Using Single Character Commands for Insertion, Deletion, Page Forward and Backward
- Dynamic Macro Command Definition
- Command Line Editing with String Search, Deletion, Insertion and Move
- Operates Under the ISIS-II Operating System on Intellec® and Intellec® Series II Microcomputer Development Systems

CREDIT is a CRT-based text editor that aids in the creation and editing of ASCII text files on Intellec Microcomputer Development Systems. Once the text has been edited to the programmer's satisfaction, it can be directed to the appropriate language processor for compilation, assembly or interpretation. CREDIT features are easy to use and simplify the change or rearrangement of text files. CREDIT runs under ISIS-II on any Intellec or Intellec Series II Microcomputer Development System with an Intel supplied CRT, disk drive(s) and 64K bytes of memory. Alternatively, it may be configured to run with non-Intel CRTs supporting cursor controls.

There are two editing modes in CREDIT: a screen mode and a command line mode. The screen mode makes full use of the display characteristics of the CRT. The cursor position is visible on the screen and can be positioned by use of the cursor control keys. Display text can be corrected in two ways—either by simply retyping the text, or by using the single-stroke control keys. Specifically, the single-character control keys are used for change, deletion, insertion and paging forward and page backward.

In addition to screen editing, there is command line editing, which includes commands for more powerful and complex editing tasks. Some examples of functions available in the command line mode are search, block move and copy, macro definition and manipulation of external files. These easily used, high-level commands facilitate complex editing and speed microcomputer development.



## CREDIT™ EDITOR FEATURES

- Two editing modes: cursor-driven screen editing and command line context editing

### CRT Editing Includes:

- Displays full page of text
- Single control key commands for insertion, deletion, page forward and backward
- Type-over correction and replacement
- Immediate feedback of the results of each operation
- The current state of the text is always represented on the display

### Command Line Editing Includes:

- String search and substitute
- String delete, change or insert
- Block move
- Block copy
- User-defined macros
- External file handling

- Change CREDIT features with ALTER command
- Conditional iteration
- User-defined tab settings
- Symbolic tag positions
- Automatic disk full warning
- Runs under ISIS-II SUBMIT facility
- Option to exit at any time with original file intact
- HELP command

## BENEFITS OF CREDIT™ EDITOR

- Speeds source program creation and editing—lowers the cost of these functions
- Easy to learn and use—
  - source text is clearly displayed
  - single command keys used for CRT editing
  - HELP command is available for easy reference when needed
- Complements existing software - source text used for PL/M, PASCAL, FORTRAN, BASIC, and Assemblers
- Aids in the management of source file libraries
- Offers full use of Intel supplied CRT cursor functions

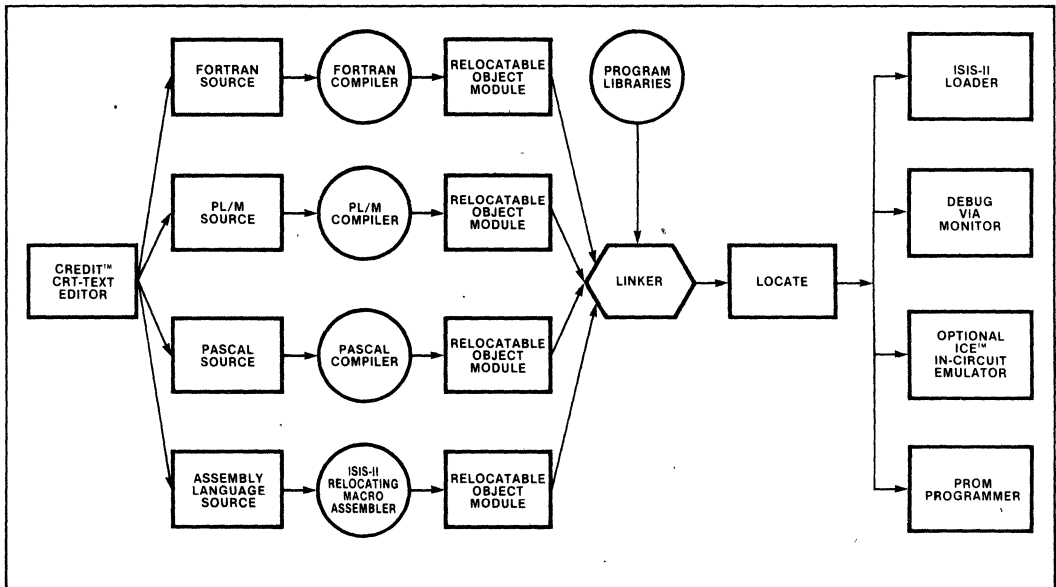
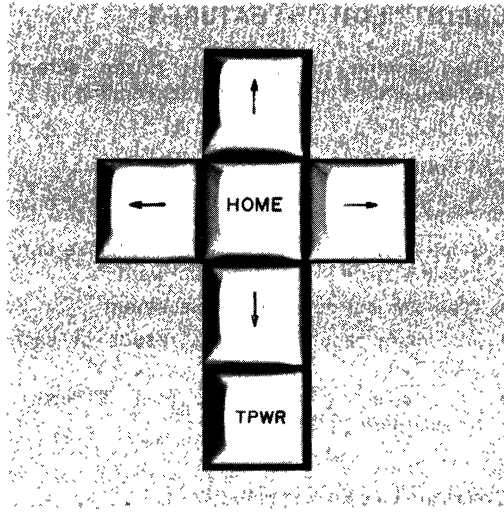


Figure 1. Microcomputer Program Development

**SCREEN MODE COMMANDS**

- MOVE CURSOR:** Use the directional arrow keys on the keyboard.
- REPLACE:** Type over existing text with replacement new text.
- INSERT:** Insert one character.  
Insert more than one character.
- DELETE:** Delete one character.  
Set boundaries and delete all text between them.
- PAGE:** Next Page: Get next screenful of text.  
Previous Page: Get previous screenful.  
View Page: Rewrite current page with possible reframing.

**COMMAND MODE COMMANDS**

- |               |   |                |  |
|---------------|---|----------------|--|
| <b>HELP:</b>  | Display summary of commands.  | <b>TAGS:</b>   | Set tag n, n = 0-9. Tag n is referenced as Tn.   |
| <b>PRINT:</b> | Print n lines or up to tag.   | <b>EXIT:</b>   | Normal exit.<br>Exit Quit: Abandon any changes to edit file.                               |
| <b>JUMP:</b>  | Move cursor position n characters or to tag.<br>Move cursor position n lines forward or backward.   | <b>INSERT:</b> | Insert before CP all text between delimiters.  |
| <b>MOVE:</b>  | Transfer Copy block of text from tag1, for n lines or through tag2, to cursor position.<br>Transfer move: like Transfer Copy but the old copy is deleted. | <b>DELETE:</b> | Delete n characters, or characters up to tag.<br>Delete n lines forward or backward.       |
|               |   | <b>FIND:</b>   | Search for text; move pointer if found.  |
|               |   | <b>SUBST:</b>  | newtext replaces oldtext if oldtext is found. (Optional query to user before replacement.) |

**ADVANCED EDITING COMMANDS**

- |                |  |               |  |
|----------------|--|---------------|--|
| <b>MACROS:</b> | Define a macro.<br>Delete a macro, or all macros if name=*.<br>Expand and execute macro contents, command mode.<br>Expand and execute macro contents, screen mode.<br>Print names and definitions of all macros. | <b>FILES:</b> | Open file "filename" for Reading or Writing.<br><br>Close the current external Read (Write) file.<br><br>Go to beginning of current Read file.<br><br>Read and insert n lines from the Read file.<br><br>Write n lines to the external Write file. |
|----------------|--|---------------|--|



GET:	Get contents of file into command line.		Do command only if Yes (Search) Flag is False.
QUERY:	Query User: set Query Flag accordingly. Do command only if Query Flag is True. Do command only if Query Flag is False.	LOOP:	Exit current iteration loop.
		ALTER:	Configure the command input keys to work with alternative CRTs.
		USER:	Copy text to the console.
YES:	Do command only if Yes (Search) Flag is False.	HELP:	Display summary of commands.

## SPECIFICATIONS

### Operating Environment

#### Required Hardware

- Intellec® Microcomputer Development System
  - Model 800 or Series II with 64k bytes of RAM memory
  - Series III
- Diskette Drive(s)
  - Single or double density
- System Console
  - Intel supplied CRT or alternative CRT supporting cursor controls

#### Optional Hardware

- Line Printer
- Additional diskette drive(s)

### Required Software

- ISIS-II Diskette Operating System
  - Single or double density

### Documentation Package

- CREDIT® (CRT-Based Text Editor) User's Guide (9800902)*
- CREDIT® Pocket Reference (9800903)*

### Shipping Media

- Flexible Diskettes
  - Single or double density

## ORDERING INFORMATION

Part Number	Description
MDS-360*	ISIS-II CREDIT CRT-Based Text Editor

Requires Software License

\***MDS** is an ordering code only, and is not used as a product name or trademark.  
MDS® is a registered trademark of Mohawk Data Sciences Corporation.

**SUPPORT CATEGORY:** Level B



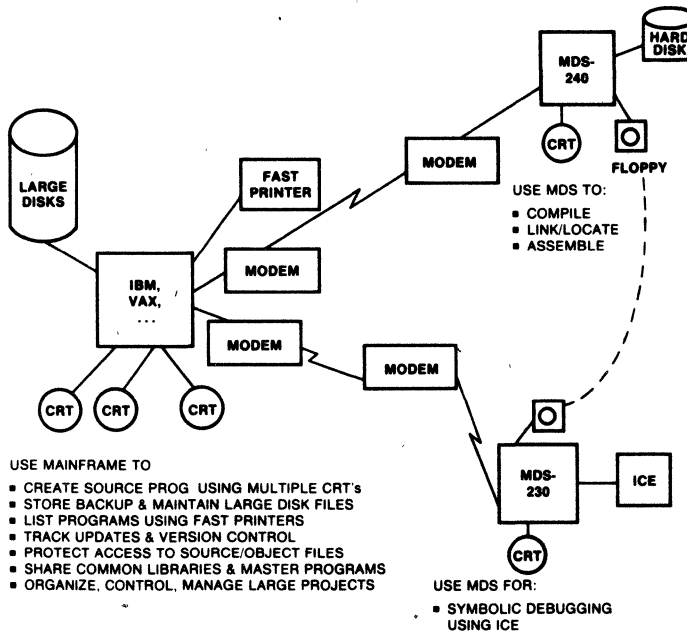


## MAINFRAME LINK FOR DISTRIBUTED DEVELOPMENT

- Integrates user mainframe resources with Inteltec® Development Systems.
- Uses IBM 2780/3780 standard BISYNC protocol supported by a majority of mainframes and minicomputers.
- Protocol supports full error detection with automatic retry.
- Software runs under ISIS-II on any Inteltec® Development System.
- Communicates with remote systems on dedicated or switched (dial-up) telephone lines.
- Package also includes tests and a connector for loop-back self-test capability.

The Mainframe Link consists of software, modem cable to connect the development system to the modem and a loopback connector for diagnostic testing. The software runs under ISIS-II on Inteltec Development Systems. It emulates the operation of an IBM 2780 or 3780 Remote Job Entry (RJE) terminal to (1) transmit ISIS-II files to a remote system or (2) receive files from a remote system using standard BISYNC 2780/3780 protocol. The remote system can be any mainframe or minicomputer which supports the IBM 2780 or 3780 communications interface standard. Files may contain ASCII or binary data so that either program source files (ASCII) or program object files (binary) may be transmitted.

The Mainframe Link allows the user to integrate in-house mainframe resources with Inteltec Microcomputer Development resources. The mainframe can be used for storage, maintenance and management of program source and object files. The program source can be downloaded to a development system for compilation, assembly, linkage, and/or location. The linked modules can be transmitted and saved on the mainframe to be shared by all programmers. The linked program can then be downloaded to a development system for debugging using ICE emulation.



The following are trademarks of Intel Corporation and may be used only to identify Intel products: i, Int, i, INTEL, INTELLEC, MCS, 'm, ICS, ICE, UPI, BXP, iSBC, iSBX, INSITE, iRMX, CREDIT, RMX/80, μScope, Multibus, PROMPT, Promware, Megachassis, Library Manager, MAIN MULTI MODULE, and the combination of MCS, ICE, SBC, RMX or ICS and a numerical suffix; e.g., iSBC-80.

## FEATURES

- Runs under ISIS-II on any Intellec® Microcomputer Development System.
- Communicates with a remote system using IBM 2780/3780 standard BISYNC protocol, which is supported by a majority of minicomputers and mainframes, on dedicated or switched (dial-up) telephone lines.
- The modem cable supplied with the package can be used to connect the Intellec® Development System to the modem (or modem eliminator) using the standard RS232C port.
- Supports user selectable data transmission rates of up to 9600 baud.
- Package includes diagnostic tests used to verify the operation of the Intellec® Development System using the loop-back connector supplied and data transmission up to the modem using the analog loop-back feature.
- System can be configured to match the requirements of the installation, i.e., using modem eliminators for connections up to fifty (50) feet, or by using modems and telephone lines.
- Software can be configured from several configuration options such as:
  - 2780, 3780 or Intel Mode
  - Transparent mode for binary data
  - Non-transparent mode for ASCII data
- Automatic translation from ASCII to EBCDIC and vice versa
- Receive chaining for receiving multiple files
- Intel mode is used mainly for file transfers between two Intellec® Development Systems. The files are duplicated exactly.
- Console commands support all standard features including:
  - SEND data in Transparent or Non-transparent mode, with or without translation to EBCDIC
  - RECEIVE in Transparent or Non-transparent mode, with or without translation to EBCDIC.
  - Support for an IBM RJE console (such as HASP)
- Special utility programs are provided. STRZ strips extra binary zero's from the end of object files. CONSOL assigns system console input to an ISIS-II disk file.
- Can process commands interactively from the console or sequentially from an ISIS-II file under the SUBMIT facility for semi-automatic batch operation.
- Error detection in line transmission and error recovery by automatic retransmission.
- A special command such as DIAGNOSE, allows logging of all data activity on the line, during transmission and reception.
- When not used for communicating with the mainframe, the Intellec® Development System is available as a complete, stand-alone system.

## BENEFITS

- Allows the customer to use an in-house mainframe or minicomputer for program source-preparation, editing, back-up and maintenance using inexpensive CRT's and multi-terminal access. The common files may be shared and others protected.
- Many programmers can use and share the high-performance devices normally available on large computer systems, e.g., fast printers to reduce listing time, the large capacity disks with their fast access time to store large program files.
- The source files can be downloaded using the Mainframe Link to an Intellec Development System (e.g., Model 240 or 245) for compilation, linking and locating.
- The compiled and/or linked object files may be transmitted back to the remote for storage. Updates and version numbers and dates can be tracked to ensure that the latest version is always used and back-up files are available. Binary object files can be later downloaded to an Intellec Development System for debugging using an ICE emulator.
- In short, provides a powerful and flexible tool combining the best of both micro and mainframe worlds, i.e., powerful CPU with large disk capacity, file sharing, multi-terminal access, etc., from a mainframe or minicomputer with Intel's versatile and compatible software support systems (including PL/M, PASCAL, FORTRAN, Assembler, R & L) and sophisticated debugging tools such as ICE emulators.

**SPECIFICATIONS****Operating Environment****Required Hardware:**

Intellec® Microcomputer Development System  
Model 800  
Models 220, 225, 230, 235, 240 or 245

64KB of Memory

One Diskette Drive  
Single or Double Density

System Console  
Intel CRT or non-Intel CRT

**Recommended Hardware for Compilation:**

Hard Disk (Models 240, 245, or Model 740 Upgrade)

Additional Hardware Required for Model 800  
iSBC-955™, iSBC-534™

**Required Software:**

ISIS-II Diskette Operating System  
Single or Double Density

**Documentation Package**

*Mainframe Link User's Guide (121565-001)*

**Shipping Media**

Flexible Diskettes  
Single and Double Density

**ORDERING INFORMATION**

<b>Part Number</b>	<b>Description</b>
*MDS-384 Kit	Mainframe Link for Distributed Development

\*MDS is an ordering code only and is not used as a product name or trademark.  
MDS® is a registered trademark of Mohawk Data Sciences Corporation.

**Remote System Requirements**

- IBM 2780/3780 BISYNC protocol as supported by a majority of mainframes and minicomputers including: all IBM-360/370 Systems, PDP-11/70, VAX-11/780, Data General ECLIPSE.
- Users should purchase this standard software package from the remote system vendor and any additional required hardware such as a synchronous communications interface.
- The operating system at the remote must be configured (SYSGEN'ed) with correct options such as line address, 2780 or 3780, . . .

**Communication Equipment Requirements**

The Intellec Development System may be connected to the remote system using any one of the following methods:

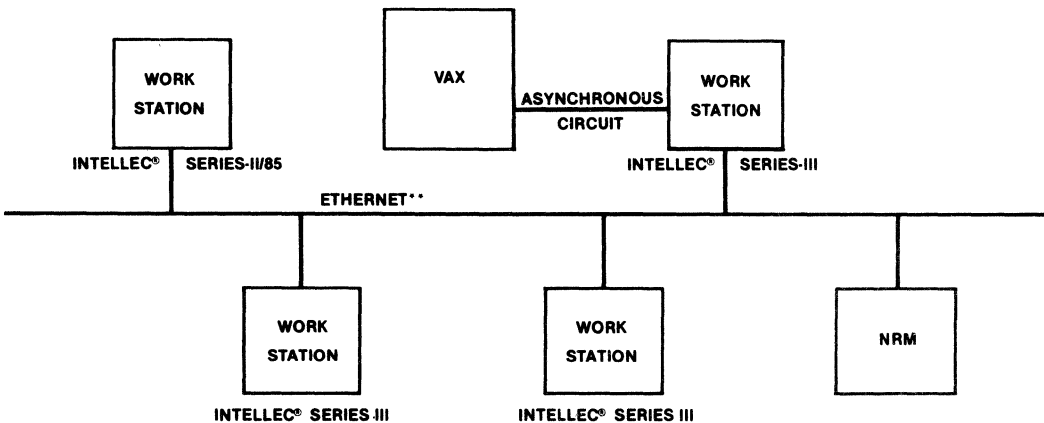
- For short distances (up to 50 feet), use a synchronous modem eliminator (e.g., SPECTRON ME-81FS-2).
- For distances up to four miles, use short haul synchronous modems and telephone lines.
- For distances greater than four miles, use synchronous modems and telephone lines. The following BELL modems or their equivalents are recommended:
  - BELL 201C 2400 bits/second  
(half duplex, switched line)
  - BELL 208A 4800 bits/second  
(full duplex, leased line)
  - BELL 208B 4800 bits/second  
(half duplex, switched line)
  - BELL 209A 9600 bits/second  
(full duplex, leased line)
- Modems at either end must be compatible.



## INTEL ASYNCHRONOUS COMMUNICATIONS LINK

- Communications software for VAX\* host computer and Intel microcomputer development systems
- Compatible with VAX/VMS\* and UNIX† operating systems
- Supports Intel's Model 800, Intellec® Series II, and Series III microcomputer development systems
- Supports NDS-II workstations
- Allows development system console to function as a host terminal!
- Operates through direct cable connection or over telephone lines
- Software selectable transmission rate from 300 to 9600 baud

Intel's Asynchronous Communications Link (ACL) enables one or more Intel microcomputer development systems to communicate with a Digital Equipment Corporation VAX family computer. The link supports Intel Model 800, Intellec Series II, or Intellec Series III development systems and NDS-II workstations. Programmers can use the editing and file management tools of the host computer and then download to the Intel microcomputer development system for debugging and execution. Programmers can use their microcomputer development system as a host terminal and control the host directly without changing terminals.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: Intel, ICE, iRMX, iSBC, iSBX, iSXM, MULTIBUS, MULTICHANNEL, MULTIMODULE and iCS. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

\*VAX and VAX/VMS are trademarks of Digital Equipment Corporation.

† UNIX is a trademark of Bell Laboratories.

\*\* Ethernet is a trademark of Xerox Corp.

© INTEL CORPORATION, 1983

## FUNCTIONAL DESCRIPTION

The Asynchronous Communication Link (ACL) consists of cooperating programs: one that runs on the host computer, and others that run on each microcomputer development system. The development system programs execute under the ISIS-II or ISIS-III(N) operating system and use its file system. They invoke the companion program on the VAX-11/7XX, which runs under either the VAX/VMS or UNIX operating system.

The link provides three modes of communication: on-line transmission, single-line transmission, and file transfer. In on-line mode, the development system functions as a host terminal, enabling the programmer to develop programs using the host computer's editing, compilation, and file-management tools directly from the development system's console. Later, switching to file transfer mode, text files and object code can be downloaded from the host to the development system for debugging and execution. Alternatively, files can be sent back to the host for editing or storage. In single line mode, the programmer can send single-line commands to the host computer while remaining in the ISIS-II or ISIS-III(N) environment.

The user can select transmission rates over the link from 300 to 9600 baud. The link transmits in encapsulated blocks. The receiver program validates the transmission by checking record-number and checksum information in each block's header. In the event of a transmission error, the receiving program recognizes a bad block and requests the sender to retransmit the correct block. The result is highly reliable data communications.

## SOFTWARE PACKAGE

The Asynchronous Communications Link Package contains either a VAX/VMS or UNIX compatible magnetic tape, a single- or double-density diskette compatible with the Intel development system, and the *Asynchronous Communications Link User's Guide* containing installation, configuration, and operation information.

## HARDWARE CONNECTION

The Link sends data over an RS232C cable. The communication line from the host computer connects directly to a development system port.

## TELECOMMUNICATIONS USING THE LINK

The ACL is ideal for cross-host program development using a commercial timesharing service. This configuration requires RS232C compatible modems and a telecommunications line. Depending on the anticipated level of usage, wide-area telephone service (WATS), a leased line, or a data communications network may be chosen to keep operating overhead low.

## NDS-II ACCESS USING THE LINK

The ACL is ideal for interconnecting VAX host computers with NDS-II. This configuration requires that an NDS-II workstation be connected to the VAX host computer using the RS232C interface and to NDS-II using the Ethernet interface.

All three modes of communication operate identically on NDS-II. In the on-line mode, the development workstation operates as a host terminal, and concurrently, as an NDS-II workstation. It is easy to transition between the VAX and ISIS-III(N) operating systems environments as LOGON/LOGOFF sequences are not required to re-enter environments.

In file transfer mode, text and object files can be transferred from the VAX directly to the Winchester Disk at the NRM without first copying the files to the workstation local floppy disk. Similarly, files residing on the NDS-II Network File System (the Winchester Disk at the NRM) can be transferred directly to the VAX without using local workstation storage.

Using the EXPORT/IMPORT mechanisms of NDS-II, a network workstation which is not directly connected to the VAX can cause files to be transferred between the VAX and NRM. For example, any NDS-II workstation can "EXPORT" ACL commands to another "IMPORT"ing NDS-II



workstation which is physically connected to a VAX. The "IMPORT"ing workstation executes the ACL command file causing the desired action to occur.

## VAX ACCESS USING THE LINK

Users who want multiple workstations concur-

rently operating as VAX terminals (the ONLINE mode) must physically connect each workstation to the VAX. However, users who want multiple workstations to be able to upload/download files, for example, must only physically connect one workstation to the VAX. By using the EXPORT/IMPORT mechanism of NDS-II as described above, the user can have multiple workstations accessing the VAX using only one connection.

---

## SPECIFICATIONS

### Software

Asynchronous Communications Link development system programs

VAX/VMS or UNIX companion program

### Media

Single- or double-density ISIS-II compatible diskette

600-ft. 1600 bpi magnetic tape, VAX/VMS or UNIX compatible

### Manual

*Asynchronous Communications Link User's Guide*, Order No. 172174-001

### Required Host Configuration

VAX-11/7XX running VAX/VMS (Version 2.4) or fourth Berkeley distribution of UNIX 32V

### Required Intel Development System Configuration

Model 800, Inteltec Series II, or Inteltec Series III under ISIS-II

### Required Connection

**RS232C compatible** — cable 3M-3349/25 or equivalent; 25-pin connector 3M-3482-1000 or equivalent

### Recommended Modems for Telecommunications

**300 baud** — Bell\* 103 modem; VADIC† 3455 modem or equivalent

**1200 baud** — Bell 202 modem; VADIC 3451 modem or equivalent

**9600 baud** — Bell 209A (full duplex, leased line) or equivalent

**Note:** Since one of the two Model 800 ports uses a current loop interface, Model 800 users need a terminal or modem that is current loop compatible, or a current loop/RS232C converter.

---

## ORDERING INFORMATION

### Product Name

Asynchronous Communications Link

### Ordering Code‡

IMDX 394 for VAX/VMS systems

IMDX 395 for UNIX systems

\*Bell is a trademark of American Telephone and Telegraph.

† VADIC is a trademark of Racal-Vadic Inc.

‡ See price book for proper suffixes for options and media selection.

# INA 960 NETWORK SOFTWARE

- **ISO Transport (8073) Class 4 services**
  - Guaranteed message integrity
  - Data rate matching (flow control)
  - Multiple connection capability
  - Variable length messages
  - Expedited delivery
  - Negotiation of virtual circuit characteristics during opens
- **Additional functionality**
  - Connectionless transport (Datagram)
  - External Data Link
- **IEEE 802.3 Data Link protocol (CSMA/CD) supported**
- **Comprehensive Network Management services**
  - Collection of network usage statistics
  - Setting and inspecting of transport and data link parameters
  - Fault isolation and detection
  - Boot Server
- **Compatible with multiple system environments**
  - Runs as an iRMX™ 86 job
  - Supports host operating system independent designs based on 8086, 8088 or 80186 and 82586 components
- **Runs on iSBC® 186/51 COMMputer™ Board**
- **Size configurable to suit specific application requirements**

iNA 960 is a general purpose local area network software package implementing the class 4 services of the ISO transport proposed specification and network management functions in system designs based on the 8086, 8088 and 80186 microprocessors and the 82586 communications co-processor. iNA 960 also supports Intel's board level LAN products, the iSBC® 550 KIT and the iSBC® 186/51. Combined with the iSBC 186/51 COMmputer™ board, iNA 960 offers a high performance, cost effective network solution for MULTIBUS®/iRMX™ 86 users. See Figure 1 for iNA 960 functionality and operating environments.

iNA 960 is a ready-to-use software building block for OEM suppliers of networked systems for both technical and commercial applications. Examples for such applications include networked design stations, manufacturing process control, communicating word processors, and financial services workstations. Using the iNA 960 software the OEM can minimize development cost and time while achieving compatibility with a growing number of equipment suppliers adapting the IEEE and ISO standards.

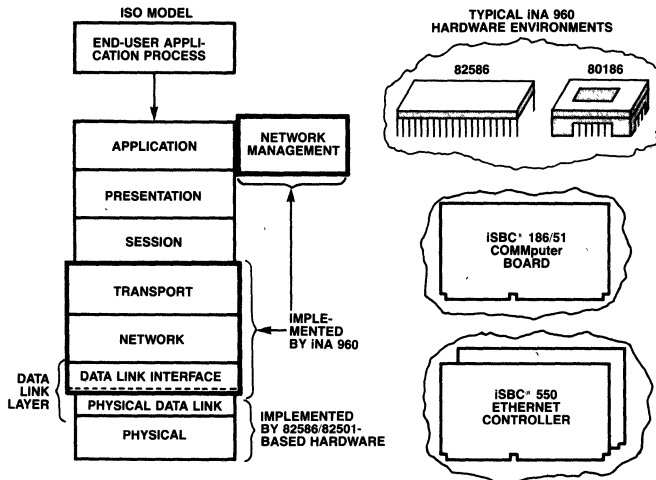


Figure 1.

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained Herein Supersedes Previously Published Specifications On These Devices From Intel.

**FUNCTIONAL OVERVIEW**

The iNA 960 design is a standard implementation of the Class 4 transport protocol defined by the ISO OSI model. The Transport Layer provides a reliable full-duplex message delivery service on top of the "best effort" IEEE 802.3 standard packet delivery service implemented by the 82586 (or equivalent) physical and data link functions.

Consisting of linkable modules, the software can be configured to implement a range of optional capabilities and interface protocols. In addition to reliable process-to-process message delivery, the options include a datagram service, a boot server, a direct user access to the Data Link Layer, and a comprehensive network management facility.

iNA 960 can be configured to run under iRMX 86 along with the user software, or to run on top of a

dedicated 8086, 8088 or 80186 processor coupled with an 82586 to provide a communications front end processor.

The software also includes a Network Management service. This facility enables the user to monitor and adjust the network's operation in order to optimize its performance.

The current release of iNA 960 includes a "null" Network Layer supporting the Data Link and Transport Layers without providing internetwork routing service. This capability will be implemented in later releases of iNA 960.

For a conceptual block diagram of iNA 960, refer to Figure 2.

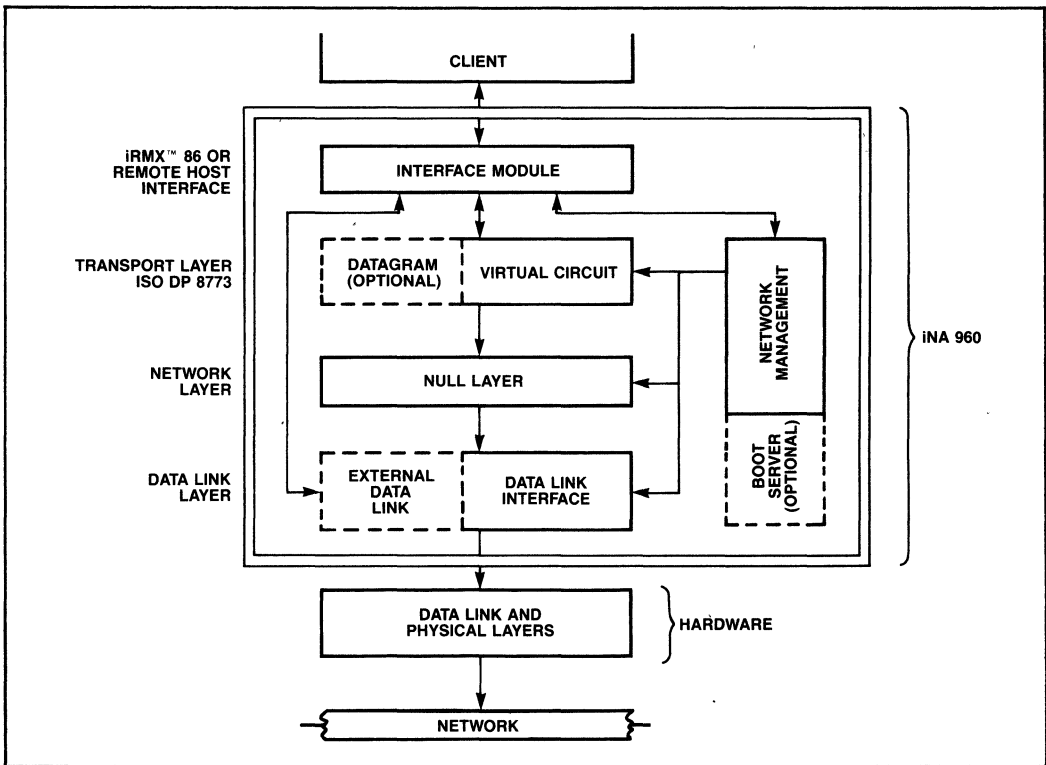


Figure 2. iNA 960 Conceptual Block Diagram



## TRANSPORT LAYER

The Transport Layer provides message delivery services between client processes running on computers (network "hosts" or "nodes") anywhere in the network.

Client processes are identified by a combination of a network address defining the node and a transport service access point defining the interface point through which the client accesses the transport services. The combined parameters, called the transport address, are supplied by the user for both the local and the remote client processes to be connected.

The iNA 960 transport layer implements two kinds of message delivery services: virtual circuit and datagram. The virtual circuit provides a reliable point-to-point message delivery service ensuring maximum data integrity, and it is fully compatible with the ISO 8073 Class 4 protocol. The datagram service provides a best effort message delivery between client processes requiring less overhead and therefore allowing higher throughput than virtual circuits.

### Virtual Circuit Services

- Reliable Delivery: Data is delivered to the destination in the exact order it was sent by the source, with no errors, duplications or losses, regardless of the quality of service available from the underlying network service.
- Data Rate Matching (flow control): The Transport Layer attempts to maximize throughput while conserving communication subsystem resources by controlling the rate at which messages are sent. That rate is based on the availability of receive buffers at the destination and its own resources.
- Multiple Connection Capability (Process Multiplexing): Several processes can be simultaneously using the Transport Layer with no risk that progress or lack of progress by one process will interfere with others.
- Variable Length Messages: The client software can submit arbitrarily short or long messages for transmittal without regard for the minimum or maximum network service data unit (NSDU) lengths supported by the underlying network services.

- Expedited Delivery: With this service the client can transmit up to 16 bytes of urgent data bypassing the normal flow control. The expedited data is guaranteed to arrive before any normal data submitted afterward.

### Connectionless Transport (Datagram) Service

The datagram service option transfers data between client processes without establishing a virtual circuit. The service is a "best effort" capability and data may be lost or misordered. Data can be transferred at one time to a single destination or to several destinations (multicast).

## NETWORK MANAGEMENT (NM)

The network management option provides the users of the network with planning, operation, maintenance and initialization services described below.

- Planning: This service captures network usage statistics on the various layers to help plan network expansion. Statistics are maintained by the layers themselves and are made available to users via an interface with the NM.
- Operation: This service allows the user to monitor network functions and to inspect and adjust network parameters. The goal is to provide the tools for performance optimization on the network.
- Maintenance: This service deals with detecting, isolating and correcting network faults. It also provides the capability to determine the presence of hosts and the viability of their connection to the network.
- Initialization: NM provides initialization and remote loading facilities.

Network management provides distributed management of the network; the user can request any of the services to be performed on a remote as well as a local node. The NM interfaces to every other network layer both to utilize their services and to access their internal data bases.

In support of the above services, the NM capabilities include layer management, echo testing, limited debugging facilities, and the ability to down line load and up line dump a remote system.

Layer management deals with manipulating the internal database of a layer. The elements of these data bases are termed objects. Some examples for objects are the number of collisions, retransmission time-out limit, the number of packets sent, and the list of nodes to boot. NM can examine and modify objects in a layer's data base.

An echo facility is provided. Using this facility the host can determine if a node is present on the network or not, test the communication path to that node and determine whether the remote node is functional.

NM enables the user to read or write memory in any host present on the network. This feature is provided as an aid to debugging.

NM can down line load any system present on the network. A simple Data Link protocol is used to ensure reliability. This facility can be used to load databases, to boot systems without local mass storage or to boot a set of nodes remotely, thus ensuring that they have the same version of software, etc.

Up line dumping is an operation equivalent to memory read from the user's standpoint; however, up line dumping uses the Data Link facilities while memory read uses the transport facilities.

## EXTERNAL DATA LINK (EDL)

The External Data Link option allows the user to access the functionalities of the Data Link Layer directly instead of having to go through the network and transport layers. This flexibility is useful when the user needs custom higher layer software, or does not need the Network Layer and Transport Layer services (e.g., when sending "best effort" messages, or running customer diagnostics).

Through the EDL the capabilities supporting the lower layers in iNA 960 are made directly available to the user. EDL enables the user to establish and delete data link connections, transmit packets to individual and multiple receivers, and configure the data link software to meet the requirements of the given network environment.

## USER ENVIRONMENT

iNA 960 is designed to run on hardware based on the 8086, 8088 or 80186 microprocessors and the 82586 LAN Coprocessor. The software can be configured to run under iRMX 86 or on a dedicated 8086, 8088 or 80186 processor separately from the host. The following section describes these two operating environments.

### iRMX Environment

In this configuration, both the user program and iNA 960 are running under iRMX 86. The communications software is implemented as an iRMX 86 job requiring the nucleus only for most operations. The only exception is the boot server option which also needs the Basic I/O System. iNA 960 will run in any iRMX environment including configurations based on the 80130. See Figure 3 for an illustration of iNA 960 running under iRMX 86.

Some of the typical hardware implementations include the iSBC 550 KIT combined with an 8086, 8088 or 80186 based host or the iSBC 186/51 COMMputer™ board integrating the host processor and the communications controller into a single, high performance MULTIBUS board. See Figure 4A and 4B for a conceptual block diagram of these configurations.

### Operating System/Processor Independent Implementation

In those systems where iRMX 86 is not the primary operating system, where off-loading the host of the communications tasks is necessary for performance reasons, or where an existing communications front-end processor configuration is being upgraded, the user may wish to dedicate a processor for communications purposes. iNA 960 can be configured to support such implementations by providing network services on an 8086, 8088 or 80186 processor. Figure 5 depicts the conceptual block diagram of this configuration.

This approach provides the component and system designer with an ISO standard communications software building block that can be adapted to his system's needs with a minimum interfacing effort. For added flexibility, iNA 960 provides the user with the alternative of using the included interface module or writing his own module, if necessary.

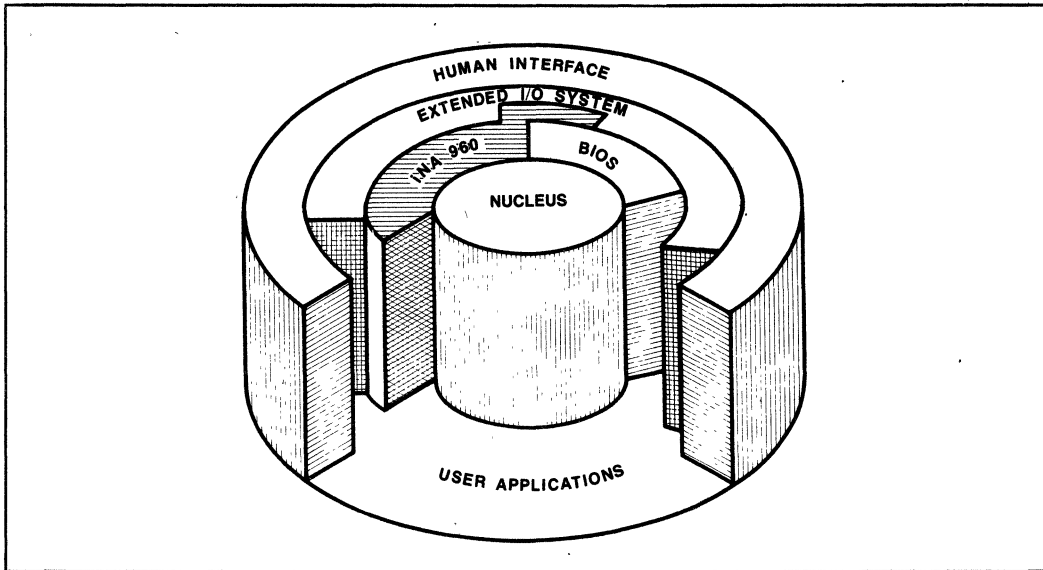


Figure 3. As an iRMX™ job, iNA 960 uses nucleus calls and, when the Boot Server is present, BIOS calls.

