intel® Series 3000 Reference Manual

$5.00

98-221A

intel

Series 3000 Family Of Computing Elements — The Total System Solution.

Since its introduction in September, 1974, the Series 3000 family of computing elements has found acceptance in a wide range of high performance applications from disk controllers to airborne CPU's.

The Series 3000 family represents more than a simple collection of bipolar components, it is a complete family of computing elements and hardware/software support that greatly simplifies the task of transforming a design from concept to production.

The Series 3000 Component Family

A complete set of computing elements that are designed as a system requiring a minimum amount of ancillary circuitry.

3001	Microprogram Control Unit.
3002	Central Processing Element.
3003	Look-Ahead Carry Generator.
3212	Multi-Mode Latch Buffer.
3214	Interrupt Control Unit.
3216/26	Parallel Bi-directional Bus Driver.
ROMs/PROMs	A complete set of bipolar ROMs and PROMs.
RAMs	A Complete family of MOS and bipolar RAMs.

The Series 3000 Support

A comprehensive support system that assists the designer in writing microprograms, debugging hardware and microcode, and programming prototype and production PROMs.

CROMIS	Cross microprogram assembler.
MDS-800	Microcomputer development system with TTY/CRT, line printer, diskette, PROM programmer and high speed paper tape reader facilities.
ICE-30	In-circuit emulation for the 3001 MCU.
ROM-SIM	ROM simulation for all of Intel's Bipolar ROMs and PROMs.
Application Notes	Central processor and disk controller designs and system timing considerations.
Customer Course	Comprehensive 3 day course covering the component family, CPU and controller designs, microprogramming and the MDS-800, ICE-30 and ROM-SIM operation.

The Series 3000 family is designed to provide a Total System Solution: high performance, minimum package count and total commitment to support.

Series 3000 Reference Manual

Contents

INTRODUCTION

A family architecture

To reduce component count as far as practical, a multi-chip LSI microcomputer set must be designed as a complete, compatible family of devices. The omission of a bus or a latch or the lack of drive current can multiply the number of miscellaneous SSI and MSI packages to a dismaying extent—witness the reputedly LSI minicomputers now being offered which need over a hundred extra TTL packages on their processor boards to support one or two custom LSI devices. Successful integration should result in a minimum of extra packages, and that includes the interrupt and the input/output systems.

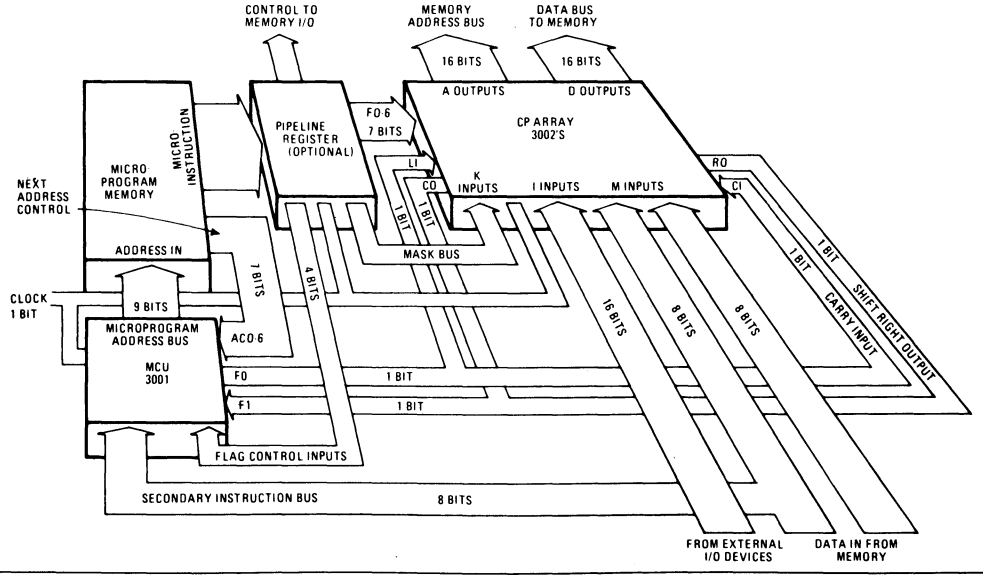
With this objective in mind, the Intel Schottky bipolar LSI microcomputer chip set was developed. Its two major components, the 3001 Microprogram Control Unit (MCU) and the 3002 Central Processing Element (CPE), may be combined by the digital designer with standard bipolar LSI memory to construct high-performance controller-processors (Fig. 1) with a minimum of ancillary logic.

Among the features that minimize package count and improve performance are: the multiple independent data and address busses that eliminate time multiplexing and the need for external latches; the three-state output buffers with high fanout that make bus drivers unnecessary except in the largest systems, and the separate output-enable logic that permits bidirectional busses to be formed simply by connecting inputs and outputs together.

Each CPE represents a complete two-bit slice through the data-processing section of a computer. Several CPEs may be arrayed in parallel to form a processor of any desired word length. The MCU, which together with the microprogram memory, controls the step-by-step operation of the processor, is itself a powerful microprogramed state sequencer.

Enhancing the performance and capabilities of these two components are a number of compatible computing elements. These include a fast look-ahead carry generator, a priority interrupt unit, and a multimode latch buffer. A complete summary of the first available members of this family of LSI computing elements and memories is given in the table on this page.

3001	Microprogram control unit
3002	Central processing element
3003	Look-ahead carry generator
3212	Multimode latch buffer
3214	Priority interrupt unit
3216	Noninverting bidirectional bus driver
3226	Inverting bidirectional bus driver
3601	256-by-4-bit programable read-only memory
3604	512-by-8-bit programable read-only memory
3301A	256-by-4-bit read-only memory
3304A	512-by-8-bit read-only memory



1. Bipolar microcomputer. Block diagram shows how to implement a typical 16-bit controller-processor with new family of bipolar computer elements. An array of eight central processing elements (CPEs) is governed by a microprogram control unit (MCU) through a separate read-only memory that carries the microinstructions for the various processing elements. This ROM may be a fast, off-the-shelf unit.

CPEs form a processor

Each CPE (Fig. 2) carries two bits of five independent busses. The three input busses can be used in several different ways. Typically, the K-bus is used for microprogram mask or literal (constant) value input, while the other two input busses, M and I, carry data from external memory or input/output devices. D-bus outputs are connected to the CPE accumulator; A-bus outputs are connected to the CPE memory address register. As the CPEs are wired together, all the data paths, registers, and busses expand accordingly.

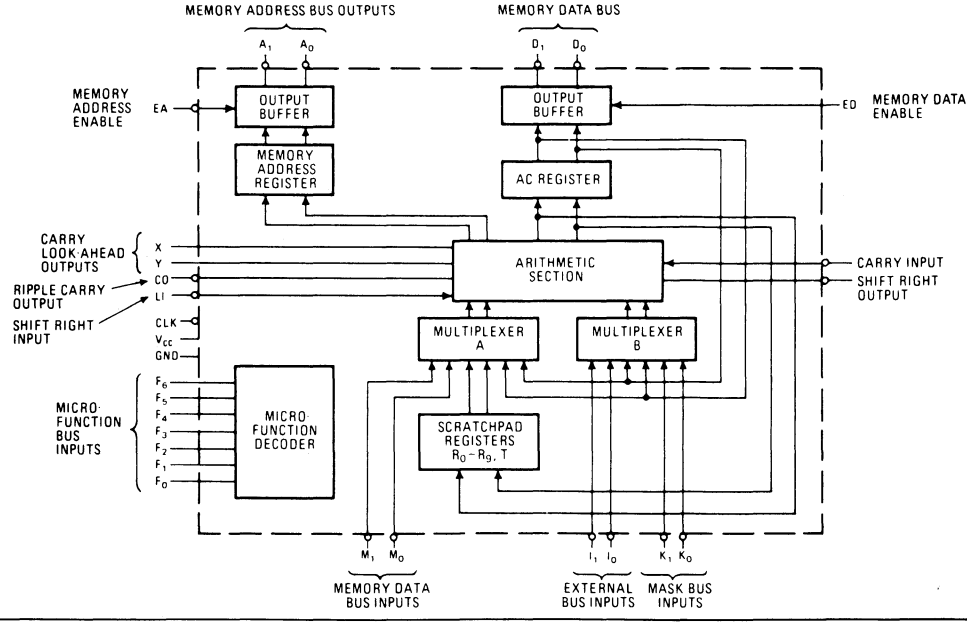
Certain data operations can be performed simply by connecting the busses in a particular fashion. For example, a byte exchange operation, often used in data-communications processors, may be carried out by wiring the D-bus outputs back to the I-bus inputs, exchanging the high-order outputs and low-order inputs. Several other discretionary shifts and rotates can be accomplished in this manner.

A sixth CPE bus, the seven-line microfunction bus, controls the internal operation of the CPE by selecting the operands and the operation to be performed. The arithmetic function section, under control of the microfunction bus decoder, performs over 40 Boolean and binary functions, including 2's complement arithmetic and logical AND, OR, NOT, and exclusive-NOR. It increments, decrements, shifts left or right, and tests for zero.

Unlike earlier MSI arithmetic-logic units, which contain many functions that are rarely used, the microfunction decoder selects only useful CPE operations. Standard carry look-ahead outputs, X and Y, are generated by the CPE for use with available look-ahead devices or the Intel 3003 Look-ahead Carry Generator. Independent carry input, carry output, shift input, and shift output lines are also available.

What's more, since the K-bus inputs are always ANDed with the B-multiplexer outputs into the arithmetic function section, a number of useful functions that in conventional MSI ALUs would require several cycles are generated in a single CPE microcycle. The type of bit masking frequently done in computer control systems can be performed with the mask supplied to the K-bus directly from the microinstruction.

Placing the K-bus in either the all-one or all-zero state will, in most cases, select or deselect the accumulator in the operation, respectively. This toggling effect of the K-bus on the accumulator nearly doubles the CPE's repertoire of microfunctions. For instance, with the K-bus in the all-zero state, the data on the M-bus may be complemented and loaded into the CPE's accumulator. The same function selected with the K-bus in the all-one state will exclusive-NOR the data on the M-bus with the accumulator contents.



2. Central processing element. This element contains all the circuits representing a two-bit-wide slice through a small computer's central processor. To build a processor of word width N, all that's necessary is to connect an array of N/2 CPEs together.

Three Innovations

The power and versatility of the CPE are increased by three rather novel techniques. The first of these is the use of the carry lines and logic during non-arithmetic operations for bit testing and zero detection. The carry circuits during these operations perform a word-wide logical OR (ORing adjacent bits) of a selected result from the arithmetic section. The value of the OR, called the carry OR, is passed along the carry lines to be ORed with the result of an identical operation taking place simultaneously in the adjacent higher-order CPE.

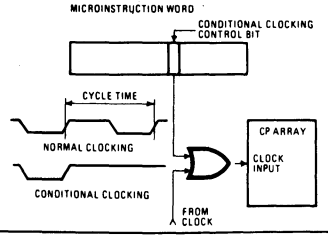
Obviously, the presence of at least one bit in the logical 1 state will result in a true carry output from the highest-order CPE. This output, as explained later, can be used by the MCU to determine which microprogram sequence to follow. With the ability to mask any desired bit, or set of bits, via the K-bus inputs included in the carry OR, a powerful bit-testing and zero-detection facility is realized.

The second novel CPE feature is the use of three-state outputs on the shift right output (RO) and carry output (CO) lines. During a right shift operation, the CO line is placed in the high-impedance (Z) state, and the shift data is active on the RO line. In all other CPE operations, the RO line is placed in the Z state, and the carry data is active on the CO line. This permits the CO and RO lines to be tied together and sent as a single rail input to the MCU for testing and branching. Left shift operations utilize the carry lines, rather than the shift lines, to propagate data.

The third novel CPE capability, called conditional clocking, saves microcode and microcycles by reducing the number of microinstructions required to perform a given test. One extra bit is used in the microinstruction to selectively control the gating of the clock pulse to the central processor (CP) array. Momentarily freezing the clock (Fig. 3) permits the CPE microfunction to be performed, but stops the results from being clocked into the specified registers. The carry or shift data that results from the operation is available because the arithmetic section is combinatorial, rather than sequential. The data can be used as a jump condition by the MCU and in this way permits a variety of nondestructive tests to be performed on register data.

Microprogram control

The classic form of microprogram control incorporates a next-address field in each microinstruction—any other approach would require some type of program counter. To simplify its logic, the MCU (Fig. 4) uses the classic approach and requires address control information from each microinstruction. This information is not, however, simply the next microprogram address. Rather, it is a highly encoded specification of the next address and one of a set of conditional tests on the MCU bus inputs and registers.



3. Conditional clock. This feature permits an extra bit in microinstruction to selectively control gating of clock pulse to CP array. Carry or shift data thus made available permits tests to be performed on data with fewer microinstructions.

The next-address logic and address control functions of the MCU are based on a unique scheme of memory addressing. Microprogram addresses are organized as a two-dimensional array or matrix. Unlike in ordinary memory, which has linearly sequenced addresses, each microinstruction is pinpointed by its row and column address in the matrix. The 9-bit microprogram address specifies the row address in the upper 5 bits and the column address in the lower 4 bits. The matrix can therefore contain up to 32 row addresses and 16 column addresses for a total of 512 microinstruction addresses.

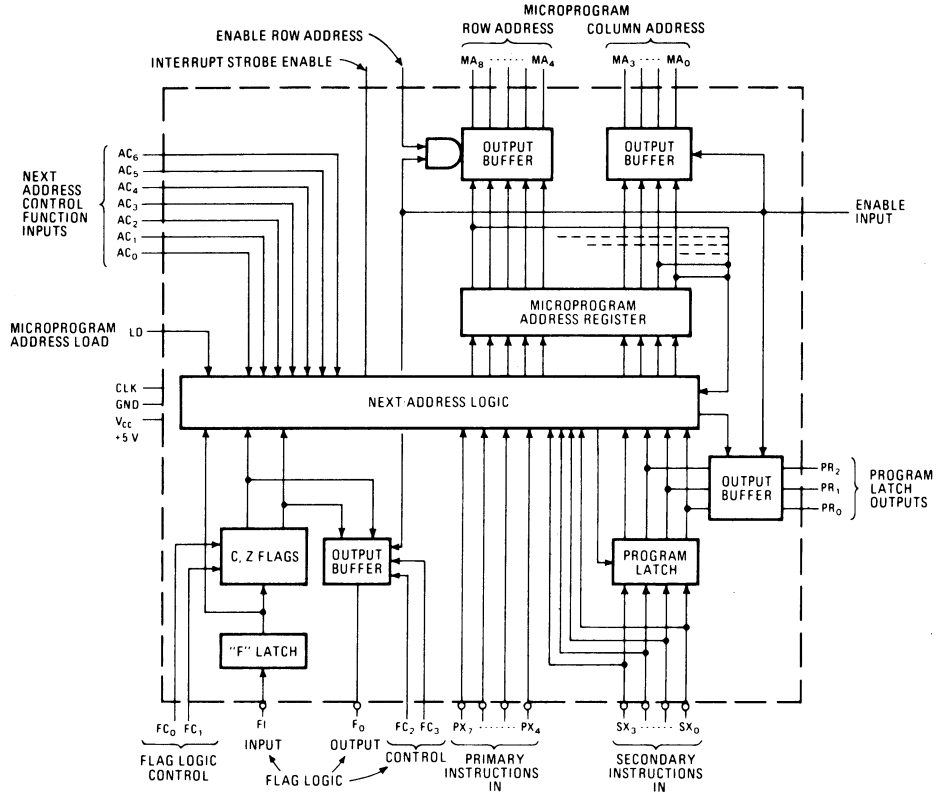
The next-address logic of the MCU makes extensive use of this addressing scheme. For example, from a particular row or column address, it is possible to jump either unconditionally to any other location in that row or column or conditionally to other specified locations, all in one operation. For a given location in the matrix there is a fixed subset of microprogram addresses that may be selected as the next address. These are referred to as a jump set, and each type of MCU address control jump function has a jump set associated with it.

Incorporating a jump operation in every microinstruction improves performance by allowing processing functions to be executed in parallel with program branches. Reductions in microcode are also obtained because common microprogram sequences can be shared without the time-space penalty usually incurred by conditional branching.