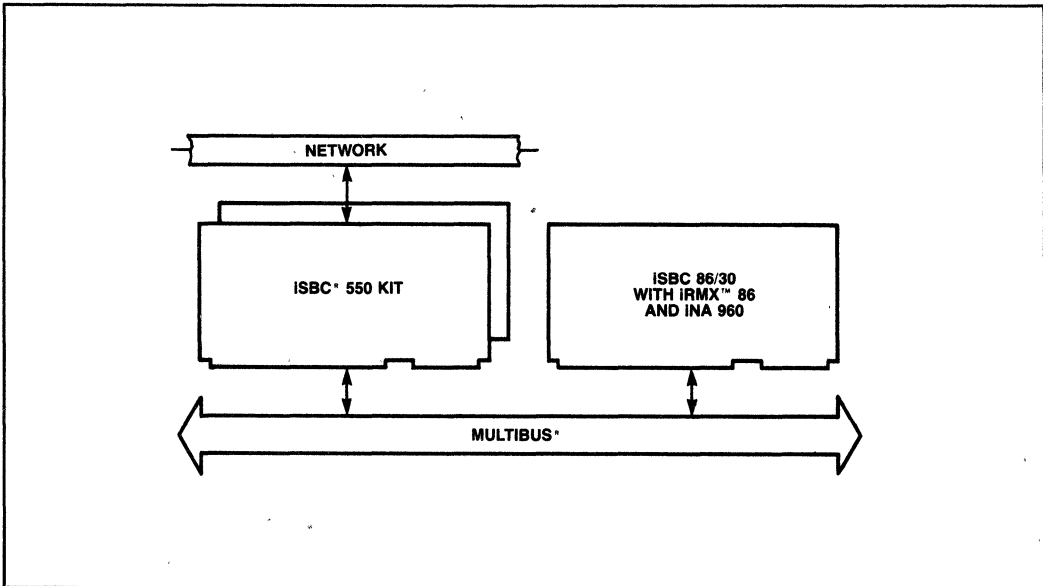


Figure 4A. Typical configuration using ISBC® 550 kit, iSBC® 86/30, iRMX 86™ and iNA 960.

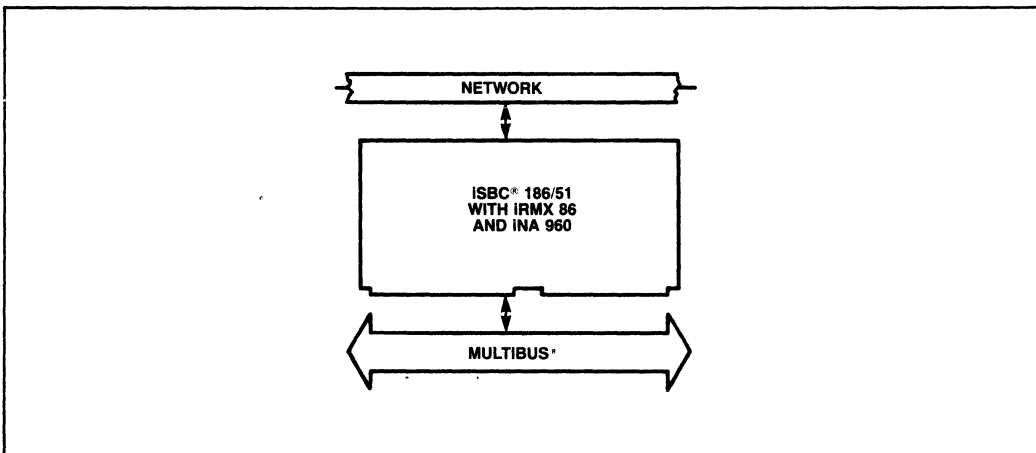


Figure 4B. Configuration using iSBC® 186/51, IRMX 86 and INA 960.

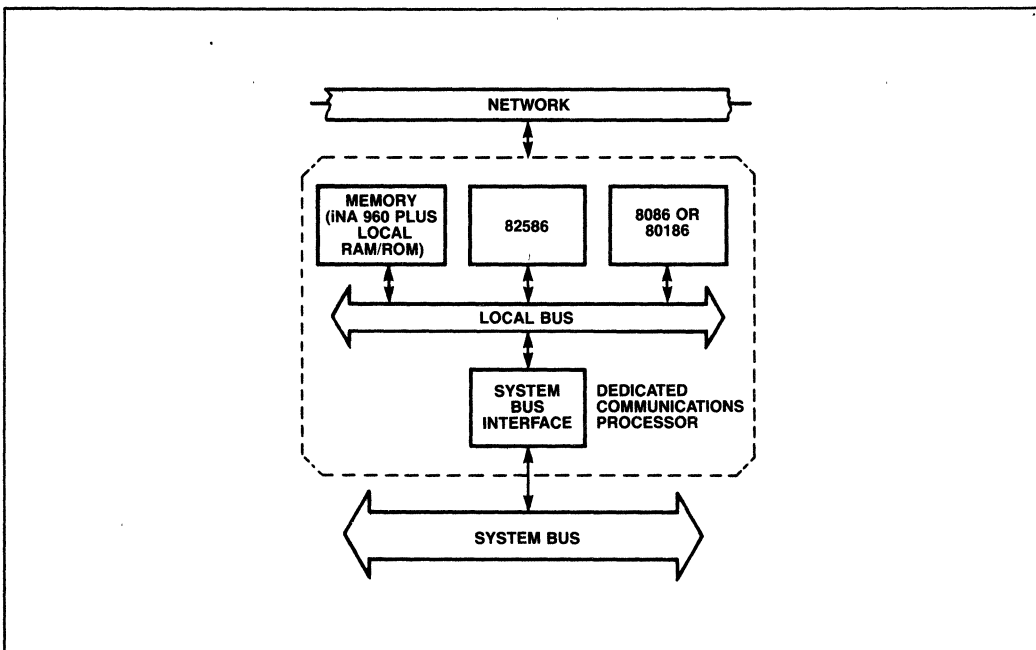


Figure 5. In the operating system/processor independent implementation INA 960 is running on a dedicated 8086, 8088 or 80186 processor.

**USER INTERFACE**

iNA 960 is designed to run both under iRMX 86 and on a dedicated communications front end processor separately from the host. In both environments, the interface is based on exchanging memory segments called request blocks between iNA 960 and the client. The format and contents of the request blocks remain the same in both configurations; only the request block delivery mechanism changes. See Figure 6 for a simplified interface diagram.

Request blocks are memory segments containing the data to be passed from the user to iNA 960 (commands), or from iNA 960 to the user (responses). The iNA 960 request blocks consist of fixed format fields identical across all user commands and argument fields unique to the individual

commands. Refer to Figure 7 for the standard request block format.

Issuing an iNA 960 command consists of filling in the request block fields and transferring the block to iNA 960 for execution. After processing the command, iNA 960 returns the request block with one of the pre-defined response codes placed in the response code field of the request block. The response code indicates whether the command was executed successfully or whether an error occurred. By examining the response code, the user can take appropriate action for that command.

For iRMX users, iNA 960 also provides a procedural interface option to simplify writing the application software interface. In this case, the allocation and formatting of request blocks are replaced by a procedure call with parameters that specify the user's command options. The procedure execution will create a request block and fill in the appropriate fields from the user's parameter list.

For component users the request block delivery mechanism is the means by which the host processor and the communications processor running iNA 960 software exchange the request blocks. iNA 960 provides several such mechanisms as well as permitting user-defined techniques to be utilized.

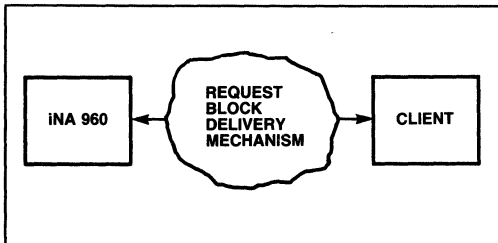


Figure 6.

<u>FIELDS</u>	<u>WORD/BYTE</u>	
<b>Reserved (2)</b>	<b>WORD</b>	} <b>FIXED FORMAT FIELDS</b>  (same for all commands)
<b>Length</b>	<b>BYTE</b>	
<b>User I.D.</b>	<b>WORD</b>	
<b>Response Port</b>	<b>BYTE</b>	
<b>Return Mailbox Token</b>	<b>WORD</b>	
<b>Segment Token</b>	<b>WORD</b>	
<b>Subsystem</b>	<b>BYTE</b>	
<b>Opcode</b>	<b>BYTE</b>	
<b>Response Code</b>	<b>WORD</b>	
<b>Arguments</b>	<b>BYTE</b>	} <b>ARGUMENTS</b>  (changes by command)
.	.	
.	.	

Figure 7. iNA 960 Request Block Format

### Transport Layer User Interface

The following table summarizes the user commands and the corresponding transport layer responses.

Command	Function
1. OPEN	Allocates memory for the connection data base of a virtual circuit (or connection) to be established. The connection database contains data concerning the connection.
2. SEND CONNECT REQUEST	Requests connection to a fully specified remote transport address using specified ISO connection negotiation options.
3. AWAIT CONNECT REQUEST/TRAN	Indicates that the transport client is willing to consider incoming connection requests based on pre-established acceptance criteria.
4. AWAIT CONNECT REQUEST/USER	Indicates that the transport client is willing to consider incoming connection requests. If the request meets the address and negotiation option criteria, it is passed to the client for further consideration.
5. ACCEPT CONNECT REQUEST	Indicates that the connection requested by a remote transport service is accepted by the client.
6. SEND DATA or SEND EOM DATA	With this command, the client requests the transmission of the data in the buffers using the normal delivery service of the specified connection.  The SEND EOM DATA command signals that the end of the data marks the end of the transport service data unit.
7. RECEIVE DATA	Posts normal receive data buffers for a specific connection or for a buffer pool used by a class of connections.
8. WITHDRAW RECEIVE BUFFER	Withdraws normal receive data buffers previously posted by a RECEIVE DATA command.
9. SEND EXPEDITED DATA	Transmits up to 16 bytes of data using the expedited delivery service. The expedited data is guaranteed to arrive at the destination before any normal data submitted afterward.
10. RECEIVE EXPEDITED DATA	Posts receive data buffers for expedited delivery for a specific connection or for a pool of buffers used by a class of connections.
11. WITHDRAW EXPEDITED DATA BUFFER	Withdraws expedited receive data buffers previously posted by a RECEIVE EXPEDITED DATA command.
12. CLOSE	Terminates an existing connection or rejects an incoming connection request. Any normal or expedited data queued up to be sent will not be sent.
13. AWAIT CLOSE	Requests notification of the client of the termination of a specified connection.
14. STATUS and DEFERRED STATUS	Places status information about transport and, optionally, about a specified connection into a supplied buffer.

<b>Command</b>	<b>Function</b>
15. SEND DATAGRAM	Requests transmission of the data in the buffers using the transport datagram service.
16. RECEIVE DATAGRAM	Posts a receive buffer for a specific receiver or a class of receivers to receive data from a transport datagram.
17. WITHDRAW DATAGRAM BUFFER	Withdraws datagram receive buffers previously posted by a RECEIVE DATAGRAM command.

**Network Management Layer User Interface**

<b>Command</b>	<b>Function</b>
1. READ OBJECT	Returns the value of the specified object to the client.
2. SET OBJECT	Sets the value of an object as specified by the client.
3. READ AND CLEAR OBJECT	Returns the value of the specified object to the client then clears the object.
4. ECHO	This function is used to determine the presence of a node, to test the communication path to the node and to ascertain the viability and functionality of the remote host addressed.
5. UP LINE DUMP	Requests a remote node to dump a specified memory area.
6. READ MEMORY	Reads memory of the specified network node.
7. SET MEMORY	Sets memory of the specified network node.
8. FORCE LOAD	Causes a node to attempt a remote load from another node.

**External Data Link Interface**

<b>Command</b>	<b>Function</b>
1. CONNECT	With this command the client establishes a data link connection.
2. DISCONNECT	Eliminates a previously established connection.
3. TRANSMIT	Transmits data contained in buffers specified by the client.
4. POST RECEIVE PACKET DESCRIPTOR	Allocates memory for maintaining records on receive data buffers. Also may be used to allocate memory for buffering receive data.
5. POST RECEIVE BUFFER	Allocates memory for buffering receive data.
6. ADD MULTICAST ADDRESS	Adds an address to the list of data link multicast addresses.
7. REMOVE MULTICAST ADDRESS	Removes an address from the list of data link multicast addresses.
8. SET DATA LINK I.D.	Sets up a unique data link I.D. for the station.

**CONFIGURING INA 960**

In order to adapt iNA 960 to his specific application, the user must configure the software to define the desired functions, to select the appropriate interface, to set the layer parameters and to set up for the required hardware configuration.

There are four optional functionalities the user may elect to implement in his application: the datagram service, the External Data Link interface, network management, and the boot server. These capabilities can be made available simply by linking in the corresponding software modules. The interface options are also implemented in a modular fashion; the user links

in the desired module to set up for the iRMX 86 or the operating system independent configurations.

Layer parameters and configuration options are first edited into layer configuration files, then assembled and linked into iNA 960. Layer parameters adjust the network's operation to match the usage pattern and the available resources. For example, within the Transport Layer, the flow control parameters, the retransmission timer parameters, the transport data base parameters, etc. can be set via this process.

The user also sets up for the required hardware configuration, such as port addresses and interrupt levels, during this process. For the flow diagram of configuring iNA 960, refer to Figure 8.

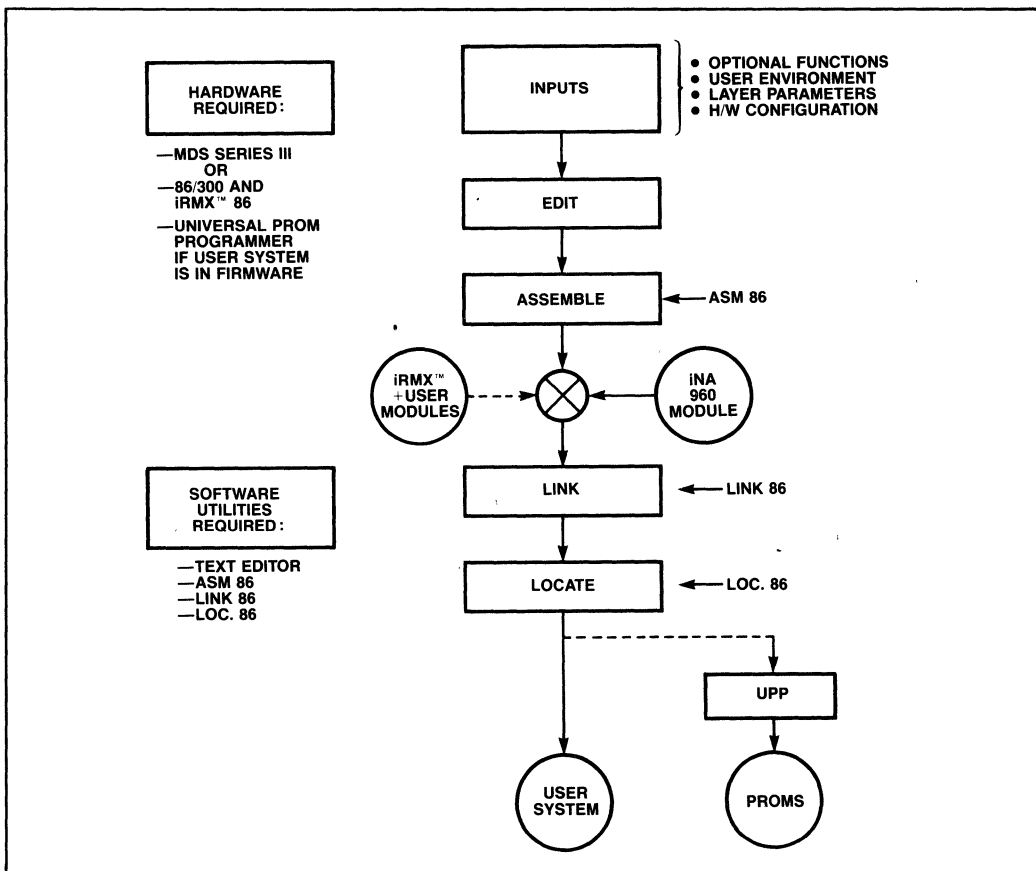


Figure 8. The Configuration Process for iNA 960



**SPECIFICATIONS****Hardware Supported:**

- iSBC 186/51 Communicating Computer.
- iSBC 550 KIT Ethernet controller board(s) configured to run with iSBC 86/30 or iSBC 86/12B Multi-bus processor boards.
- Custom designs based on 8086, 8088 and 80186 microprocessors and the 82586 Local Communications Controller.

**Typical Throughput at transport:****Environments:**

186/51 and iRMX 86	50K to 200K bytes/sec
Dedicated 80186/ 82586 COMMengine	100K to 300K bytes/sec

**Memory Requirements: (in bytes)**

Base Transport System	32K plus configurable Buffer Memory
Net Management Option	1K to 5K
Datagram Option	2K plus Data Base Memory
External Data Link Option	3K
Boot Server Option	5K

**Available Literature/  
Reference Materials:**

- INA 960 Programmer's Reference Manual (11/83)
- iSBC 186/51 Data Sheet (Now)
- iSBC 186/51 Hardware Reference Manual (11/83)

**Ordering Information**

The following is a list of ordering options for the iNA 960 Network Software. All options include a full year of update service that provides a periodic NEWSLETTER, Software Problem Report Service, and copies of system updates that occur during this period. All of the object code options listed are available on either ISIS or RMX compatible double density diskettes.

As with all Intel software, purchase of any of these options requires the execution of a standard Intel Master Software License. The specific rights granted to users depend on the specific option and the License signed.

Order Code	Description
iNA 960 BRO	OEM object code license requiring the payment of incorporation fees for each derivative work based on iNA 960; ISIS formatted diskettes
iNA 960 ERO	Same as above; RMX formatted diskettes
iNA 960 BST	Object code license to use the product at a second site or facility; ISIS formatted diskettes
iNA 960 EST	Same as above; RMX formatted diskettes
iNA 960 BBY	Object code buy-out license requiring no further payment of incorporation fees; ISIS formatted diskettes
iNA 960 EBY	Same as above; RMX formatted diskettes
iNA 960 BSU	Object code single use license only; ISIS formatted diskettes
iNA 960 ESU	Same as above; RMX formatted diskettes
iNA 960 BSR	License for machine readable source code of iNA 960; ISIS formatted diskettes
iNA 960 ESR	Same as above; RMX formatted diskettes
iNA 960 LST	Source listing of iNA 960 provided on microfiche under a special source code license agreement
iNA 960 LWX	One year extension of software support service for source listings
iNA 960 BWX	Same as above for ISIS customers
iNA 960 EWX	Same as above for RMX customers
iNA 960 RF	Order code for the payment of incorporation fees

## NDS-II ELECTRONIC MAIL

- Improves Project Coordination and Communication
- Minimizes "Phone Tag" and Excess Paperwork
- Users Can Send and Receive Text or Object Files
- MAIL Operates Either Interactively or in Command-Tail Format
- User, Group, and "Bulletin Board" Mailboxes Can Be Created
- Operates on any Workstation in the NDS-II Development Environment

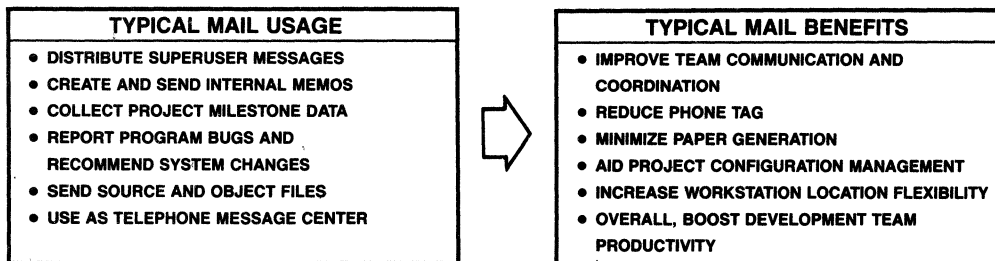
Electronic Mail enables users to send and receive messages and files between any nodes on the NDS-II network. In doing so, Electronic Mail improves the communication and coordination between members, reduces "phone tag" and paper generation, aids project configuration management by enabling simplified file transfers, and increases flexibility in workstation location.

Mail maintains a directory, called the "post office," which contains user, group, and bulletin board mailboxes. Each NDS-II user has a mailbox which is only accessible to that user. Group mailboxes are accessible by a defined group of users, and bulletin board mailboxes are accessible by all users. Both group and bulletin board mailboxes can be easily created by any system user.

Users can send a message to any of the mailbox types listed above. Messages can consist of text generated when Mail is invoked, or a text or object file. Options available when sending mail include using a subject string to categorize a message, specifying a message expiration date and time, delaying message delivery until a specific date and time, marking the message URGENT, and maintaining a log of all messages sent.

Users can interactively read their mail and perform the following operations: print messages on their workstation console, delete messages from a mailbox, save messages in a file, forward messages to other users, and reply to message senders. In addition, users can request a mailbox summary which includes, for each message, the sender's name, date sent, subject, urgency, code type (text or object), and message number.

NDS-II Electronic Mail executes on all existing NDS-II workstations using either the iNDX or ISIS-III(N)/ISIS Cluster operating systems.



## NDS-II ELECTRONIC MAIL

**OPERATING ENVIRONMENT****Required Hardware**

NDS-II Environment with any 8 or 16 bit Microcomputer Development System Workstation

**Required Software**

iNDX or ISIS-III(N)/ISIS Cluster System Software

**DOCUMENTATION**

"NDS-II Electronic Mail User's Guide"  
(122146)

**SOFTWARE SUPPORT**

This product includes a 90-day initial support consisting of new software releases, updates, subscription services (software performance reports and technical reports), and telephone hotline support. Additional software support services are available separately.

**ORDERING INFORMATION****Product Code**

iMDX-337

**Description**

NDS-II Electronic Mail



---

# **Systems and Applications Software**

---

**5**

Handwritten text at the top of the page, possibly a title or header, which is mostly illegible due to blurriness.

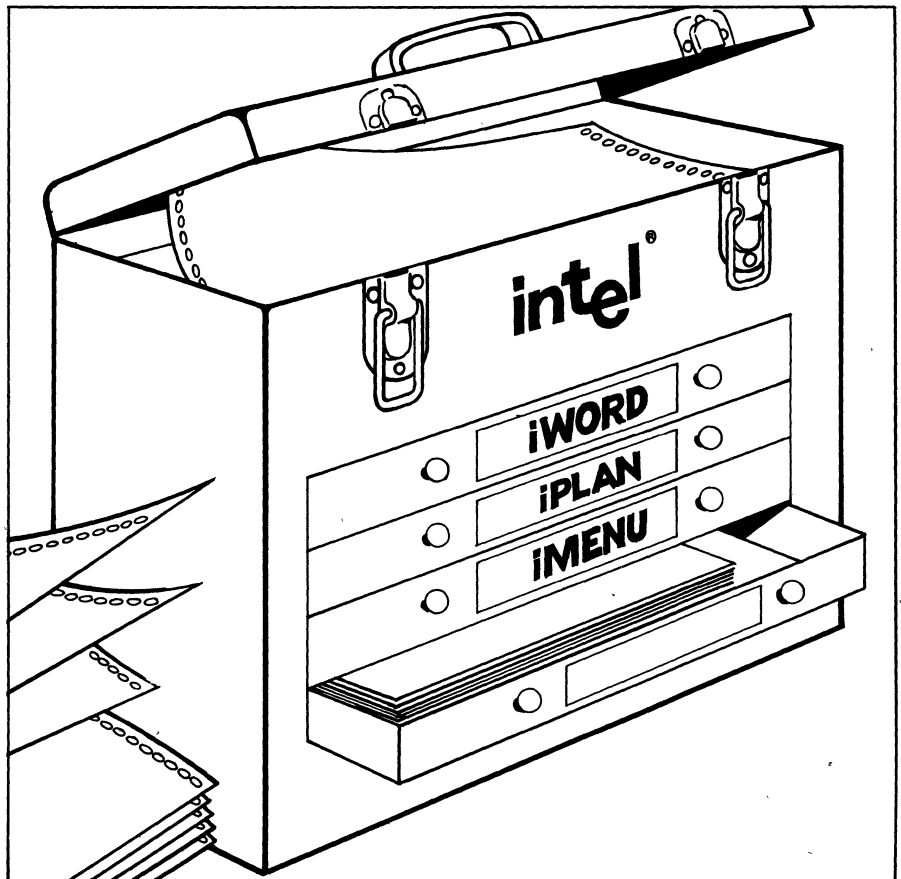
Main body of handwritten text, consisting of several lines of cursive script. The text is extremely faint and difficult to decipher, but appears to be a continuous paragraph or list of items.

Handwritten text at the bottom of the page, possibly a signature or footer, which is also illegible.



**XENIX\***  
**Productivity**  
**Software**  
**Tools**

- iWORD Processing
- iPLAN (Multiplan\*) Spreadsheet
- iMENU Development System





## INTRODUCTION

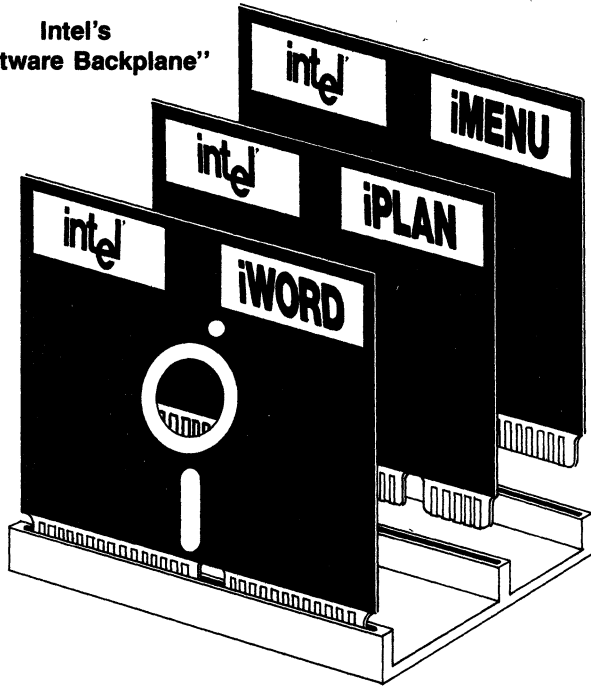
### Software tools for the XENIX environment

Intel's productivity software tools are designed to meet the basic information processing needs of the office environment. Tailored specifically for the XENIX\* operating system, the software tools are available as individual packages which can be applied to specific end-user tasks. Intel's application packages are also offered as a Seamless™ set of software tools, integrated with a hardware/software system such as Intel's Database Information System (iDIS™ 86/735). Seamless software tools support the transparent sharing of data files among various application packages with complete data integrity. With Seamless software, results from one application package are readily accessible and compatible as input for another form of processing.

### iWORD\*

- Standard text editing/formatting commands
- Designed especially for the XENIX operating system
- Easy-to-use for beginners, powerful for experts
- Full-screen text editor
- Access to XENIX typesetter and printer drivers
- Embedded commands for global formatting
- On-screen display of formatted text
- On-line Help facility, spelling/dictionary module, and mail/merge facility
- Worldwide service and support

### Intel's "Software Backplane"



### The powerful, versatile word processing tool

Intel's iWORD package is a sophisticated, yet friendly word processing tool for preparing business documents, such as reports, letters, memoranda, technical papers, and more. Written in the "C" language and tailored to the XENIX operating system, the iWORD package can run in both multi-user and single-user environments. Menu-driven and screen-oriented, the iWORD package supports all standard text editing, storage, and formatting development functions.

### An efficient, easy-to-use text processor

Inexperienced users will find the iWORD software concepts intuitively easy. For example, the user accesses a document file by opening a "drawer," and editing commands follow familiar "cut and paste" procedures.

All commands are in plain English. No memorization is necessary, and many operations are executed by a

single keystroke. Concise command menus and an on-line Help facility are continuously available so that novice users can quickly advance in their word processing abilities. The iWORD system is sufficiently powerful to meet the needs of more experienced users as well.

### Designed around office needs

Intel's iWORD software is based on the simple concept of an office file cabinet, defined by a collection of drawers, each of which contains files (documents).

The user may:

- Open an existing drawer or make a new drawer
- Create a new file or select an existing file
- Rename a drawer or file
- Add to, change, copy, move or delete the selected file.

There is no limit to the number of user-created drawers other than the availability of disk storage space.

\*iWORD is a version of Horizon Word Processing, a trademark of Horizon Software Systems, Inc.

### On-screen display of formatted text

The word processor allows users to visually format documents and print them as they are displayed on the terminal, or to format them with the powerful text processing facilities inherent to the XENIX operating system. This on-screen display capability is particularly helpful in preparing documents for typesetting.

The results of text formatting commands appear immediately on the screen. Examples of these commands include:

- Right justification
- Underlining
- Indentation
- Centering
- Alignment.

### Advanced word processing features

Inexperienced users may execute commands from a simple menu (and related Help screens), while more proficient users may opt to use up to 64 function keys without accessing the menu.

The iWORD system allows simultaneous support for multiple character and/or line printers at the local or system level; printer selection is an operator option at print time.

The iWORD package includes a spelling checker and correction facility with an extensive on-line dictionary.

A Mail/Merge facility is available to combine mailing lists and document files (e.g., form letters) for printer output. Mail/Merge also provides the capability of incorporating paragraphs from a third file.

The iWORD processing allows on-screen sorting of numeric or alphabetic text.

### Special editing commands

- Find commands ("string search") to locate characters or words in text for possible changes or additions
- Deletion commands for removing words, sentences, lines, paragraphs and entire files
- Fill commands to fit as many words as possible in a finite space
- Form command to type over existing text
- Command to mark location of the cursor within text
- Paste-in command to copy a section of text
- Replace command that replaces one text area with another
- Tab setting commands

### Embedded "dot" commands

When formatting or printing needs are complex, the user has easy access to more powerful embedded "dot" commands. "Dot" commands are most useful for medium-sized and long documents requiring sophisticated formatting functions like subscripts, superscripts, and footnotes. "Dot" commands are fully compatible with NROFF and TROFF, the XENIX-supplied printing and typesetting utilities. The results of "dot" commands are displayed on-screen before the document is printed.

Embedded commands for global formatting include:

- Page layout
- Justification
- Automatic hyphenation
- Running headers and footers
- Footnotes
- Superscripts and subscripts
- Automatic page numbering
- XENIX typesetting commands (TROFF)
- XENIX printing commands (NROFF).

### Editing two files simultaneously

The iWORD package provides a moveable "window" into a file for full-screen text editing. The user may simultaneously display two areas of the same document or two different documents in two screen "windows." Employing this "split screen" capability, the user may review two different files at the same time, as well as move text between files.

### Designed for experienced and novice users

The iWORD software provides the experienced word processing user the full strength of XENIX text preparation commands, such as NROFF and TROFF. The iWORD package also offers a comprehensive menu shell which makes the word processing software easy to use for even the most inexperienced computer user. In conclusion, the iWORD system is a powerful, "user friendly," and flexible word processing package intended for all levels of computer proficiency.

**iPLAN\***

- Industry standard advanced electronic spreadsheet functions
- Sophisticated formatting options
- Easy-to-use English commands
- Extensive, on-line Help facilities
- Scrolling features and multi-window/multi-table display
- Links and updates multiple inter-related spreadsheets
- Automatically updates calculations
- Worldwide service and support

**The most advanced electronic spreadsheet**

The iPLAN Multiplan Spreadsheet software is one of the most powerful, easy-to-use "electronic worksheet" programs available. Developed by Microsoft Corporation, Multiplan has been enhanced for Intel hardware environments operating under XENIX.

The iPLAN package is a multi-purpose tool capable of a wide variety of business and scientific applications: financial modeling, planning, forecasting, tabulations, calculation of engineering formulas, and much more. It supports "what-if" decision-modeling with a versatile two-dimensional matrix that can be custom-tailored for specific use.

Unlike other spreadsheets, the iPLAN system is designed to meet the needs of both inexperienced and sophisticated computer users. It also offers versatile presentation and reporting capabilities.

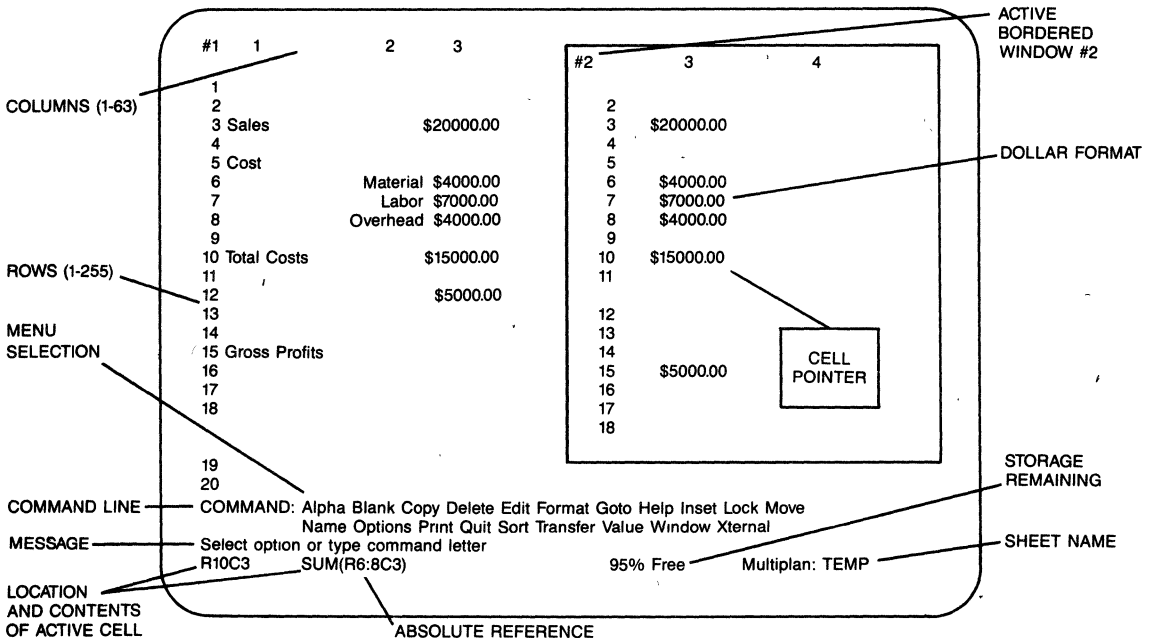
**The iPLAN matrix format**

The iPLAN software displays numerical data, text, or formulas in matrix (row/column) format. The spreadsheet screen is divided into 'cells' which are referenced by row and column numbers. Cells may contain numeric data, formulas, text, or labels. Commands are listed at the bottom of the screen along with the current addressed cell, the amount of unused spreadsheet storage space, and the name of the file in use.

**Designed for ease-of-use**

Beginning iPLAN users can start building worksheets after a couple hours of initial use. While simple to operate, the iPLAN system functionality is enhanced by the skill of the user.

\* iPLAN is a version of Microsoft Multiplan, a trademark of Microsoft Corporation.



**Typical Multiplan Screen Display**

The iPLAN package does not use cryptic, abbreviated commands or reference codes (e.g. "AZ23"). Instead, it uses plain English commands (e.g. COPY) and reference names (e.g. COSTS or SALES). Completely menu-driven, the iPLAN software prompts the user with simple commands that can be executed with a single keystroke. To help in command selection, the user can access a reference guide.

Notable iPLAN features include:

- Ability to build formulas by highlighting cells
- Menu-driven functions and command prompting
- Plain English command words and formulas
- Comprehensive on-line reference guide
- Eight-window display option
- Full-screen display of worksheet formulas.

### **A dynamic, versatile workspace**

The iPLAN package offers an effective workspace that is 63 columns wide by 255 rows long. Worksheets are easily designed to fit project requirements. Moreover, worksheets can be linked to automatically receive or transmit data into other related iPLAN worksheets. Column width can be varied to accept long (or short) words and numerals; lines of text can be typed across several columns.

Up to eight windows are available with vertical and horizontal scrolling, such that different areas of a very large worksheet can be viewed simultaneously. The windows can be aligned, scrolled together, opened, or closed at the user's choice.

### **Built-in data security**

The iPLAN software features cell locking to protect worksheet data. When data and formulas have been entered, the specified information can be "locked" in place so that vital data cannot be accidentally erased or altered.

### **Flexible presentation features**

The iPLAN system enables users to produce printed reports of professional caliber. The program includes special formatting, alignment, and printing functions that support the printing of presentation-quality reports. The iPLAN software can automatically break a spreadsheet into multiple pages, and the user can specify the appropriate margins.

### **Powerful modeling capabilities**

Highlights of iPLAN modeling features include:

- Alphabetical or numerical sorting capabilities
- Links and automatically updates up to eight interrelated worksheets
- Automatically updates subtotals, totals, percentages, growth curves and other calculations
- Performs multiple iterations to solve closed-loop problems
- Automatically revises formulas when reordering rows and columns in displays
- Cells and areas can be named for clarity
- Continuous formatting allows entries across cell boundaries
- Formulas moved to various worksheet locations without retyping
- Includes special editing area for quick additions or deletions
- Sheet display may be redesigned or formatted in various ways without altering the stored data
- Formulas, words, or numerals can be entered into any location so that printed sheets have titles and descriptions
- Offers a rich repertoire of advanced math functions and operators.

### **iMENU\***

- **Hierarchical control of menu screens to organize application program use**
- **On-line application development/maintenance system**
- **A menu screen design and menu development system for non-programmers**
- **On-line Help facility**
- **Written in the "C" language, specifically for XENIX**
- **Supports turnkey application development**
- **Worldwide service and support**

### **Simplifies use, development and maintenance of XENIX-based applications**

The iMENU software package is a hierarchical user interface and application development tool that ties together XENIX-based applications to achieve a high level of software integration. The iMENU package allows applications developers to create integrated, logical, and friendly interfaces to XENIX applications.

In effect, the iMENU system allows the XENIX operating system to appear transparent to the non-technical user, while it offers all the power and functionality inherent to XENIX to the more experienced user. The iMENU package interfaces with virtually all character-oriented terminals.

### **Aids application development and software packaging**

Programmers and experienced users can apply the iMENU system in maintaining or creating menus, forms, or Help screens for existing or new applications.

♦

\* iMENU is a version of Schmidt's /menus, a trademark of Schmidt Associates.

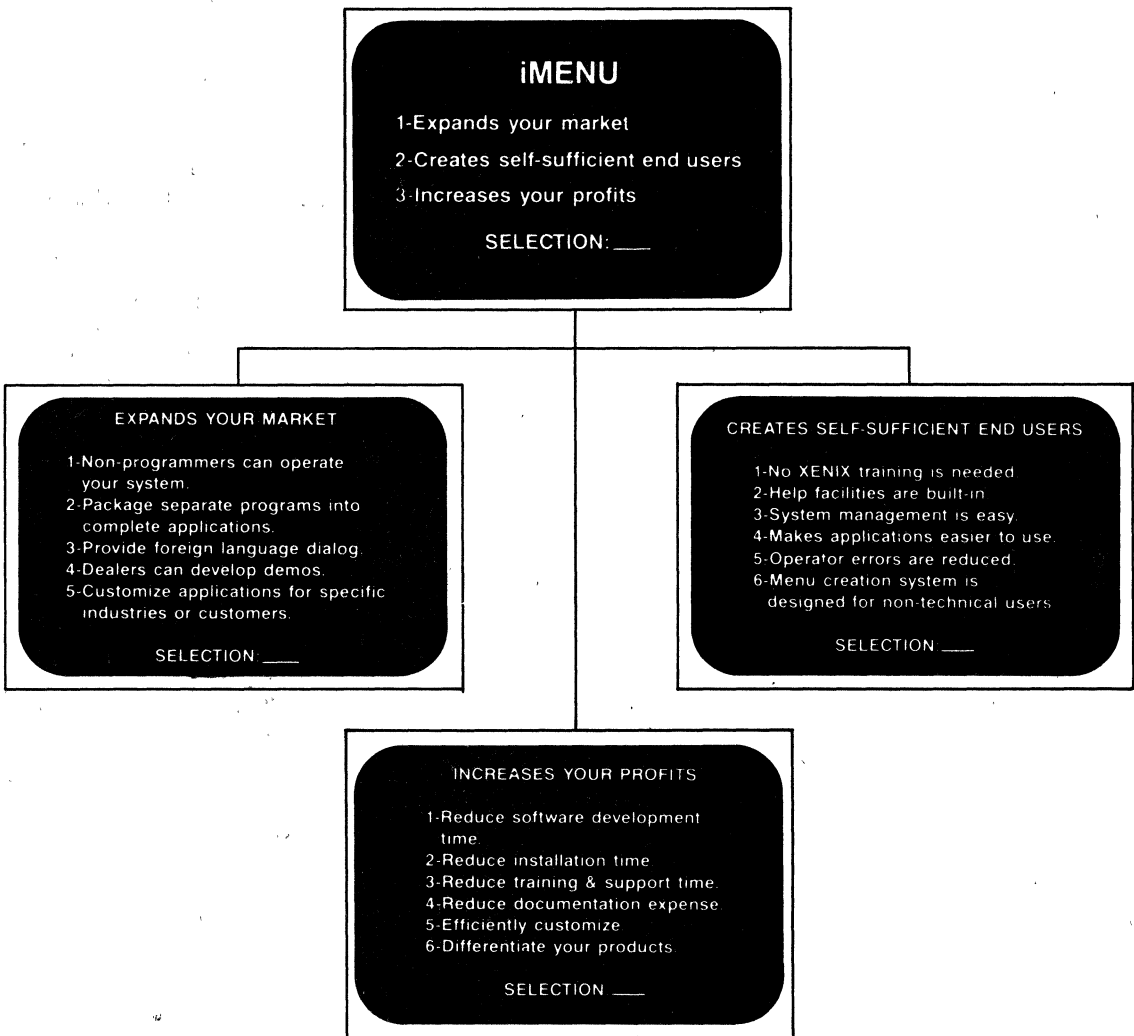
Full XENIX functionality is retained and simplified with the iMENU software. Experienced users have the option of skipping step-by-step menu selection via the fast menu selection mode. Advanced users can also modify the menu system to reflect changes in existing applications or to incorporate new applications.