Independently controlled flag logic in the MCU is available for latching and controlling the value of the carry and shift inputs to the CP array. Two flags, called C and Z, are used to save the state of the flag input line. Under microprogram control, the flag logic simultaneously sets the state of the flag output line, forcing the line to logical 0, logical 1, or the value of the C or Z flag.

The jump decisions are made by the next-address logic on the basis of: the MCU's current microprogram address; the address control function on the accumulator inputs; and the data that's on the macroinstruction (X) bus or in the program latch or in the flags. Jump decisions may also be based on the instantaneous state of the flag input line without loading the value in one of the flags. This feature eliminates many extra microinstructions that would be required if only the flag flip-flop could be tested.

Microinstruction sequences are normally selected by the operation codes (op codes) supplied by the microinstructions, such as control commands or user instructions in main memory. The MCU decodes these commands by using their bit patterns to determine which is to be the next microprogram address. Each decoding results in a 16-way program branch to the desired microinstruction sequence.



4. Microprogram control unit. The MCU's two major control functions include controlling the sequence of microprograms fetched from the microprogram memory, and keeping track of the carry inputs and outputs of the CP array by means of the flag logic control.

Cracking the op codes

For instance, the MCU can be microprogramed to directly decode conventional 8-bit op codes. In these op codes the upper 4 bits specify one of up to 16 instruction classes or address modes, such as register, indirect, or indexed. The remaining bits specify the particular subclass such as ADD, SKIP IF ZERO, and so on. If a set of op codes is required to be in a different format, as may occur in a full emulation, an external pre-decoder, such as ROM, can be used in series with the X-bus to reformat the data for the MCU.

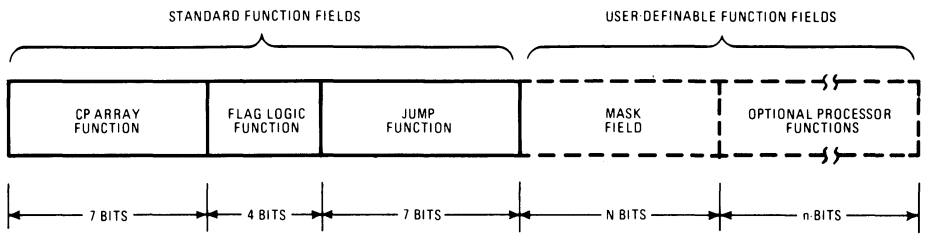
In rigorous decoding situations where speed or space is critical, the full 8-bit macroinstruction bus can be used for a single 256-way branch. Pulling down the load line of the MCU forces the 8 bits of data on the X-bus (typically generated by a predecoder) directly into the microprogram address register.

The data thus directly determines the next microprogram address which should be the start of the desired microprogram sequence. The load line may also be used by external logic to force the MCU, at power-up, into the system re-initialization sequence.

From time to time, a microprocessor must examine the state of its interrupt system to determine whether an interrupt is pending. If one is, the processor must suspend its normal execution sequence and enter an interrupt sequence in the microprogram. This requirement is handled by the MCU in a simple but elegant manner.

When the microprogram flows through address row 0 and column 15, the interrupt strobe enable line of the MCU is raised. The interrupt system, an Intel 3214 Interrupt Control Unit, responds by disabling the row address outputs of the MCU via the enable row address line, and by forcing the row entry address of the microprogram interrupt sequence onto the row address bus. The operation is normally performed just before the macroinstruction fetch cycle, so that a macroprogram is interrupted between, not during, macroinstructions.

The 9-bit microprogram address register and address bus of the MCU directly address 512 microinstructions. This is about twice as many as required by the typical 16-bit disk-controller or central processor.



5. Microinstruction format. Only a generalized microinstruction format can be shown since allocation of bits for the mask field and optional processor functions depends on the wishes of the designer and the tradeoffs he decides to make.

Moreover, multiple 512 microinstruction memory planes can easily be implemented simply by adding an extra address bit to the microinstruction each time the number of extra planes is doubled. Incidentally, as the number of bits in the microinstruction is increased, speed is not reduced. The additional planes also permit program jumps to take place in three address dimensions instead of two.

Because of the tremendous design flexibility offered by the Intel computing elements, it is impossible to describe every microinstruction format exactly. But generally speaking, the formats all derive from the one in Fig. 5. The minimum width is 18 bits: 7 bits for the address control functions, plus 4 bits for the flag logic control; plus 7 bits for the CPE microfunction control.

More bits can be added to the microinstruction format to provide such functions as mask field input to the CP array, external memory control, conditional clocking, and so on. Allocation of these bits is left to the designer who organizes the system. He is free to trade off memory costs, support logic, and microinstruction cycles to meet his cost/performance objectives.

Microprograming technology

- **Microprogram:** A type of program that directly controls the operation of each functional element in a microprocessor.
- **Microinstruction:** A bit pattern that is stored in a microprogram memory word and specifies the operation of the individual LSI computing elements and related subunits, such as main memory and input/output interfaces.
- **Microinstruction sequence:** The series of microinstructions that the microprogram control unit (MCU) selects from the microprogram to execute a single macroinstruction or control command. Microinstruction sequences can be shared by several macroinstructions.
- **Macroinstruction:** Either a conventional computer instruction (e.g. ADD MEMORY TO REGISTER, INCREMENT, and SKIP, etc.) or device controller command (e.g., SEEK, READ, etc.).

The cost/performance spectrum

The total flexibility of the Intel LSI computing elements is demonstrated by the broad cost/performance spectrum of the controllers and processors that can be constructed with them. These include:

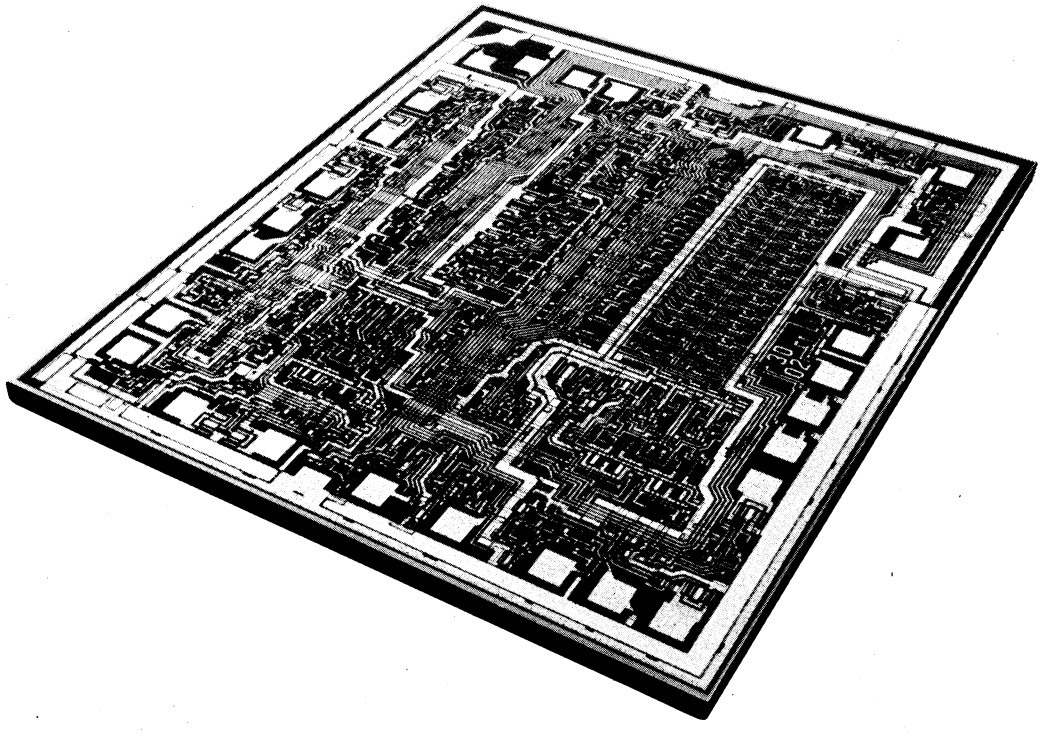
- High-speed controllers, built with a stand-alone ROM-MCU combination that sequences at up to 10 megahertz; it can be used without any CPEs as a system state controller.
- Pipelined look-ahead carry controller-processors, where the overlapped microinstruction fetch/execute cycles and fast-carry logic reduce the 16-bit add time to less than 125 nanoseconds.
- Ripple-carry controller processors (a 16-bit design adds the contents of two registers in 300 nanoseconds).
- Multiprocessors, or networks of any of the above controllers and processors, to provide computation, interrupt supervision, and peripheral control.

These configurations represent a range of microinstruction execution rates of from 3 million to 10 million instructions per second, or up to two orders of magnitude faster, for example, than p-channel microprocessors. Moreover, the increases in processor performance are achieved with relative simplicity. A ripple-carry 16-bit processor uses one MCU, eight CPEs, plus microprogram memory. One extra computing element, the 3003 Look-ahead Carry Generator, enhances the processor with fast carry. Increasing speed further by pipelining, the overlap of microinstruction fetch and execute cycles, requires a few D-type MSI flip-flops.

At the multiprocessor level, the microprogram memory, MCU, or CPE devices can be shared. A 16-bit processor, complete with bus control and microprogram memory, requires some 20 bipolar LSI packages and half that many small-scale ICs. In this configuration, it replaces an equivalent MSI TTL system having more than 200 packages.

Furthermore, systems built with this large-scale integrated circuitry are much smaller and less costly and consume less energy than equivalent designs using lower levels of transistor-transistor-logic integration. Even allowing for ancillary logic circuits, the new bipolar computing elements cut 60% to 80% off the package count in realizing most of today's designs made with small- or medium-scale-integrated TTL.

COMPONENT FAMILY



SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

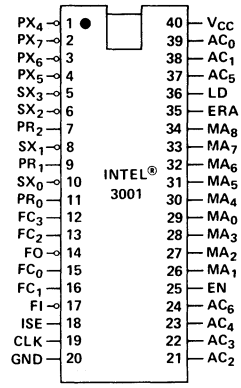
3001 MICROPROGRAM CONTROL UNIT

The INTEL® 3001 Microprogram Control Unit (MCU) controls the sequence in which microinstructions are fetched from the microprogram memory. Its functions include the following:

- Maintenance of the microprogram address register.
- Selection of the next microinstruction based on the contents of the microprogram address register.
- Decoding and testing of data supplied via several input busses to determine the microinstruction execution sequence.
- Saving and testing of carry output data from the central processor (CP) array.
- Control of carry/shift input data to the CP array.
- Control of microprogram interrupts.

- High Performance – 85 ns Cycle Time
- TTL and DTL Compatible
- Fully Buffered Three-State and Open Collector Outputs
- Direct Addressing of Standard Bipolar PROM or ROM
- 512 Microinstruction Addressability
- Advanced Organization
  - 9-Bit Microprogram Address Register and Bus
  - 4-Bit Program Latch
  - Two Flag Registers
- Eleven Address Control Functions
  - Three Jump and Test Latch Functions
  - 16-way Jump and Test Instruction Bus Function
- Eight Flag Control Functions
  - Four Flag Input Functions
  - Four Flag Output Functions
- 40 Pin DIP

PACKAGE CONFIGURATION



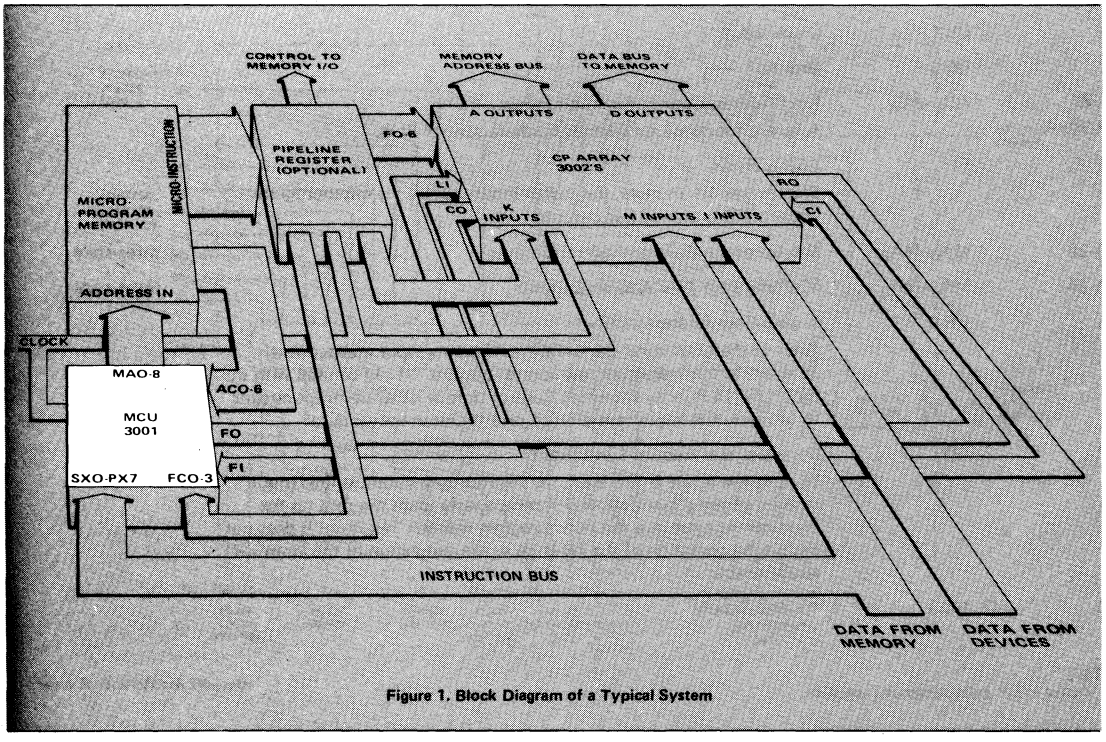


Figure 1. Block Diagram of a Typical System

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE[1]
1-4	PX_4-PX_7	Primary Instruction Bus Inputs Data on the primary instruction bus is tested by the JPX function to determine the next microprogram address.	active LOW
5, 6, 8, 10	SX_0-SX_3	Secondary Instruction Bus Inputs Data on the secondary instruction bus is synchronously loaded into the PR-latch while the data on the PX-bus is being tested (JPX). During a subsequent cycle, the contents of the PR-latch may be tested by the JPR, JLL, or JRL functions to determine the next microprogram address.	active LOW
7, 9, 11	PR_0-PR_2	PR-Latch Outputs The PR-latch outputs are asynchronously enabled by the JCE function. They can be used to modify microinstructions at the outputs of the microprogram memory or to provide additional control lines.	open collector
12, 13, 15, 16	FC_0-FC_3	Flag Logic Control Inputs The flag logic control inputs are used to cross-switch the flags (C and Z) with the flag logic input (FI) and the flag logic output (FO).	
14	FO	Flag Logic Output The outputs of the flags (C and Z) are multiplexed internally to form the common flag logic output. The output may also be forced to a logical 0 or logical 1.	active LOW three-state
17	FI	Flag Logic Input The flag logic input is demultiplexed internally and applied to the inputs of the flags (C and Z). Note: the flag input data is saved in the F-latch when the clock input (CLK) is low.	active LOW
18	ISE	Interrupt Strobe Enable Output The interrupt strobe enable output goes to logical 1 when one of the JZR functions are selected (see Functional Description, page 6). It can be used to provide the strobe signal required by the INTEL 3214 Priority Interrupt Control Unit or other interrupt circuits.	
19	CLK	Clock Input	
20	GND	Ground	
21-24 37-39	AC_0-AC_6	Next Address Control Function Inputs All jump functions are selected by these control lines.	
25	EN	Enable Input When in the HIGH state, the enable input enables the microprogram address, PR-latch and flag outputs.	
26-29	MA_0-MA_3	Microprogram Column Address Outputs	three-state
30-34	MA_4-MA_8	Microprogram Row Address Outputs	three-state
35	ERA	Enable Row Address Input When in the LOW state, the enable row address input independently disables the microprogram row address outputs. It can be used with the INTEL 3214 Priority Interrupt Control Unit or other interrupt circuits to facilitate the implementation of priority interrupt systems.	
36	LD	Microprogram Address Load Input When in the active HIGH state, the microprogram address load input inhibits all jump functions and synchronously loads the data on the instruction busses into the microprogram register. However, it does not inhibit the operation of the PR-latch or the generation of the interrupt strobe enable.	
40	VCC	+5 Volt Supply	

NOTE:
(1) Active HIGH unless otherwise specified.