Standard iMENU package functions include:

- Definition of login IDs
- Add, delete, and list login IDs
- Authorization control
- Define, update, delete and list menu items and attributes
- Screen and forms building
- Interaction with XENIX shell commands
- Powerful macros
- Fast menu selection mode for experienced users
- On-line Help facility.

The iMENU software includes a Menu Development Subsystem, which is a menu-driven set of maintenance functions allowing:

- Menu screen maintenance
- Form screen maintenance
- Help screen maintenance
- Menu selection maintenance
- Macro maintenance
- Shellsript maintenance
- Login ID maintenance
- Deauthorization maintenance
- Backup/restore.



### Comprehensive on-line Help system

The Help system is an interactive user-assistance facility. The Help system is completely integrated with the iMENU package so the user need not rely on bulky reference manuals to operate a particular application. In the event the user encounters problems or has questions, explanatory solutions can be made available at all times. Using the iMENU system itself, programmers and experienced users can extend or modify the Help system to include new applications.

### Expands markets and increases profits

Systems integrators will find the iMENU system to be an indispensable tool for packaging XENIX-based applications software and integrated hardware/software systems.

Using the iMENU package, application developers can:

- Extend the user-interface to wrap around new and existing applications software
- Customize existing applications to meet varying customer needs
- Develop application demos
- Create applications that are easily used by non-programmers
- Package separate programs into integrated applications.

The iMENU software enhances the cost-effectiveness of application development by:

- Improving time to market for new software products
- Reducing software development time
- Reducing the need for training and support
- Decreasing software installation time
- Cutting documentation expenses
- Unifying a family of software products for consistent screen appearance and operation.

### Integrated software for the iDIS system

Intel's Database Information System (iDIS 86/735) provides end users and systems builders with a vehicle for incorporating a Seamless set of software productivity tools. Seamless software supports the transparent sharing of data files among application packages with

**SOFTWARE PRODUCTIVITY TOOLS**

- Technology tailored to Intel's XENIX® system
- Experienced, world-wide Intel service and support

**iWORD**

- All-purpose office word processing

**iPLAN**

- Comprehensive "what-if" decision modeling

**iMENU**

- Menu-driven system management, application development, application operation

complete data integrity. The iDIS system is a multi-user, multi-tasking XENIX-based microcomputer available with the iWORD processor, the iPLAN spreadsheet, the iMENU development system, iXTRACT communication facilities for downloading mainframe databases, the iDB DBMS for local relational database management, and software that supports networking of personal computers. Application development tools and high-level programming languages are also offered with the iDIS system. Data can be transferred among the Seamless software packages, and the iDIS menu system and iHELP facility provide a friendly, common user interface. The iDIS system is an example of how Intel provides hardware/software components at all levels of integration to meet individual system needs.

### Worldwide service and support

All Intel software included under an active software maintenance agreement is fully supported by Intel's staff of trained software engineers. Depending on the system configuration, several levels of support are available. Each package is offered with complete documentation, including a comprehensive user manual and installation guide.

### SPECIFICATIONS

#### Required Hardware:

- Any 8086 or 80286-based system including Intel's SYSTEM 86/300, 286/300 family and iDIS systems
- Minimum of 128 KB memory
- At least two floppy disks or one hard disk
- One 8 in. or 5.25 in. double-density floppy disk drive for distribution media

#### Required Software:

- Intel's XENIX 86/286 Operating System

#### Warranty:

90 days for:  
Software Updates and application support. Continuing support services available with subscription to a Software maintenance agreement.



The following are trademarks of Intel Corporation and may be used only to describe Intel products: BXP, CREDIT, i, ICE, iICE, ICS, iDBP, iDIS, iLBX, i<sub>m</sub>, iMMX, Insite, INTEL, Intelevison, Inteltec, Intelligent Identifier™, InteIBOS, intelligent Programming™, Intellink, iOSP, iPDS, iRMS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, Micromainframe, MULTIBUS, Multichannel™ Plug-A-Bubble, Seamless, MULTIMODULE, PROMPT, Ripplemode, RMX/80, RUPI, SYSTEM 2000, Data Pipeline, iDIS, iDBP, and UPI, and the combination of ICE, ICS, iRMX, iSBX, MCS, or UPI and a numerical suffix. Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other patent licenses are implied. Specifications are subject to change without notice.

iWORD is a version of Horizon Word Processing, a trademark of Horizon Software Systems, Inc. iPLAN is a version of Microsoft's multiplan, a trademark of Microsoft Corporation. iMENU is a version of Schmidt's /menus, a trademark of Schmidt Associates.

Information contained herein supercedes previously published specifications on these devices from Intel.

# **iTPS Transaction Processing Systems Terminal Application Processing System (iTAPS)**

- Complete on-line transaction processing monitor
- Easy-to-use, interactive screen building utilities
- Flexible, on-line transaction development facilities
- Reduced user coding with standardized functions and processing modules
- Variable indexed sequential database structure
- Powerful interactive query language with Boolean logic capability
- Multilevel user-defined security
- Real-time report writer
- Menu-prompting



## FUNCTIONAL DESCRIPTION

Intel's Terminal Application Processing System (iTAPS) is the on-line transaction processing software package customized for Intel's Transaction Processing System (iTPS). iTAPS consists of an interactive application development facility and a reliable high-performance run-time transaction processing monitor. In addition to providing ready-to-use software for building data files and maintaining data relationships and structures, iTAPS interfaces with user-specific processing modules written in COBOL or Pascal, iTAPS also supports a powerful query language that provides direct, easy and fast access to data contained in user database files.

## FEATURES AND BENEFITS

### Application development

As a complete transaction processing application development tool, iTAPS frees programmers from the coding complexities normally associated with development environments that include database management, telecommunications and multiprogramming-multiuser on-line systems. iTAPS enables programmers to write complex transaction processing applications with simple commands and easy-to-use interactive utilities. iTAPS is primarily intended for application programmers and requires minimal systems programmer involvement.

iTAPS furnishes an on-line facility to define screen formats, including the variables to be processed and their video characteristics. iTAPS simply requests the programmer to "paint" the screen as it should appear to the system user, and fields are then defined interactively by filling in a form on the screen.

iTAPS also provides a number of standard modules used repetitively in any transaction processing application. These modules greatly simplify the development and maintenance tasks by automating such functions as:

- Initialization
- Addition of a record to the database
- Modification of a record in the database
- Deletion of a record from the database

- Index structure update
- Data validation
- Uniqueness checking
- Dynamic specification of display format
- File writing
- Database searching
- Sorting

In addition to the standard iTAPS modules, the programmer may write user-specific COBOL or Pascal programs to address specific application processing requirements. These programs combined with the iTAPS standard modules make up an application transaction. The COBOL or Pascal program calls upon iTAPS to provide all database and terminal I/O functions. The program can also access non-iTAPS files using standard I/O programming.

All these features add up to increased system and programmer productivity. Programmers can spend more time designing and implementing their applications and less time worrying about multiuser on-line systems requirements. As a result, applications come on-line faster. Furthermore, the consistent organization of iTAPS application systems minimizes maintenance and enhancement problems.

iTAPS produces extensive application documentation. Even if the program's author has departed, very little time will be lost reconstructing or enhancing the application. This means that iTAPS reduces the cost of application software both in the development phase and in the maintenance phase.

### Run-time transaction processing

At run-time, iTAPS provides a complete multiuser on-line transaction processing environment with five levels of predefined security checks: sign-on, application system selection, data add/delete, field read/write, and field index. Two menu levels to select the application system and the desired transaction are provided.

When iTAPS receives a request from a terminal to initiate execution of a particular transaction, it allocates the resources between that terminal and

the specified transaction, and manages all the disk input/output. Because Intel's Transaction Processing System has been optimized for transaction processing, the system overhead is kept at a minimum. Superior performance results as measured in terms of both throughput and response time.

iTAPS provides first level data validation. Errors are detected as they occur and before the more complex application processing takes place. Thus iTAPS significantly assists not only in validation and immediate correction of the input data, but also in realizing a more efficiently performing system by reducing unnecessary processing.

iTAPS supports a powerful query facility based on predefined commands and on the use of English words to describe each data element. Compound Boolean inquiries can be formulated using aliases, data values and logical operators. The response to an inquiry can be displayed at the terminal or printed on the system printer.

### System Organization

iTAPS is composed of five modules:

- The Executive
- The Communication Manager
- The Application Manager
- The Data Manager
- The iTAPS terminal

As shown, these modules interact with the operating system, the user's software, the terminal, and the database. This segmentation enables programmers to develop each module as an independent entity and to modify it without having to alter the entire application. Each module is designed to simplify application development, implementation and execution.

### The iTAPS Executive

The Executive provides the interface with the iRMX 86 operating system and acts as the run-time transaction processing monitor. The Executive controls sign-on and menu displays, performs security-checking, and maintains a log which posts the appropriate "before" and/or "after" image of any data file change. Based on this log, the Executive provides defined recovery capabilities:

- At the network level: Following a terminal or communication line failure, the last transaction is retrieved from storage and retransmitted to the terminal upon entry of a command by the operator.
- At the file level: Following a system failure the recovery log is read to update the file and back-out the incomplete transactions. Full recovery is also provided by the recovery log in conjunction with the last back-up.

In addition, the Executive provides statistical services to evaluate system use and performance, auditing facilities

to isolate illegal transmissions, and documentation facilities to record the application and data structures.

### The iTAPS Communication Manager

The Communication Manager controls the communications network. It collects data that has been transmitted from a terminal, sends messages back to the terminals, and maintains the network buffers. Supporting both block and TAPS modes, the Communication Manager is designed to optimize line transmission.

The Communication Manager is completely transparent to the application programmer as well as to the user.

### The iTAPS Application Manager

The Application Manager controls the application processing sequence of each terminal within the network. Upon receipt of a request from the Communication Manager, the Application Manager determines the transaction type, schedules the

appropriate processing modules, transfers data, and generally manages the logic of the application.

### The iTAPS Data Manager

The Data Manager interfaces iTAPS with the database. It performs the additions, changes, deletions and extractions requested by the Application Manager. It can be executed from a user module during the processing of a transaction or when the query language is used.

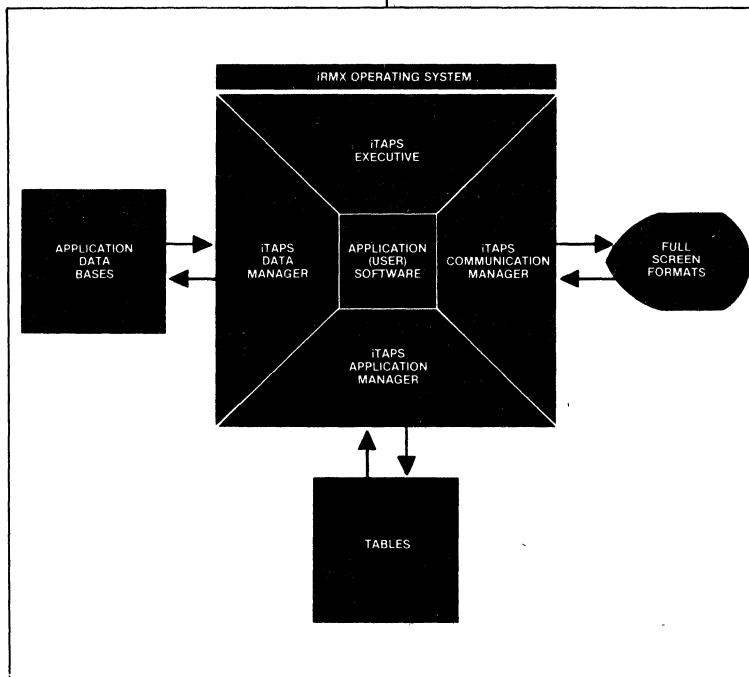
The Data Manager uses a valued, inverted structure called Variable Indexed Sequential Access Method (VISAM) to format the database. Each VISAM database is composed of a keyed access index file and a direct access primary file. The primary file contains the data records, and the index file contains all the pointers to the primary file records. Virtually all fields in the database can be accessed by keyed value. Both structured and unstructured (text) fields can be indexed.

### The iTAPS terminal

When the TAPS mode feature is used in the iTAPS terminal, major iTAPS functions are handled at the terminal level. Screen-formatting, edit-checking, and data-validation are performed by the terminal. This frees the central processor from the burden of performing these tasks. iTAPS supports both block and TAPS modes in the intelligent terminals. The TAPS mode terminal feature enables an operator to correct an error while the source document is readily available.

### Summary

iTAPS has been specifically designed to satisfy the requirements of transaction processing. It is a complete software system with an efficient architecture, a simplified programming style, and a wide array of features designed to optimize both applications development and on-line transaction processing. When Intel's iTAPS is enhanced with iTAPS, the result is application development productivity and a complete user friendly secure run-time system.



# iTPS Transaction Processing Systems Communications

- Extensive array of remote communication connection options
- Emulation of IBM 2780/3780 RJE station
- Emulation of IBM 3270 BISYNC cluster controller
  - Intel Transaction Processing System (iTPS) looks like IBM 3271 control unit with up to 32 devices attached
  - iTPS terminals look like IBM 3277 terminals
  - iTPS printers look like IBM 3284 printers
- Emulation of IBM 3270 SDLC/SNA cluster controller
  - iTPS looks like IBM 3274 control unit with up to 16 devices attached
  - iTPS terminals look like IBM 3278 model 2 terminals
  - iTPS printers look like the IBM 3287 printer
- Communication speeds up to 19,200 bits per second (bps)
- Leased or switched communication lines

## FUNCTIONAL DESCRIPTION

A variety of host communications options are available on the iTPS family of systems. Emulation of an IBM 2780 or 3780 Remote Job Entry station provides an easy and commonly recognized method of file transfer and host communication. Emulation of an IBM 3271 control unit and its attached devices using binary synchronous (BISYNC) protocol is provided for interactive communication requirements. For IBM System Network Architecture (SNA) users, emulation of IBM 3274 or 3276 control unit and its attached devices using synchronous data link control (SDLC) protocol provides interactive SNA communications. Thus, both batch and interactive communication requirements are satisfied.

## FEATURES AND BENEFITS IBM 2780/3780 emulator

The IBM 2780 and 3780 data communications terminals were designed to allow the transmission and reception of large volumes of data at communications line speed. The devices consist of an 80 column card reader, a printer and an optional 80 column card

punch. The iTPS 2780 and 3780 emulators perform the same functions, using card images from disk files for transfer to the host and accepting print or punch records from the host for transfer to iTPS files or spooling to the printer.

The following IBM 2780/3780 features are supported by the emulators:

- Full data link control
- Full time-out control
- Cyclic redundancy checking
- Printer carriage control decoding
- Component selection
- Horizontal format control
- EBCDIC transparency
- Processor interrupt
- Space compression/expansion (3780 mode only)
- Multiple record transmission (2780 only)
- Leased communications lines

The emulators provide automatic or operator selectable ASCII/EBCDIC code translation. Operator selectable device modes are: 2780, 3780, 2770, 3741 or CPU. CPU mode is useful in computer-to-computer communications where record padding and truncation are not

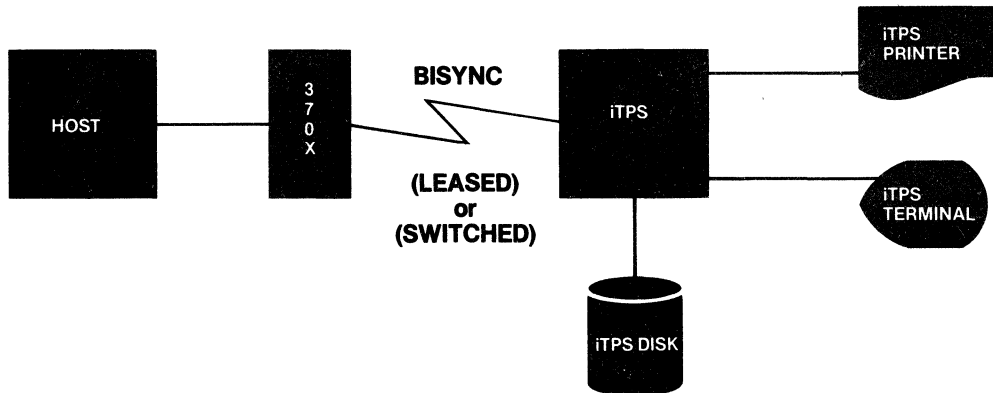
desired. The CPU mode allows one iTPS to communicate with other iTPS systems. The console operation can be optionally assigned to the disk file, minimizing operator interaction.

The emulators may be operated at data rates up to 19,200 bps using:

- Point-to-point (leased or switched) lines with synchronous modems, or
- Hard-wired connection with short-haul synchronous modems, or
- Hard-wired connections with a synchronous modem eliminator.

Interface to the modem or modem eliminator is via an 11 conductor cable equipped with DB-type connectors on both ends. The emulators communicate in half-duplex mode and require only half-duplex communications facilities.

The emulators can communicate with a host system which supports an IBM 2780 or 3780 and has a compatible modem. Typical hosts include IBM 360, IBM 370, IBM 30XX, 430X, System 34, Series 1 and Office System 6 systems.



IBM 2780/3780 Emulation

### IBM 3270 BISYNC emulator

This offering provides emulation of an IBM 3271 model 2 BISYNC cluster controller. iTPS terminals and printers emulate IBM 3277 model 2 terminals and IBM 3284 printers, respectively. Thus, the iTPS system appears as a "real" 3271 cluster operating on a BISYNC line with up to 32 devices attached. The iTPS can offer a wide range of capabilities to end-users, like execution of transactions under CICS, software development activities under TSO, or execution of other applications developed on a host system which interacts with 3270 devices.

Functions of the 3270 such as screen-formatting, polling responses, data link control, time-out control and cyclic redundancy checking are all supported. The emulator may be configured to

define any terminal keyboard character as any 3270 keyboard character. It is possible, then, to define 3270 keys not present on the actual keyboard. An ideal use of this capability is for definition of the 3270 keys ENTER, PRINT, RESET, TAB BACK, TAB, PFn and PAn.

A hard copy of the information currently displayed on the screen may be obtained by pressing the local print key. This image will be directed to a spool file, if so desired.

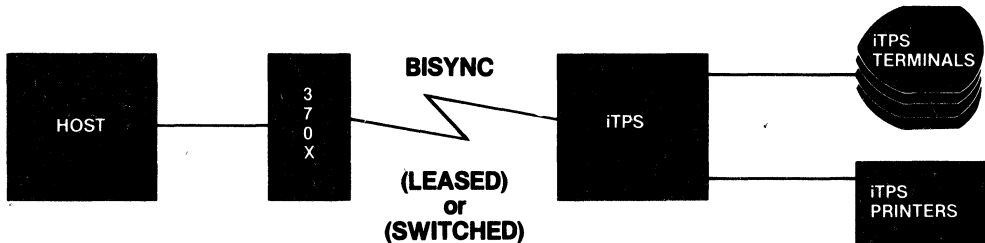
The 3270 BISYNC emulator communicates with the host in half-duplex mode on point-to-point lines. A synchronous modem is used at speeds of 1200 bps to 19,200 bps. The connection to the host may also be hard-wired using inexpensive short-haul

synchronous modems. The iTPS may be multidropped along with other 3270 clusters.

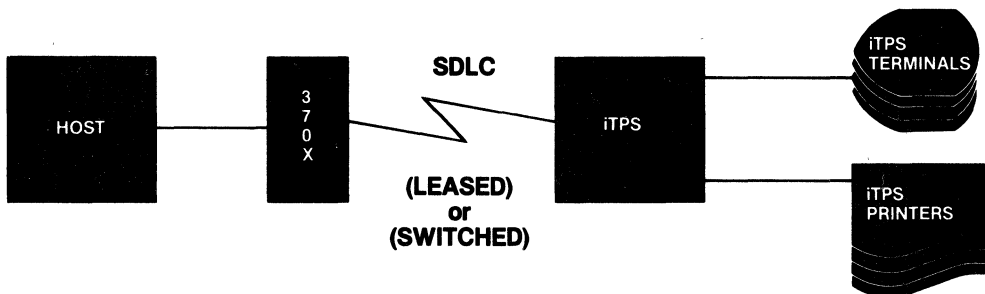
### IBM 3270 SDLC/SNA emulator

Emulation of an IBM 3274 lc and 51c or SDLC/SNA cluster controller is also available. iTPS terminals emulate IBM 3278 model 2 terminals. iTPS printers emulate the IBM 3287 model 2 printer. Thus, the iTPS system appears as a "real" 3274 cluster operating on an SDLC line with up to 16 devices attached. The iTPS appears as a type 2 logical unit (LU-2) as defined in IBM's System Network Architecture for support of display devices and a type 1 and a type 3 logical unit (LU-3) for support of printer devices.

iTPS 3270 SDLC provides for all field display attributes except for the



IBM 3270 BISYNC Emulation



IBM 3270 SDLC/SNA Emulation

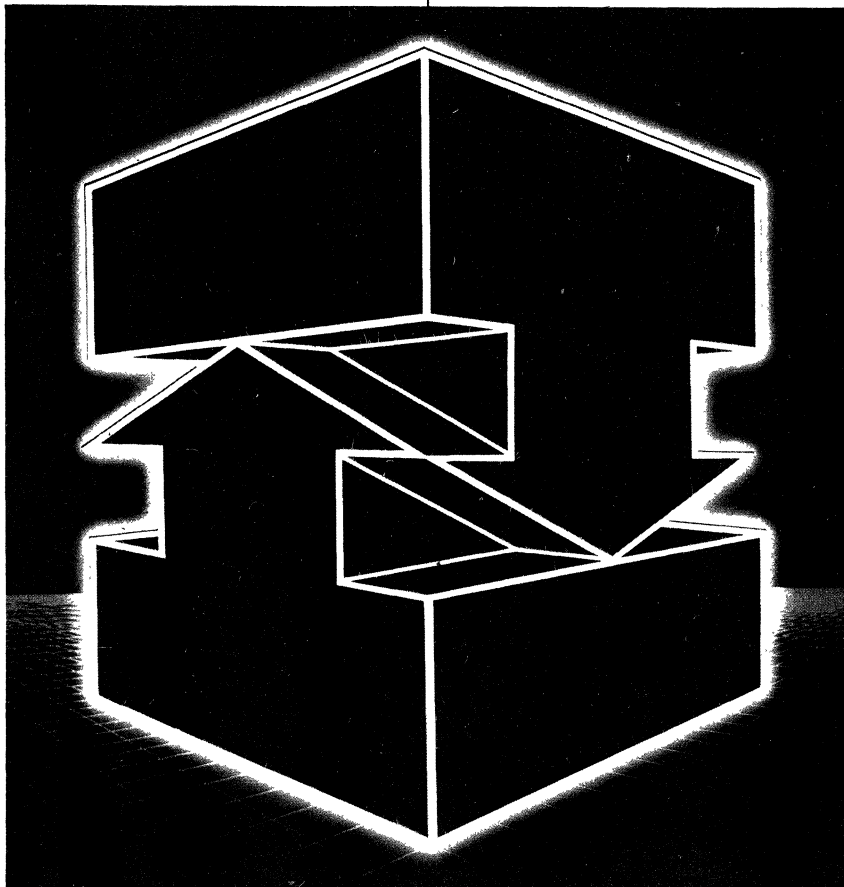
extended attributes for color, extended highlighting or programmed symbols. The 25th line on the iTPS terminals is used to display symbols for INSERT MODE, INPUT INHIBITED, SYSTEM READ, JOB ACTIVE, SYSTEM ACTIVE and CHECK. All keys supported by iTPS are supported by iTPS 3270 SDLC/SNA. In addition, attention, cursor select, delete and insert keys may be defined. The system request key, if defined, may be used to access an SSCP-LU session from a LU-LU session, as with the real 3270.

A hard copy of the information currently displayed on the screen can be obtained by pressing the local print key. The image will be directed to a pool file, if so desired.

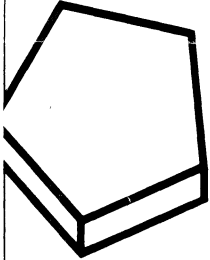
Transmission speeds from 1200 bps to 19,200 bps are supported. The iTPS can be multidropped along with other SDLC/SNA devices.

**SYSTEM 2000<sup>®</sup>**  
**Database**  
**Management**  
**System**  
**Sperry (Univac)**  
**1100 Series**

- Fully integrated facilities
- Controlled sharing of corporate data
- Relational query
- Screens and transactions without programming
- Interactive report generator
- Application development tools
- Decision-assist microcomputer system
- Vendor service and support



## SYSTEM TECHNICAL DESCRIPTION



### Fully integrated facilities

IDD, the integrated Data Dictionary, controls all database-related information and processes. All access to SYSTEM 2000® is through the IDD, which acts as a control point for Data Processing. Operating at the core of Intel's SYSTEM 2000, IDD stores information on database structures and processes in a form that is compatible with the DBMS functions. Both end users and applications programmers rely on IDD functions for DBMS requests. IDD handles definition, control, and reporting functions. All updates are integrated through IDD, and user-defined procedure and other facilities operate through IDD, thus creating a central definition for all access to data. IDD security features protect the information resource from unauthorized access and usage. IDD enables Data Processing to retain greater control over the environment, provides greater systems reliability, and allows for reduced application development time.

### Controlled sharing of corporate data

With SYSTEM 2000, the data administrator can maintain central control of database files. All access by end users can thus be controlled and scrutinized. System integrity and control features allow multiple levels of database recovery spanning database archiving, roll-forward, and update logs. In case of program or machine failure, data is recoverable. Concurrent access and update are allowed and protected without application support, thus saving application time.

IDD has been designed with inherent security features to protect the information resource from unauthorized access and usage. The key security feature is a password procedure which prevents unauthorized database creation or revision. The data administrator applies the password at database, record, and item levels to impose any combination of retrieval, update, or search restrictions for each item in a database. The data administrator also uses IDD to specify restart and recovery parameters for each physical database in the environment.

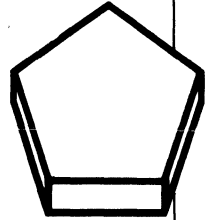
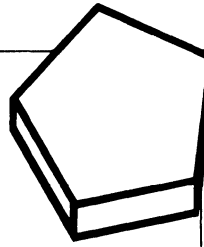
SYSTEM 2000 supports concurrent processing, including concurrent updating, in all environments. Users, databases, and resources are coordinated via the Concurrent Update facility. This coordination ensures the integrity of the database and makes the most efficient utilization of resources to support production processing.

### Relational query

QUEST is an English-like query/update language, designed for the end user in support of ad hoc access to the database, as well as for the application developer in support of database testing and prototyping. Since this free-form, natural language is designed for ease of use and has powerful search, display and update capability, it is ideally suited for all database access needs.

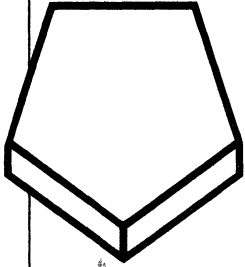
For the end user, QUEST represents a powerful and user-friendly relational-like language to support on-line, ad hoc access to SYSTEM 2000. For the application programmer, QUEST is a convenient facility for trying out basic report and update procedures and as a debug tool for application program development. By giving the end user direct access to data, QUEST can ease application backlogs and substantially contribute to improving an organization's overall productivity.

QueX, Query/Update by Example, is a powerful aid for the end user environment. QueX extends the power of SYSTEM 2000 beyond QUEST, Report Writer, and PLEX by providing function-driven, screen-oriented query/update to the non-Data Processing user. QueX supports full networking and multiple database access and requires only one hour of training for non-DP end users.





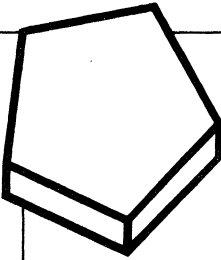
QueX provides a user with a fill-in-the-blank approach. QueX performs all concurrency control automatically and helps the user through each retrieval operation, continually displaying the current record. QueX also enables the user to perform data entry, deletion, and modification where so authorized. Once the appropriate databases have been created, QueX systems can be developed and in production in minutes. In short, QueX is the solution to immediate end user productivity.



### **Screens and transactions without programming**

Screen Writer is an extension to QueX which allows screens to be defined and conditionally processed without any programming. Screen formats as well as the logic to process those screens are defined during an interactive session. Transaction-oriented systems can be developed in hours instead of months. Powerful security, editing, and validation features make Screen Writer an extremely effective tool for entering and updating data. Editing and validation rules can even be specified at the character level.

Both QueX and Screen Writer are available on a wide range of synchronous and asynchronous terminals.

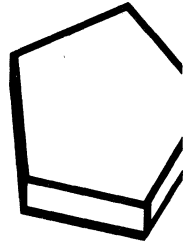


### **Interactive report generator**

Report Writer is a facility which supplies end users and programmers with demand and batch reporting capabilities for both simple and complex requirements.

Report Writer can generate reports against all or selected portions of a database. Report Writer provides data editing, page formatting, calculations, summaries, and ordering of printed information in either demand or batch mode. By providing a quick and easy way to produce reports, Report Writer eases the burden on Data Processing, thereby reducing the application development costs.

Report Writer has a conversational option called Genius. Based on the user's response to a series of prompts, Genius generates distribution-quality reports in just minutes, without programming. Genius has the ability to store report specifications and generate the format at a later date. It not only provides the error detection and recovery capabilities needed for inexperienced users, it also makes the programmer's job easier by eliminating coding in report production. A graphics option allows the user to generate bar, pie, line and plot charts in a conversational manner.



### **Programmer language interfaces**

SYSTEM 2000 addresses programmer needs by offering programmer productivity tools which assist in reducing application development time for applications which may require programming. These productivity tools work in concert to provide the benefits of data independence, on-line applications support, complete documentation of the environment, and reduced programmer time.

With PLEX, the Programming Language Extension to SYSTEM 2000, the application developer has powerful database search and manipulation capabilities at his disposal. Programmers can focus on specific application solutions, leaving SYSTEM 2000 to address the physical database environment. PLEX is ideally suited for production-oriented applications in demand and batch environments.

PLEX offers the application developer a choice of COBOL or FORTRAN programming languages to support database manipulation. PLEX commands are analyzed by a preprocessor that interacts directly with the IDD, thus ensuring system integrity.

PLEX commands are high level English-like verbs which eliminate the need to manipulate pointers or navigate data structures, improving programmer productivity and reducing program maintenance. Capabilities include:

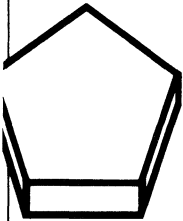
- **Dynamic Networking:** Using PLEX LINK, network relationships among data in one or more databases may be established.

- **Database Loading:** The high speed LOAD utility supports database creation and large volume additions.

- **Data Selection, Retrieval and Updating:** The WHERE clause controls data selection; LOCATE and GET verbs are for retrieval; MODIFY, REMOVE, and INSERT verbs are for updating.

- **Program Independence:** PLEX programs are independent of the physical structure of the databases which they access, a benefit which significantly improves programmer productivity over the traditional file environment.

Features such as Screen Writer, QueX, QUEST, and Genius can be used in most instances to build entire applications without any programming.



### Decision-assist microcomputer system

iDIS™ 86/735 (the Intel Database Information System) is a fully integrated XENIX\*-based microcomputer that allows decision-assist needs to be supported on corporate data resources. iDIS solves the problem of uncontrolled microcomputer proliferation by distributing the database where it is needed while enabling Data

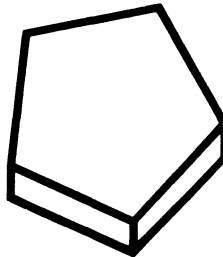
\*Xenix is a Trademark of Microsoft Corporation

Processing to maintain control of the database. As a means of distributed processing, iDIS still observes centralized rules on database access, thereby preserving the investment in corporate data. SYSTEM 2000 and iDIS work in concert to extract data from SYSTEM 2000 down to the local iDIS databases, all operating under IDD control.

Many desirable features are inherent in the iDIS system:

- Local relational database operation
- Word processing and graphics capability
- User-friendly environment
- Compact desk-top integrated microsystem
- High-level language support

The hardware for iDIS is Intel's 86/735. iDIS communicates with mainframe via a configurable communications subsystem.



### Complete service and support

**Documentation:** Intel provides comprehensive modular documentation, incorporating a user-friendly approach, to support its full line of DBMS products. At present, 25 different documentation manuals and pocket references are available for various systems audiences.

**Education:** To augment documentation, Intel offers nine complete, cost-effective training classes, including instructor-directed courses and workshops held publicly or at the customer's site. Video-based instruction is also available. The self-paced courses are designed to meet the varying knowledge requirements of the professional and non-technical end user.

**Field Support:** Whatever the level of support required, Intel responds. Intel's field offices are located throughout the U.S., Canada, and Europe. All software under a current license agreement is supported by the Intel Austin Customer Service Department and by the local sales office. Intel maintains a customer service group of highly trained, experienced information systems professionals who have the technical expertise to handle customer needs. A Customer Support Representative (CSR) is assigned to each customer to provide personalized service; the CSR can be reached via the 24-hour TOLL FREE Customer Service Hotline.

**Innovation:** Intel is committed to improving and enhancing its product family, thereby increasing customer productivity and extending their useful system life cycle.

Other Environments:

- IBM OS
- IBM DOS/VSE
- IBM VM/CMS
- CDC 6000, CYBER

**PROGRAMMERS**



**Additional  
Features**

**Application  
Development**

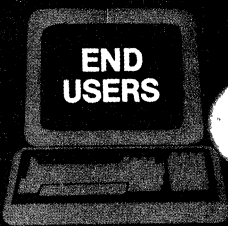
**iDIS**

**DEVELOPMENT  
CENTER**

**Report  
Writing**

**Relational  
Query**

**Graphics**





## SPECIFICATIONS

### Hardware support

Sperry (Univac) 1100 Series

### Operating systems

OS 1100

### Communications

Demand or batch mode

### Access methods

Sequential, indexed sequential

### Main memory requirements

Minimum recommended-30K

### Number of databases

230 maximum (concurrent access)

### Size of data bank

1380 billion characters data (concurrent access)

### Data model

Hierarchical

### Semantic models

Relational, network, hierarchical

### Number of on-line users

230 (concurrent access)

### Data types

Character, text, date, integer, decimal, money

### Functions

- Database definition
- Data dictionary
- Program development
- On-line query/update
- Batch query/update
- Report generator
- On-line data entry
- Multiple database access
- Item level security
- Recovery
- Reorganization utilities
- Concurrency control
- Multi-user/concurrent update
- Full Boolean logic



---

# Component Software

6

---

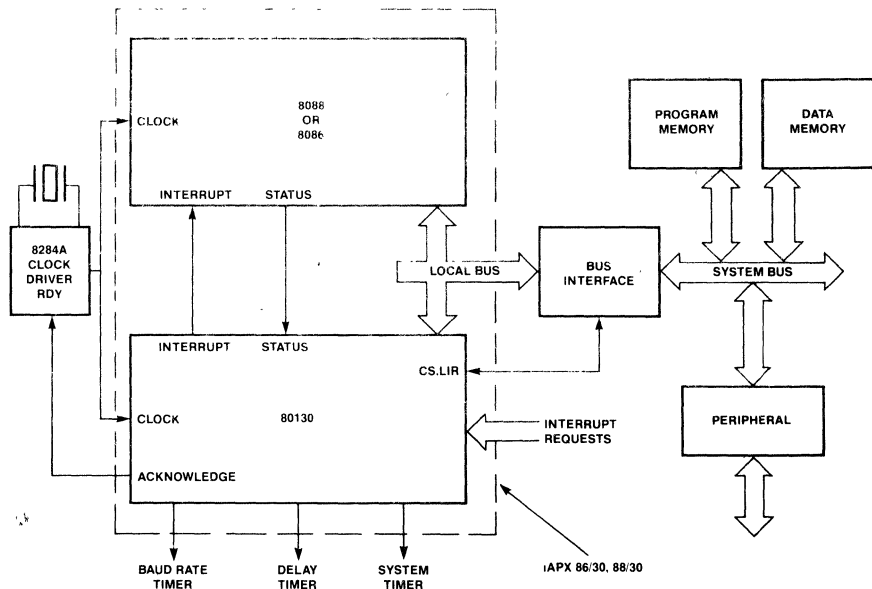


# 80130/80130-2 iAPX 86/30, 88/30, 186/30, 188/30 iRMX 86 OPERATING SYSTEM PROCESSORS

- High-Performance 2-Chip Data Processors Containing Operating System Primitives
  - Standard iAPX 86/10, 88/10 Instruction Set Plus Task Management, Interrupt Management, Message Passing, Synchronization and Memory Allocation Primitives
  - Fully Extendable To and Compatible With iRMX<sup>®</sup> 86
  - Supports Five Operating System Data
- Types: Jobs, Tasks, Segments, Mailboxes, Regions
  - 35 Operating System Primitives
  - Built-In Operating System Timers and Interrupt Control Logic Expandable From 8 to 57 Interrupts
  - 8086/80150/80150-2/8088/80186/80188 Compatible At Up To 8 MHz Without Wait States
  - MULTIBUS<sup>®</sup> System Compatible Interface

The Intel iAPX 86/30 and iAPX 88/30 are two-chip microprocessors offering general-purpose CPU (8086) instructions combined with real-time operating system support. They provide a foundation for multiprogramming and multitasking applications. The iAPX 86/30 consists of an iAPX 86/10 (16-bit 8086 CPU) and an Operating System Firmware (OSF) component (80130). The 88/30 consists of the OSF and an iAPX 88/10 (8-bit 8088 CPU). (80186 or 80188 CPUs may be used in place of the 8086 or 8088.)

Both components of the 86/30 and 88/30 are implemented in N-channel, depletion-load, silicon-gate technology (HMOS), and are housed in 40-pin packages. The 86/30 and 88/30 provide all the functions of the iAPX 86/10, 88/10 processors plus 35 operating system primitives, hardware support for eight interrupts, a system timer, a delay timer and a baud rate generator.