LOGICAL DESCRIPTION

The MCU performs two major control functions. First, it controls the sequence in which microinstructions are fetched from the microprogram memory. For this purpose, the MCU contains a microprogram address register and the associated logic for selecting the next microinstruction address. The second function of the MCU is the control of the two flag flip-flops that are included for interaction with the carry input and carry output logic of the CP array. The logical organization of the MCU is shown in Figure 2.

NEXT ADDRESS LOGIC

The next address logic of the MCU provides a set of conditional and unconditional address control functions. These address control functions are used to implement a jump or jump/test operation as part of every microinstruction. That is to say, each microinstruction typically contains a jump operation field that specifies the address control function, and hence, the next microprogram address.

In order to minimize the pin count of the MCU, and reduce the complexity of the next address logic, the microprogram address space is organized as a two dimensional array or matrix. Each microprogram address corresponds to a unit of the matrix at a particular row and column location. Thus, the 9-bit microprogram address is treated as specifying not one, but two addresses – the row address in the upper five bits and the column address in the lower four bits. The address matrix can therefore contain, at most, 32 row addresses and 16 column addresses for a total of 512 microinstructions.

The next address logic of the MCU makes extensive use of this two component addressing scheme. For example, from a particular row or column address, it is possible to jump unconditionally in one operation anywhere in that row or column. It is not possible, however, to jump anywhere in the address matrix. In fact, for a given location in the matrix, there is a fixed subset of microprogram addresses that may be selected as the next address. These possible jump target addresses are referred to as a jump set. Each type of MCU address control (jump) function has a jump set associated with it. Appendix C illustrates the jump set for each function.

FLAG LOGIC

The flag logic of the MCU provides a set of functions for saving the current value of the carry output of the CP array and for controlling the value of the carry input to the CP array. These two distinct flag control functions are called flag input functions and flag output functions.

The flag logic is comprised of two flip-flops, designated the C-flag and the Z-flag, along with a simple latch, called the F-latch, that indicates the current state of the carry output line of the CP array. The flag logic is used in conjunction with the carry and shift logic of the CP array to implement a variety of shift/rotate and arithmetic functions.

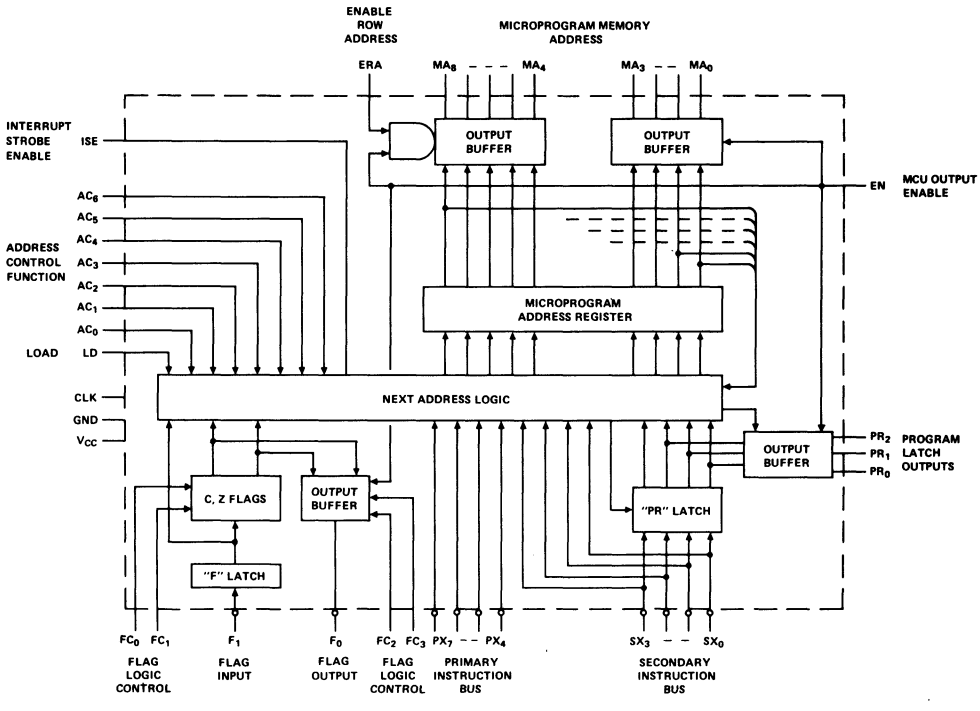


Figure 2. 3001 Block Diagram

FUNCTIONAL DESCRIPTION

ADDRESS CONTROL FUNCTIONS

The address control functions of the MCU are selected by the seven input lines designated AC_0-AC_6. On the rising edge of the clock, the 9-bit microprogram address generated by the next address logic is loaded into the microprogram address register. The next microprogram address is delivered to the microprogram memory via the nine output lines designated MA_0-MA_8. The microprogram address outputs are organized into row and column addresses as:

$$\underbrace{MA_8\ MA_7\ MA_6\ MA_5\ MA_4}_{\text{row address}}$$

$$\underbrace{MA_3\ MA_2\ MA_1\ MA_0}_{\text{column address}}$$

Each address control function is specified by a unique encoding of the data on the function input lines. From three to five bits of the data specify the particular function while the remaining bits are used to select part of either the row or column address desired. Function code formats are given in Appendix A, "Address Control Function Summary."

The following is a detailed description of each of the eleven address control functions. The symbols shown below are used throughout the description to specify row and column addresses.

Symbol	Meaning
row_n	5-bit next row address where n is the decimal row address.
col_n	4-bit next column address where n is the decimal column address.

UNCONDITIONAL ADDRESS CONTROL (JUMP) FUNCTIONS

The jump functions use the current microprogram address (i.e., the contents of the microprogram address register prior to the rising edge of the clock) and several bits from the address control inputs to generate the next microprogram address.

Mnemonic	Function Description
JCC	Jump in current column. AC_0-AC_4 are used to select 1 of 32 row addresses in the current column, specified by MA_0-MA_3, as the next address
JZR	Jump to zero row. AC_0-AC_3 are used to select 1 of 16 column addresses in row_0, as the next address.
JCR	Jump in current row. AC_0-AC_3 are used to select 1 of 16 addresses in the current row, specified by MA_4-MA_8, as the next address.
JCE	Jump in current column/row group and enable PR-latch outputs. AC_0-AC_2 are used to select 1 of 8 row addresses in the current row group, specified by MA_7-MA_8, as the next row address. The current column is specified by MA_0-MA_3. The PR-latch outputs are asynchronously enabled.

FLAG CONDITIONAL ADDRESS CONTROL (JUMP/TEST) FUNCTIONS

The jump/test flag functions use the current microprogram address, the contents of the selected flag or latch, and several bits from the address control function to generate the next microprogram address.

Mnemonic	Function Description
JFL	Jump/test F-Latch. AC_0-AC_3 are used to select 1 of 16 row addresses in the current row group, specified by MA_8, as the next row address. If the current column group, specified by MA_3, is col_0-col_7, the F-latch is used to select col_2 or col_3 as the next column address. If MA_3 specifies column group col_8-col_{15}, the F-latch is used to select col_{10} or col_{11} as the next column address.
JCF	Jump/test C-flag. AC_0-AC_2 are used to select 1 of 8 row addresses in the current row group, specified by MA_7 and MA_8, as the next row address. If the current column group specified by MA_3 is col_0-col_7, the C-flag is used to select col_2 or col_3 as the next column address. If MA_3 specifies column group col_8-col_{15}, the C-flag is used to select col_{10} or col_{11} as the next column address.
JZF	Jump/test Z-flag. Identical to the JCF function described above, except that the Z-flag, rather than the C-flag, is used to select the next column address.

PX-BUS AND PR-LATCH CONDITIONAL ADDRESS CONTROL (JUMP/TEST) FUNCTIONS

The PX-bus jump/test function uses the data on the primary instruction bus (PX_4-PX_7), the current mircoprogram address, and several selection bits from the address control function to generate the next microprogram address. The PR-latch jump/test functions use the data held in the PR-latch, the current microprogram address, and several selection bits from the address control function to generate the next microprogram address.

Mnemonic	Function Description
JPR	Jump/test PR-latch. AC_0-AC_2 are used to select 1 of 8 row addresses in the current row group, specified by MA_7 and MA_8, as the next row address. The four PR-latch bits are used to select 1 of 16 possible column addresses as the next column address.

Mnemonic	Function Description
JLL	Jump/test leftmost PR-latch bits. AC_0-AC_2 are used to select 1 of 8 row addresses in the current row group, specified by MA_7 and MA_8, as the next row address. PR_2 and PR_3 are used to

FUNCTIONAL DESCRIPTION (con't)

select 1 of 4 possible column addresses in col_4 through col_7 as the next column address.

JRL — Jump/test rightmost PR-latch bits. AC_0 and AC_1 are used to select 1 of 4 high-order row addresses in the current row group, specified by MA_7 and MA_8, as the next row address. PR_0 and PR_1 are used to select 1 of 4 possible column addresses in col_{12} through col_{15} as the next column address.

JPX — Jump/test PX-bus and load PR-latch. AC_0 and AC_1 are used to select 1 of 4 row addresses in the current row group, specified by MA_6-MA_8, as the next row address. PX_4-PX_7 are used to select 1 of 16 possible column addresses as the next column address. SX_0-SX_3 data is locked in the PR-latch at the rising edge of the clock.

FLAG CONTROL FUNCTIONS

The flag control functions of the MCU are selected by the four input lines designated FC_0-FC_3. Function code formats are given in Appendix B, "Flag Control Function Summary."

The following is a detailed description of each of the eight flag control functions.

FLAG INPUT CONTROL FUNCTIONS

The flag input control functions select which flag or flags will be set to the current value of the flag input (FI) line. Data on FI is stored in the F-latch when the clock is low. The content of the F-latch is loaded into the C and/or Z flag on the rising edge of the clock.

Mnemonic	Function Description
SCZ	Set C-flag and Z-flag to FI. The C-flag and the Z-flag are both set to the value of FI.
STZ	Set Z-flag to FI. The Z-flag is set to the value of FI. The C-flag is unaffected.
STC	Set C-flag to FI. The C-flag is set to the value of FI. The Z flag is unaffected.
HCZ	Hold C-flag and Z-flag. The values in the C-flag and Z-flag are unaffected.

FLAG OUTPUT CONTROL FUNCTIONS

The flag output control functions select the value to which the flag output (FO) line will be forced.

Mnemonic	Function Description
FF0	Force FO to 0. FO is forced to the value of logical 0.
FFC	Force FO to C. FO is forced to the value of the C-flag.
FFZ	Force FO to Z. FO is forced to the value of the Z-flag.
FF1	Force FO to 1. FO is forced to the value of logical 1.

LOAD AND INTERRUPT STROBE FUNCTIONS

The load function of the MCU is controlled by the input line designated LD. If the LD line is active HIGH at the rising edge of the clock, the data on the primary and secondary instruction busses, PX_4-PX_7 and SX_0-SX_3, is loaded into the microprogram address register. PX_4-PX_7 are loaded into MA_0-MA_3 and SX_0-SX_3 are loaded into MA_4-MA_7. The high-order bit of the microprogram address register MA_8 is set to a logical 0. The bits from the primary instruction bus select 1 of 16 possible column addresses. Likewise, the bits from the secondary instruction bus select 1 of the first 16 row addresses.

The interrupt strobe enable of the MCU is available on the output line designated ISE. The line is placed in the active high state whenever a JZR to col_{15} is selected as the address control function. Customarily, the start of a macroinstruction fetch sequence is situated at row_0 and col_{15} so that the INTEL 3214 Priority Interrupt Control Unit may be enabled at the beginning of the fetch/execute cycle. The priority interrupt control unit may respond to the interrupt by pulling the enable row address (ERA) input line down to override the selected next row address from the MCU. Then by gating an alternative next row address on to the row address lines of the microprogram memory, the microprogram may be forced to enter an interrupt handling routine. The alternative row address placed on the microprogram memory address lines does not alter the contents of the microprogram address register. Therefore, subsequent jump functions will utilize the row address in the register, and not the alternative row address, to determine the next microprogram address.

Note, the load function always overrides the address control function on AC_0-AC_6. It does not, however, override the latch enable or load sub-functions of the JCE or JPX instruction, respectively. In addition, it does not inhibit the interrupt strobe enable or any of the flag control functions.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . −65°C to +160°C

All Output and Supply Voltages . . . . . −0.5V to +7V

All Input Voltages . . . . . −1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may affect device reliability.

T_A = 0°C to 70°C V_{CC} = 5.0V ±5%

SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		−0.8	−1.0	V	I_C = −5 mA
I_F	Input Load Current:					
	CLK Input		−0.075	−0.75	mA	V_F = 0.45V
	EN Input		−0.05	−0.50	mA	
	All Other Inputs		−0.025	−0.25	mA	
I_R	Input Leakage Current:					
	CLK			120	μA	V_R = 5.25V
	EN Input			80	μA	
	All Other Inputs			40	μA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	
I_{CC}	Power Supply Current (2)		170	240	mA	
V_{OL}	Output Low Voltage (All Output Pins)		0.35	0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage (MA_0-MA_8, ISE, FO)	2.4	3.0		V	I_{OH} = −1 mA
I_{OS}	Output Short Circuit Current (MA_0-MA_8, ISE, FO)	−15	−28	−60	mA	V_{CC} = 5.0V
$I_{O(off)}$	Off-State Output Current:					
	MA_0–MA_8, FO			−100	μA	V_O = 0.45V
	MA_0–MA_8, FO, PR_0–PR_2			100	μA	V_O = 5.25V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) EN input grounded, all other inputs and outputs open.

A.C. CHARACTERISTICS AND WAVEFORMS T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%

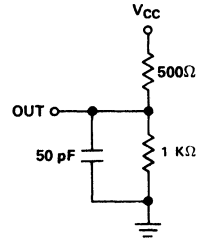
SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT
t_{CY}	Cycle Time (2)	85	60		ns
t_{WP}	Clock Pulse Width	30	20		ns
	Control and Data Input Set-Up Times:				
t_{SF}	LD, AC_0-AC_6	10	0		ns
t_{SK}	FC_0, FC_1	0			ns
t_{SX}	SX_0-SX_3, PX_4-PX_7	35	25		ns
t_{SI}	FI	15	5		ns
	Control and Data Input Hold Times:				
t_{HF}	LD, AC_0-AC_6	5	0		ns
t_{HK}	FC_0, FC_1	0			ns
t_{HX}	SX_0-SX_3, PX_4-PX_7	20	5		ns
t_{HI}	FI	20	8		ns
t_{CO}	Propagation Delay from Clock Input (CLK) to Outputs (MA_0-MA_8, FO)	10	30	45	ns
t_{KO}	Propagation Delay from Control Inputs FC_2 and FC_3 to Flag Out (FO)		16	30	ns
t_{FO}	Propagation Delay from Control Inputs AC_0-AC_6 to Latch Outputs (PR_0-PR_2)		26	40	ns
t_{EO}	Propagation Delay from Enable Inputs EN and ERA to Outputs (MA_0-MA_8, FO, PR_0-PR_2)		21	32	ns
t_{FI}	Propagation Delay from Control Inputs AC_0-AC_6 to Interrupt Strobe Enable Output (ISE)		24	40	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) $t_{CY} = t_{WP} + t_{SF} + t_{CO}$

TEST CONDITIONS:

Input pulse amplitude of 2.5 volts.
Input rise and fall times of 5 ns between 1 volt and 2 volts.
Output load of 10 mA and 50 pF.
Speed measurements are taken at the 1.5 volt level.

TEST LOAD CIRCUIT:



CAPACITANCE(2) T_A = 25°C

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
C_{IN}	Input Capacitance:				
	CLK, EN		11	16	pF
	All Other Inputs		5	10	pF
C_{OUT}	Output Capacitance		6	12	pF

NOTE:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5V and T_A = 25°C.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . −55°C to +125°C

Storage Temperature . . . . . −65°C to +160°C

All Output and Supply Voltages . . . . . −0.5V to +7V

All Input Voltages . . . . . −1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may affect device reliability.