**Figure 1. iAPX 86/30, 88/30 Block Diagram**



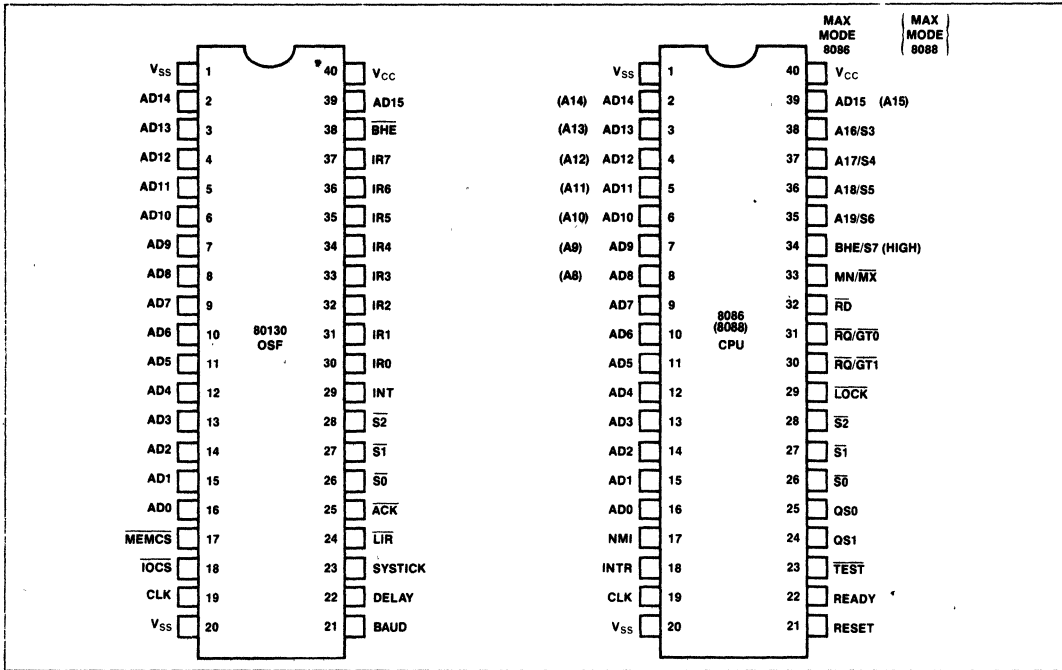


Figure 2. iAPX 86/30, 88/30 Pin Configuration

Table 1. 80130 Pin Description

Symbol	Type	Name and Function																																
AD <sub>15</sub> -AD <sub>0</sub>	I/O	<b>Address Data:</b> These pins constitute the time multiplexed memory address (T <sub>1</sub> ) and data (T <sub>2</sub> , T <sub>3</sub> , T <sub>W</sub> , T <sub>4</sub> ) bus. These lines are active HIGH. The address presented during T <sub>1</sub> of a bus cycle will be latched internally and interpreted as an 80130 internal address if MEMCS or IOCS is active for the invoked primitives. The 80130 pins float whenever it is not chip selected, and drive these pins only during T <sub>2</sub> -T <sub>4</sub> of a read cycle and T <sub>1</sub> of an INTA cycle.																																
BHE/S <sub>7</sub>		<b>Bus High Enable:</b> The 80130 uses the $\overline{\text{BHE}}$ signal from the processor to determine whether to respond with data on the upper or lower data pins, or both. The signal is active LOW. BHE is latched by the 80130 on the trailing edge of ALE. It controls the 80130 output data as shown. <table style="margin-left: 40px;"> <tr> <td><math>\overline{\text{BHE}}</math></td> <td>A<sub>0</sub></td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>Word on AD<sub>15</sub>-AD<sub>0</sub></td> </tr> <tr> <td>0</td> <td>1</td> <td>Upper byte on AD<sub>15</sub>-AD<sub>8</sub></td> </tr> <tr> <td>1</td> <td>0</td> <td>Lower byte on AD<sub>7</sub>-AD<sub>0</sub></td> </tr> <tr> <td>1</td> <td>1</td> <td>Upper byte on AD<sub>7</sub>-AD<sub>0</sub></td> </tr> </table>	$\overline{\text{BHE}}$	A <sub>0</sub>		0	0	Word on AD <sub>15</sub> -AD <sub>0</sub>	0	1	Upper byte on AD <sub>15</sub> -AD <sub>8</sub>	1	0	Lower byte on AD <sub>7</sub> -AD <sub>0</sub>	1	1	Upper byte on AD <sub>7</sub> -AD <sub>0</sub>																	
$\overline{\text{BHE}}$	A <sub>0</sub>																																	
0	0	Word on AD <sub>15</sub> -AD <sub>0</sub>																																
0	1	Upper byte on AD <sub>15</sub> -AD <sub>8</sub>																																
1	0	Lower byte on AD <sub>7</sub> -AD <sub>0</sub>																																
1	1	Upper byte on AD <sub>7</sub> -AD <sub>0</sub>																																
$\overline{\text{S}}_2, \overline{\text{S}}_1, \overline{\text{S}}_0$	I	<b>Status:</b> For the 80130, the status pins are used as inputs only. 80130 encoding follows: <table style="margin-left: 40px;"> <tr> <td><math>\overline{\text{S}}_2</math></td> <td><math>\overline{\text{S}}_1</math></td> <td><math>\overline{\text{S}}_0</math></td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>INTA</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>IORD</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>IOWR</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Passive</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Instruction fetch</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>MEMRD</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>Passive</td> </tr> </table>	$\overline{\text{S}}_2$	$\overline{\text{S}}_1$	$\overline{\text{S}}_0$		0	0	0	INTA	0	0	1	IORD	0	1	0	IOWR	0	1	1	Passive	1	0	0	Instruction fetch	1	0	1	MEMRD	1	1	X	Passive
$\overline{\text{S}}_2$	$\overline{\text{S}}_1$	$\overline{\text{S}}_0$																																
0	0	0	INTA																															
0	0	1	IORD																															
0	1	0	IOWR																															
0	1	1	Passive																															
1	0	0	Instruction fetch																															
1	0	1	MEMRD																															
1	1	X	Passive																															

Table 1. 80130 Pin Description (Continued)

Symbol	Type	Name and Function																																																						
CLK	I	<b>Clock:</b> The system clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing. The 80130 uses the system clock as an input to the SYSTICK and BAUD timers and to synchronize operation with the host CPU.																																																						
INT	O	<b>Interrupt:</b> INT is HIGH whenever a valid interrupt request is asserted. It is normally used to interrupt the CPU by connecting it to INTR.																																																						
IR <sub>7</sub> -IR <sub>0</sub>	I	<b>Interrupt Requests:</b> An interrupt request can be generated by raising an IR input (LOW to HIGH) and holding it HIGH until it is acknowledged (Edge-Triggered Mode), or just by a HIGH level on an IR input (Level-Triggered Mode).																																																						
ACK	O	<b>Acknowledge:</b> This line is LOW whenever an 80130 resource is being accessed. It is also LOW during the first INTA cycle and second INTA cycle if the 80130 is supplying the interrupt vector information. This signal can be used as a bus ready acknowledgement and/or bus transceiver control.																																																						
MEMCS	I	<b>Memory Chip Select:</b> This input must be driven LOW when a kernel primitive is being fetched by the CPU. AD <sub>13</sub> -AD <sub>0</sub> are used to select the instruction.																																																						
IOCS	I	<p><b>Input/Output Chip Select:</b> When this input is low, during an IORD or IOWR cycle, the 80130's kernel primitives are accessing the appropriate peripheral function as specified by the following table:</p> <table border="1"> <thead> <tr> <th>BHE</th> <th>A<sub>3</sub></th> <th>A<sub>2</sub></th> <th>A<sub>1</sub></th> <th>A<sub>0</sub></th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>X</td> <td>X</td> <td>X</td> <td>X</td> <td>Passive</td> </tr> <tr> <td>X</td> <td>X</td> <td>X</td> <td>X</td> <td>1</td> <td>Passive</td> </tr> <tr> <td>X</td> <td>0</td> <td>1</td> <td>X</td> <td>X</td> <td>Passive</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>X</td> <td>0</td> <td>Interrupt Controller</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>Systick Timer</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>Delay Counter</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>Baud Rate Timer</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>Timer Control</td> </tr> </tbody> </table>	BHE	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		0	X	X	X	X	Passive	X	X	X	X	1	Passive	X	0	1	X	X	Passive	1	0	0	X	0	Interrupt Controller	1	1	0	0	0	Systick Timer	1	1	0	1	0	Delay Counter	1	1	1	0	0	Baud Rate Timer	1	1	1	1	0	Timer Control
BHE	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>																																																				
0	X	X	X	X	Passive																																																			
X	X	X	X	1	Passive																																																			
X	0	1	X	X	Passive																																																			
1	0	0	X	0	Interrupt Controller																																																			
1	1	0	0	0	Systick Timer																																																			
1	1	0	1	0	Delay Counter																																																			
1	1	1	0	0	Baud Rate Timer																																																			
1	1	1	1	0	Timer Control																																																			
LIR	O	<b>Local Bus Interrupt Request:</b> This signal is LOW when the interrupt request is for a non-slave input or slave input programmed as being a local slave.																																																						
V <sub>CC</sub>		<b>Power:</b> V <sub>CC</sub> is the +5V supply pin.																																																						
V <sub>SS</sub>		<b>Ground:</b> V <sub>SS</sub> is the ground pin.																																																						
SYSTICK	O	<b>System Clock Tick:</b> Timer 0 Output. Operating System Clock Reference. SYSTICK is normally wired to IR <sub>2</sub> to implement operating system timing interrupt.																																																						
DELAY	O	<b>DELAY Timer:</b> Output of timer 1. Reserved by Intel Corporation for future use.																																																						
BAUD	O	<b>Baud Rate Generator:</b> 8254 Mode 3 compatible output. Output of 80130 Timer 2.																																																						

## FUNCTIONAL DESCRIPTION

The increased performance and memory space of iAPX 86/10 and 88/10 microprocessors have proven sufficient to handle most of today's single-task or single-device control applications with performance to spare, and have led to the increased use of these microprocessors to control *multiple* tasks or devices in real-time. This trend has created a new challenge to designers—development of real-time, multitasking application systems and software. Examples of such systems include control systems that monitor and react to external events in real-time, multifunction desktop and personal computers, PABX equip-

ment which constantly controls the telephone traffic in a multiphone office, file servers/disk subsystems controlling and coordinating multiple disks and multiple disk users, and transaction processing systems such as electronics funds transfer.

## The iAPX 86/30, 88/30 Operating System Processors

The Intel iAPX 86/30, 88/30 Operating System Processors (OSPs) were developed to help solve this

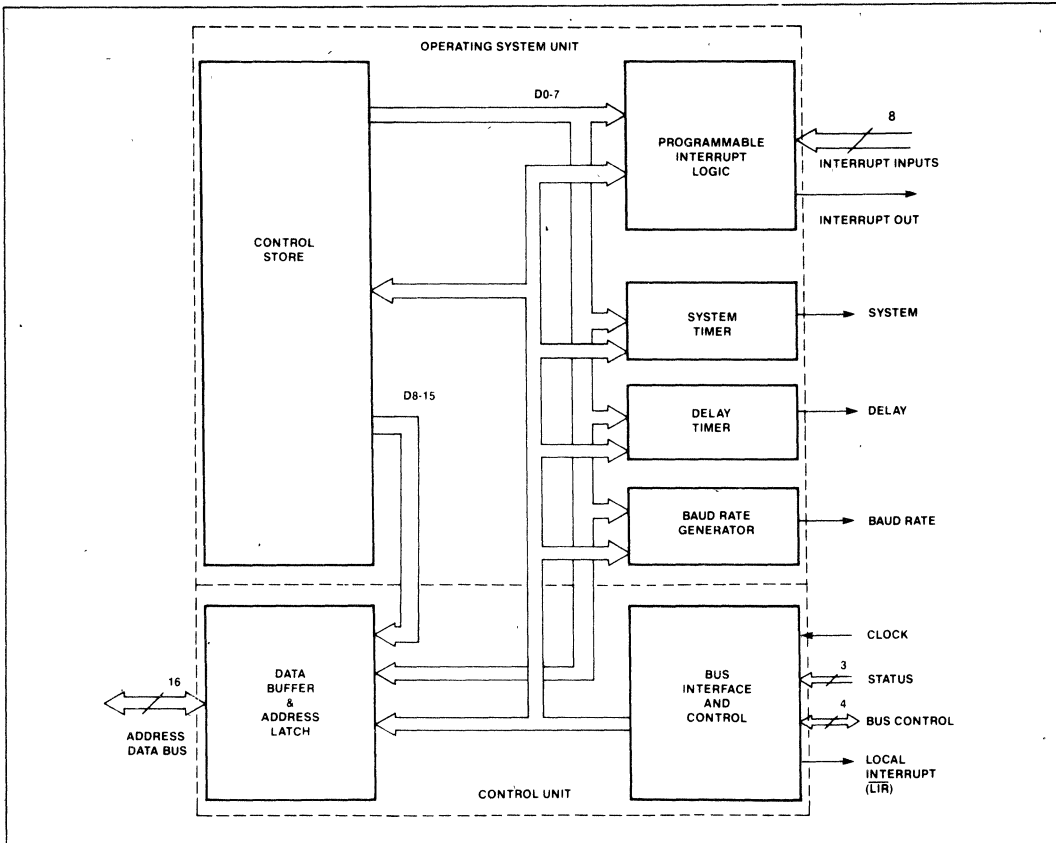


Figure 3. OSF Internal Block Diagram

problem. Their goal is to simplify the design of multi-tasking application systems by providing a well-defined, fully debugged set of operating system primitives implemented directly in the hardware, thereby removing the burden of designing multitasking operating system primitives from the application programmer.

Both the 86/30 and the 88/30 OSPs are two-chip sets consisting of a main processor, an 8086 or 8088 CPU, and the Intel 80130, Operating System Firmware component (OSF) (see Figure 1). The 80130 provides a set of multitasking kernel primitives, kernel control storage, and the additional support hardware, including system timers and interrupt control, required by these primitives. From the application programmer's viewpoint, the OSF extends the base iAPX 86, 88 architecture by providing 35 operating system primitive instructions, and supporting five new system data types, making the OSF a logical and

easy-to-use architectural extension to iAPX 86, 88 system designs.

### The OSP Approach

The OSP system data types (SDTs) and primitive instructions allocate, manage and share low-level processor resources in an efficient manner. For example, the OSP implements task context management (managing a task state image consisting of both hardware register set and software control information) for either the basic 86/10 context or the extended 86/20 (8086+8087) numerics context. The OSP manages the entire task state image both while the task is actively executing and while it is inactive. Tasks can be created, put to sleep for specified periods, suspended, executed to perform their functions, and dynamically deleted when their functions are complete.

The Operating System Processors support event-oriented systems designs. Each event may be processed by an individual responding task or along with other closely related events in a common task. External events and interrupts are processed by the OSP interrupt handler primitives using its built-in interrupt controller subsystem as they occur in real-time. The multiple tasks and the multiple events are coordinated by the OSP integral scheduler whose preemptive, priority-based scheduling algorithm and system timers organize and monitor the processing of every task to guarantee that events are processed as they occur in order of relative importance. The 86/30 also provides primitives for intertask communication (by mailboxes) and for mutual exclusion (by regions), essential functions for multitasking applications.

### Programming Language Support

Programs for the OSP can be written in ASM 86/88 or PL/M 86/88, Intel's standard system languages for iAPX 86,88 systems.

The Operating System Processor Support Package (iOSP 86) provides an interface library for application programs written in any model of PL/M-86. This library also provides 80130 configuration and initialization support as well as complete user documentation.

### OSF PROGRAMMING INTERFACE

The OSF provides 35 operating system kernel primitives which implement multitasking, interrupt management, free memory management, intertask communication and synchronization. Table 4 shows each primitive, and Table 5 gives the execution performance of typical primitives.

OSP primitives are executed by a combination of CPU and OSF (80130) activity. When an OSP primitive is called by an application program task, the iAPX CPU registers and stacks are used to perform the appropriate functions and relay the results to the application programs.

### OSP Primitive Calling Sequences

A standard, stack-based, calling sequence is used to invoke the OSF primitives. Before a primitive is called, its operand parameters must be pushed on the task stack. The SI register is loaded with the offset of the last parameter on the stack. The entry code for the primitive is loaded into AX. The primitive invocation call is made with a CPU software interrupt

(Table 4). A representative ASM86 sequence for calling a primitive is shown in Figure 4. In PL/M the OSP programmer uses a call to invoke the primitive.

SAMPLE ASSEMBLY LANGUAGE PRIMITIVE CALL	
PUSH P <sub>1</sub>	.PUSH PARAMETER 1
PUSH P <sub>2</sub>	.PUSH PARAMETER 2
.	.
.	.
PUSH P <sub>N</sub>	.PUSH PARAMETER N
PUSH BP	.STACK CALLING CONVENTION
MOV BP,SP	
LEA SI,SS:NUM__BYTES__PARAM	2(BP)
	.SS SI POINTS TO FIRST
	.PARAMETER ON STACK
MOV AX, ENTRY CODE	.AX SETS PRIMITIVE ENTRY CODE
INT 184	.OSF INTERRUPT
	OSP PRIMITIVE INVOKED
POP BP	.POP PARAMETERS
RET NUM__BYTES__PARAM.	.CX CONTAINS EXCEPTION CODES
	.DL CONTAINS PARAMETER NUMBER
	. THAT CAUSED EXCEPTION (IF
	. CX IS NON ZERO)
	.AX CONTAINS WORD RETURN VALUE
	.ES BX CONTAINS POINTER
	. RETURN VALUE

Figure 4. ASM/86 OSP Calling Convention

### OSP Functional Description

Each major function of the OSP is described below. These are:

- Job and Task Management
- Interrupt Management
- Free Memory Management
- Intertask Communication
- Intertask Synchronization
- Environmental Control

The system data types (or SDTs) supported by the OSP are capitalized in the description. A short description of each SDT appears in Table 2.

### JOB and TASK Management

Each OSP JOB is a controlled environment in which the applications program executes and the OSF system data types reside. Each individual application program is normally a separate OSP JOB, whether it has one initial task (the minimum) or multiple tasks. JOBS partition the system memory into pools. Each memory pool provides the storage areas in which the OSP will allocate TASK state images and other system data types created by the executing TASKs, and free memory for TASK working space. The OSP supports multiple executing TASKs within a JOB by managing the resources used by each, including the CPU registers, NPX registers, stacks, the system data types, and the available free memory space pool.

When a TASK is created, the OSP allocates memory (from the free memory of its JOB environment) for the TASK's stack and data area and initializes the additional TASK attributes such as the TASK priority level and its error handler location. (As an option, the caller of CREATE TASK may assign previously defined stack and data areas to the TASK.) Task priorities are integers between 0 and 255 (the lower the priority number the higher the scheduling priority of the TASK). Generally, priorities up to 128 will be assigned to TASKs which are to process interrupts. Priorities above 128 do not cause interrupts to be disabled, these priorities (129 to 255) are appropriate for non-interrupt TASKs. If an 8087 Numerics Processor Extension is used, the error recovery interrupt level assigned to it will have a higher priority than a TASK executing on it, so that error handling is performed correctly.

### EXECUTION STATUS

A TASK has an execution status or execution state. The OSP provides five execution states: RUNNING, READY, ASLEEP, SUSPENDED, and ASLEEP-SUSPENDED.

- A TASK is RUNNING if it has control of the processor.
- A TASK is READY if it is not asleep, suspended, or asleep-suspended. For a TASK to become the running (executing) TASK, it must be the highest priority TASK in the ready state.
- A TASK is ASLEEP if it is waiting for a request to be granted or a timer event to occur. A TASK may put itself into the ASLEEP state.
- A TASK is SUSPENDED if it is placed there by another TASK or if it suspends itself. A TASK may have multiple suspensions, the count of suspensions is managed by the OSP as the TASK suspension depth.
- A TASK is ASLEEP-SUSPENDED if it is both waiting and suspended.

TASK attributes, the CPU register values, and the 8087 register values (if the 8087 is configured into the application) are maintained by the OSP in the TASK state image. Each TASK will have a unique TASK state image.

### SCHEDULING

The OSP schedules the processor time among the various TASKs on the basis of priority. A TASK has an execution priority relative to all other TASKs in the system, which the OSP maintains for each TASK in its TASK state image. When a TASK of higher priority than the executing TASK becomes ready to execute,

the OSP switches the control of the processor to the higher priority TASK. First, the OSP saves the outgoing (lower priority) TASK's state including CPU register values in its TASK state image. Then, it restores the CPU registers from the TASK state image of the incoming (higher priority) TASK. Finally, it causes the CPU to start or resume executing the higher priority TASK.

TASK scheduling is performed by the OSP. The OSP's priority-oriented preemptive scheduler determines which TASK executes by comparing their relative priorities. The scheduler insures that the highest priority TASK with a status of READY will execute. A TASK will continue to execute until an interrupt with a higher priority occurs, or until it requests unavailable resources, for which it is willing to wait, or until it makes specific resources available to a higher priority TASK waiting for those resources.

TASKs can become READY by receiving a message, receiving control, receiving an interrupt, or by timing out. The OSP always monitors the status of all the TASKs (and interrupts) in the system. Preemptive scheduling allows the system to be responsive to the external environment while only devoting CPU resources to TASKs with work to be performed.

### TIMED WAIT

The OSP timer hardware facilities support timed waits and timeouts. Thus, in many primitives, a TASK can specify the length of time it is prepared to wait for an event to occur, for the desired resources to become available or for a message to be received at a MAILBOX. The timing interval (or System Tick) can be adjusted, with a lower limit of 1 millisecond.

### APPLICATION CONTROL OF TASK EXECUTION

Programs may alter TASK execution status and priority dynamically. One TASK may suspend its own execution or the execution of another TASK for a period of time, then resume its execution later. Multiple suspensions are provided. A suspended TASK may be suspended again.

The eight OSP Job and TASK management primitives are:

- |                    |  |
|--------------------|--|
| <b>CREATE JOB</b>  | Partitions system resources and creates a TASK execution environment.  |
| <b>CREATE TASK</b> | Creates a TASK state image. Specifies the location of the TASK code instruction stream, its execution priority, and the other TASK attributes. |

- DELETE TASK**      Deletes the TASK state image, removes the instruction stream from execution and deallocates stack resources. Does not delete INTERRUPT TASKS.
  
- SUSPEND TASK**    Suspends the specified TASK or, if already suspended, increments its suspension depth by one. Execute state is SUSPEND.
  
- RESUME TASK**     Decrements the TASK suspension depth by one. If the suspension depth is then zero, the primitive changes the task execution status to READY, or ASLEEP (if ASLEEP/SUSPENDED).
  
- SLEEP**            Places the requesting TASK in the ASLEEP state for a specified number of System Ticks. (The TICK interval can be configured down to 1 millisecond.)
  
- SET PRIORITY**    Alters the priority of a TASK.

### Interrupt Management

The OSP supports up to 256 interrupt levels organized in an interrupt vector, and up to 57 external interrupt sources of which one is the NMI (Non-Maskable Interrupt). The OSP manages each interrupt level independently. The OSF INTERRUPT SUBSYSTEM provides two mechanisms for interrupt management: INTERRUPT HANDLERS and INTERRUPT TASKS. INTERRUPT HANDLERS disable all maskable interrupts and should be used only for servicing interrupts that require little processing time. Within an INTERRUPT HANDLER only certain OSF Interrupt Management primitives (DISABLE, ENTER INTERRUPT, EXIT INTERRUPT, GET LEVEL, SIGNAL INTERRUPT) and basic CPU instructions can be used, other OSP primitives cannot be. The INTERRUPT TASK approach permits all OSP primitives to be issued and masks only lower priority interrupts.

Work flow between an INTERRUPT HANDLER and an INTERRUPT TASK assigned to the same level is regulated with the SIGNAL INTERRUPT and WAIT INTERRUPT primitives. The flow is asynchronous. When an INTERRUPT HANDLER signals an INTERRUPT TASK, the INTERRUPT HANDLER becomes immediately available to process another interrupt. The number of interrupts (specified for the level) the

INTERRUPT HANDLER can queue for the INTERRUPT TASK can be limited to the value specified in the SET INTERRUPT primitive. When the INTERRUPT TASK is finished processing, it issues a WAIT INTERRUPT primitive, and is immediately ready to process the queue of interrupts that the INTERRUPT HANDLER has built with repeated SIGNAL INTERRUPT primitives while the INTERRUPT TASK was processing. If there were no interrupts at the level, the queue is empty and the INTERRUPT TASK is SUSPENDED. See the Example (Figure 5) and Figures 6 and 7.

OSP external INTERRUPT LEVELS are directly related to internal TASK scheduling priorities. The OSP maintains a single list of priorities including both tasks and INTERRUPT LEVELS. The priority of the executing TASK automatically determines which interrupts are masked. Interrupts are managed by INTERRUPT LEVEL number. The OSP supports eight levels directly and may be extended by means of slave 8259As to a total of 57.

The nine Interrupt Management OSP primitives are:

- DISABLE**            Disables an external INTERRUPT LEVEL.
  
- ENABLE**            Enables an external INTERRUPT LEVEL.
  
- ENTER INTERRUPT**    Gives an Interrupt Handler its own data segment, separate from the data segment of the interrupted task.
  
- EXIT INTERRUPT**    Performs an "END of INTERRUPT" operation. Used by an INTERRUPT HANDLER which does not invoke an INTERRUPT TASK. Reenables interrupts, when the INTERRUPT HANDLER gives up control.
  
- GET LEVEL**         Returns the interrupt level number of the executing INTERRUPT HANDLER.
  
- RESET INTERRUPT**    Cancels the previous assignment made to an interrupt level by SET INTERRUPT primitive request. If an INTERRUPT TASK has been assigned, it is also deleted. The interrupt level is disabled.
  
- SET INTERRUPT**     Assigns an INTERRUPT HANDLER to an interrupt level and, optionally, an INTERRUPT TASK.

```

/* CODE EXAMPLE A INTERRUPT TASK TO KEEP TRACK OF TIME-OF-DAY
DECLARE SECONDSCOUNT BYTE,
MINUTESCOUNT BYTE,
HOURSACCOUNT BYTE;
TIMESTASK: PROCEDURE;
DECLARE TIMEEXCEPTSCODE WORD,
ACSCYCLESCOUNT=0;
CALL ROSSETSINTERRUPT(ACSINTERRUPTSLEVEL, 01H),
@ACSHANDLER.0,@TIMEEXCEPTSCODE);
CALL ROSRESUMESTASK(INITTASKSTOKEN,@TIMEEXCEPTSCODE);
DO HOURSACCOUNT=0 TO 23;
DO MINUTESCOUNT=0 TO 59;
DO SECONDSCOUNT=0 TO 59;
CALL ROSWAITSINTERRUPT(ACSINTERRUPTSLEVEL,
@TIMEEXCEPTSCODE),
IF SECONDSCOUNT MOD 5=0
THEN CALL PROTECTEDSCRTSOUT(BEL);
END, /* SECOND LOOP */
END, /* MINUTE LOOP */
END, /* HOUR LOOP */
CALL ROSRESETSINTERRUPT(ACSINTERRUPTSLEVEL,@TIMEEXCEPTSCODE);
END TIMESTASK;
/* CODE EXAMPLE B INTERRUPT HANDLER TO SUBDIVIDE A.C. SIGNAL BY 60. */
DECLARE ACSCYCLESCOUNT BYTE,
ACSHANDLER: PROCEDURE INTERRUPT 59,
DECLARE ACSEXCEPTSCODE WORD,
ACSCYCLESCOUNT=ACSCYCLESCOUNT +1,
IF ACSCYCLESCOUNT>=60 THEN DO,
ACSCYCLESCOUNT=0,
CALL ROSSIGNALSINTERRUPT(ACSINTERRUPTSLEVEL,@ACSEXCEPTSCODE),
END;
END ACSHANDLER;

```

Figure 5. OSP Examples

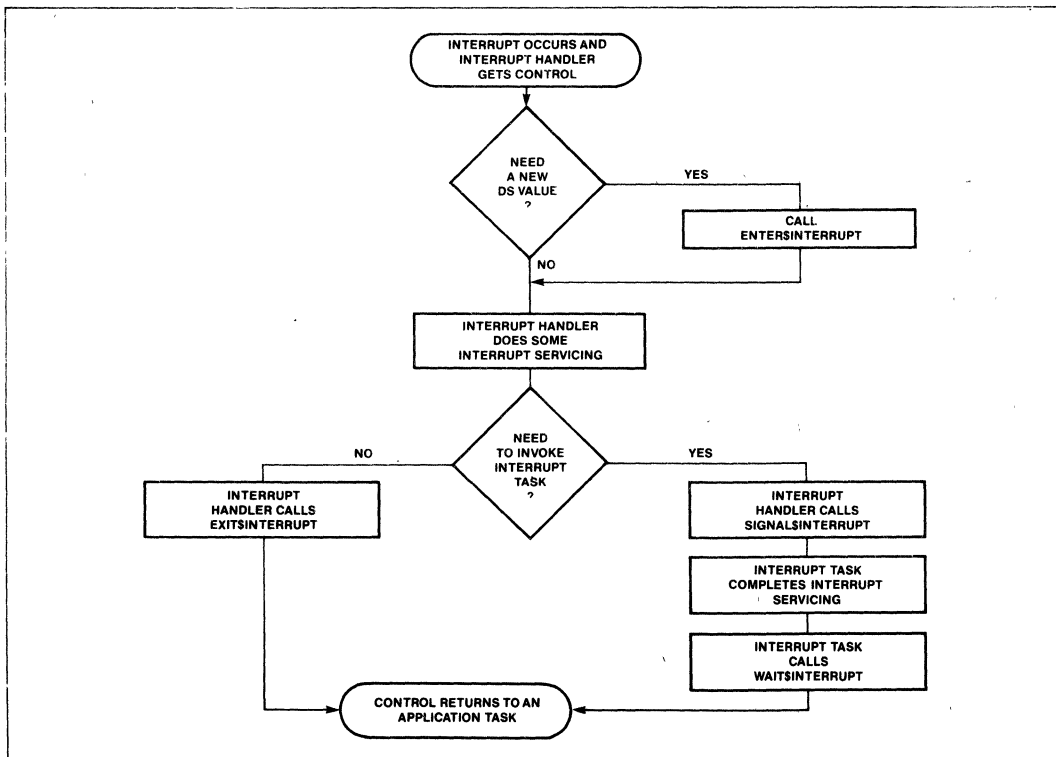


Figure 6. Interrupt Handling Flowchart

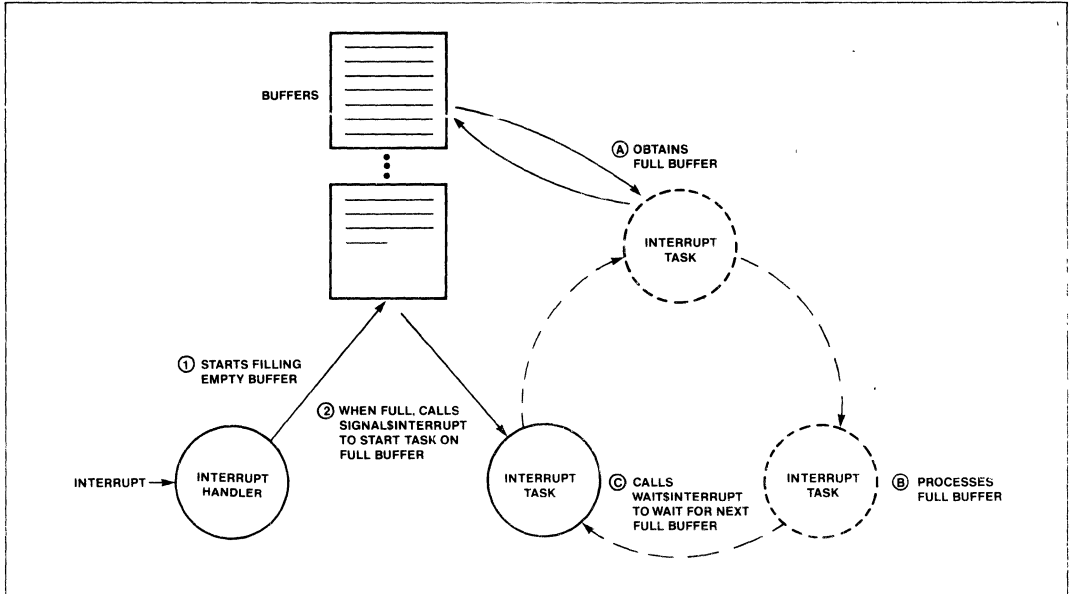


Figure 7. Multiple Buffer Example

**SIGNAL INTERRUPT** Used by an INTERRUPT HANDLER to activate an Interrupt Task.

**WAIT INTERRUPT** Suspends the calling Interrupt Task until the INTERRUPT HANDLER performs a SIGNAL INTERRUPT to invoke it. If a SIGNAL INTERRUPT for the task has occurred, it is processed.

**FREE MEMORY MANAGEMENT**

The OSP Free Memory Manager manages the memory pool which is allocated to each JOB for its execution needs. (The CREATE JOB primitive allocates the new JOB's memory pool from the memory pool of the parent JOB.) The memory pool is part of the JOB resources but is not yet allocated between the tasks of the JOB. When a TASK, MAILBOX, or REGION system data type structure is created within that JOB, the OSP implicitly allocates memory for it from the JOB's memory pool, so that a separate call to allocate memory is not required. OSP primitives that use free memory management implicitly include CREATE JOB, CREATE TASK, DELETE TASK, CREATE MAILBOX, DELETE MAILBOX, CREATE REGION, and DELETE REGION. The

CREATE SEGMENT primitive explicitly allocates a memory area when one is needed by the TASK. For example, a TASK may explicitly allocate a SEGMENT for use as a memory buffer. The SEGMENT length can be any multiple of 16 bytes between 16 bytes and 64K bytes in length. The programmer may specify any number of bytes from 1 byte to 64 KB, the OSP will transparently round the value up to the appropriate segment size.

The two explicit memory allocation/deallocation primitives are:

**CREATE SEGMENT** Allocates a SEGMENT of specified length (in 16-byte-long paragraphs) from the JOB Memory Pool.

**DELETE SEGMENT** Deallocates the SEGMENT's memory area, and returns it to the JOB memory pool.

**Intertask Communication**

The OSP has built-in intertask synchronization and communication, permitting TASKS to pass and share information with each other. OSP MAILBOXes contain controlled handshaking facilities which guarantee that a complete message will always be sent from a sending TASK to a receiving TASK. Each MAILBOX consists of two interlocked queues, one of TASKS



and the other of Messages. Four OSP primitives for intertask synchronization and communication are provided:

CREATE MAILBOX	Creates intertask message exchange.
DELETE MAILBOX	Deletes an intertask message exchange.
RECEIVE MESSAGE	Calling TASK receives a message from the MAILBOX.
SEND MESSAGE	Calling TASK sends a message to the MAILBOX.

The CREATE MAILBOX primitive allocates a MAILBOX for use as an information exchange between TASKs. The OSP will post information at the MAILBOX in a FIFO (First-In First-Out) manner when a SEND MESSAGE instruction is issued. Similarly, a message is retrieved by the OSP if a TASK issues a RECEIVE MESSAGE primitive. The TASK which creates the MAILBOX may make it available to other TASKs to use.

If no message is available, the TASK attempting to receive a message may choose to wait for one or continue executing.

The queue management method for the task queue (FIFO or PRIORITY) determines which TASK in the MAILBOX TASK queue will receive a message from the MAILBOX. The method is specified in the CREATE MAILBOX primitive.

## Intertask Synchronization and Mutual Exclusion

Mutual exclusion is essential to multiprogramming and multiprocessing systems. The REGION system data type implements mutual exclusion. A REGION is represented by a queue of TASKs waiting to use a resource which must be accessed by only one TASK at a time. The OSP provides primitives to use REGIONS to manage mutually exclusive data and resources. Both critical code sections and shared data structures can be protected by these primitives from simultaneous use by more than one task. REGIONS support both FIFO (First-In First-Out) or Priority queueing disciplines for the TASKs seeking to enter the REGION. The REGION SDT can also be used to implement software locks.

Multiple REGIONS are allowed, and are automatically exited in the reverse order of entry. While in a REGION, a TASK cannot be suspended by itself or any other TASK, and thereby avoids deadlock.

There are five OSP primitives for mutual exclusion:

CREATE REGION	Create a REGION (lock).
SEND CONTROL	Give up the REGION.
ACCEPT CONTROL	Request the REGION, but do not wait if it is not available.
RECEIVE CONTROL	Request a REGION, wait if not immediately available.
DELETE REGION	Delete a REGION.

The OSP also provides dynamic priority adjustment for TASKs within priority REGIONS: If a higher-priority TASK issues a RECEIVE CONTROL primitive, while a (lower-priority) TASK has the use of the same REGION, the lower-priority TASK will be transparently, and temporarily, elevated to the waiting TASK's priority until it relinquishes the REGION via SEND CONTROL. At that point, since it is no longer using the critical resource, the TASK will have its normal priority restored.

## OSP Control Facilities

The OSP also includes system primitives that provide both control and customization capabilities to a multitasking system. These primitives are used to control the deletion of SDTs and the recovery of free memory in a system, to allow interrogation of operating system status, and to provide uniform means of adding user SDTs and type managers.

### DELETION CONTROL

Deletion of each OSP system data type is explicitly controlled by the applications programmer by setting a deletion attribute for that structure. For example, if a SEGMENT is to be kept in memory until DMA activity is completed, its deletion attribute should be disabled. Each TASK, MAILBOX, REGION, and SEGMENT SDT is created with its deletion attribute enabled (i.e., they may be deleted). Two OSP primitives control the deletion attribute: ENABLE DELETION and DISABLE DELETION.

### ENVIRONMENTAL CONTROL

The OSP provides inquiry and control operations which help the user interrogate the application environment and implement flexible exception handling. These features aid in run-time decision making and in application error processing and recovery. There are five OSP environmental control primitives.

### OS EXTENSIONS

The OSP architecture is defined to allow new user-defined System Data Types and the primitives to manipulate them to be added to OSP capabilities

provided by the built-in System Data Types. The type managers created for the user-defined SDTs are called user OS extensions and are installed in the system by the SET OS EXTENSION primitive. Once installed, the functions of the type manager may be invoked with user primitives conforming to the OSP interface. For well-structured extended architectures, each OS extension should support a separate user-defined system data type, and every OS extension should provide the same calling sequence and program interface for the user as is provided for a built-in SDT. The type manager for the extension would be written to suit the needs of the application. OSP interrupt vector entries (224–255) are reserved for user OS extensions and are not used by the OSP. After assigning an interrupt number to the extension, the extension user may then call it with the standard OSP call sequence (Figure 4), and the unique software interrupt number assigned to the extension.

ENABLE DELETION	Allows a specific SEGMENT, TASK, MAILBOX, or REGION SDT to be deleted.
DISABLE DELETION	Prevents a specific SEGMENT, TASK, MAILBOX, or REGION SDT from being deleted.
GET TYPE	Given a token for an instance of a system data type, returns the type code.
GET TASK TOKENS	Returns to the caller information about the current task environment.
GET EXCEPTION HANDLER	Returns information about the calling TASK's current information handler: its address, and when it is used.
SET EXCEPTION HANDLER	Provides the address and usage of an exception handler for a TASK.
SET OS EXTENSION	Modifies one of the interrupt vector entries reserved for OS extensions (224–255) to point to a user OS extension procedure.
SIGNAL EXCEPTION	For use in OS extension error processing.