T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%

SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		−0.8	-1.2	V	I_C = −5 mA
I_F	Input Load Current:					
	CLK Input		-75	-750	μA	V_F = 0.45V
	EN Input		-50	-500	μA	
	All Other Inputs		-25	-250	μA	
I_R	Input Leakage Current:					
	CLK			120	μA	V_R = 5.5V
	EN Input			80	μA	
	All Other Inputs			40	μA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	
I_{CC}	Power Supply Current (2)		170	250	mA	
V_{OL}	Output Low Voltage (All Output Pins)		0.35	0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage (MA_0-MA_8, ISE, FO)	2.4	3.0		V	I_{OH} = −1 mA
I_{OS}	Output Short Circuit Current (MA_0-MA_8, ISE, FO)	−15	−28	−60	mA	V_{CC} = 5.0V
$I_{O(off)}$	Off-State Output Current:					
	MA_0–MA_8, FO			−100	μA	V_O = 0.45V
	MA_0–MA_8, FO, PR_0–PR_2			100	μA	V_O = 5.5V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) EN input grounded, all other inputs and outputs open.

A.C. CHARACTERISTICS AND WAVEFORMS T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%

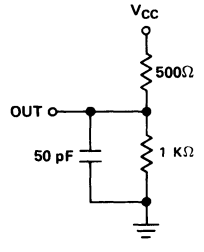
SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT
t_{CY}	Cycle Time (2)	95	60		ns
t_{WP}	Clock Pulse Width	40	20		ns
	Control and Data Input Set-Up Times:				
t_{SF}	LD, AC_0-AC_6	10	0		ns
t_{SK}	FC_0, FC_1	0			ns
t_{SX}	SX_0-SX_3, PX_4-PX_7	35	25		ns
t_{SI}	FI	15	5		ns
	Control and Data Input Hold Times:				
t_{HF}	LD, AC_0-AC_6	5	0		ns
t_{HK}	FC_0, FC_1	0			ns
t_{HX}	SX_0-SX_3, PX_4-PX_7	25	5		ns
t_{HI}	FI	22	8		ns
t_{CO}	Propagation Delay from Clock Input (CLK) to Outputs (MA_0-MA_8, FO)	10	30	45	ns
t_{KO}	Propagation Delay from Control Inputs FC_2 and FC_3 to Flag Out (FO)		16	50	ns
t_{FO}	Propagation Delay from Control Inputs AC_0-AC_6 to Latch Outputs (PR_0-PR_2)		26	50	ns
t_{EO}	Propagation Delay from Enable Inputs EN and ERA to Outputs (MA_0-MA_8, FO, PR_0-PR_2)		21	35	ns
t_{FI}	Propagation Delay from Control Inputs AC_0-AC_6 to Interrupt Strobe Enable Output (ISE)		24	40	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) $t_{CY} = t_{WP} + t_{SF} + t_{CO}$

TEST CONDITIONS:

Input pulse amplitude of 2.5 volts.
Input rise and fall times of 5 ns between 1 volt and 2 volts.
Output load of 10 mA and 50 pF.
Speed measurements are taken at the 1.5 volt level.

TEST LOAD CIRCUIT:

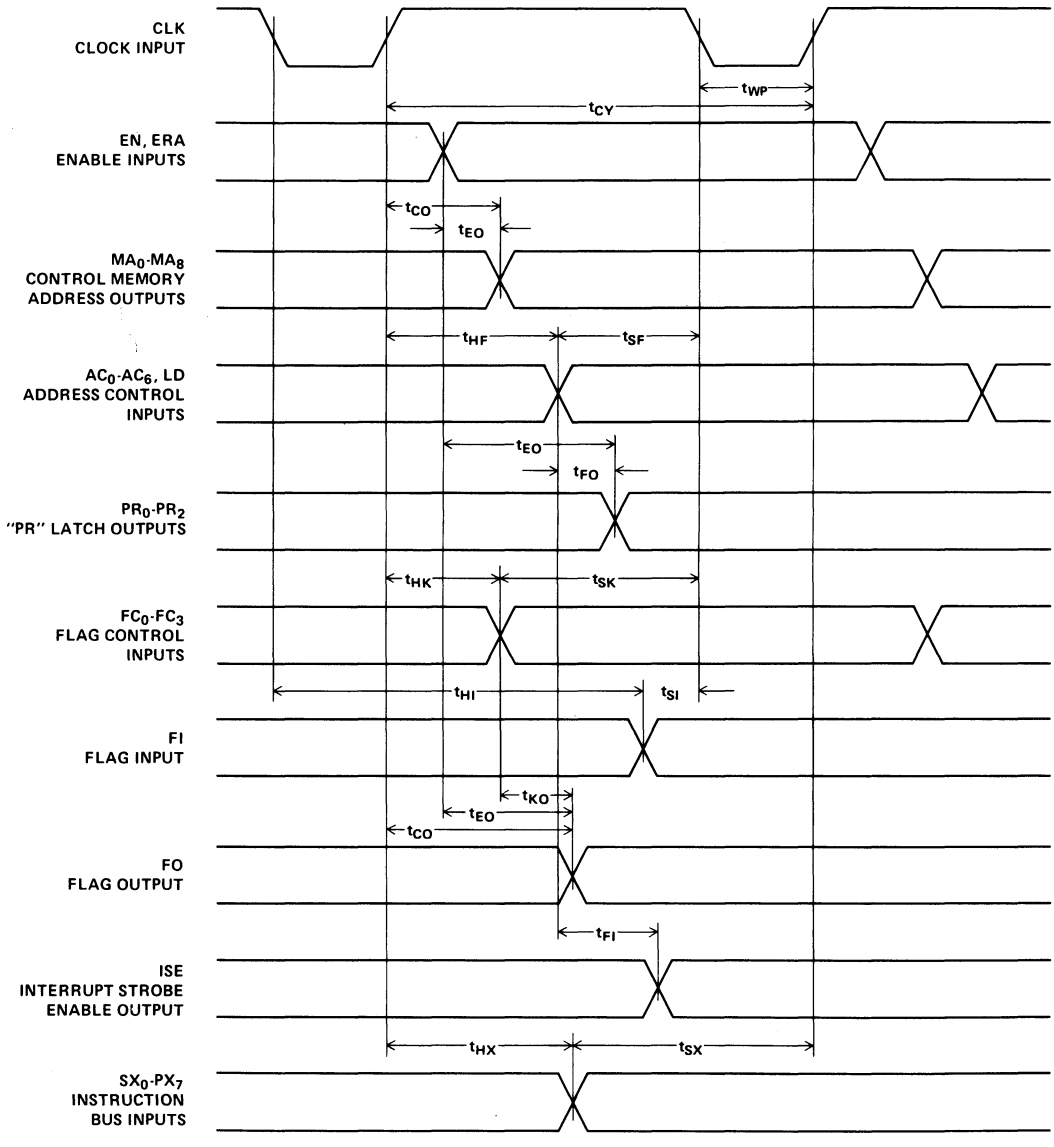


CAPACITANCE(2) T_A = 25°C

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
C_{IN}	Input Capacitance:				
	CLK, EN		11	16	pF
	All Other Inputs		5	10	pF
C_{OUT}	Output Capacitance		6	12	pF

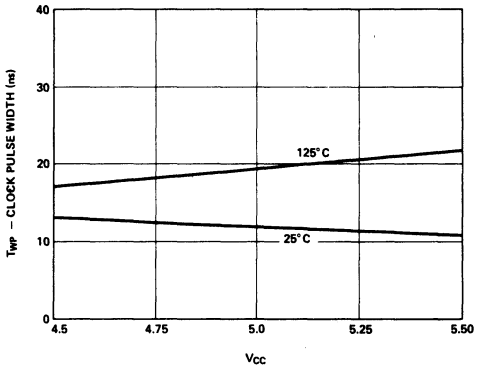
NOTE:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5V and T_A = 25°C.