## EXCEPTION HANDLING

The OSP supports exception handlers. These are similar to CPU exception handlers such as OVERFLOW and ILLEGAL OPERATION. Their purpose is to

allow the OSP primitives to report parameter errors in primitive calls, and errors in primitive usage. Exception handling procedures are flexible and can be individually programmed by the application. In general, an exception handler if called will perform one or more of the following functions:

- Log the Error.
- Delete/Suspend the Task that caused the exception.
- Ignore the error, presumably because it is not serious.

An EXCEPTION HANDLER is written as a procedure. If PLM/86 is used, the “compact,” “medium” or “large” model of computation should be specified for the compilation of the program. The mode in which the EXCEPTION HANDLER operates may be specified in the SET EXCEPTION HANDLER primitive. The return information from a primitive call is shown in Figure 4. CX is used to return standard system error conditions. Table 7 shows a list of these conditions, using the *default* EXCEPTION HANDLER of the OSP.

## HARDWARE DESCRIPTION

The 80130 operates in a closely coupled mode with the iAPX 86/10 or 88/10 CPU. The 80130 resides on the CPU local multiplexed bus (Figure 8). The main processor is always configured for maximum mode operation. The 80130 automatically selects between its 88/30 and 86/30 operating modes.

The 80130 used in the 86/30 configuration, as shown in Figure 8 (or a similar 88/30 configuration), operates at both 5 and 8 MHz without requiring processor wait states. Wait state memories are fully supported, however. The 80130 may be configured with both an 8087 NPX\* and an 8089 IOP, and provides full context control over the 8087.

The 80130 (shown in Figure 3) is internally divided into a control unit (CU) and operating system unit (OSU). The OSU contains facilities for OSP kernel support including the system timers for scheduling and timing waits, and the interrupt controller for interrupt management support.

## iAPX 86/30, iAPX 88/30 System Configuration

The 80130 is both I/O and memory mapped to the local CPU bus. The CPU's status S0/S2/ is decoded along with IOCS/ (with BHE and AD<sub>3</sub>–AD<sub>0</sub>) or MEMCS/ (with AD<sub>13</sub>–AD<sub>0</sub>). The pins are internally latched. See Table 1 for the decoding of these lines.

## Memory Mapping

Address lines A<sub>19</sub>-A<sub>14</sub> can be used to form MEMCS/ since the 80130's memory-mapped portion is aligned along a 16K-byte boundary. The 80130 can reside on any 16K-byte boundary excluding the highest (FC000H-FFFFFH) and lowest (00000H-003FFH). The 80130 control store code is position-independent except as limited above, in order to make it compatible with many decoding logic designs. AD<sub>13</sub>-AD<sub>0</sub> are decoded by the 80130's kernel control store.

## I/O Mapping

The I/O-mapped portion of the 80130 must be aligned along a 16-byte boundary. Address lines A<sub>15</sub>-A<sub>4</sub> should be used to form IOCS/.

## System Performance

The approximate performance of representative OSP primitives is given in Table 5. These times are shown for a typical iAPX 86/30 implementation with an 8 MHz clock. These execution times are very comparable to the execution times of similar functions in minicomputers (where available) and are an order of magnitude faster than previous generation microprocessors.

## Initialization

Both application system initialization and OSP-specific initialization/configuration are required to use the OSP. Configuration is based on a "database" provided by the user to the iOSP 86 support package. The OSP-specific initialization and configuration information area is assigned to a user memory address adjacent to the 80130's memory-mapped location. (See Application Note 130 for further details.) The configuration data defines whether 8087 support is configured in the system, specifies if slave 8259A interrupt controllers are used in addition to the 80130, and sets the operating system time base (Tick Interval). Also located in the configuration area are the exception handler control parameters, the address location of the (separate) application system configuration area and the OSP extensions in use. The OSP application system configuration area may be located anywhere in the user memory and must include the starting address of the application instruction code to be executed, plus the locations of the RAM memory blocks to be managed by the OSP free memory manager. Complete application system support and the required 80130 configuration support are provided by the iAPX 86/30 and iAPX 88/30 OPERATING SYSTEM PROCESSOR SUPPORT PACKAGE (IOSP 86).

## RAM Requirements

The OSP manages its own interrupt vector, which is assigned to low RAM memory. Working RAM storage is required as stack space and data area. The memory space must be allocated in user RAM.

OSP interrupt vector memory locations 0H-3FFH must be RAM based. The OSP requires 2 bytes of allocated RAM. The processor working storage is dynamically allocated from free memory. Approximately 300 bytes of stack should be allocated for each OSP task.

## TYPICAL SYSTEM CONFIGURATION

Figure 8 shows the processing cluster of a "typical" iAPX 86/30 or iAPX 88/30 OSP system. Not shown are subsystems likely to vary with the application. The configuration includes an 8086 (or 8088) operating in maximum mode, an 8284A clock generator and an 8288 system controller. Note that the 80130 is located on the CPU side of any latches or transceivers. See Intel Application Note 130 for further details on configuration.

## OSP Timers

The OSP Timers are connected to the lower half of the data bus and are addressed at even addresses. The timers are read as two successive bytes, always LSB followed by MSB. The MSB is always latched on a read operation and remains latched until read. Timers are not gatable.

## Baud Rate Generator

The baud rate generator is 8254 compatible (square wave mode 3). Its output, BAUD, is initially high and remains high until the Count Register is loaded. The first falling edge of the clock after the Count Register is loaded causes the transfer of the internal counter to the Count Register. The output stays high for  $N/2$  [ $(N+1)/2$  if  $N$  is odd] and then goes low for  $N/2$  [ $(N-1)/2$  if  $N$  is odd]. On the falling edge of the clock which signifies the final count for the output in low state, the output returns to high state and the Count Register is transferred to the internal counter. The whole process is then repeated. Baud Rates are shown in Table 6.

The baud rate generator is located at 0CH (12), relative to the 16-byte boundary in the I/O space in which the 80130 component is located ("OSF" in the following example), the timer control word is located at

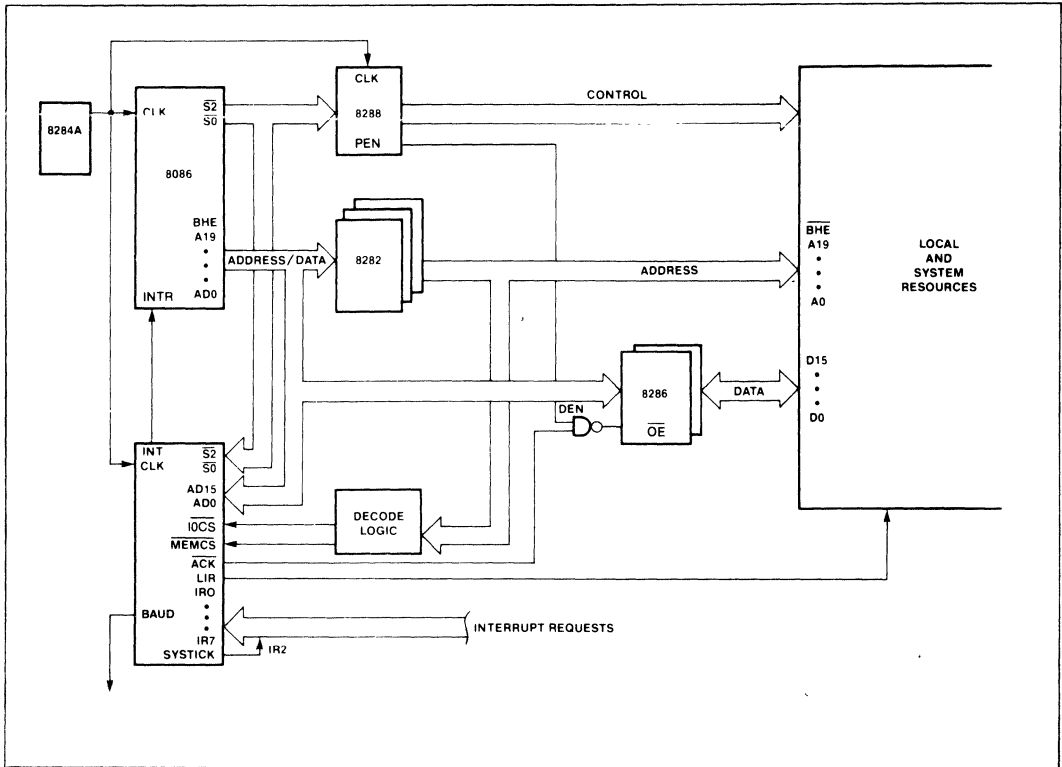


Figure 8. Typical OSP Configuration

relative address, 0EH(14). Timers are addressed with IOCS=0. Timers 0 and 1 are assigned to the use by the OSP, and should not be altered by the user.

For most baud-rate generator applications, the command byte

0B6H Read/Write Baud-Rate Delay Value

will be used. A typical sequence to set a baud rate of 9600 using a count value of 52 follows (see Table 6):

```
MOV AX,0B6H ;Prepare to Write Delay to
              Timer 3.
OUT OSF+14,AX ;Control Word.
MOV AX, 52
OUT OSF+12,AL ;LSB written first
XCHG AL,AH
OUT OSF+12,AL ;MSB written after.
```

The 80130 timers are subset compatible with 8254 timers.

### Interrupt Controller

The Programmable Interrupt Controller (PIC), is also an integral unit of the 80130. Its eight input pins handle eight vectored priority interrupts. One of these pins must be used for the SYSTICK time function in timing waits, using an external connection as shown. During the 80130 initialization and configuration sequence, each 80130 interrupt pin is individually programmed as either level or edge sensitive. External slave 8259A interrupt controllers can be used to expand the total number of OSP external interrupts to 57.

In addition to standard PIC functions, 80130 PIC unit has an LIR output signal, which when low indicates an interrupt acknowledge cycle. LIR=0 is provided to control the 8289 Bus Arbiter SYSB/RESB pin. This will avoid the need of requesting the system bus to acknowledge local bus non-slave interrupts. The user defines the interrupt system as part of the configuration.

## INTERRUPT SEQUENCE

The OSP interrupt sequence is as follows:

1. One or more of the interrupts is set by a low-to-high transition on edge-sensitive IR inputs or by a high input on level-sensitive IR inputs.
2. The 80130 evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an interrupt acknowledge cycle which is encoded in  $S_2-S_0$ .
4. Upon receiving the first interrupt acknowledge from the CPU, the highest-priority interrupt is set by the 80130 and the corresponding edge detect latch is reset. The 80130 does not drive the address/data bus during this bus cycle but does acknowledge the cycle by making  $\overline{ACK}=0$  and sending the  $\overline{LIR}$  value for the IR input being acknowledged.
5. The CPU will then initiate a second interrupt acknowledge cycle. During this cycle, the 80130 will supply the cascade address of the interrupting input at  $T_1$  on the bus and also release an 8-bit pointer onto the bus if appropriate, where it is read by the CPU. If the 80130 does supply the pointer, then  $\overline{ACK}$  will be low for the cycle. This cycle also has the value  $\overline{LIR}$  for the IR input being acknowledged.
6. This completes the interrupt cycle. The ISR bit remains set until an appropriate EXIT INTERRUPT primitive (EOI command) is called at the end of the Interrupt Handler.

## OSP APPLICATION EXAMPLE

Figure 5 shows an application of the OSP primitives to keep track of time of day in a simplified example. The system design uses a 60 Hz A.C. signal as a time base. The power supply provides a TTL-compatible

signal which drives one of 80130 edge-triggered interrupt request pins once each A.C. cycle. The Interrupt Handler responds to the interrupts, keeping track of one second's A.C. cycles. The Interrupt Task counts the seconds and after a day deletes itself. In typical systems it might perform a data logging operation once each day. The Interrupt Handler and Interrupt Task are written as separate modular programs.

The Interrupt Handler will actually service interrupt 59 when it occurs. It simply counts each interrupt, and at a count of 60 performs a SIGNAL INTERRUPT to notify the Interrupt Task that a second has elapsed. The Interrupt Handler (ACS HANDLER) was assigned to this level by the SET INTERRUPT primitive. After doing this, the Interrupt Task performed the Primitive RESUME TASK to resume the application task (INITS TASKS TOKEN).

The main body of the task is the counting loop. The Interrupt Task is signaled by the SIGNAL INTERRUPT primitive in the Interrupt Handler (at interrupt level ACS INTERRUPTS LEVEL). When the task is signaled by the Interrupt Handler it will execute the loop exactly one time, increasing the time count variables. Then it will execute the WAIT INTERRUPT primitive, and wait until awakened by the Interrupt Handler. Normally, the task will now wait some period of time for the next signal. However, since the interface between the Handler and the Task is asynchronous, the handler may have already queued the interrupt for servicing, the writer of the task does not have to worry about this possibility.

At the end of the day, the task will exit the loop and execute RESET INTERRUPT, which disables the interrupt level, and deletes the interrupt task. The OSP now reclaims the memory used by the Task and schedules another task. If an exception occurs, the coded value for the exception is available in TIMES EXCEPTS CODE after the execution of the primitive.

A typical PL/M-86 calling sequence is illustrated by the call to RESET INTERRUPT shown in Figure 5.

Table 2. OSP System Data Type Summary

Job	Jobs are the means of organizing the program environment and resources. An application consists of one or more jobs. Each iAPX 86/30 system data type is contained in some job. Jobs are independent of each other, but they may share access to resources. Each job has one or more tasks, one of which is an initial task. Jobs are given pools of memory, and they may create subordinate offspring jobs, which may borrow memory from their parents.
Task	Tasks are the means by which computations are accomplished. A task is an instruction stream with its own execution stack and private data. Each task is part of a job and is restricted to the resources provided by its job. Tasks may perform general interrupt handling as well as other computational functions. Each task has a set of attributes, which is maintained for it by the iAPX 86/30, which characterize its status. These attributes are: <ul style="list-style-type: none"> <li>its containing job</li> <li>its register context</li> <li>its priority (0–255)</li> <li>its execution state (asleep, suspended, ready, running, asleep/suspended).</li> <li>its suspension depth</li> <li>its user-selected exception handler</li> <li>its optional 8087 extended task state</li> </ul>
Segment	Segments are the units of memory allocation. A segment is a physically contiguous sequence of 16-byte, 8086 paragraph-length, units. Segments are created dynamically from the free memory space of a Job as one of its Tasks requests memory for its use. A segment is deleted when it is no longer needed. The iAPX 86/30 maintains and manages free memory in an orderly fashion, it obtains memory space from the pool assigned to the containing job of the requesting task and returns the space to the job memory pool (or the parent job pool) when it is no longer needed. It does not allocate memory to create a segment if sufficient free memory is not available to it, in that case it returns an error exception code.
Mailbox	Mailboxes are the means of intertask communication. Mailboxes are used by tasks to send and receive message segments. The iAPX 86/30 creates and manages two queues for each mailbox. One of these queues contains message segments sent to the mailbox but not yet received by any task. The other mailbox queue consists of tasks that are waiting to receive messages. The iAPX 86/30 operation assures that waiting tasks receive messages as soon as messages are available. Thus at any moment one or possibly both of two mailbox queues will be empty.
Region	Regions are the means of serialization and mutual exclusion. Regions are familiar as "critical code regions." The iAPX 86/30 region data type consists of a queue of tasks. Each task waits to execute in mutually exclusive code or to access a shared data region, for example to update a file record.
Tokens	The OSP interface makes use of a 16-bit TOKEN data type to identify individual OSF data structures. Each of these (each instance) has its own unique TOKEN. When a primitive is called, it is passed the TOKENs of the data structures on which it will operate.

**Table 3. System Data Type Codes and Attributes**

S.D.T.	Code	Attributes
Jobs	1	Tasks Memory Pool S.D.T. Directory
Tasks	2	Priority Stack Code State Exception Handler
Mailboxes	3	Queue of S.D.T.s (generally segments) Queue of Tasks waiting for S.D.T.s
Region	5	Queue of Tasks waiting for mutually exclusive code or data
Segments	6	Buffer Length

**Table 4. OSP Primitives**

Class	OSP Primitive	Interrupt Number	Entry Code in AX	Parameters On Caller's Stack
J O B	CREATE JOB	184	0100H	*See 80130 User Manual
T A S K	CREATE TASK	184	0200H	Priority, IP Ptr, Data Segment, Stack Seg, Stack Size Task Information, ExcptPtr
	DELETE TASK	184	0201H	TASK, ExcptPtr
	SUSPEND TASK	184	0202H	TASK, ExcptPtr
	RESUME TASK	184	0203H	TASK, ExcptPtr
	SET PRIORITY	184	0209H	TASK, Priority, ExcptPtr
	SLEEP	184	0204H	Time Limit, ExcptPtr
I N T E R R U P T	DISABLE	190	0705H	Level, ExcptPtr
	ENABLE	184	0704H	Level #, ExcptPtr
	ENTER INTERRUPT	184	0703H	Level #, ExcptPtr
	EXIT INTERRUPT	186	NONE	Level #, ExcptPtr
	GET LEVEL	188	0702H	Level #, ExcptPtr
	RESET INTERRUPT	184	.0706H	Level #, ExcptPtr
	SET INTERRUPT	184	0701H	Level, Interrupt Task Flag Interrupt Handler Ptr, Interrupt Handler DataSeg ExcptPtr
	SIGNAL INTERRUPT WAIT INTERRUPT	185 187	NONE NONE	Level, ExcptPtr Level, ExcptPtr
S E G M E N T	CREATE SEGMENT	184	0600H	Size, ExcptPtr
	DELETE SEGMENT	184	0603H	SEGMENT, ExcptPtr

Table 4. OSP Primitives (Continued)

Class	OSP Primitive	Interrupt Number	Entry Code in AX	Parameters On Caller's Stack
M A I L B O X	CREATE MAILBOX	184	0300H	Mailbox flags, ExcptPtr MAILBOX, ExcptPtr MAILBOX, Time Limit ResponsePtr, ExcptPtr MAILBOX, Message Response, ExcptPtr
	DELETE MAILBOX	184	0301H	
	RECEIVE MESSAGE	184	0303H	
	SEND MESSAGE	184	0302H	
R E G I O N	ACCEPT CONTROL	184	0504H	REGION, ExcptPtr
	CREATE REGION	184	0500H	Region Flags, ExcptPtr
	DELETE REGION	184	0501H	REGION, ExcptPtr
	RECEIVE CONTROL	184	0503H	REGION, ExcptPtr
	SEND CONTROL	184	0502H	ExcptPtr
E N V I R O N M E N T A L	DISABLE DELETION	184	0001H	TOKEN, ExcptPtr
	ENABLE DELETION	184	0002H	TOKEN, ExcptPtr
	GET EXCEPTION HANDLER	184	0800H	Ptr, ExcptPtr
	GET TYPE	184	0000H	TOKEN, ExcptPtr
	GET TASK TOKENS	184	0206H	Request, ExcptPtr
	SET EXCEPTION HANDLER	184	0801H	Ptr, ExcptPtr
	SET OS EXTENSION	184	0700H	Code, InstPtr, ExcptPtr
	SIGNAL			
	EXCEPTION	184	0802H	Exception Code, Parameter Number, StackPtr, 0, 0, ExcptPtr

**NOTES:**

All parameters are pushed onto the OSP stack. Each parameter is one word. See Figure 3 for Call Sequence.

**Explanation of the Symbols**

JOB	OSP JOB SDT Token
TASK	OSP TASK SDT Token
REGION	OSP REGION SDT Token
MAILBOX	OSP MAILBOX SDT Token
SEGMENT	OSP SEGMENT SDT Token
TOKEN	Any SDT Token
Level	Interrupt Level Number
ExcptPtr	Pointer to Exception Code
Message	Message Token
Ptr	Pointer to Code, Stack etc. Address
Seg	Value Loaded into appropriate Segment Register
----	Value Parameter



**Table 5. OSP Primitive Performance Examples**

<b>Datatype Class</b>	<b>Primitive Execution Speed* (microseconds)</b>	
JOB	CREATE JOB	2950
	CREATE TASK (no preemption)	1360
TASK	CREATE SEGMENT	700
SEGMENT	SEND MESSAGE (with task switch)	475
	SEND MESSAGE (no task switch)	265
MAILBOX	RECEIVE MESSAGE (task waiting)	540
	RECEIVE MESSAGE (message waiting)	260
REGION	SEND CONTROL	170
	RECEIVE CONTROL	205

\*8 MHz iAPX 86/30 OSP Configuration.

**Table 6. Baud Rate Count Values (16X)**

<b>Baud Rate</b>	<b>8 MHz Count Value</b>	<b>5 MHz Count Value</b>
300	1667	1042
600	833	521
1200	417	260
2400	208	130
4800	104	65
9600	52	33

**Table 7a. Mnemonic Codes for Unavoidable Exceptions**

E\$OK	Exception Code Value = 0 the operation was successful
E\$TIME	Exception Code Value = 1 the specified time limit expired before completion of the operations was possible
E\$MEM	Exception Code Value = 2 insufficient nucleus memory is available to satisfy the request
E\$BUSY	Exception Code Value = 3 specified region is currently busy
E\$LIMIT	Exception Code Value = 4 attempted violation of a job, semaphore, or system limit
E\$CONTEXT	Exception Code Value = 5 the primitive was called in an illegal context (e.g., call to enable for an already enabled interrupt)
E\$EXIST	Exception Code Value = 6 a token argument does not currently refer to any object; note that the object could have been deleted at any time by its owner
E\$STATE	Exception Code Value = 7 attempted illegal state transition by a task
E\$NOT\$CONFIGURED	Exception Code Value = 8 the primitive called is not configured in this system
E\$INTERRUPT\$SATURATION	Exception Code Value = 9 The interrupt task on the requested level has reached its user specified saturation point for interrupt service requests. No further interrupts will be allowed on the level until the interrupt task executes a WAIT\$INTERRUPT. (This error is only returned, in line, to interrupt handlers.)
E\$INTERRUPT\$OVERFLOW	Exception Code Value = 10 The interrupt task on the requested level previously reached its saturation point and caused an E\$INTERRUPT\$SATURATION condition. It subsequently executed an ENABLE allowing further interrupts to come in and has received another SIGNAL\$INTERRUPT call, bringing it over its specified saturation point for interrupt service requests. (This error is only returned, in line, to interrupt handlers.)

**Table 7b. Mnemonic Codes for Avoidable Exceptions**

E\$ZERO\$DIVIDE	Exception Code Value = 8000H divide by zero interrupt occurred
E\$OVERFLOW	Exception Code Value = 8001H overflow interrupt occurred
E\$TYPE	Exception Code Value = 8002H a token argument referred to an object that was not of required type
E\$BOUNDS	Exception Code Value = 8003H an offset argument is out of segment bounds
E\$PARAM	Exception Code Value = 8004H a (non-token, non-offset) argument has an illegal value
E\$BAD\$CALL	Exception Code Value = 8005H an entry code for which there is no corresponding primitive was passed
E\$ARRAY\$BOUNDS = 8006H	Hardware or Language has detected an array overflow
E\$NDP\$ERROR	Exception Code Value = 8007H an 8087 (Numeric data Processor) error has been detected; (the 8087 status information is contained in a parameter to the exception handler)

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bins ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to 150°C  
 Voltage on Any Pin With  
   Respect to Ground ..... -1.0V to +7V  
 Power Dissipation ..... 1.0 Watts

*\*NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.*

**D.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{CC} = 4.5$  to  $5.5\text{V}$ )

Symbol	Parameter	Min.	Max.	Units	Test Conditions
$V_{IL}$	Input Low Voltage	- 0.5	0.8	V	
$V_{IH}$	Input High Voltage	2.0	$V_{CC} + .5$	V	
$V_{OL}$	Output Low Voltage		0.45	V	$I_{OL} = 2\text{mA}$
$V_{OH}$	Output High Voltage	2.4		V	$I_{OH} = -400\mu\text{A}$
$I_{CC}$	Power Supply Current		200	mA	$T_A = 25^\circ\text{C}$
$I_{LI}$	Input Leakage Current		10	$\mu\text{A}$	$0 < V_{IN} < V_{CC}$
$I_{LR}$	IR Input Load Current		10 -300	$\mu\text{A}$	$V_{IN} = V_{CC}$ $V_{IN} = 0$
$I_{LO}$	Output Leakage Current		10	$\mu\text{A}$	$.45 = V_{IN} = V_{CC}$
$V_{CLI}$	Clock Input Low		0.6	V	
$V_{CHI}$	Clock Input High	3.9		V	
$C_{IN}$	Input Capacitance		10	pF	
$C_{IO}$	I/O Capacitance		.15	pF	
$I_{CLI}$	Clock Input Leakage Current		10 150 10	$\mu\text{A}$	$V_{IN} = V_{CC}$ $V_{IN} = 2.5\text{V}$ $V_{IN} = 0\text{V}$

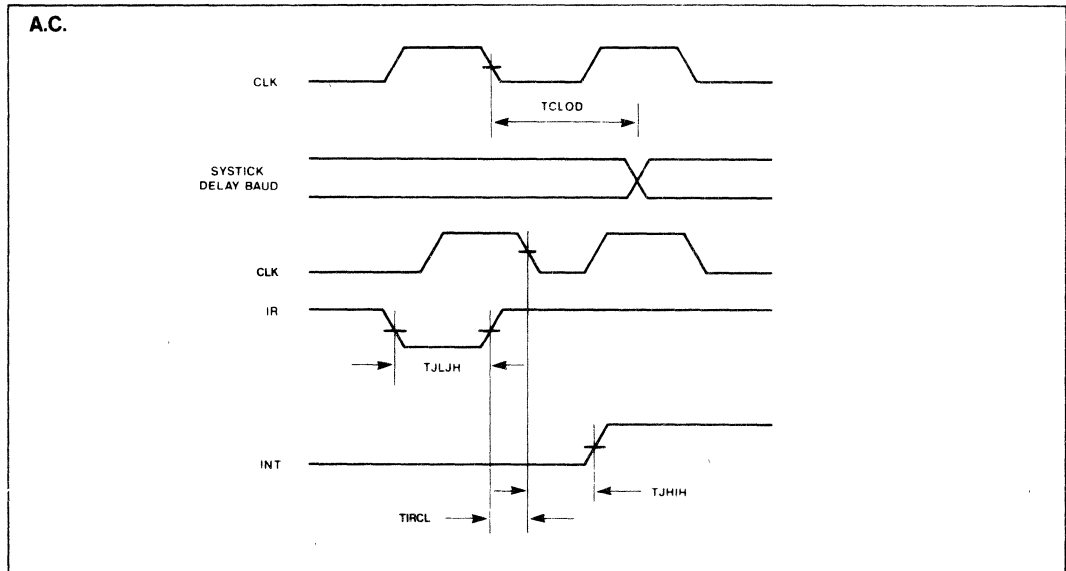
**A.C. CHARACTERISTICS** ( $T_A = 0-70^\circ\text{C}$ ,  $V_{CC} = 4.5-5.5$  Volt,  $V_{SS} = \text{Ground}$ )

Symbol	Parameter	80130		80130-2		Units	Test Conditions
		Min.	Max.	Min.	Max.		
$T_{CLCL}$	CLK Cycle Period	200	-	125	-	ns	
$T_{CLCH}$	CLK Low Time	90	-	55	-	ns	
$T_{CHCL}$	CLK High Time	69	2000	44	2000	ns	
$T_{SVCH}$	Status Active Setup Time	80	-	65	-	ns	
$T_{CHSV}$	Status Inactive Hold Time	10	-	10	-	ns	
$T_{SHCL}$	Status Inactive Setup Time	55	-	55	-	ns	
$T_{CLSH}$	Status Active Hold Time	10	-	10	-	ns	
$T_{ASCH}$	Address Valid Setup Time	8	-	8	-	ns	
$T_{CLAH}$	Address Hold Time	10	-	10	-	ns	
$T_{CSCL}$	Chip Select Setup Time	20	-	20	-	ns	
$T_{CHCS}$	Chip Select Hold Time	0	-	0	-	ns	
$T_{DSCL}$	Write Data Setup Time	80	-	60	-	ns	
$T_{CHDH}$	Write Data Hold Time	10	-	10	-	ns	
$T_{JLJH}$	IR Low Time	100	-	100	-	ns	
$T_{CLDV}$	Read Data Valid Delay	-	140	-	105	ns	$C_L = 200\text{pE}$
$T_{CLDH}$	Read Data Hold Time	10	-	10	-	ns	
$T_{CLDX}$	Read Data to Floating	10	100	10	100	ns	
$T_{CLCA}$	Cascade Address Delay Time	-	85	-	65	ns	

**A.C. CHARACTERISTICS (Continued)**

Symbol	Parameter	80130		80130-2		Units	Notes
		Min.	Max.	Min.	Max.		
$T_{CLCF}$	Cascade Address Hold Time	10	-	10	-	ns	
$T_{IAVE}$	INTA Status t Acknowledge	-	80	-	80	ns	
$T_{CHEH}$	Acknowledge Hold Time	0	-	0	-	ns	
$T_{CSAK}$	Chip Select to ACK	-	110	-	110	ns	
$T_{SACK}$	Status to ACK	-	140	-	140	ns	
$T_{AACK}$	Address to ACK	-	90	-	90	ns	
$T_{CLOD}$	Timer Output Delay Time	-	200	-	200	ns	$C_L = 100$ pF
$T_{CLOD1}$	Timer1 Output Delay Time	-	200	-	200	ns	$C_L = 100$ pF
$T_{JHIH}$	INT Output Delay	-	200	-	200	ns	"
$T_{IRCL}$	IR Input Set Up	20				ns	

**WAVEFORMS**







# 80150/80150-2 iAPX 86/50, 88/50, 186/50, 188/50 CP/M-86\* OPERATING SYSTEM PROCESSORS

ADVANCE INFORMATION

- High-Performance Two-Chip Data Processors Containing the Complete CP/M-86 Operating System
- Standard On-Chip BIOS (Basic Input/Output System) Contains Drivers for 8272A, 8274, 8255A, 8251A
- BIOS Extensible with User-Supplied Peripheral Drivers
- User Intervention Points Allow Addition of New System Commands
- Memory Disk Makes Possible Diskless CP/M-86 Systems
- No License or Serialization Required
- Built-in Operating System Timers and Interrupt Controller
- 8086/80150/80150-2/8088/80186/80188 Compatible At Up To 8 MHz Without Wait States

The Intel iAPX 86/50, 88/50, 186/50, and 188/50 are two-chip microprocessors offering general-purpose CPU instructions combined with the CP/M-86 operating system. Respectively, they consist of the 8- and 16-bit software compatible 8086, 8088, 80186, and 80188 CPU plus the 80150 CP/M-86 operating system extension.

CP/M-86 is a single-user operating system designed for computers based on the Intel iAPX 86, 88, 186, and 188 microprocessors. The system allows full utilization of the one megabyte of memory available for application programs. The 80150 stores CP/M-86 in its 16K bytes of on-chip memory. The 80150 will run third-party applications software written to run under standard Digital Research CP/M-86.

The 80150 is implemented in N-Channel, depletion-load, silicon-gate technology (HMOS), and is housed in a 40-pin package. Included on the 80150 are the CP/M-86 operating system, Version 1.1, plus hardware support for eight interrupts, a system timer, a delay timer, and a baud rate generator.

\*CP/M-86 is a trademark of Digital Research, Inc.

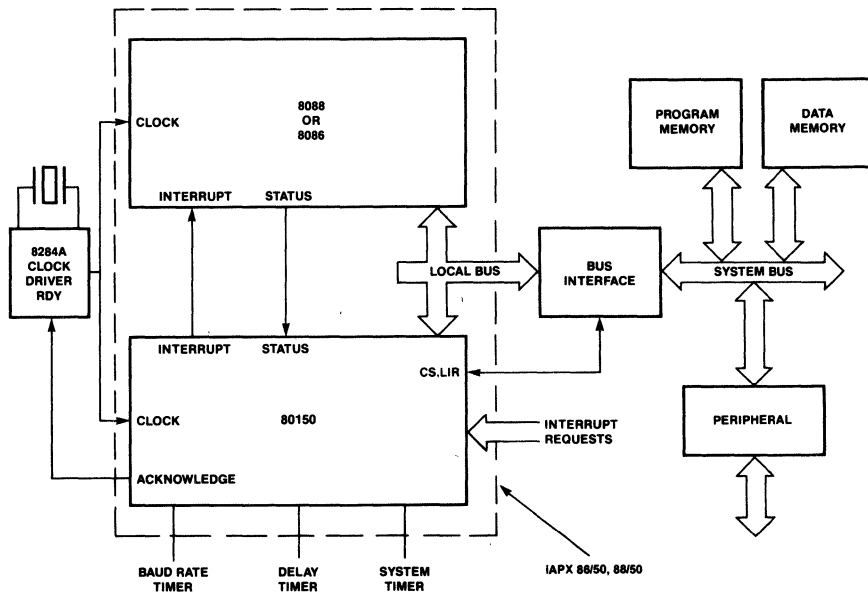


Figure 1. iAPX 86/50, 88/50 Block Diagram

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products. BXP, CREDIT, i, ICE, iCS, iM, Insite, Intel, INTEL, Intelevison, Intellink, Inteltec, iMMX, iOSP, iPDS, iRMX, iSBC, iSBX, Library Manager, MCS, MULTIMODULE, Megachassis, Micromainframe, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RUPi, RMX/80, System 2000, UPI, and the combination of iCS, iRMX, iSBC, iSBX, ICE, i<sup>2</sup>ICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied. ©INTEL CORPORATION, 1982. SEPTEMBER 1982

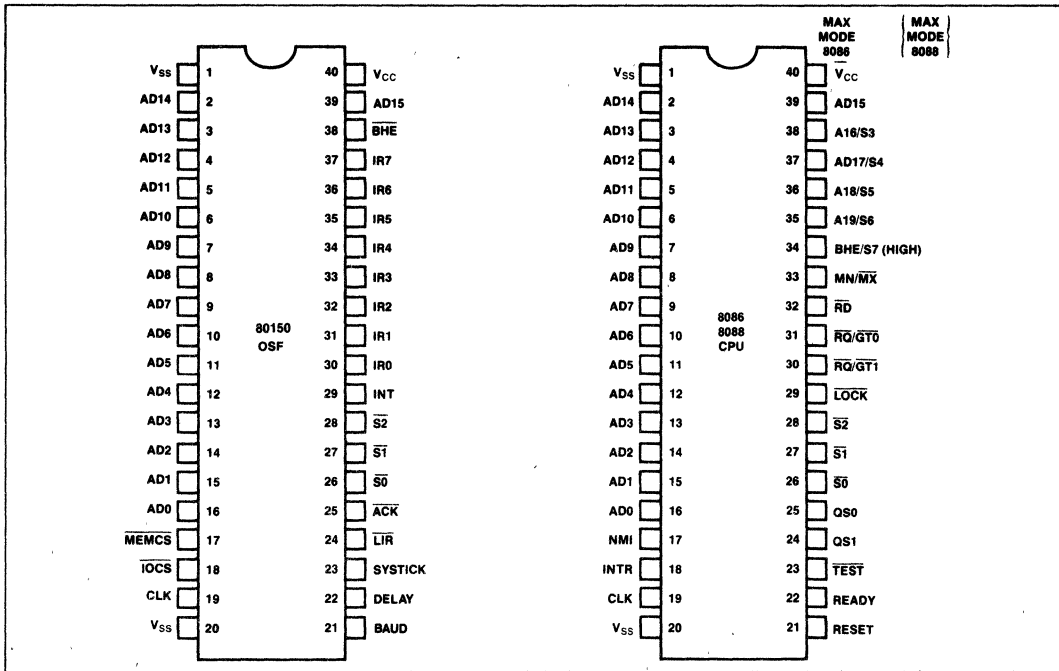


Figure 2. iAPX 86/50, 88/50 Pin Configuration

Table 1. 80150 Pin Description

Symbol	Type	Name and Function																																
AD <sub>15</sub> -AD <sub>0</sub>	I/O	<b>Address Data:</b> These pins constitute the time multiplexed memory address (T <sub>1</sub> ) and data (T <sub>2</sub> , T <sub>3</sub> , T <sub>W</sub> , T <sub>4</sub> ) bus. These lines are active HIGH. The address presented during T <sub>1</sub> of a bus cycle will be latched internally and interpreted as an 80150 internal address if MEMCS or IOCS is active for the invoked primitives. The 80150 pins float whenever it is not chip selected, and drive these pins only during T <sub>2</sub> - T <sub>4</sub> of a read cycle and T <sub>1</sub> of an INTA cycle.																																
BHE/S <sub>7</sub>	I	<b>Bus High Enable:</b> The 80150 uses the BHE signal from the processor to determine whether to respond with data on the upper or lower data pins, or both. The signal is active LOW. BHE is latched by the 80150 on the trailing edge of ALE. It controls the 80150 output data as shown. <table style="margin-left: 40px;"> <tr> <td>BHE</td> <td>A<sub>0</sub></td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>Word on AD<sub>15</sub>-AD<sub>0</sub></td> </tr> <tr> <td>0</td> <td>1</td> <td>Upper byte on AD<sub>15</sub> - AD<sub>8</sub></td> </tr> <tr> <td>1</td> <td>0</td> <td>Lower byte on AD<sub>7</sub>-AD<sub>0</sub></td> </tr> <tr> <td>1</td> <td>1</td> <td>Upper byte on AD<sub>7</sub>-AD<sub>0</sub></td> </tr> </table>	BHE	A <sub>0</sub>		0	0	Word on AD <sub>15</sub> -AD <sub>0</sub>	0	1	Upper byte on AD <sub>15</sub> - AD <sub>8</sub>	1	0	Lower byte on AD <sub>7</sub> -AD <sub>0</sub>	1	1	Upper byte on AD <sub>7</sub> -AD <sub>0</sub>																	
BHE	A <sub>0</sub>																																	
0	0	Word on AD <sub>15</sub> -AD <sub>0</sub>																																
0	1	Upper byte on AD <sub>15</sub> - AD <sub>8</sub>																																
1	0	Lower byte on AD <sub>7</sub> -AD <sub>0</sub>																																
1	1	Upper byte on AD <sub>7</sub> -AD <sub>0</sub>																																
S <sub>2</sub> , S <sub>1</sub> , S <sub>0</sub>	I	<b>Status:</b> For the 80150, the status pins are used as inputs only. 80150 encoding follows: <table style="margin-left: 40px;"> <tr> <td>S<sub>2</sub></td> <td>S<sub>1</sub></td> <td>S<sub>0</sub></td> <td></td> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> <td>INTA</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>IORD</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>IOWR</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Passive</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>Instruction fetch</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>MEMRD</td> </tr> <tr> <td>1</td> <td>1</td> <td>X</td> <td>Passive</td> </tr> </table>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>		0	0	0	INTA	0	0	1	IORD	0	1	0	IOWR	0	1	1	Passive	1	0	0	Instruction fetch	1	0	1	MEMRD	1	1	X	Passive
S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>																																
0	0	0	INTA																															
0	0	1	IORD																															
0	1	0	IOWR																															
0	1	1	Passive																															
1	0	0	Instruction fetch																															
1	0	1	MEMRD																															
1	1	X	Passive																															

**Table 1. 80150 Pin Description (Continued)**

Symbol	Type	Name and Function																																																						
CLK	I	<b>Clock:</b> The system clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing. The 80130 uses the system clock as an input to the SYSTICK and BAUD timers and to synchronize operation with the host CPU.																																																						
INT	O	<b>Interrupt:</b> INT is HIGH whenever a valid interrupt request is asserted. It is normally used to interrupt the CPU by connecting it to INTR.																																																						
IR <sub>7</sub> -IR <sub>0</sub>	I	<b>Interrupt Requests:</b> An interrupt request can be generated by raising an IR input (LOW to HIGH) and holding it HIGH until it is acknowledged (Edge-Triggered Mode), or just by a HIGH level on an IR input (Level-Triggered Mode).																																																						
ACK	O	<b>Acknowledge:</b> This line is LOW whenever an 80150 resource is being accessed. It is also LOW during the first INTA cycle and second INTA cycle if the 80150 is supplying the interrupt vector information. This signal can be used as a bus ready acknowledgement and/or bus transceiver control.																																																						
MEMCS	I	<b>Memory Chip Select:</b> This input must be driven LOW when a kernel primitive is being fetched by the CPU. AD <sub>13</sub> -AD <sub>0</sub> are used to select the instruction.																																																						
IOCS	I	<p><b>Input/Output Chip Select:</b> When this input is low, during an IORD or IOWR cycle, the 80150's kernel primitives are accessing the appropriate peripheral function as specified by the following table:</p> <table border="1"> <thead> <tr> <th>BHE</th> <th>A<sub>3</sub></th> <th>A<sub>2</sub></th> <th>A<sub>1</sub></th> <th>A<sub>0</sub></th> <th></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>X</td> <td>X</td> <td>X</td> <td>X</td> <td>Passive</td> </tr> <tr> <td>X</td> <td>X</td> <td>X</td> <td>X</td> <td>1</td> <td>Passive</td> </tr> <tr> <td>X</td> <td>0</td> <td>1</td> <td>X</td> <td>X</td> <td>Passive</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>X</td> <td>0</td> <td>Interrupt Controller</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>Systick Timer</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>1</td> <td>0</td> <td>Delay Counter</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>Baud Rate Timer</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>Timer Control</td> </tr> </tbody> </table>	BHE	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>		0	X	X	X	X	Passive	X	X	X	X	1	Passive	X	0	1	X	X	Passive	1	0	0	X	0	Interrupt Controller	1	1	0	0	0	Systick Timer	1	1	0	1	0	Delay Counter	1	1	1	0	0	Baud Rate Timer	1	1	1	1	0	Timer Control
BHE	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>																																																				
0	X	X	X	X	Passive																																																			
X	X	X	X	1	Passive																																																			
X	0	1	X	X	Passive																																																			
1	0	0	X	0	Interrupt Controller																																																			
1	1	0	0	0	Systick Timer																																																			
1	1	0	1	0	Delay Counter																																																			
1	1	1	0	0	Baud Rate Timer																																																			
1	1	1	1	0	Timer Control																																																			
LIR	O	<b>Local Bus Interrupt Request:</b> This signal is LOW when the interrupt request is for a non-slave input or slave input programmed as being a local slave.																																																						
V <sub>CC</sub>		<b>Power:</b> V <sub>CC</sub> is the +5V supply pin.																																																						
V <sub>SS</sub>		<b>Ground:</b> V <sub>SS</sub> is the ground pin.																																																						
SYSTICK	O	<b>System Clock Tick:</b> Timer 0 Output.																																																						
DELAY	O	<b>DELAY Timer:</b> Output of timer 1.																																																						
BAUD	O	<b>Baud Rate Generator:</b> 8254 Mode 3 compatible output. Output of 80150 Timer 2.																																																						

The 80150 breaks new ground in operating system software-on-silicon components. It is unique because it is the first time that an industry-standard personal/small business computer operating system is being put in silicon. The 80150 contains Digital Research's CP/M-86 operating system, which is designed for Intel's line of software- and interface-compatible iAPX 86, 88, 186, and 188 microprocessors. Since the entire CP/M-86 operating system is contained on the chip, it is now possible to design a diskless computer that runs proven and commonly available applications software. The 80150 is a

true operating system extension to the host microprocessor, since it also integrates key operating system-related peripheral functions onto the chip.

### MODULAR DESIGN

Based on a proven, modular design, the system includes the:

- CCP: Console Command Processor

The CCP is the human interface to the operating system and performs decoding and



execution of user commands.

- **BDOS: Basic Disk Operating System**

The BDOS is the logical, invariant portion of the operating system; it supports a named file system with a maximum of 16 logical drives, containing up to 8 megabytes each for a potential of 128 megabytes of on-line storage.

- **BIOS: Basic Input/Output System**

The physical, variant portion of the operating system, the BIOS contains the system-dependent input/output device handlers.

### CP/M\* COMPATIBILITY

CP/M-86 files are completely compatible with CP/M for 8080- and 8085-based microcomputer systems. This simplifies the conversion of software developed under CP/M to take full advantage of iAPX 86, 88, 186, 188-based systems.

The user will notice no significant difference between CP/M and CP/M-86. Commands such as DIR, TYPE, REN, and ERA respond the same way in both systems.

CP/M-86 uses the iAPX 86, 88, 186, 188 registers corresponding to 8080 registers for system call and return parameters to further simplify software transport. The 80150 allows application code and data segments to overlap, making the mixture of code and data that often appears in CP/M applications acceptable to the iAPX 86, 88, 186, 188.

### Unique Capabilities of CP/M-86 in Silicon

1. CP/M-86 on-a-chip reduces software development required by the system designer. It can change the implementation of the operating system into the simple inclusion of the 80150 on the CPU board.

As described later, the designer can either simply incorporate the Intel chip without the need for writing even a single line of additional code, or he can add additional device drivers by writing only the small amount of additional code required.

2. The 80150 is the most cost-effective way to implement CP/M-86 in a microcomputer. The integration of CP/M-86 with the 16K bytes of system memory it requires, the two boot ROMS required in a diskette-based CP/M-86, and the on-chip peripherals (interrupt controller and timers) lead to savings in software, parts cost, board space, and interconnect wiring.

3. The reliability of the microcomputer is in-

creased significantly. Since CP/M-86 is now always in the system as a standard hardware operating system, a properly functioning system diskette is not required. CP/M-86 in hardware can no longer be overwritten accidentally by a runaway program. System reliability is enhanced by the decreased dependence on floppy disks and fewer chips and interconnections required by the highly integrated 80150.

4. The microcomputer system boots up CP/M-86 on power-on, rather than requiring the user to go through a complicated boot sequence, thus lowering the user expertise required.
5. Diskless CP/M-based systems are now easy to design. Since CP/M is already in the microcomputer hardware, there is no need for a disk drive in the system if it is not desired. Without a disk drive, a system is more portable, simpler to use, less costly, and more reliable.
6. The administrative costs associated with distributing CP/M-86 are eliminated. Since CP/M-86 is now resident on the 80150 in the microcomputer system, there is no end-user licensing required nor is there any serialization requirement for the 80150 (because no CP/M diskette is used).
7. End-users will value having their CP/M operating system resident in their computer rather than on a diskette. They will no longer have to back up the operating system or have a diskette working properly to bring the system up in CP/M, increasing their confidence in the integrity, reliability, and usability of the system.

### 80150 FUNCTIONAL DESCRIPTION

The 80150 is a processor extension that is fully compatible with the 8086, 8088, 80186, and 80188 microprocessors. When the 80150 is combined with the microprocessor, the two-chip set is called an Operating System Processor and is denoted as the iAPX 86/50, 88/50, 186/50, or 188/50. The basic system configuration is shown in Figure 1. The 80150 connects directly to the multiplexed address/data bus and runs up to 8 MHz without wait states.

- A. **Hardware.** Figure 3 is a functional diagram of the 80150 itself. CP/M-86 is stored in the 16K-bytes of control store. The timers are compatible with the standard 8254 timer. The interrupt controller, with its eight programmable interrupt inputs and one interrupt output, is compatible with the 8259A Programmable Interrupt Controller. External slave 8259A Inter-

\*CP/M is a registered trademark of Digital Research, Inc.

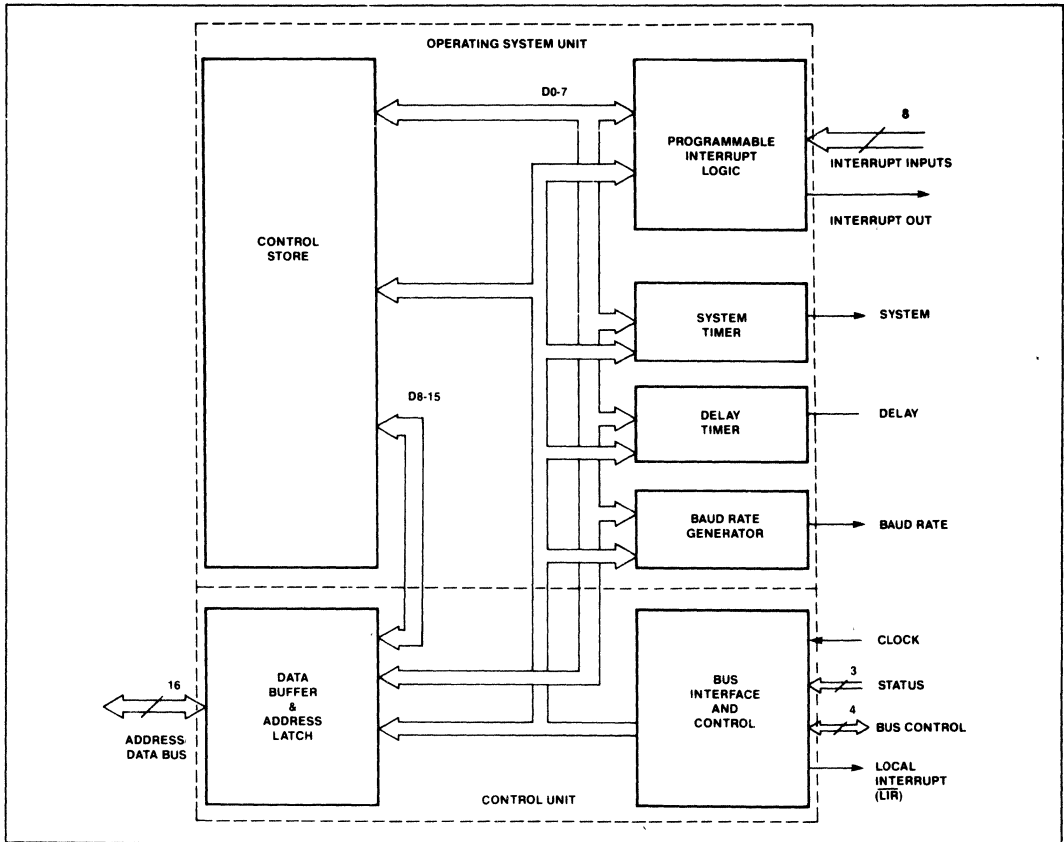


Figure 3. 80150 Internal Block Diagram

rupt controllers can be cascaded with the 80150 to expand the total number of interrupts to 57.

- B. Software. Digital Research's version 1.1 of CP/M-86 forms the basis of the 80150. CP/M consists of three major parts: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input/Output System (BIOS). Details on CP/M-86 are provided in Digital Research's *CP/M-86 Operating System User's Guide* and *CP/M-86 Operating System System Guide*.

### CCP - Console Command Processor

The CCP provides all of the capabilities provided by Digital Research's CCP. Built-in commands have been expanded to include capabilities normally included as transient utilities on the Digital Research CP/M-86 diskette. Commands are pro-

vided to format diskettes, transfer files between devices (based on Digital Research's Peripheral Interchange Program PIP), and alter and display I/O device and file status (based on Digital Research's STAT).

Through User Intervention Points, the standard CP/M-86 CCP is enhanced to allow the user to add new built-in commands to further customize a CP/M-86 system.

### BDOS - Basic Disk Operating System

Once the CCP has parsed a command, it sends it to the BDOS, which performs system services such as managing disk directories and files. Some of the standard BDOS functions provide:

- Console Status
- Console Input and Output
- List Output
- Select Drive
- Set Track and Sector

**Read/Write Sector Load Program**

The BDOS in the 80150 provides the same functions as the standard Digital Research CP/M-86 BDOS.

**BIOS - Basic Input/Output System**

The BIOS contains the system-dependent I/O drivers. The 80150 BIOS offers two fundamental configuration options:

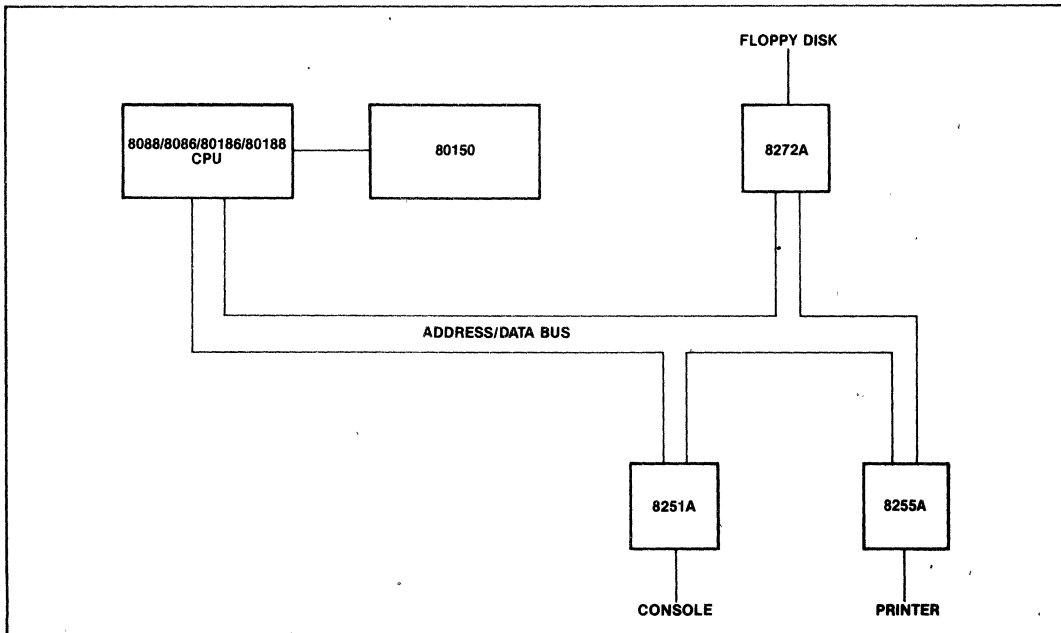
1. A **predefined configuration** which supports minimum cost CP/M-86 microcomputer systems and which requires no operating system development by the system designer.
2. An **OEM-configurable mode**, where the designer can choose among several drivers of

ferred on the 80150 or substitute or add any additional device drivers of his choice.

These two options negate the potential software-on-silicon pitfall of inflexibility in system design. The OEM can customize the end system as desired.

The **predefined configuration** offers a choice among several peripheral chip drivers included on the 80150. Drivers for the following chips are included in the 80150 BIOS:

8251A	Universal Synchronous/Asynchronous Receiver/Transmitter (USART)
8274	Multi-Protocol Serial Controller (MPSC)
8255A	Programmable Parallel Interface (PPI)
8272A	Floppy Disk Controller



**Figure 4. Predefined Configuration**

Even in the predefined configuration, the system designer (or end user, if the system designer desires) may select parameters such as the baud rates for the console and printer, and the floppy disk size (standard 8" or 5 1/4" mini-floppy) and format (FM single density or MFM double density, single-sided or double-sided).

Drivers for the 80150 on-chip timers and interrupt controller are also included in the BIOS.

The 80150 takes advantage of the 80186 and 80188 on-chip peripherals in an iAPX 186/50 or 188/50 system. For example, the integrated DMA controller is used. Also fully utilized are the integrated memory chip selects and I/O chip selects.

Since all microcomputer configurations cannot be anticipated, the **OEM-configurable mode** allows the system designer to use any set of peripheral chips desired. This configuration is shown in Figure 5.

By simply changing the jump addresses in a configuration table, the designer can also gain the flexibility of adding custom BIOS drivers for other

peripheral chips, such as bubble memories or more complex CRT controllers. These drivers would be stored in memory external to the 80150 itself. By providing the configurability option, the 80150 is applicable to a far broader range of designs that it would be with an inflexible BIOS.

### MEMORY ORGANIZATION

When using the **predefined configuration** of the 80150 BIOS, the 80150 must be placed in the top 16K of the address space of the microprocessor (starting at location FC000H) so that the 80150 gains control when the microprocessor is reset. Upon receipt of control, the 80150 writes a **configuration block** into the bottom of the microprocessor's address space, which must be in RAM. The 80150 uses the area after the interrupt vectors for system configuration information and scratch-pad storage.

When using the **OEM-configurable mode** of the 80150 BIOS, the 80150 is placed on any 16K bound-

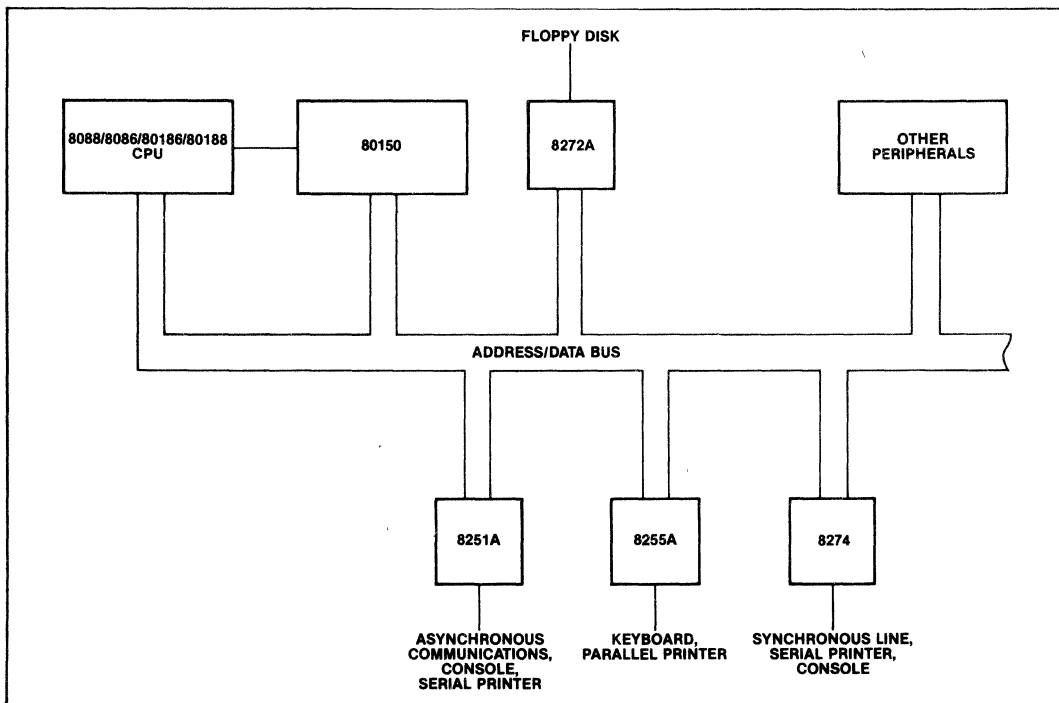


Figure 5. OEM Configurable System

dary of memory **except** the highest (FC000H) or lowest (00000H). The user writes interface code (in the form of a simple boot ROM) to incorporate and link additional features and changes into the standard 80150 environment. The configuration block may be located as desired in the address space, and its size may vary widely depending on the application.

### Memory Disk

A unique capability offered by the 80150 is the Memory Disk. The Memory Disk consists of a block of RAM whose size can be selected by the designer. The Memory Disk is treated by the BDOS as any standard floppy disk, and is one of the 16 disks that CP/M can address. Thus files can be opened and closed, programs stored, and statistics gathered on the amount of Memory Disk space left.

The Memory Disk opens the possibility of a portable low-cost diskless microcomputer or network station. Applications software can be provided in

a number of ways:

- a. telephone lines via a modem.
- b. ROM-based software.
- c. a network.
- d. bubble memory based software.
- e. low-cost cassettes.

### TYPICAL SYSTEM CONFIGURATION

Figure 6 shows the processing cluster of a "typical" iAPX 86/50 or iAPX 88/50 OSP system. Not shown are subsystems likely to vary with the application. The configuration includes an 8086 (or 8088) operating in maximum mode, an 8284A clock generator and an 8288 system controller. Note that the 80150 is located on the CPU side of any latches or transceivers.

### Timers

The Timers are connected to the lower half of the data bus and are addressed at even addresses. The timers are read as two successive bytes,

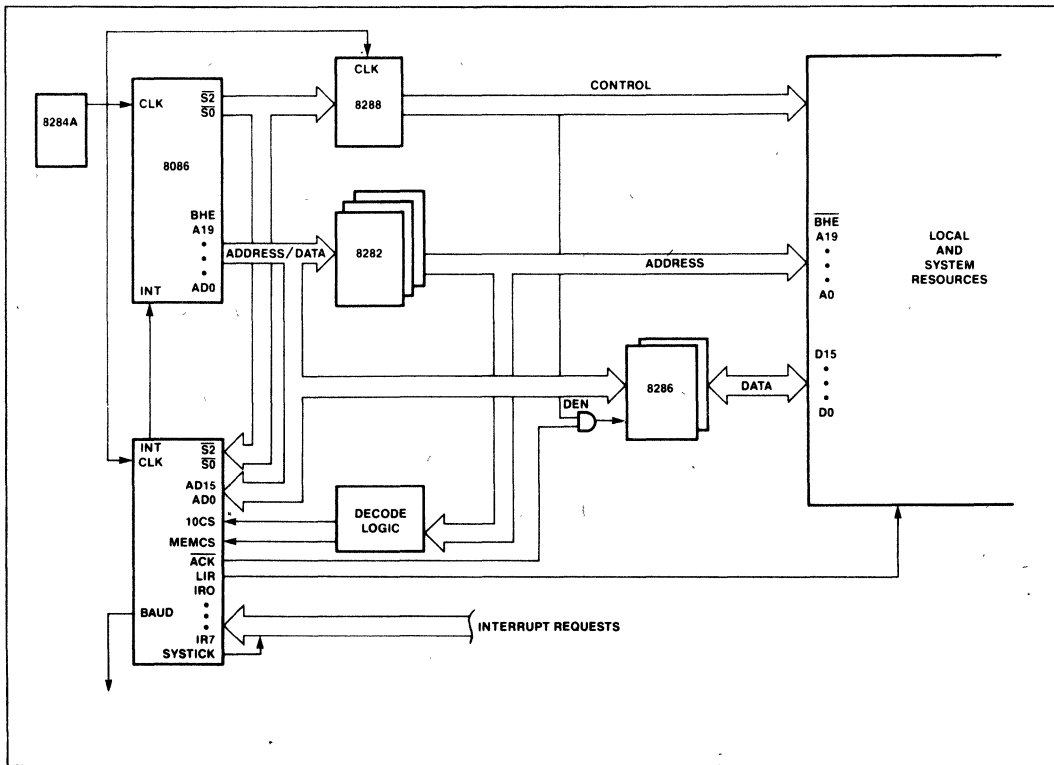


Figure 6. Typical OSP Configuration  
6-30

always LSB followed by MSB. The MSB is always latched on a read operation and remains latched until read. Timers are not gatable. An external 8254 Programmable Interval Timer may be added to the system.

### Baud Rate Generator

The baud rate generator operates like an 8254 (square wave mode 3). Its output, BAUD, is initially high and remains high until the Count Register is loaded. The first falling edge of the clock after the Count Register is loaded causes the transfer of the internal counter to the Count Register. The output stays high for  $N/2$  [( $N + 1$ )/2 if  $N$  is odd] and then goes low for  $N/2$  [( $N - 1$ )/2 if  $N$  is odd]. On the falling edge of the clock which signifies the final count for the output in low state, the output returns to high state and the Count Register is transferred to the internal counter. The baud rates can vary from 300 to 9600 baud.

The baud rate generator is located at 0CH (12), relative to the 16-byte boundary in the I/O space in which the 80150 component is located. The timer control word is located at relative address, 0EH(14). Timers are addressed with IOCS=0. Timers 0 and 1 are assigned to use by the OSP, and should not be altered by the user.

The 80150 timers are subset compatible with 8254 timers.

### Interrupt Controller

The Programmable Interrupt Controller (PIC), is also an integral unit of the 80150. Its eight input pins handle eight vectored priority interrupts. One of these pins must be used for the SYSTICK time function in timing waits, using an external connection as shown. During the 80150 initialization and configuration sequence, each 80150 interrupt pin is individually programmed as either level or edge sensitive. External slave 8259A interrupt controllers can be used to expand the total number of interrupts to 57.

In addition to standard PIC functions, the 80150 PIC unit has an LIR output signal, which when low indicates an interrupt acknowledge cycle.  $\overline{\text{LIR}} = 0$  is provided to control the 8289 Bus Arbiter SYSB/RESB pin. This will avoid the need of requesting the system bus to acknowledge local bus non-slave interrupts. The user defines the interrupt system as part of the configuration.

### INTERRUPT SEQUENCE

The interrupt sequence is as follows:

1. One or more of the interrupts is set by a low-to-high transition on edge-sensitive IR inputs or by a high input on level-sensitive IR inputs.
2. The 80150 evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an interrupt acknowledge cycle which is encoded in  $\overline{\text{S}}_2 - \overline{\text{S}}_0$ .
4. Upon receiving the first interrupt acknowledge from the CPU, the highest-priority interrupt is set by the 80150 and the corresponding edge detect latch is reset. The 80150 does not drive the address/data bus during this bus cycle but does acknowledge the cycle by making  $\overline{\text{ACK}} = 0$  and sending the LIR value for the IR input being acknowledged.
5. The CPU will then initiate a second interrupt acknowledge cycle. During this cycle, the 80150 will supply the cascade address of the interrupting input at  $T_1$  on the bus and also release an 8-bit pointer onto the bus if appropriate, where it is read by the CPU. If the 80150 does supply the pointer, then  $\overline{\text{ACK}}$  will be low for the cycle. This cycle also has the value LIR for the IR input being acknowledged.
6. This completes the interrupt cycle. The ISR bit remains set until an appropriate EXIT INTERRUPT primitive (EOI command) is called at the end of the Interrupt Handler.

**ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to 150°C  
 Voltage on Any Pin With Respect to Ground ..... -1.0V to +7V  
 Power Dissipation ..... 1.0 Watts

*\*NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.*

**D.C. CHARACTERISTICS** (T<sub>A</sub> = 0°C to 70°C, V<sub>CC</sub> = 4.5 to 5.5V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V <sub>IL</sub>	Input Low Voltage	-0.5	0.8	V	
V <sub>IH</sub>	Input High Voltage	2.0	V <sub>CC</sub> + .5	V	
V <sub>OL</sub>	Output Low Voltage		0.45	V	I <sub>OL</sub> = 2mA
V <sub>OH</sub>	Output High Voltage	2.4		V	I <sub>OH</sub> = -400µA
I <sub>CC</sub>	Power Supply Current		200	mA	T <sub>A</sub> = 25 C
I <sub>LI</sub>	Input Leakage Current		10	µA	0 < V <sub>IN</sub> < V <sub>CC</sub>
I <sub>LR</sub>	IR Input Load Current		10 -300	µA µA	V <sub>IN</sub> = V <sub>CC</sub> V <sub>IN</sub> = 0
I <sub>LO</sub>	Output Leakage Current		10	µA	.45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>
V <sub>CLI</sub>	Clock Input Low		0.6	V	
V <sub>CHI</sub>	Clock Input High	3.9		V	
C <sub>IN</sub>	Input Capacitance		10	pF	
C <sub>IO</sub>	I/O Capacitance		15	pF	
I <sub>CLI</sub>	Clock Input Leakage Current		10 150 10	µA µA µA	V <sub>IN</sub> = V <sub>CC</sub> V <sub>IN</sub> = 2.5V V <sub>IN</sub> = 0V

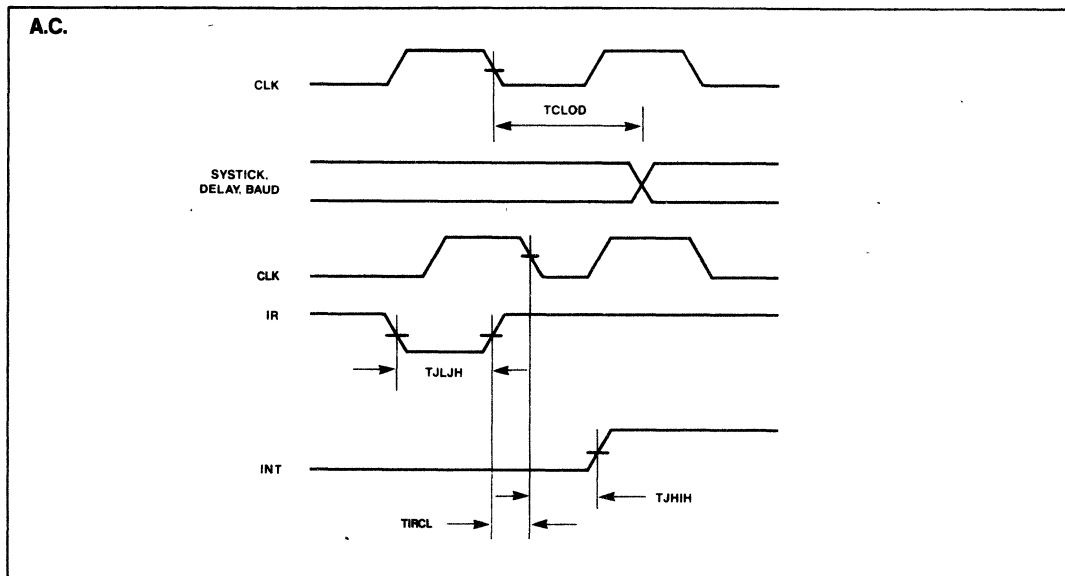
**A.C. CHARACTERISTICS** (T<sub>A</sub> = 0-70°C, V<sub>CC</sub> = 4.5-5.5 Volt, V<sub>SS</sub> = Ground)

Symbol	Parameter	80150		80150-2		Units	Test Conditions
		Min.	Max.	Min.	Max.		
T <sub>CLCL</sub>	CLK Cycle Period	200	-	125	-	ns	
T <sub>CLCH</sub>	CLK Low Time	90	-	55	-	ns	
T <sub>CHCL</sub>	CLK High Time	69	2000	44	2000	ns	
T <sub>SVCH</sub>	Status Active Setup Time	80	-	65	-	ns	
T <sub>CHSV</sub>	Status Inactive Hold Time	10	-	10	-	ns	
T <sub>SHCL</sub>	Status Inactive Setup Time	55	-	55	-	ns	
T <sub>CLSH</sub>	Status Active Hold Time	10	-	10	-	ns	
T <sub>ASCH</sub>	Address Valid Setup Time	8	-	8	-	ns	
T <sub>CLAH</sub>	Address Hold Time	10	-	10	-	ns	
T <sub>CSCL</sub>	Chip Select Setup Time	20	-	20	-	ns	
T <sub>CHCS</sub>	Chip Select Hold Time	0	-	0	-	ns	
T <sub>DSDL</sub>	Write Data Setup Time	80	-	60	-	ns	
T <sub>CHDH</sub>	Write Data Hold Time	10	-	10	-	ns	
T <sub>ILJH</sub>	IR Low Time	100	-	100	-	ns	
T <sub>CLDV</sub>	Read Data Valid Delay	-	140	-	105	ns	C <sub>L</sub> = 200 pF
T <sub>CLDH</sub>	Read Data Hold Time	10	-	10	-	ns	
T <sub>CLDX</sub>	Read Data to Floating	10	100	10	100	ns	
T <sub>CLCA</sub>	Cascade Address Delay Time	-	85	-	65	ns	

### A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	80150		80150-2		Units	Notes
		Min.	Max.	Min.	Max.		
$T_{CLCF}$	Cascade Address Hold Time	10	—	10	—	ns	
$T_{IAVE}$	INTA Status t Acknowledge	—	80	—	80	ns	
$T_{CHEH}$	Acknowledge Hold Time	0	—	0	—	ns	
$T_{CSAK}$	Chip Select to ACK	—	110	—	110	ns	
$T_{SACK}$	Status to ACK	—	140	—	140	ns	
$T_{AACK}$	Address to ACK	—	90	—	90	ns	
$T_{CLOD}$	Timer Output Delay Time	—	200	—	200	ns	$C_L = 100 \text{ pF}$
$T_{CLOD1}$	Timer1 Output Delay Time	—	200	—	200	ns	$C_L = 100 \text{ pF}$
$T_{JHIH}$	INT Output Delay	—	200	—	200	ns	
$T_{IRCL}$	IR Input Set Up	20	—	20	—	ns	

### WAVEFORMS







---

# User Library

7

---



## **USER LIBRARY**

The Insite User's Program Library is an Intel-sponsored software library supporting Intel microcomputer products. There are currently over 325 programs in the Library collection.

Insite offices are located in the U.S., Brussels, Paris, Germany, the U.K., and Japan, serving about 1,500 members worldwide.

As the Library collection is built on programs submitted by Intel employees as well as customers, we encourage and welcome all program contributions. These contributions are essential to the growth and success of Insite.

In the following pages you will be introduced to more in-depth information about Insite. Membership and program submittal forms, including a complete program index listing, are also included for your convenience.



## INSITE™ USER'S PROGRAM LIBRARY

- **Programs for 8048, 8051, 8080/8085, and 8086/8087/8088 Processors**
- **Accepted Program Submittals Entitle You to a Free Membership or Free Program Package**
- **Worldwide Offices to Serve You**
- **Diskettes, Paper Tapes, and Listings Available for Library Programs**
- **Program Library Catalog Offering Hundreds of Programs**
- **Updates of New Programs Sent During Subscription Period**

Insite, Intel's Software Index and Technology Exchange Library, is a varied collection of programs and routines that have been written by users of Intel microcomputers, single-board computers, and development systems. This expanding library of programs covers a broad range of software tools that includes monitors, conversion routines, peripheral drivers, translators, math packages, and even games. As a library member, you can acquire a copy of any program within the library on any of its available types of media. By taking advantage of the availability of existing library programs, numerous hours of coding and debugging time can be saved and routine or redundant programming operations can be eliminated. The Insite Program Library also serves as a learning tool for individuals unfamiliar with assembly or high-level languages associated with Intel's family of microcomputers.

**Membership.** Membership in Insite is available on an annual basis. Intel customers may become members through an accepted program contribution or paid membership fee.

**Program Submittals.** The Insite Library is built on program submittals contributed by users. Customers are encouraged to submit their programs. (Details and forms are available through the Insite Library.) For each accepted program, submitters will receive a choice of three free programs (A, B, C, or D category), or free membership with Insite for one year.

**Program Library Service.** PAPER TAPES, DISKETTES OR SOURCE LISTINGS are available for every program in Insite. Diskettes are available on single or double density. Membership is required to purchase programs.

**Insite™ Program Library Catalog.** Each member will be sent the Program Library Catalog consisting of an abstract for each program indicating the function of the routine, required hardware and software, and memory requirements.

Insite members will be updated with abstracts of new programs submitted to the Library during the subscription period. For catalog and yearly subscription fee please refer to the Intel OEM Price List or contact the nearest Insite or Intel Sales Office.

INSITE OFFICES ARE WORLDWIDE, WITH FIVE LOCATIONS TO SERVE YOU:

### **NORTH AMERICA**

Intel Corporation  
3065 Bowers Avenue  
Santa Clara, California 95051  
ATTN: Insite User's Program Library  
Telephone: 408-987-8080

### **THE ORIENT**

Intel Japan K.K.  
5-6 Tohkohdai, Toyosato-cho,  
Tsukuba-gun, Ibaraki, 300-26, Japan  
ATTN: Insite User's Program Library  
Telephone: 029747-8511

### **EUROPE**

Intel Corporation S.A.R.L.  
5 Place de la Balance  
Silic 223  
94528 Rungis Cedex, France  
ATTN: Insite User's Program Library  
Telephone: 0687-22-21

Intel Semiconductor GmbH  
Seidlstrasse 27  
8000 Muenchen 2  
West Germany  
ATTN: Insite User's Program Library  
Telephone: 089-5389-1

Intel Corporation (U.K.) Ltd.  
Pipers Way  
Swindon SN3 LRJ  
Wiltshire, England  
ATTN: Insite User's Program Library  
Telephone: 0793-488-388

---

## **SUBMITTAL REQUIREMENTS**

Programs submitted for Insite review must follow the guidelines listed below:

Programs must be written in a language capable of compilation and assembly by the currently-supported version of an Intel standard compiler/assembler. Accepted languages are documented in the following manuals available through Intel's Literature Department.

- BASIC-80 Reference Manual, Order No. 980758
- iCIS-COBOL Language Reference Manual, Order No. 980927
- FORTRAN-80 Programming Manual, Order No. 980481
- FORTRAN-86 User's Guide, Order No. 121570
- Pascal-80 User's Guide, Order No. 981015
- Pascal-86 User's Guide, Order No. 121539
- PL/M-80 Programming Manual, Order No. 980268
- PL/M-86 Programming Manual, Order No. 980466
- MCS-48 and UPI-41A Assembly Language Manual, Order No. 980255
- MCS-86 Macro Assembly Language Reference Manual, Order No. 121703
- 8080/8085 Assembly Language Programming Manual, Order No. 980940
- 8086/8087/8088 Macro Assembly Language Reference Manual for 80/85 Based Development System, Order No. 121623
- 8086/8087/8088 Macro Assembly Language Reference Manual for 80/86 Based Development System, Order No. 121703
- 8089 Assembly Language Reference Manual, Order No. 980255
- Microsoft BASIC Compiler Reference Manual, Order No. 121805
- Microsoft BASIC-80 Reference Manual, Order No. 121806
- Microsoft BASIC Reference Book, Order No. 121857
- Microsoft FORTRAN-80 Reference Manual, Order No. 121798
- Microsoft FORTRAN-80 User's Manual, Order No. 121799
- Microsoft M/Sort Reference Manual, Order No. 121809
- Microsoft Utility Software Manual, Order No. 121797

A well-documented source code furnished on an ISIS-formatted 8" diskette, CP/M-formatted 8" diskette, PDS 5 1/4" diskette, or ASCII-coded paper tape.

A source listing of the program must be included. This must be the output listing of a compilation or an assembly. No consideration will be given to incomplete programs or duplications of programs already in the Library.

A link and locate listing.

A demonstration program which assures the validity of the contributed program must be included. This must show the accurate operation of the program.

A complete submittal form.

Licensed software or copyrighted material must be accompanied by a written release from the appropriate, authorized person.



## INSITE™ USER'S PROGRAM LIBRARY SUBMITTAL FORM

**Processor**

8048  
  8051  
  8080/8085  
  8086/8087/8088  
  Other \_\_\_\_\_  
 Indicate the MDS series model the program was created on by checking the appropriate box, and identify other MDS series models the program may be compatible with.

**Program Title**

**Function**

**Required Hardware**

**Required Software**

**Input Parameters**

**Output Results**

<b>Registers Modified:</b>	<b>Programmer:</b>
<b>RAM Required:</b>	<b>Company:</b>
<b>ROM Required:</b>	<b>Address:</b>
<b>Maximum Subroutine Nesting Level:</b>	<b>City:</b>
<b>Assembler/Compiler Used:</b>	<b>State:</b>
<b>Programming Language:</b>	<b>Telephone:</b>

### ACKNOWLEDGEMENT AND AGREEMENT

To the best of my knowledge, I have the right to contribute this program material without breaching any obligation concerning nondisclosure of proprietary or confidential information of other persons or organizations. I am contributing this program material on a nonconfidential nonobligatory basis to the Insite User's Library for inclusion in its program library, and I agree that the Library may use, duplicate, modify, publish, and sell the program material without obligation or liability of any kind. The Insite User's Library may publish my name and address, as the contributor, to facilitate user inquiries pertaining to this program material.

Signature \_\_\_\_\_ Date \_\_\_\_\_



## LIST OF PROGRAMS ALPHABETICAL, BY APPLICATION

Program Title	Order No.
ADD AND SUBTRACT: BCD Numbers	CB11
ASSEMBLER: 8080 MACRO, V4.1	BF4
ASSEMBLER, CROSS: 8008 Code	BC5
ASSEMBLER, CROSS: 8048 On DG Nova	BC6
ASSEMBLER, CROSS: DEC PDP-8 or PDP-11	BC2
ASSEMBLER, CROSS: DEC PDP-11	BC3
ASSEMBLER, CROSS: DEC PDP-11	BC4
ASSEMBLER, CROSS: MCS-48	BC1
ASSEMBLER, ON-LINE	BF5
BAUD RATE: Modify	BG25
BAUD RATE: Modify Under CP/M	BG26
BIT HANDLING: 8048	BG35
BRANCH: MCS-48 Branch Table Routine	BG37
BREAKPOINT: 8089	BD15
CALCULATE: CHECKSUM	BD16
CALCULATE: Sine or Cosine Routine	CB13
CALCULATE: Square Root	CB5
CALCULATION: Least Squares Quadratic Fitting	CB3
CALCULATION: Natural Logarithm	CB4
CHANGE: Load Addresses, iAPX-86/88 Object File	BG42
CHECKBOOK	BA6
CLOCK: 8748 Clock and LCD Tachometer	BG30
CLOCK: MICRO/SYS MC1460 Real Time Clock Board Utilities	BG31
CLOCK: Real Time	BG29
COMMANDS: Meta-Programs	BG38
COMMUNICATION: DEC PDP-11 to Intel Development System	BB16
COMMUNICATION: HP Calculator with Intel Development System-800	AD1
COMMUNICATION: Intel Development System 220/230 with SDK-85, V1.0	AD4
COMMUNICATION: Intel Model 220/230 to Timesharing Computer	AD6
COMMUNICATION: Intel Model 800 to/from DEC PDP-10	AD8
COMMUNICATION: Intel Development System to/from DEC	AD10
COMMUNICATION: Intel Development System to/from Tektronix 8001	AD11
COMMUNICATION: Intel Development System Series-II with Minicomputer	AD9
COMMUNICATION: Intel Development System Series-II with PROMPT-48	AD2
COMMUNICATION: Intel Development System to PROMPT-48 or -80	AD3
COMMUNICATION: Intel System to Serial Output Device	AD14
COMMUNICATION: Intel Development System to/from Hewlett-Packard Computer	AD15
COMMUNICATION: Intel Development System to/from VAX 11	AD13
COMMUNICATION: Intel MDS-Data I/O Programmer Interface	BE8
COMMUNICATION: NDS-II to/from iPDS Running CP/M-80	AD17
COMMUNICATION: Tektronix DAS 9100 Digital Analysis System to Intel Development System	AD12
COMMUNICATION: Two Intel Series-II Development Systems	AD7
COMMUNICATION: Xerox File Transfer Facility	AD16
COMPARE: 8048 or 8049 ROMS	AE11
COMPARE: Files	BD11
COMPILER: Pascal	BF1
CONSOLE ACCESS: Input and Output for Series III	BD36
CONTROLLER: 8278 Keyboard/Display	AC3
CONTROLLER: 8292 on 8741A	AC4
CONTROLLER: Dual Floppy Disk Drive	AB11
CONTROLLER: Firmware for iSBC-589	AC7





Program Title	Order No.
CONTROLLER: PID Control Loops	AB20
CONTROLLER: PROMPT-48 Interactive	AB2
CONTROLLER: UP1-41 8-Digit LED Display	AC1
CONTROLLER: UP1-41A/42 Digital Cassette, V2.5	AC5
CONVERSION: ASCII-Decimal to/from FPAL Number	BB13
CONVERSION: ASCII Floating Point Numbers to AM9711 and Intel 8231 4 Byte FP Format	BB5
CONVERSION: ASCII to Floating Point	BB14
CONVERSION: ASCII to/from EBCDIC	BB1
CONVERSION: ASCII to/from Floating Point	BB11
CONVERSION: ASCII Code to/from Intel Floating Point	BB12
CONVERSION: Binary to BCD	BB6
CONVERSION: Binary to BCD	BB7
CONVERSION: Convert/Format/Print	BB8
CONVERSION: Decimal to/from Floating Point	BB9
CONVERSION: FORTRAN or FPAL Floating Point to/from Decimal	BB10
CONVERSION: Hex to ASCII	BB2
CONVERSION: ISIS-II to/from CP/M	BB18
CONVERSION: MCON-6800 Source Code to 8086/88 Source Code	BB3
CONVERSION: ZCON-Z80 to 8086/88 Source Converter	BB4
CONVERT: Doubleword to ASCII String	BB22
CONVERT: Fixed Point to Floating Point	BB21
COPY: Disk	BG28
COPY: Diskette	BG27
COPY: Diskette	BG43
COPY: iPDS CP/M-80 Diskette	BG45
COPY: PDP-11 Disk File to Intel ISIS-II Disk File	BB15
COUNT: ICE-80 Machine Cycles	BD10
COUNT: Program Usage	BG40
CREDIT: Tutorial	E6
CREDIT: Used on Modified Hazeltine 1500	BG33
DEBUG: CAT.88 (iRMX88 Task Debugger)	BD34
DEMO: 208	AE7
DEMO: iAPX-88	AE13
DEMO: iRMX 86 Multitasking Spectrum Analysis	AE8
DEMO SOFTWARE: 8275	AE6
DEVICE, I/O: UPI-41A Combination	AC2
DIAGNOSTIC: 8080 I/O	AE2
DIAGNOSTIC: Microcomputer Development System 230	AE9
DISASM	BD6
DISASSEMBLER: 8048 Object Code	BD8
DISASSEMBLER: 8080 Code	BD1
DISASSEMBLER: 8080 Code	BD4
DISASSEMBLER: 8080 Object Code	BD2
DISASSEMBLER: ICE-80 Ver 2.1	BD3
DISASSEMBLER: ISIS-II Object Files	BD5
DIVISION: 32-Bit by 16-Bit	CB12
DOWNLOAD: iPDS to Serial Port	AD18
DRIVER: 8048 Seven-Segment Display	AB5
DRIVER: 8085 Serial I/O	AB1
DRIVER: Audio Cassette Recorder	AB6
DRIVER: Bios and Boot Program for CP/M-80	AB22
DRIVER: Cassette Operating System	AB7
DRIVER: Dumb Terminal Simulator	AB10
DRIVER: Intellec Development System Series-II as Dumb Terminal	AB9
DRIVER: iPDS Dumb Terminal	AB23



Program Title	Order No.
DRIVER: iSBC 86/12 Real Time Clock Driver	AB19
DRIVER: PROM Programmer	BE7
DRIVER: RMX-80, for the iSBC 254 Bubble Memory with 80/10 Board	AB14
DRIVER: RMX-80, for the iSBC 254 Bubble Memory with 80/20/30 Board	AB15
DRIVER: RMX-86, for the iSBC 254 Bubble Memory Board	AB16
DRIVER: RMX-80 for iSBC 534	AB12
DRIVER: RMX-80 for SBC 215 Controller Board	AB13
DRIVER: RMX-86, for the iPAB-128, iPAB-256, iSBX-251 Bubble Memory Products	AB17
DRIVER: RMX-86, High Performance Driver for iSBC-550 Ethernet Communications Controller	AB18
DRIVER: SYCOR 135 Cassette Operating System	AB8
DRIVER: Tektronix 4010 Graphic Screen	AB3
DRIVER: T.I. Omni 810 Lineprinter	AB4
DRIVER: USART for iSBC-86/XX	AB21
DUMP: Diskette	BD27
DUMP: Diskette File	BD28
DUMP: Diskette File	BD26
DUMP: iAPX-86/88 Absolute Object File	BD30
DUMP: iSBC 86/12 Memory	BD29
DUMP: Symbol Table	BD21
EDIT: Disk	BD33
EDIT: Hex File	BD31
EDIT: Inspect and Change File	BD32
EDIT: Text	BA4
EDITOR: Text, Intel X111	BA3
EXECUTIVE: Real Time	AA8
EXERCISE: Data Translation MULTIBUS Analog I/O Boards	BE6
FIFO	BG13
FIFO	BG12
GAME: Bandit	D3
GAME: Black Box	D15
GAME: Breakout	D13
GAME: Craps	D5
GAME: Darts	D6
GAME: Fruit Machine	D4
GAME: Hangman	D7
GAME: Mastermind	D9
GAME: Maze	D2
GAME: Maze	D1
GAME: Othello	D10
GAME: Poker	D14
GAME: Slalom, V1.4	D8
GAME: Tiny Chess 86	D12
GENERATE: 16-Bit Random Number	CB2
GENERATE: Calendar	BA8
GENERATE: CCITT Cyclic Redundancy Check	BD37
GENERATE: Disk Directory Library	BA15
GENERATE: Fast Generation of IBM Bi-Sync CRC16	BD20
GENERATE: Graph	CB7
GENERATE: High and Low Bytes from 8086 Hex File	BD35
GENERATE: Histogram	CB8
GENERATE: IBM Bi-Sync CRC16	BD19
GENERATE: Music for SDK-85	D11
GENERATE: Output Signal	BG5



Program Title	Order No.
GENERATE: PL/M Cross Reference	BD25
GENERATE: PROM Checksum Calculation	BD18
GENERATE: Public Symbol Cross Reference	BD38
GENERATE: Random Number	CB6
GENERATE: Software Documentation	BA14
GENERATE: Stochastic Variates and Histograms	CA23
GENERATE: Symbol List	BD24
GENERATE: Symbol Table for BASIC-80	BD23
GENERATE: Tabs	BA16
GENERATE: X-Y Graph	CB9
HANDLER: RMX/80 Minimal Terminal	BE2
INCREMENT: Program Counter	BG39
INITIALIZE: Baud Rate	BG24
INITIALIZE: Baud Rate	BG23
INTERPRETER: 8086 Tiny BASIC	BF9
INTERPRETER: Interactive 8087 Instruction Interpreter	AA12
INTERPRETER: LISP	BF3
INTERPRETER: LLL BASIC-II	BF7
INTERPRETER: LLL/Chernack BASIC	BF8
INTERPRETER: MCS-51 Tiny BASIC, V2.2	BF10
INTERPRETER: PILOT-80	BF2
INTERPRETER: RMX/80 Command Line	BG4
INTERPRETER: Single-Step	BD7
LINKAGE: Series III i8087 Linkage Modules	BG36
LIST: Directory, ISIS Diskette/NDS Disk	BG18
LIST: Diskette Directory	BG17
LIST: File	BG15
LIST: File	BG16
LIST: File Errors	BD12
LIST: PL/M Compiler Errors	BD13
LIST/PRINT/TYPE	BG14
LIST: Save Error	BD14
LOAD/SAVE: RAM	BG1
MACROS: Block Structures	BG10
MACROS: Block Structures	BG11
MAIL LIST	BA9
MAIL LIST	BA11
MAIL LISTS FOR BASIC 80	BA12
MATH PACKAGE 8231	CA17
MATH PACKAGE 8051	CA18
MATH PACKAGE: 8080/8085 Fundamental Support Package	CA20
MATH PACKAGE: 8231 Arithmetic Processing Unit	CA16
MATH PACKAGE: Arithmetic Functions	CA11
MATH PACKAGE: Arithmetic Functions for MCS-48	CA22
MATH PACKAGE: Double Precision Floating Point	CA12
MATH PACKAGE: Double Precision Integer	CA4
MATH PACKAGE: Fixed and Floating Point	CA5
MATH PACKAGE: Floating Point	CA2
MATH PACKAGE: Floating Point	CA1
MATH PACKAGE: Floating Point	CA7
MATH PACKAGE: Floating Point	CA6
MATH PACKAGE: Floating Point Library/8086	CA13
MATH PACKAGE: Floating Point Utilities for FPAL.LIB	CA8



Program Title	Order No.
MATH PACKAGE: High Speed Binary Math Package for 8031/8051	CA21
MATH PACKAGE: Multiple Precision Arithmetic/8086	CA14
MATH PACKAGE: Multiply/Divide	CA15
MATH PACKAGE: Optimized Floating Point	CA9
MATH PACKAGE: Optimized Floating Point	CA10
MATH PACKAGE: PL/M Multiple Precision	CA3
MATH PACKAGE: Recursive Computation of Mean and Standard Deviation	CA19
MERGE: Mailing List	BA10
MONITOR: Intellec 8/MOD80	AA1
MONITOR: Bubble Memory Development Software for Intel BPK-72	AA10
MONITOR: HSE-49 Expansion Monitor	AA13
MONITOR: Intellec Development System, V2.0	AA6
MONITOR: iSBC 250 1-Megabit Bubble Memory	AA9
MONITOR: iSBC 254 Bubble Memory Board Monitor	AA11
MONITOR: iSBC 544	AA7
MONITOR: iSBC 80/05 or 80/04	AA14
MONITOR: iSBC 80/10	AA15
MONITOR: iSBC 80/10 or 80/10A	AA16
MONITOR: iSBC 80/20 or 80/20-4	AA17
MONITOR: iSBC 80/24	AA18
MONITOR: iSBC 80/30	AA19
MONITOR: iSBC 86/12	AA2
MONITOR: SDK-85, V2.0	AA3
MONITOR: SDK-86 Keypad	AA5
MONITOR: SDK-85 Serial, V1.1	AA4
MONITOR: Super Monitor 80	AA20
MONITOR: Super Monitor 86	AA21
MONITOR: Super Monitor 86 for the iSBC 88/45	AA22
MORSE CODE TUTOR V2.0	E3
MULTIPLICATION: 8748 BCD	CB10
MULTIPLICATION: 40-Bit	CB14
PRINT: Cover Page	BA1
PRINT: Discounted Cash Flow	BA7
PRINT: File	BA17
PRINT: Files	BA18
PRINT: Files	BA19
PRINT: High Speed Utility	BG32
PROCEDURE: Pascal 86 Screen/Cursor Control	BG34
PROCEDURE: PL/M DOCASE	BG9
PROCEDURES: PL/M Output	BG8
PROCEDURES: PL/M Utilities	BG7
PROCESSOR: Macro	BF6
PROCESSOR: Text	BA5
PROGRAM: 8741A as iSBC 941	AC6
PROGRAMMER: EPROM-8755A	BE5
PROGRAMMER: EPROMS 2708/16/32	BE4
READ: Paper Tape to SDK-85 RAM	BE3
RECEIVE	AD5
RECOVER: Diskette	BG2
RECOVERY: Diskette File	BA2
RELOCATE	BG41
REPORT: Status of Exported Job	BG44
SIMULATE: iACX-96	BD40
SIMULATOR: 8048/49 Code, V1.3	BB19



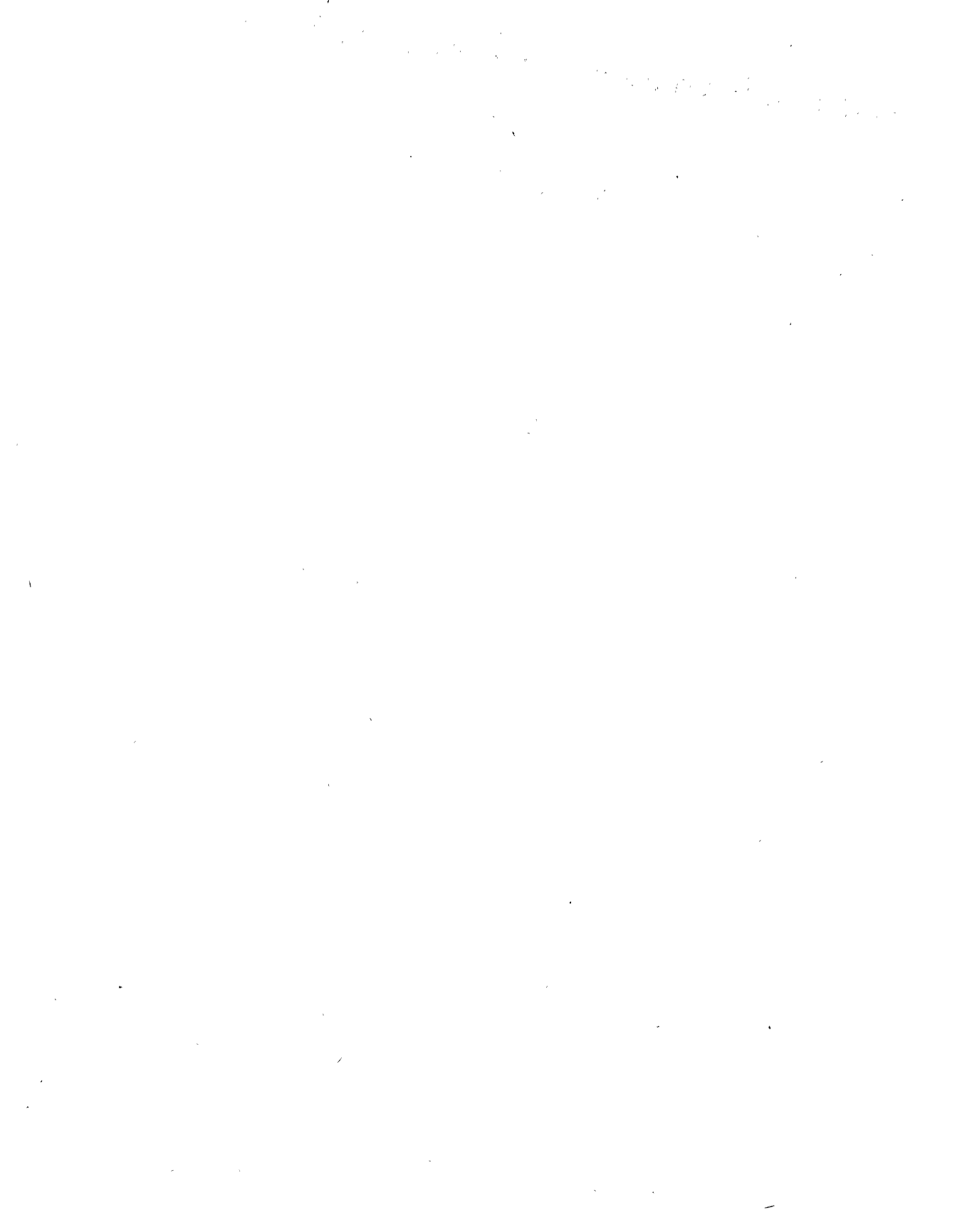
Program Title	Order No.
SIMULATOR: 8048/49 Simulator	BB20
SORT: Bubble Sort and Binary Search Routines	BG22
SORT: Disk Directory	BG19
SORT: Disk Directory	BG20
SORT: Diskette File	BG21
SORT: General	BA13
SORT: Public Symbols	BD39
SORT: Symbol Table from an Absolute File	BD22
SOURCE FILES: iAPX-86/88 System Workshop Summary and Review	E1
SOURCE FILES: MCS-80/85 System Workshop Summary and Review	E2
SPELL	BA21
SUBMIT: ISIS Command String	BG6
TEST: 8080 CPU	AE1
TEST: iSBC 80/10 I/O Ports	AE3
TEST: Error Correcting Code	AE12
TEST: MCS-48 Family CPU	AE10
TEST: Memory	AE5
TEST: Memory	AE4
TEST: PROM/ROM Checksum Self-Test	BD17
THERMOMETER: Thermistor Controlled	BE1
TRACE: ICE-80	BD9
TRANSFORM: Discrete Fourier	CB1
UTILITIES: Circular Lists	BG3
UTILITIES: Menu	E5
UTILITIES: RT11 Diskette Utility for Intellec 800	BB17
UTILITIES: Talk	E4
WORD PROCESSOR	BA20

---

# Appendix

# A

---



## INTEL SOFTWARE STANDARDS

Intel's software is built on standards which facilitate software portability and provide an open system for software.

Intel's software utilizes the emerging standards in graphics, networking, database and portable operating system interfaces. This base system software provides a mapping from architectural and operating system dependencies to a standard interface. High-level applications are built from the standard interfaces and remain portable across multiple configurations and operating systems. Figure 1 illustrates the open software model relationship.

**OPEN SOFTWARE MODEL**

<b>Applications</b>	Languages R & L	Word Processing	Spread Sheet	Mail & Document Filling	Business Graphics	Project Management
<b>Base System Software</b>	UDI	Language Run-Time	LAN	Graphics	Database	
<b>Operating Systems</b>	RMX, Xenix CP/M, MS-DOS					
<b>Micro- processors</b>	86, 186, 286, 386					

**Figure 1**

Intel has supported its fundamental software across multiple operating systems through the UDI operating system interface. By writing software to use the UDI interface which provides memory management, I/O routines, and exception handling—Intel is able to port high level languages, language run-time, and fundamental software to a new release or new operating system in minimal time. Thus Intel's software is operating system independent.

Intel's local area networking products use the IEEE 802.3 CSMA/CD Access Method and physical layer. The work for this standard was done jointly by Intel, DEC, and Xerox and is commonly known as the Ethernet protocol. The transport layer uses the proposed ISO transport protocol specification.





Intel supports the ANSI graphics standardization effort and will offer products which utilize these standards. The Virtual Device Interface (VDI) standard developed by the ANSI X3H3 committee is intended to provide a single standard which supports multiple graphics devices with the same set of graphics functions. A companion standard Virtual Device Metafile (VDM) will give a means of storing or transmitting pictures as streams of VDI functions. A third graphics standard being supported by Intel is the North American Presentation Level Protocol Syntax (NAPLPS). NAPLPS is suited for raster-scan display (both CRT and hardcopy) and is currently in final approval stage by ANSI.

Intel's Pascal and FORTRAN adhere to the ANSI standard and support optional extensions. All Intel languages (ASM-86, PL/M, FORTRAN, and Pascal) use common data types and parameter passing conventions to allow inter-language calls. The real number data types in these languages utilize the IEEE real math standard and use the numerics coprocessor or an emulator to support the real math operations and functions. The object module formats (OMF) are commonly used by all of the 86 language products as well as many language products supplied by independent software vendors.

### INTEL SOFTWARE STANDARDS DOCUMENTS

Standard	Document
UDI	Run-Time Support Manual For iAPX 86, 88 Applications, Appendix A (Intel 121776-002)
NAPLPS	Intel's Guide To Understanding The ANSI Videotex Presentation Level Protocol (Intel 145412-001)
VDM, VDI	Draft of proposed American Standard for the Virtual Device Metafile, ANSI X3H33 Virtual Device Interface Task Group
Local Area Network	
— Data link and physical layer (Ethernet)	Draft IEEE Standard 802.3 CSMA/CD Access Method and Physical Layer Specifications, IEEE Computer Society
— Transport layer	ISO draft proposal 8073 Information Processing Systems, Open Systems Interconnection-Connection Oriented Transport Protocol Specification
FORTRAN 77	Intel's extension to ANSI FORTRAN 77 specified in FORTRAN-86 User's Guide (Intel 121570-002)
Pascal	Intel's extension to ANSI Pascal specified in Pascal-86 User's Guide (Intel 121539-001)
Real Math	Draft 10.0 of IEEE Task P754, December 1982. 8087 Support Library Reference Manual (Intel 121725-001).
Language Data Types and Parameter Passing	
— Pascal	Pascal-86 User's Guide, Appendix J (Intel 121539-003)
— FORTRAN	FORTRAN-86 User's Guide, Appendix H (Intel 121570-002)
Object Module Formats	
— 86	8086 Relocatable Object Module Formats (Intel 121748-001)
— 286	The Concrete Representation of 80286 Object Modules, Intel internal document iAPX 286 Compilers Writer's Guide, (Intel-in preparation)

---

# Appendix

# B

---





# SOFTWARE SUPPORT SERVICES

## A FULL SERVICE SUPPORT PROGRAM

Intel's Software Support Services is a comprehensive range of post-sales support programs for software and systems purchased from Intel. Its objectives are to maximize the system's performance and minimize unnecessary downtime for greater productivity. These services are provided for all Intel developed and most Intel marketed third-party software.

## DESCRIPTION OF SERVICES

Software support services will be available for the Customer as follows:

### Initial Support

Initial support provides each Intel system and licensed product with 90-day support after product delivery. It includes automatic updates and new releases, Software Performance Reporting (SPR) Service, technical reports and monthly technical bulletin. Also available on designated products will be telephone assistance for product specific technical information and assistance with work-arounds, patches, and other solutions for Intel defined product deficiencies.

### SUBSCRIPTION SERVICE (available for individual software products)

#### 1. Technical Reports

A technical report will be published quarterly for active products and semi-annually for mature products. This will contain a Configuration and Compatibility Guide, a product performance exceptions list providing solutions to known problems and a review of important current Software Performance Reports (SPR's) submitted by customers. A listing of product manuals available from Intel is also provided.

#### 2. Software Performance Reporting Service (SPR)

Intel will respond to written questions (submitted on a standard SPR form) on product-specific software, system, or documentation issues. Intel will verify receipt of the SPR within 48 hours and will respond within 3 weeks. Intel does not guarantee a resolution will always be available to specific problems.

## SOFTWARE INFORMATION SERVICE

This service provides the Customer with direct communication with an Intel Software Support Engineer (SSE). The Customer may call a single service number (U.S.) between 8:00 A.M. and 5:00 P.M., (Pacific Time) for product-specific inquiries.

This service enables the Customer to:

1. Obtain assistance in using the product.
  - \_\_\_ documentation clarification.
  - \_\_\_ operational understanding.
2. Obtain product specific information.
  - \_\_\_ problem identification.
  - \_\_\_ work-around, patch, or other solution when available.
  - \_\_\_ information on existing SPRs.
3. If the reported condition is not an already documented SPR, obtain assistance in problem isolation techniques.

As part of the Software Information Service, Software Support Services maintain a list of reported problems and problem resolutions. Software Information Services does not include user application or engineering time to derive a resolution to a problem if none is currently available. (See Phone Consulting under Consulting Services). However, the Software Support Engineer will submit a problem report into the SPR system under the Customer name when appropriate.

Software Information Service is offered as a supplemental tool for obtaining maximum utilization of Intel software products. It is expected that the Customer will avail himself of training classes as appropriate, and will make reasonable efforts to utilize all product documentation.

The Customer must designate one System Manager and one alternate as authorized callers. Additional persons can be authorized for an additional charge. This service is offered on a one-year period.



## SYSTEM/SOFTWARE SUPPORT PACKAGES

We have structured very specific support packages to assist our customers with the installation and reconfiguration of such systems as NDS-II and 86/330.

NDS-II Installation Package.	Includes software installation, system generation, and network orientation.
NDS-II Network Reconfiguration Pkg.	Includes regeneration and installation of NDS-II System Software.
RMX System Installation Package.	Includes installation and customer orientation of the RMX operating system on the 86/330 or 86/380 System.
RMX 86 Installation Package.	Includes one day installation, configuration and/or assistance; a training credit for one RMX86 class; 16 hours of phone consulting.
XENIX System Installation Package.	Includes up to 2 days of installation and customer orientation of the XENIX Operating System on the 86/330 AX or 86/380 AX System.
I2ICE Installation Package.	Includes the installation of and orientation to the host and probe software for the I2ICE on a Series III or Series IV Intellec Development System.

## CONSULTING SERVICES

Consulting Services provides customized support for system, board, and component level customers. Consulting services provide a wide range of support - from system designs to solving difficult development problems to complete project management and project implementation.

1. **Field Consulting** — The Customer may contract for an Intel Software Support Engineer to come on-site to assist and advise the customer in utilizing Intel software products. This service is available on a Time and Material basis. Minimum period: 1 day (8 hours). Travel time and expenses are billed separately as specified in the price list.
2. **Phone Consulting** — The Customer may contract for an Intel Software Support Engineer in the Customer Support Group to provide customer or application-specific research, effort, or consultation. Blocks of time may be purchased and utilized in minimum fifteen (15) minute increments.

## UPDATES

Updates and new releases for licensed products are offered on a per update basis. Each update will be separately priced for any registered Intel customer to purchase. Notification of updates and who requires which updates, will be provided through the monthly technical bulletin, ;COMMENTS, and each product-specific technical report.

## INSITE USER'S PROGRAM LIBRARY

Intel's Software Index and Technology Library is a library of programs that have been submitted by users of Intel microcomputers, single-board computers, and development systems. Membership in INSITE enables the Customer to order programs at a nominal charge. Members are provided a program catalog and catalog updates.

## LIMITATIONS

- A. Software Support Services are limited to standard Intel system configurations supported by software products, as defined in the applicable software product data sheet. Services will be performed within a 12-month period from effective date of the purchased services.
- B. Software support services do not include hardware maintenance.
- C. Any change in the equipment site of the system within the U.S. may affect Intel's ability to deliver the support services ordered and may result in increased charges. If the system is moved outside the continental U.S., it shall not be eligible for continued service as ordered, but may be eligible for continued service under Intel's local terms and conditions then in effect for a like system in the country or territory of reinstallation.



## OTHER INFORMATION

- A. Software Support Services is Intel's commitment to providing the customer with consistent, high-quality, post-sales software and system support. It is our way of delivering guaranteed support which the customer can rely on. To tailor a full service software/system support program that addresses specific needs, contact the local Intel sales or service office for more information.
- B. Term: Service will be provided for the period specified in the price list. Subscription services will automatically be renewed on an annual basis unless specified otherwise in writing by the customer.
- C. Charges: There are three kinds of billings utilized with the service offerings: front-end billing, monthly billing (not less than \$100 per month), and post-service billing. The customer will be billed on one of the referenced types of billings, depending on the type of service. Prices will be those specified in the current Intel price list.
- D. In order to obtain maximum service from Software Support Services, it is advised that the customer maintain the system to the latest revision level, and assign a System Manager who will be the key contact for Software Support Services.
- E. Guidelines:
1. The customer must have signed an Intel Master Software License agreement. All services and materials made available to the customer through Software Support Services, including documentation and program materials, are subject to the terms and conditions of the license/sale.
  2. The License Fee or List Price for covered software products includes a period of Initial Support as defined in individual product descriptions. Additional support services may be obtained as listed in the price list.
  3. Updates are available separately for software products. Each update to a specified software product has a single charge and is purchased separately. The customer must have a valid software license that covers the software product to obtain any update or new release.

The following chart summarizes the services available for various operating system environments at different levels of integration.

Software Support Services

PRODUCT NAME	PART NUMBER	OPERATING SYSTEM				
		ISIS	INDX	IRMX*	XENIX*	CP/M*-80
<b>INITIAL SUPPORT</b>	Included in license fee or price of each product	SYS	SYS	SYS BRD	SYS BRD	SYS
<b>CONTRACT SERVICE</b>						
Subscription Service	SPR-TECH-REP	SYS	SYS	SYS BRD	SYS BRD	SYS
Hotline Service	HOTLINE	SYS	SYS	SYS BRD	SYS BRD	SYS
<b>UPDATES</b>	Each update has a unique part number	SYS	SYS	SYS BRD	SYS BRD	SYS
<b>SUPPORT PACKAGES</b>						
NDS-II	SPNDS2INSTALL		SYS			
NDS-II	SPNDS2RECON		SYS			
RMX-86	SPRMX86INSTALL			SYS, BRD		
RMX Systems	SP86330RINSTALL			SYS		
XENIX Sys	SP86330XINSTALL				SYS	
PRICE*	SP11520INSTALL	SYS	SYS			
<b>CONSULTING SERVICES</b>						
Phone Consulting	CONSULT-PHONE	SYS	SYS	SYS, BRD COMP	SYS, BRD COMP	SYS
Field Consulting	CONSULT-FIELD	SYS	SYS	SYS, BRD COMP	SYS, BRD COMP	SYS COMP
Long-term Consulting	CONSULT-LT	SYS	SYS	SYS, BRD COMP	SYS, BRD COMP	SYS COMP
Expenses	CONSULT-EXP	Travel and living expenses incurred for delivering Consulting and Support Package				
<b>INSITE USER'S PROGRAM LIBRARY</b>	N/A	User submitted and non-licensed software products and programs used on or in conjunction with Intel Components, Boards and Systems				

KEY    SYS = All Intel standard system hardware, languages and software packages designed to operate on or within the particular Operating System  
        BRD = Service provided for this Operating System and associated language and software designed to operate on standard Intel Single Board Computer configurations  
        COMP = Support during the development of a user's system based on an Intel microprocessor component and using an Intel operating system and its associated languages and software

\*CP/M is a trademark of Digital Research, Inc

\*XENIX is a trademark of Microsoft Corp





## DOMESTIC SALES OFFICES

### ALABAMA

Intel Corp  
303 Williams Avenue, S W  
Suite 1422  
Huntsville 35801  
Tel (205) 533-9353

### ARIZONA

Intel Corp  
11225 N 28th Drive  
Suite 2140  
Phoenix 85029  
Tel (602) 869-4980

### CALIFORNIA

Intel Corp  
1010 Hurley Way  
Suite 300  
Sacramento 95825  
Tel (916) 929-4078

Intel Corp  
7670 Opportunity Road  
Suite 135  
San Diego 92111  
(714) 268-3563

Intel Corp \*  
2000 East 4th Street  
Suite 100  
Santa Ana 92705  
Tel (714) 835-9642  
TWX 910-595-1114

Intel Corp \*  
1350 Shorebird Way  
Mt View 94043  
Tel (415) 968-8086  
TWX 910-339-9279  
910-338-0255

Intel Corp \*  
5530 Corbin Avenue  
Suite 120  
Tarzana 91356  
Tel (213) 708-0333  
TWX 910-495-2045

### COLORADO

Intel Corp  
4445 Northpark Drive  
Suite 100  
Colorado Springs 80907  
Tel (303) 594-8622

Intel Corp \*  
650 S. Cherry Street  
Suite 720  
Denver 80222  
Tel (303) 321-8086  
TWX 910-931-2289

### CONNECTICUT

Intel Corp  
36 Padanaram Road  
Danbury 06810  
Tel (203) 792-8366  
TWX 710-456-1199

EMC Corp  
393 Center Street  
Wallingford 06492  
Tel (203) 265-6991

### FLORIDA

Intel Corp  
1500 N W 62nd Street  
Suite 104  
Ft Lauderdale 33309  
Tel (805) 771-0600  
TWX 510-956-9407

Intel Corp  
500 N Millland  
Suite 205  
Milland 32751  
Tel (805) 828-2383  
TWX 810-653-9219

### GEORGIA

Intel Corp  
3300 Holcombe Bridge Road  
Suite 225  
Norcross 30092  
Tel (404) 449-0541

### ILLINOIS

Intel Corp \*  
2550 Golf Road  
Suite 815  
Rolling Meadows 60008  
Tel (312) 981-7200  
TWX 910-651-5681

### INDIANA

Intel Corp  
9100 Purdue Road  
Suite 400  
Indianapolis 46208  
Tel (317) 875-0623

### IOWA

Intel Corp  
St Andrews Building  
1930 St Andrews Drive N E  
Cedar Rapids 52402  
Tel (319) 393-5510

### KANSAS

Intel Corp  
8400 W 110th Street  
Suite 170  
Overland Park 66210  
Tel (913) 642-8080

### LOUISIANA

Industrial Digital Systems Corp  
2332 Severn Avenue  
Suite 202  
Metairie 70001  
Tel (504) 831-8492

### MARYLAND

Intel Corp \*  
7257 Parkway Drive  
Hanover 21076  
Tel (301) 796-7500  
TWX 710-862-1944

Intel Corp  
1620 Elton Road  
Silver Spring 20903  
Tel (301) 431-1200

### MASSACHUSETTS

Intel Corp \*  
27 Industrial Avenue  
Chelmsford 01824  
Tel (617) 256-1800  
TWX 710-343-6333

EMC Corp  
385 Elliot Street  
Newton 02164  
Tel (617) 244-4740  
TWX 922531

### MICHIGAN

Intel Corp \*  
26500 Northwestern Hwy  
Suite 401  
Southfield 48075  
Tel (313) 353-0920  
TWX 810-244-4915

### MINNESOTA

Intel Corp  
3500 W 80th Street  
Suite 360  
Bloomington 55431  
Tel (612) 835-6722  
TWX 910-576-2867

### MISSOURI

Intel Corp  
4203 Earth City Expressway  
Suite 121  
Earth City 63045  
Tel (314) 291-1990

### NEW JERSEY

Intel Corp \*  
Raritan Plaza III  
Raritan Center  
Eriason 08837  
Tel (201) 225-3000  
TWX 710-480-6238

### NEW MEXICO

Intel Corp  
1120 Junco Tabo N E  
Albuquerque 87112  
Tel (505) 292-8086

### NEW YORK

Intel Corp \*  
300 Vanderbilt Motor Parkway  
Hauppauge 11788  
Tel (516) 231-0300  
TWX 510-227-6236

Intel Corp  
80 Washington Street  
Poughkeepsie 12601  
Tel (914) 473-2303  
TWA 510-248-0060

Intel Corp \*  
211 White Spruce Boulevard  
Rochester 14623  
Tel (716) 424-1050  
TWX 510-253-7391

T-Squared  
6443 Ridings Road  
Syracuse 13206  
Tel (315) 463-8592  
TWX 710-541-0554

T-Squared  
7353 Pittsford  
Victor Road  
Tel (716) 924-9101  
TWX 510-254-8547

### NORTH CAROLINA

Intel Corp  
2306 W Meadowview Road  
Suite 206  
Greensboro 27407  
Tel (919) 294-1541

### OHIO

Intel Corp \*  
8500 Poe Avcnue  
Dayton 45414  
Tel (613) 890-5360  
TWX 810-450-2528

Intel Corp \*  
Chagrin-Branard Bldg., No 300  
28001 Chagrin Boulevard  
Cleveland 44122  
Tel (216) 464-6915  
TWX 810-427-9298

### OKLAHOMA

Intel Corp  
4157 S Harvard Avenue  
Suite 125  
Tulsa 74135  
Tel (918) 749-8688

### OREGON

Intel Corp  
10700 S W Beaverlton  
Hillsdale Highway  
Suite 22  
Beaverlton 97005  
Tel (503) 841-8096  
TWX 910-467-5741

### PENNSYLVANIA

Intel Corp \*  
510 Pennsylvania Avenue  
Fort Washington 19034  
Tel (215) 641-1000  
TWX 510-661-2077

Intel Corp \*  
221 Penn Center Boulevard  
Suite 301W  
Pittsburgh 15225  
Tel (412) 823-4970

Q.E.D., Electronics  
300 N York Road  
Hatboro 19040  
Tel (215) 674-9600

### TEXAS

Intel Corp \*  
12300 Ford Road  
Suite 380  
Dallas 75234  
Tel (214) 241-9097  
TWX 910-860-5617

Intel Corp \*  
7322 S W Freeway  
Suite 1490  
Houston 77074  
Tel (713) 968-8086  
TWX 910-881-2490

Industrial Digital Systems Corp  
5925 Sovereign  
Suite 101  
Houston 77036  
Tel (713) 868-9421

Intel Corp  
315 E Anderson Lane  
Suite 314  
Austin 78752  
Tel (512) 454-3628

### UTAH

Intel Corp  
258 West 400 South  
Salt Lake City 84101  
Tel (801) 533-8086

### VIRGINIA

Intel Corp  
1603 Santa Rosa Road  
Suite 109  
Richmond 23288  
Tel (804) 282-5668

### WASHINGTON

Intel Corp  
110 110th Avenue N E  
Suite 510  
Bellevue 98004  
Tel (206) 453-8086  
TWX 910-443-3002

### WISCONSIN

Intel Corp  
450 N Sunnyslope Road  
Suite 130  
Brookfield 53005  
Tel (414) 784-9060

### CANADA

#### ONTARIO

Intel Semiconductor of Canada, Ltd  
39 Hwy 7, Bell Mews  
Napran K01 8R2  
Tel (613) 829-9714  
TELEX 053-4115

Intel Semiconductor of Canada, Ltd  
50 Galaxy Boulevard  
Suite 12  
Rexdale M9W 4Y5  
Tel (416) 875-2105  
TELEX 06983574

Intel Semiconductor of Canada, Ltd  
201 Consumers Road  
Suite 200  
Willowdale M2J 4G8  
Tel (416) 494-6831  
TELEX 4946831

#### QUEBEC

Intel Semiconductor of Canada, Ltd  
3860 Cole Vertu Road  
Suite 210  
St Laurent H4R 1V4  
Tel (514) 334-0560  
TELEX 05-824172

\*Field Application Location





## DOMESTIC DISTRIBUTORS

### ALABAMA

†Arrow Electronics, Inc  
3611 Memorial Parkway So  
Huntsville 35405  
Tel (205) 892-2730  
†Hamilton/Avnet Electronics  
4812 Commercial Drive N.W.  
Huntsville 35805  
Tel (205) 837-7210  
TWX 810-728-2162  
†Pioneer/Huntsville  
1207 Putnam Drive N.W.  
Huntsville 35805  
Tel (205) 837-8300  
TWX 810-726-2197

### ARIZONA

†Hamilton/Avnet Electronics  
505 S Madison Drive  
Tempe 85281  
Tel (602) 231-5140  
TWX 910-950-0077  
†Wyle Distribution Group  
8155 N 24th Street  
Phoenix 85021  
Tel (602) 249-2332  
TWX 910-951-4282

### CALIFORNIA

†Arrow Electronics, Inc  
521 Weddell Drive  
Sunnyvale 94086  
Tel (408) 745-8600  
TWX 910-339-9371  
†Arrow Electronics, Inc  
19748 Dearborn Street  
Chatsworth 91311  
Tel (313) 701-7500  
TWX 910-493-2086  
†Hamilton/Avnet Electronics  
350 McCormick Avenue  
Costa Mesa 92626  
Tel (714) 754-6051  
TWX 910-595-1928  
†Hamilton/Avnet Electronics  
19515 So Vermont Avenue  
Irvine 92612  
Tel (313) 615-3909  
TWX 910-349-6263  
†Hamilton/Avnet Electronics  
1175 Bordeaux Drive  
Sunnyvale 94086  
Tel (408) 743-3300  
TWX 910-339-9332  
†Hamilton/Avnet Electronics  
4545 Viewridge Avenue  
San Diego 92123  
Tel (714) 641-1850  
TWX 910-595-2638  
†Hamilton/Avnet Electronics  
10912 W Washington Boulevard  
Culver City 20230  
Tel (213) 558-2458  
TWX 910-340-6364  
†Hamilton Avnet/Electronics  
21050 Erwin Street  
Woodland Hills 91367  
Tel (213) 883-0000  
TWX 910-494-2207  
†Hamilton Electro Sales  
3170 Pullman Street  
Costa Mesa 92626  
Tel (714) 641-4109  
TWX 910-595-2638  
†Hamilton/Avnet Electronics  
4103 Northgate Boulevard  
Sacramento 95834  
Tel (916) 920-3150  
†Kieruff Electronics, Inc  
3969 E Bayshore Road  
Palo Alto 94303  
Tel (415) 968-6292  
TWX 910-379-6430  
†Kieruff Electronics, Inc  
14101 Franklin Avenue  
Tustin 92680  
Tel (714) 731-5711  
TWX 910-595-2599  
†Kieruff Electronics, Inc  
2385 Commerce Way  
Los Angeles 90040  
Tel (213) 725-0325  
TWX 910-580-3686  
†Wyle Distribution Group  
124 Maryland Street  
E Segundo 90245  
Tel (313) 322-8100  
TWX 910-348-7140 or 7111

### CALIFORNIA (Cont'd)

†Wyle Distribution Group  
9525 Chesapeake Drive  
San Diego 92123  
Tel (714) 565-9171  
TWX 910-335-1590  
†Wyle Distribution Group  
3000 Bowers Avenue  
Santa Clara 95051  
Tel (408) 727-2500  
TWX 910-338-0296  
†Wyle Distribution Group  
17872 Cowan Avenue  
Irvine 92714  
Tel (714) 641-1600  
TWX 910-595-1572

### COLORADO

†Wyle Distribution Group  
451 E 124th Avenue  
Thornton 80241  
Tel (303) 457-9953  
TWX 910-936-0770  
†Hamilton/Avnet Electronics  
8765 Orchard Road  
Suite 708  
Englewood 80111  
Tel (303) 740-1017  
TWX 910-935-0787

### CONNECTICUT

†Arrow Electronics, Inc  
12 Beaumont Road  
Wallingford 06492  
Tel (203) 265-7741  
TWX 710-476-0162

†Hamilton/Avnet Electronics  
Commerce Industrial Park  
Commerce Drive  
Danbury 06810  
Tel (203) 737-2800  
TWX 710-456-9974  
†Harvey Electronics  
112 Main Street  
Norwalk 06851  
Tel (203) 853-1515  
TWX 710-466-3373

### FLORIDA

†Arrow Electronics, Inc  
1001 N W 62nd Street  
Suite 108  
Ft Lauderdale 33309  
Tel (305) 776-7790  
TWX 510-955-9456  
†Arrow Electronics, Inc  
W Woodlake Drive W  
Bldg B  
Palm Bay 32905  
Tel (305) 725-1480  
TWX 510-959-6337  
†Hamilton/Avnet Electronics  
8801 N W 15th Way  
Ft Lauderdale 33309  
Tel (305) 971-3900  
TWX 510-956-3097  
†Hamilton/Avnet Electronics  
3197 Tech Drive North  
St Petersburg 33702  
Tel (813) 576-3930  
TWX 810-863-0374  
†Pioneer/Alta Monte Springs  
221 N Lake Blvd  
Suite 412  
Alta Monte Springs 32701  
Tel (305) 859-3600  
TWX 810-855-0594  
†Pioneer/Ft Lauderdale  
1500 62nd Street N.W.  
Suite 506  
Ft Lauderdale 33309  
Tel (305) 771-7520  
TWX 510-955-9653

### GEORGIA

†Arrow Electronics, Inc  
2979 Pacific Drive  
Norcross 30071  
Tel (404) 449-8252  
TWX 810-766-0439  
†Hamilton/Avnet Electronics  
5825 D Peachtree Corners  
Norcross 30092  
Tel (404) 447-7500  
TWX 810-766-0432  
†Pioneer/Georgia  
5835B Peachtree Corners E  
Norcross 30092  
Tel (404) 448-1711  
TWX 810-766-4515

### ILLINOIS

†Arrow Electronics, Inc  
2000 E Algonquin Street  
Schaumburg 60195  
Tel (312) 397-3440  
TWX 910-291-3544  
†Hamilton/Avnet Electronics  
1130 Thorndale Avenue  
 Bensenville 60106  
Tel (312) 860-7780  
TWX 910-227-0060  
†Pioneer/Chicago  
1551 Carmen Drive  
Elk Grove Village 60007  
Tel (312) 437-9680  
TWX 910-222-1834

### INDIANA

†Arrow Electronics, Inc  
2718 Rand Road  
Indianapolis 45241  
(317) 243-9353  
TWX 810-341-3119  
†Hamilton/Avnet Electronics  
485 Grade Drive  
Carmel 46032  
Tel (317) 844-9333  
TWX 810-260-3966  
†Pioneer/Indiana  
6408 Castleplace Drive  
Indianapolis 46250  
Tel (317) 849-7300  
TWX 810-260-1794

### KANSAS

†Hamilton/Avnet Electronics  
9219 Quivera Road  
Overland Park 66215  
Tel (913) 888-8900  
TWX 910-743-0005  
†Mesa Technology Corporation  
16021 Industrial Drive  
Gaitersburg 20977  
Tel (301) 948-4350TWX 710-828-9702  
†Pioneer  
8100 Gaither Road  
Gaitersburg 20977  
Tel (301) 948-0710  
TWX 710-828-0545

### MARYLAND

†Hamilton/Avnet Electronics  
6822 Oak Hill Lane  
Columbia 21045  
Tel (301) 995-3500  
TWX 710-862-1861  
†Mesa Technology Corporation  
16021 Industrial Drive  
Gaitersburg 20977  
Tel (301) 948-4350TWX 710-828-9702  
†Pioneer  
8100 Gaither Road  
Gaitersburg 20977  
Tel (301) 948-0710  
TWX 710-828-0545

### MASSACHUSETTS

†Arrow Electronics, Inc  
1 Arrow Drive  
Woburn 01801  
Tel (617) 893-8130  
TWX 710-393-6770  
†Hamilton/Avnet Electronics  
50 Tower Office Park  
Woburn 01801  
Tel (617) 935-9700  
TWX 710-393-0382  
†Harvey/Boston  
44 Hartwell Avenue  
Lexington 02173  
Tel (617) 863-1200  
TWX 710-326-6617

### MICHIGAN

†Arrow Electronics, Inc  
3810 Varady Drive  
Ann Arbor 48104  
Tel (313) 971-8220  
TWX 810-223-6020  
†Pioneer/Michigan  
13485 Stamford  
Livonia 48150  
Tel (313) 525-1800  
TWX 810-242-8775  
†Hamilton/Avnet Electronics  
32487 Schoolcraft Road  
Livonia 48150  
Tel (313) 522-4700  
TWX 810-242-8775  
†Hamilton/Avnet Electronics  
2215 29th Street S.E.  
Space A5  
Grand Rapids 49508  
Tel (616) 243-8805  
TWX 810-273-6921

### MINNESOTA

†Arrow Electronics, Inc  
5230 W 73rd Street  
Edina 55435  
Tel (612) 830-1800  
TWX 910-576-3125  
†Hamilton/Avnet Electronics  
10300 Bren Road East  
Minnetonka 55343  
Tel (612) 932-9600  
TWX (910) 576-2720  
†Pioneer/Twin Cities  
10203 Bren Road East  
Minnetonka 55343  
Tel (612) 935-5444  
TWX 910-576-2738

### MISSOURI

†Arrow Electronics, Inc  
2380 Schuetz  
St. Louis 63141  
Tel (314) 567-6888  
TWX 910-764-0882  
†Hamilton/Avnet Electronics  
13743 Shoreline Court  
Earth City 63045  
Tel (314) 344-1200  
TWX 910-762-0684

### NEW HAMPSHIRE

†Arrow Electronics, Inc  
1 Penmar Road  
Manchester 03103  
Tel (603) 668-6969  
TWX 710-220-1684

### NEW JERSEY

†Arrow Electronics, Inc  
Pleasant Valley Avenue  
Moorestown 08057  
Tel (215) 828-1800  
TWX 710-897-0829  
†Arrow Electronics, Inc  
2 Industrial Road  
Fairfield 07006  
Tel (201) 575-5300  
TWX 910-996-2206

†Hamilton/Avnet Electronics  
1 Keystone Avenue  
Bldg 38  
Cherry Hill 08003  
Tel (609) 424-0110  
TWX 710-940-0262

†Hamilton/Avnet Electronics  
10 Industrial  
Fairfield 07006  
Tel (201) 575-3390  
TWX 710-734-4388

†Harvey Electronics  
45 Route 46  
Pinebrook 07058  
Tel (201) 575-3510  
TWX 710-734-4382  
†MTI Systems Sales  
383 Route 46 W  
Fairfield 07006  
Tel (201) 227-5552

### NEW MEXICO

†Alliance Electronics Inc  
11030 Cochiti S.E.  
Albuquerque 87123  
Tel (505) 292-3360  
TWX 910-969-1151  
†Hamilton/Avnet Electronics  
2524 Baylor Drive S.E.  
Albuquerque 87106  
Tel (505) 765-1500  
TWX 910-989-0614

### NEW YORK

†Arrow Electronics, Inc  
900 Broad Hollow Road  
Farmingdale 11735  
Tel (516) 694-8800  
TWX 510-224-6126  
†Arrow Electronics, Inc  
3000 South Winton Road  
Rochester 14623  
Tel (716) 275-0300  
TWX 510-253-4766  
†Arrow Electronics, Inc  
7705 Maitage Drive  
Liverpool 13088  
Tel (315) 652-1000  
TWX 710-545-0230  
†Arrow Electronics, Inc  
20 Oser Avenue  
Hauppauge 11788  
Tel (516) 231-1000  
TWX 510-227-6623



## DOMESTIC DISTRIBUTORS

### NEW YORK (Cont'd)

†Hamilton/Avnet Electronics  
333 Metro Park  
Rochester 14623  
Tel. (716) 475-9130  
TWX 510-253-5470

†Hamilton/Avnet Electronics  
19 Corporate Circle  
E. Syracuse 13057  
Tel. (315) 437-2641  
TWX 710-541-1560

†Hamilton/Avnet Electronics  
5 Hub Drive  
Melville, Long Island 11747  
Tel. (516) 454-0020  
TWX 510-224-6166

†Harvey Electronics  
P.O. Box 1208  
Binghamton 13902  
Tel. (607) 748-8211  
TWX 510-252-0893

†Harvey Electronics  
60 Crossway Park West  
Woodbury, Long Island 11797  
Tel. (516) 921-8700  
TWX 510-221-2184

†Harvey/Rochester  
840 Fairport Park  
Fairport 14651  
Tel. (716) 381-7070  
TWX 510-253-7001

†MTI Systems Sales  
38 Harbor Park Drive  
Port Washington 11050  
Tel. (516) 821-8200  
TWX 510-223-0846

### NORTH CAROLINA

†Arrow Electronics, Inc.  
938 Burke Street  
Winston-Salem 27101  
Tel. (919) 725-8711  
TWX 510-931-3169

†Hamilton/Avnet Electronics  
3510 Spring Forest Drive  
Raleigh 27604  
Tel. (919) 878-0819  
TWX 910-928-1936

†Pioneer/Carolina  
103 Industrial Avenue  
Greensboro 27406  
Tel. (919) 273-4441  
TWX 910-925-1114

### OHIO

†Arrow Electronics, Inc.  
7520 McEwen Road  
Centerville 45459  
Tel. (513) 435-5583  
TWX 910-459-1611

†Arrow Electronics, Inc.  
6238 Cochran Road  
Solon 44139  
Tel. (216) 249-3900  
TWX 910-427-9409

†Hamilton/Avnet Electronics  
954 Senate Drive  
Dayton 45459  
Tel. (513) 433-0610  
TWX 910-450-2531

†Hamilton/Avnet Electronics  
4588 Emery Industrial Parkway  
Warrensville Heights 44128  
Tel. (216) 831-3500  
TWX 910-427-9452

### OHIO (Cont'd)

†Pioneer/Dayton  
4433 Interpoint Boulevard  
Dayton 45424  
Tel. (513) 236-9900  
TWX 910-459-1622

†Pioneer/Cleveland  
4900 E. 131st Street  
Cleveland 44105  
Tel. (216) 587-3800  
TWX 910-422-2211

### OKLAHOMA

†Arrow Electronics, Inc.  
4719 S. Memorial Drive  
Tulsa 74145  
Tel. (918) 665-7700

### OREGON

†Almac Electronics Corporation  
8022 S.W. Nimbus, Bldg 7  
Beaverton 97005  
Tel. (503) 641-9070  
TWX 910-467-8743

†Hamilton/Avnet Electronics  
8024 S.W. Jean Road  
Bldg. C, Suite 10  
Lake Oswego 97034  
Tel. (503) 635-7848  
TWX 910-455-8179

### PENNSYLVANIA

†Arrow Electronics, Inc.  
650 Seco Road  
Monroeville 15146  
Tel. (412) 856-7000

†Pioneer/Pittsburgh  
259 Kappa Drive  
Pittsburgh 15238  
Tel. (412) 782-2300  
TWX 710-795-3122

†Pioneer/Delaware Valley  
201 Gibraltar Road  
Horsham 19044  
Tel. (215) 674-4000  
TWX 910-865-6778

### TEXAS

†Arrow Electronics, Inc.  
13715 Gamma Road  
Dallas 75224  
Tel. (214) 386-7500  
TWX 910-860-5377

†Arrow Electronics, Inc.  
10899 Kinghurst  
Suite 100  
Houston 77099  
Tel. (713) 530-4700  
TWX 910-860-4439

†Arrow Electronics, Inc. 10125  
Metropolitan  
Austin 78758  
Tel. (512) 835-4100  
TWX 910-874-1348

†Hamilton/Avnet Electronics  
2401 Rutland  
Austin 78757  
Tel. (512) 837-8911  
TWX 910-874-1319

†Hamilton/Avnet Electronics  
2111 W. Walnut Hill Lane  
Irving 75082  
Tel. (214) 659-4100  
TWX 910-860-8929

### TEXAS (Cont'd)

†Hamilton/Avnet Electronics  
8750 West Park  
Houston 77063  
Tel. (713) 780-1771  
TWX 910-881-5523

†Pioneer/Austin  
9901 Burnet Road  
Austin 78758  
Tel. (512) 835-4000  
TWX 910-874-1323

†Pioneer/Dallas  
13710 Omega Road  
Dallas 75234  
Tel. (214) 386-7300  
TWX 910-850-5583

†Pioneer/Houston  
5853 Point West Drive  
Houston 77036  
Tel. (713) 989-5555  
TWX 910-881-1606

### UTAH

†Hamilton/Avnet Electronics  
1585 West 2100 South  
Salt Lake City 84119  
Tel. (801) 975-2800  
TWX 910-925-4018

Wyle Distribution Group  
1859 South 4130 West, Unit B  
Salt Lake City 84104  
Tel. (801) 974-9653

### WASHINGTON

†Almac Electronics Corporation  
14380 S.E. Eastgate Way  
Bellevue 98007  
Tel. (206) 643-9992  
TWX 910-444-2067

†Arrow Electronics, Inc.  
14320 N.E. 21st Street  
Bellevue 98007  
Tel. (206) 643-4800  
TWX 910-444-2017

†Hamilton/Avnet Electronics  
14212 N.E. 21st Street  
Bellevue 98005  
Tel. (206) 453-5874  
TWX 910-443-2489

### WISCONSIN

†Arrow Electronics, Inc.  
430 W. Rauson Avenue  
Oakcreek 53154  
Tel. (414) 764-8800  
TWX 910-282-1193

†Hamilton/Avnet Electronics  
2975 Moorland Road  
New Berlin 53151  
Tel. (414) 784-4510  
TWX 910-282-1182

### CANADA

#### ALBERTA

†Hamilton/Avnet Electronics  
2816 21st Street N.E.  
Calgary T2E 6Z3  
Tel. (403) 230-3586  
TWX 03-827-642

†L.A. Varah, Ltd.  
4742 14th Street N.E.  
Calgary T2D 6L7  
Tel. (403) 230-1235  
TWX 038-258-97

#### ALBERTA (Cont'd)

Zenitronics  
Bay #1  
3500 14th Avenue N.E.  
Calgary T2A 6J4  
Tel. (403) 272-1021

#### BRITISH COLUMBIA

L.A. Varah, Ltd.  
2077 Alberta Street  
Vancouver V5Y 1C4  
Tel. (604) 873-3211  
TWX: 810-929-1068

Zenitronics  
108-11400 Bridgeport Road  
Richmond V6X 1T2  
Tel. (604) 273-5575  
TWX 04-5077-89

#### MANITOBA

L.A. Varah, Ltd.  
12-1832 King Edward Street  
Winnipeg R5R 0N1  
Tel. (204) 633-6190  
TWX 07-55-365

Zenitronics  
590 Berry Street  
Winnipeg R3H 0S1  
Tel. (204) 775-8661

#### ONTARIO

Hamilton/Avnet Electronics  
6845 Rexwood Road  
Unit 6 & H  
Mississauga L4V 1R2  
Tel. (416) 677-7432  
TWX 610-492-8867

Hamilton/Avnet Electronics  
210 Colomede Road South  
Napain K2E 7L5  
Tel. (613) 226-1700  
TWX 05-349-71

L.A. Varah, Ltd.  
505 Kenora Avenue  
Hamilton L8E 3P2  
Tel. (416) 561-3311  
TWX 061-6349

Zenitronics  
8 Tilbury Court  
Brampton L6T 3T4  
Tel. (416) 451-8600  
TWX 06-976-78

Zenitronics  
564/10 Weber Street North  
Waterloo N2L 5G8  
Tel. (519) 884-5700

Zenitronics  
590 Berry Street  
Winnipeg R3H 0S1  
Tel. (204) 775-8661

#### QUEBEC

Hamilton/Avnet Electronics  
2670 Sabourin Street  
St. Laurent H4S 1M2  
Tel. (514) 331-8443  
TWX 610-421-3731

Zenitronics  
505 Locke Street  
St. Laurent H4T 1X7  
Tel. (514) 735-5361  
TWX 05-827-535



## EUROPEAN SALES OFFICES

### BELGIUM

Intel Corporation S A  
Parc Sany  
Rue du Moulin a Papier 51  
Boite  
S-1160 Brussels  
Tel (02)67 07 11  
TELEX 28414

### DENMARK

Intel Denmark A/S\*  
Lyngbyvej 32F 2nd Floor  
DK-2100 Copenhagen East  
Tel (01) 18 20 00  
TELEX 19567

### FINLAND

Intel Finland Oy  
Hameentie 103  
SF - 00550 Helsinki 55  
Tel 07/19 355  
TELEX 123 332

### FRANCE

Intel Corporation, S A R L \*  
5 Place de la Balance  
Salle 223  
94528 Rungis Cedex  
Tel (01) 687 22 21  
TELEX 270475

### FRANCE (Cont'd)

Intel Corporation, S A R L  
immeuble B5C  
4 Quai des Etoiles  
69005 Lyon  
Tel (7) 842 40 89  
TELEX 305153

### WEST GERMANY

Intel Semiconductor GmbH\*  
Siedlstrasse 27  
D-8000 Muenchen 2  
Tel (89) 53951  
TELEX 05-23177 INTL D

Intel Semiconductor GmbH\*  
Manzor Strasse 75  
D-6200 Weesbaden 1  
Tel (6121) 70 08 74  
TELEX 04186183 INTW D

Intel Semiconductor GmbH  
Bruecksstrasse 61  
7012 Fellbach  
West Germany  
Tel (711) 58 00 82  
TELEX 7254826 INTS D

Intel Semiconductor GmbH\*  
Hohenzollern Strasse 57  
3000 Hannover 1  
Tel (511) 34 40 81  
TELEX 923625 INTH D

Intel Semiconductor GmbH  
Ober-Ratherstrasse 2  
D-4000 Dusseldorf 30  
Tel (211) 65 10 54  
TELEX 08-58977 INTL D

### ISRAEL

Intel Semiconductor Ltd \*  
P.O. Box 1659  
Herla  
Tel 4/524  
TELEX 46511

### ITALY

Intel Corporation Italia Spa\*  
Milanlon, Palazzo E  
20094 Assago (Milano)  
Tel (02) 824 00 06  
TELEX 315183 INTMIL

### NETHERLANDS

Intel Semiconductor Nederland B V\*  
Alexanderpoort Building  
Marten Meesweg 93  
3068 Rotterdam  
Tel (10) 21 23 77  
TELEX 22283

### NORWAY

Intel Norway A/S  
P.O. Box 92  
Hvamveien 4  
N-2013  
Sveithus  
Tel (2) 742 420  
TELEX 18018

### SWEDEN

Intel Sweden A.B.\*  
Box 20092  
Archimedesvagen 5  
S-16120 Bromma  
Tel (08) 38 53 85  
TELEX 12261

### SWITZERLAND

Intel Semiconductor A.G.\*  
Forchstrasse 95  
CH 8032 Zurich  
Tel (01) 55 45 02  
TELEX 57989 ICH CH

### UNITED KINGDOM

Intel Corporation (U.K.) Ltd \*  
5 Hospital Street  
Nantwich, Cheshire CW5 5RE  
Tel (0270) 626 560  
TELEX 36620

Intel Corporation (U.K.) Ltd \*  
Pipers Way  
Swindon, Wiltshire SN3 1RJ  
Tel (0793) 488 368  
TELEX 444447 INT SWN

\*Field Application Location

## EUROPEAN DISTRIBUTORS/REPRESENTATIVES

### AUSTRIA

Bacher Elektronische Gerate GmbH  
Rotemuehlengasse 26  
A 1120 Vienna  
Tel (222) 83 83 96  
TELEX 11532 BASAT A

### BELGIUM

Inelco Belguim S A  
Ave des Croix de Guerre 94  
B1120 Brussels  
Tel (02) 216 01 60  
TELEX 25441

### DENMARK

Multikomponent A/S  
Fabrikparken 31  
DK-2600 Glostrup  
Tel (02) 45 66 45  
TX 33555

Scandinavian Semiconductor  
Supply A/S  
Narnasgade 18  
DK-2200 Copenhagen  
Tel (01) 83 50 90  
TELEX 19037

### FINLAND

Oy Fintronik AB  
Melkonkatu 24 A  
SF-00210  
Helsinki 21  
Tel (0) 692 80 22  
TELEX 124 224 Firon SF

### FRANCE

Generm  
21 de Courtabouef  
Avenue de la Baltique  
91943 Les Ulis Cedex-B.P.88  
Tel (6) 907 78 79  
TELEX F691700

Jermyn S A  
rue Jules Ferry 35  
93170 Bagnolet  
Tel (1) 859 04 04  
TELEX 21810 F

Matrologie  
La Tour d'Asnieres  
1, Avenue Laurent Cely  
92606-Asnieres  
Tel (1) 791 44 44  
TELEX 611-448

### FRANCE (Cont'd)

Tekelec Artronc  
Cite des Bruyeres  
Rue Carie Vernet  
F-92010 Suresse  
Tel (01) 534 75 35  
TELEX 204552

### WEST GERMANY

Electronic 2000 Vertriebs A G  
Neumarkter Strasse 75  
D-8000 Munich 80  
Tel (89) 43 40 81  
TELEX 522561 EIEC D

Jermyn GmbH  
Postfach 1180  
Schulstrasse 48  
D-6277 Bad Camberg  
Tel (06434) 231  
TELEX 484426 JERM D

Celdis Elektronik Systems GmbH  
Gutenbergstrasse 4  
2359 Henstedt-Ulzburg 1  
Tel (04193) 4026  
TELEX 2180260

Proelectron Vertriebs GmbH  
Max Planck Strasse 1-3  
6072 Dreieich bei Frankfurt  
Tel (6103) 33564  
TELEX 417983

### IRELAND

Micro Marketing  
Glenagery Office Park  
Glenagery  
Co Dublin  
Tel (1) 85 62 88  
TELEX 31584

### ISRAEL

Eastronics Ltd  
11 Rozans Street  
P.O. Box 39300  
Tel Aviv 61890  
Tel (3) 47 51 51  
TELEX 33638

### ITALY

Eledra S S P A  
Viale Elvezia, 18  
I 20154 Milano  
Tel (2) 34 97 51  
TELEX 332332

### ITALY (Cont'd)

Intesi  
Milanlon Pal E/5  
20090 Assago  
Milano  
Tel (02) 82470  
TELEX 311351

### NETHERLANDS

Koning & Hartman  
Kopervort 30  
P.O. Box 43220  
2544 EN's Gravenhage  
Tel 31 (70) 210 101  
TELEX 31528

### NORWAY

Nordisk Elektronik (Norge) A/S  
Postoffice Box 122  
Smedsvengen 4  
1364 Hvalstad  
Tel (2) 796 210  
TELEX 77546

### PORTUGAL

Ditam  
Componentes E Electronica LDA  
Av Miguel Bombarda, 133  
P1000 Lisboa  
Tel (18) 545 313  
TELEX 14182 BReks-P

### SPAIN

Interface S A  
Florida San Pedro 22,3  
Barcelona 10  
Tel (3) 301 78 51  
TWX 51608  
ITT SESA  
Miguel Angel 23-3  
Madrid 10  
Tel (1) 419 54 00  
TELEX 27707

### SWEDEN

AB Gosta Backstrom  
Box 12009  
Aistromergatan 22  
S-20221 Stockholm 12  
Tel (8) 541 060  
TELEX 10135

### SWEDEN (Cont'd)

Nordisk Elektronik AB  
Box 27301  
Sandhammsgatan 71  
S-10254 Stockholm  
Tel (8) 635 040  
TELEX 10547

### SWITZERLAND

Industrade AG  
Gemmenstrasse 2  
Postfach 90 - 21190  
CH-8021 Zurich  
Tel (01) 363 23 20  
TELEX 56788 INDEL CH

### UNITED KINGDOM

Bytech Ltd  
Unit 57  
London Road  
Earley, Reading  
Berkshire  
Tel (0734) 61031  
TELEX 848215

Comway Microsystems Ltd  
Market Street  
UK-Bracknell, Berkshire  
Tel 44 (344) 55333  
TELEX 847201

Jermyn Industries  
Vestry Estate  
Sevenoaks, Kent  
Tel (0732) 450144  
TELEX 95142

M E D L  
East Lane Road  
North Wembley  
Middlesex HA8 7PP  
Tel (01) 904 93 07  
TELEX 2P817

Rapid Recall, Ltd  
Rapid House/Denmark St  
High Wycombe  
Berk, England HP11 2ER  
Tel (0494) 26 271  
TELEX 837931

### YUGOSLAVIA

H R Microelectronics Enterprises  
P.O. Box 5604  
San Jose, California 95150  
Tel 408/978-8000  
TELEX 278-559



## INTERNATIONAL SALES OFFICES

### AUSTRALIA

Intel Semiconductor Pty Ltd  
Spectrum Building  
200 Pacific Highway  
Level 6  
Crowe Nest, NSW, 2089  
Australia  
Tel 011-61-2-436-2744  
TELEX 790-20097  
FAX 011-61-2-923-2632

### HONG KONG

Intel Semiconductor Ltd  
13/F Hong Kong Trade Centre  
161-167 Des Voeux Road Central  
Tel 011-852-6-450-885  
TELEX 63969 ISLHKHX

### JAPAN

Intel Japan KK  
5-6 Tokodai, Toyosato-machi  
Tsukuba-gun, Ibaraki-ken 300-26  
Tel 029747-8511  
TELEX 03656-160

### JAPAN (Cont'd)

Intel Japan K K \*  
2-1-15 Naka-machi  
Atsugi, Kanagawa 243  
Tel 0462-23-3511

Intel Japan K K \*  
2-51-2 Kojima-cho  
Chofu, Tokyo 182  
Tel 0424-88-3151

Intel Japan K K \*  
2-69 Hon-cho  
Kumagaya, Saitama 360  
Tel 0485-24-6871

### JAPAN (Cont'd)

Intel Japan K K \*  
2-4-1 Terauchi  
Toyonaka, Osaka 560  
Tel 06-863-1091

Intel Japan K K  
1-5-1 Marunouchi  
Chiyoda-ku, Tokyo 100  
Tel 03-201-3621/3681

Intel Japan K K \*  
1-23-9 Shimomachi  
Setagaya-ku, Tokyo 154  
Tel 03-426-2231

\*Field Application Location

## INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

### ARGENTINA

VLC SRL  
Sarmiento 1630, 1 Pso  
1042 Buenos Aires  
Tel 35-1201/9242  
TELEX Public Booth 9900 or 9901

Mailing Address  
Soimex International Corporation  
15 Park Row, Room #1730  
New York, New York 10038  
(212) 406-3052  
Attn: Gaston Briones

### AUSTRALIA

Total Electronics  
9 Harker Street  
Burwood  
Victoria 3125  
Tel 61 3 288-4044  
TELEX AA 31261

Mailing Address  
Private Bag 250  
Burwood, Victoria 3125  
Australia

Total Electronics  
#1 Johnstone Lane  
Lane Cove, N S W 2086  
TELEX 26297

### BRAZIL

Icotron S.A.  
05110 Av. Mutanga 3650-6 Andar  
Pinheiros Sao Paulo  
Tel 261-0211  
TELEX 1122274/ICOTBR

### CHILE

DIN  
AV VIC MCKENNA 204  
Casilla 6055  
Santiago  
Tel 227 564  
TELEX 352 003

### COLUMBIA

International Computer Machines  
Carrera 7 No 72-34  
Apto Aereo 19403  
Bogota 1  
Tel 211-7282  
TELEX 45716 ICM CO

### HONG KONG

Schmidt & Co Ltd  
Wing on Centre, 28th Floor  
111 Connaught Road Central  
Tel 5 8521 222  
TELEX 74766 SCHMC HK

### INDIA

Micronic Devices  
104/109C, Nirmal Industrial Estate  
Sion (E)  
Bombay 400022  
Tel 486-170  
TELEX 011-71447 MDEV IN

### JAPAN

Asahi Electronics Co Ltd  
KMM Bldg Room 407  
2-14-1 Asano, Kofu  
Kita-ku, Kitakyushu City 802  
Tel (093) 511-6471  
TELEX AECKY 7126-16

### JAPAN (Cont'd)

Hamilton-Annet Electronics Japan Ltd  
YU and YOU Bldg 1-4 Horidome-  
cho  
Nihonbashi Chuo-Ku, Tokyo 103  
Tel (03) 862-9911  
TELEX 2523774

Ryoyo Electric Corp  
Konwa Bldg  
1-12-22, Tsukiji  
Chuo-Ku, Tokyo 104  
Tel (03) 543-7711/541-7311  
Tokyo Electron Ltd  
Shin Juku, Nomura Bldg  
26-2 Nishi-Shin Juku-ichome  
Shin Juku-Ku, Tokyo 160  
Tel (03) 343-4411  
TELEX 232-2220 LABTEL J

### KOREA

Koram Digital  
2nd Floor, Government Pension Bldg  
1-589, Yoido-Dong  
Youngangpo-Ku  
Seoul 150  
Tel 782-8039 or 8049  
TELEX KODIGI K25 299

### NEW ZEALAND

McLean Information Technology Ltd  
459 Kyber Pass Road, Newmarket,  
P O Box 9464, Newmarket  
Auckland 1, New Zealand  
Tel 501-801, 501-219, 587-037  
TELEX NZ21570 THERMAL

### SINGAPORE

General Engineers Corporation Pty  
Ltd  
18 Peair Panjang Road  
11-05/08 PSA Multi-Storey Complex  
Singapore 0511  
Tel 011-65-271-3163  
TELEX RS23987 GENERCO

### SOUTH AFRICA

Electronic Building Elements, Pty Ltd  
P O Box 4609  
Hazelwood, Pretoria 0001  
Tel 011-27-12-46-9221 or 9227  
TELEX 3-0181 SA

### TAIWAN

Taiwan Automation Corp \*  
3rd Floor #75, Section 4  
Nanking East Road  
Taipei  
Tel 771-0940 or 0941  
TELEX 11942 TAUAUTO

### YUGOSLAVIA

H R Microelectronics Enterprises  
P O Box 5604  
San Jose, California 95150  
Tel (408) 978-8000  
TELEX 278-559

\*Field Application Location



## U.S. SERVICE OFFICES

### CALIFORNIA

Intel Corp  
1350 Shorebird Way  
Mt View 94043  
Tel (415) 968-8211  
TWX 910-338-9279  
910-338-0255

Intel Corp  
2000 E 4th Street  
Suite 110  
Santa Ana 92705  
Tel (714) 835-5577  
TWX 910-595-2475

Intel Corp  
7670 Opportunity Road  
San Diego 92111  
Tel (714) 268-3563

Intel Corp  
5530 N Corbin Avenue  
Suite 120  
Tarzana 91356  
Tel (213) 708-0333

### COLORADO

Intel Corp  
850 South Cherry  
Suite 720  
Denver 80222  
Tel (303) 321-8086  
TWX 910-931-2289

### CONNECTICUT

Intel Corp  
36 Padanaram Road  
Danbury 06810  
Tel (203) 792-8366

### FLORIDA

Intel Corp  
1500 N W 62nd Street  
Suite 104  
Ft Lauderdale 33309  
Tel (305) 771-0600  
TWX 510-956-9407

Intel Corp  
500 N Mailand Avenue  
Suite 205  
Mailand 32751  
Tel (305) 628-2393  
TWX 910-853-9219

Intel Corp  
5151 Adanson Street  
Orlando 32804  
Tel (305) 628-2393

### GEORGIA

Intel Corp  
3300 Holcombe Bridge Road  
Suite 225  
Norcross 30092  
Tel (404) 441-1171

### ILLINOIS

Intel Corp  
2550 Golf Road  
Suite 815  
Rolling Meadows 60008  
Tel (312) 991-7270  
TWX 910-253-1825

### KANSAS

Intel Corp  
8400 W 110th Street  
Suite 170  
Overland Park 66210  
Tel (913) 642-8080

### MARYLAND

Intel Corp 7257 Parkway Drive  
Hanover 21076  
Tel (301) 796-7500  
TWX 710-862-1944

### MASSACHUSETTS

Intel Corp  
27 Industrial Avenue  
Chelmsford 01824  
Tel (617) 636-1800  
TWX 710-343-8333

### MICHIGAN

Intel Corp  
26500 Northwestern Highway  
Suite 401  
Southfield 48075  
Tel (313) 354-1540  
TWX 810-244-4915

### MINNESOTA

Intel Corp  
7401 Metro Boulevard  
Suite 355  
Edina 55435  
Tel (612) 835-6722  
TWX 910-567-2867

### MISSOURI

Intel Corp  
4203 Earth City Expressway  
Suite 143  
Earth City 63045  
Tel (314) 291-2015

### NEW JERSEY

Intel Corp  
385 Sylvan Avenue  
Englewood Cliffs 07632  
Tel (201) 567-0820  
TWX 710-991-8593

### NEW YORK

Intel Corp  
2255 Lyell Avenue  
Fochester 14606  
Tel (716) 254-6120

### NORTH CAROLINA

Intel Corp  
5600 Executive Drive  
Suite 113  
Charlotte 28212  
Tel (704) 568-8966

Intel Corp  
2306 W Meadowview Road  
Suite 206  
Greensboro 27407  
Tel (919) 294-1541

### OHIO

Intel Corp  
Chagrin-Brainard Blvd  
Suite 305  
28001 Chagrin Boulevard  
Cleveland 44122  
Tel (216) 464-6915  
TWX 910-427-9236

Intel Corp  
6500 Poe Avenue  
Dayton 45414  
Tel (603) 325-4415  
TWX 910-450-2528

### OKLAHOMA

Intel Corp  
4157 S Harward  
Suite 123  
Tulsa 74101  
Tel (918) 744-8068

### OREGON

Intel Corp  
10700 S W Beaverton-Hilledale  
Highway  
Suite 22  
Beaverton 97005  
Tel (503) 641-8086  
TWX 910-467-8741

### PENNSYLVANIA

Intel Corp  
500 Pennsylvania Avenue  
Fort Washington 19034  
Tel (215) 641-1000  
TWX 510-661-2077

Intel Corp  
201 Penn Center Boulevard  
Suite 301 W  
Pittsburgh 15235  
Tel (313) 354-1540

### TEXAS

Intel Corp  
313 E Anderson Lane  
Suite 314  
Austin 78752  
Tel (612) 454-3628  
TWX 910-874-1347

Intel Corp  
12300 Ford Road  
Suite 380  
Dallas 75234  
Tel (214) 241-8087  
TWX 910-660-5617

Intel Corp  
7322 S W Freeway  
Suite 1490  
Houston 77074  
Tel (713) 986-8098  
TWX 910-881-2490

### VIRGINIA

Intel Corp  
7700 Leesburg Pike  
Suite 412  
Falls Church 22043  
Tel (703) 734-9707  
TWX 710-931-0625

### WASHINGTON

Intel Corp  
110 110th Avenue N.E.  
Suite 510  
Bellevue 98004  
Tel 1-800-538-0662  
TWX 910-443-3002

### WISCONSIN

Intel Corp  
150 S Sunnyslope Road  
Suite 148  
Brookfield 53005  
Tel (414) 784-9060



Intel Corporation  
3065 Bowers Avenue  
Santa Clara, CA 95051

Intel International (U.K.) Ltd.  
Piper's Way  
Swindon, SN3 1RJ  
Wiltshire, England

Intel Japan K.K.  
5-6 Tokodai Toyosato-machi  
Tsukuba-gun, Ibaraki-ken 300-26  
Japan