3001 WAVEFORMS

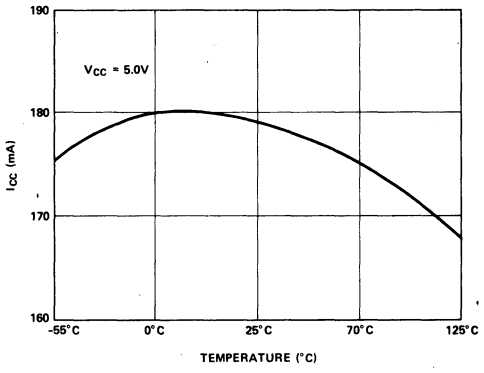


TYPICAL AC AND DC CHARACTERISTICS

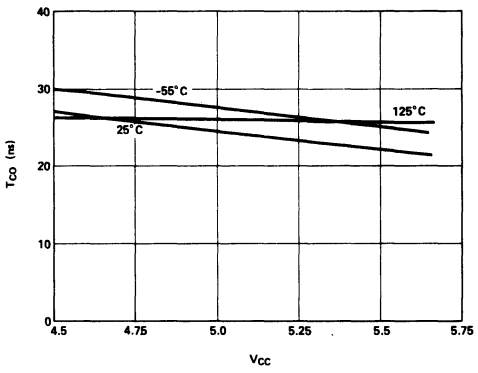
CLOCK PULSE WIDTH VS. V_{CC} AND TEMPERATURE



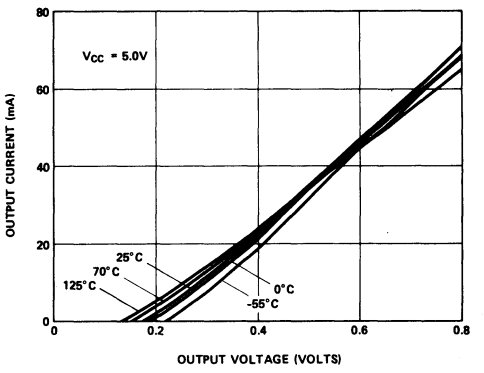
I_{CC} VS. TEMPERATURE



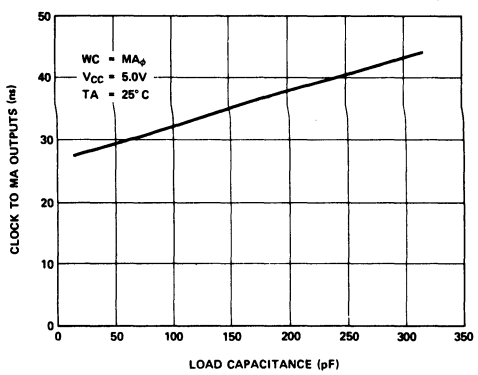
CLOCK TO mA OUTPUTS VS. LOAD CAPACITANCE



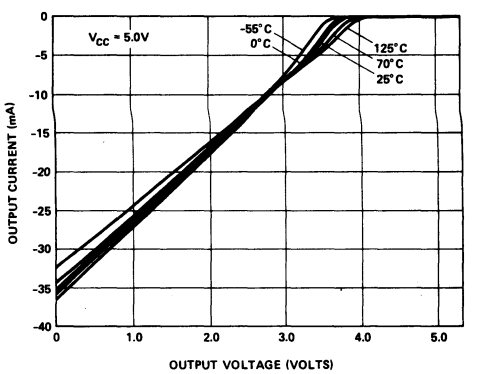
OUTPUT CURRENT VS. OUTPUT LOW VOLTAGE



CLOCK TO MA OUTPUTS VS. LOAD CAPACITANCE



OUTPUT CURRENT VS. OUTPUT HIGH VOLTAGE



APPENDIX A ADDRESS CONTROL FUNCTION SUMMARY

MNEMONIC	DESCRIPTION	FUNCTION							NEXT ROW					NEXT COL			
		AC_6	5	4	3	2	1	0	MA_8	7	6	5	4	MA_3	2	1	0
JCC	Jump in current column	0	0	d_4	d_3	d_2	d_1	d_0	d_4	d_3	d_2	d_1	d_0	m_3	m_2	m_1	m_0
JZR	Jump to zero row	0	1	0	d_3	d_2	d_1	d_0	0	0	0	0	0	d_3	d_2	d_1	d_0
JCR	Jump in current row	0	1	1	d_3	d_2	d_1	d_0	m_8	m_7	m_6	m_5	m_4	d_3	d_2	d_1	d_0
JCE	Jump in column/enable	1	1	1	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	m_2	m_1	m_0
JFL	Jump/test F-latch	1	0	0	d_3	d_2	d_1	d_0	m_8	d_3	d_2	d_1	d_0	m_3	0	1	f
JCF	Jump/test C-flag	1	0	1	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	0	1	c
JZF	Jump/test Z-flag	1	0	1	1	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	m_3	0	1	z
JPR	Jump/test PR-latches	1	1	0	0	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	p_3	p_2	p_1	p_0
JLL	Jump/test left PR bits	1	1	0	1	d_2	d_1	d_0	m_8	m_7	d_2	d_1	d_0	0	1	p_3	p_2
JRL	Jump/test right PR bits	1	1	1	1	1	d_1	d_0	m_8	m_7	1	d_1	d_0	1	1	p_1	p_0
JPX	Jump/test PX-bus	1	1	1	1	0	d_1	d_0	m_8	m_7	m_6	d_1	d_0	x_7	x_6	x_5	x_4

SYMBOL	MEANING
d_n	Data on address control line n
m_n	Data in microprogram address register bit n
p_n	Data in PR-latch bit n
x_n	Data on PX-bus line n (active LOW)
f, c, z	Contents of F-latch, C-flag, or Z-flag, respectively

APPENDIX B FLAG CONTROL FUNCTION SUMMARY

TYPE	MNEMONIC	DESCRIPTION	FC_1	0
Flag Input	SCZ	Set C-flag and Z-flag to f	0	0
	STZ	Set Z-flag to f	0	1
	STC	Set C-flag to f	1	0
	HCZ	Hold C-flag and Z-flag	1	1

TYPE	MNEMONIC	DESCRIPTION	FC_3	2
Flag Output	FF0	Force FO to 0	0	0
	FFC	Force FO to C-flag	0	1
	FFZ	Force FO to Z-flag	1	0
	FF1	Force FO to 1	1	1

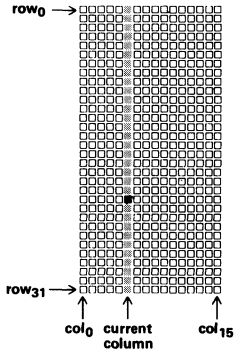
LOAD FUNCTION	NEXT ROW					NEXT COL			
LD	MA_8	7	6	5	4	MA_3	2	1	0
0	see Appendix A					see Appendix A			
1	0	x_3	x_2	x_1	x_0	x_7	x_6	x_5	x_4

SYMBOL	MEANING
f	Contents of the F-latch
x_n	Data on PX- or SX-bus line n (active LOW)

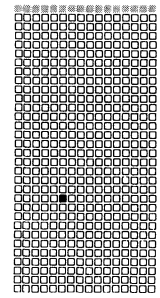
APPENDIX C JUMP SET DIAGRAMS

The following ten diagrams illustrate the jump set for each of the eleven jump and jump/test functions of the MCU. Location 341, indicated by the black square, represents one current row (row_{21}) and current column (col_5) address. The grey boxes indicate the microprogram locations that may be selected by the particular function as the next address.

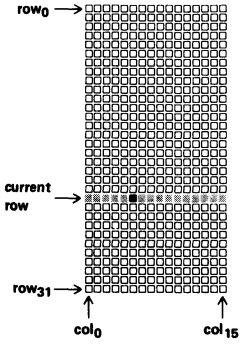
JCC
Jump in Current Column



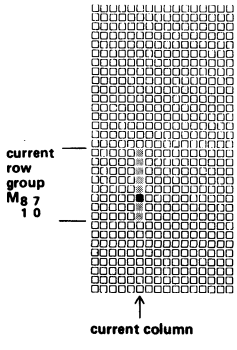
JZR
Jump to Zero Row



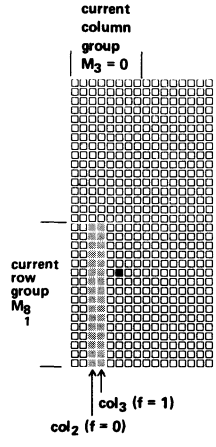
JCR
Jump in Current Row



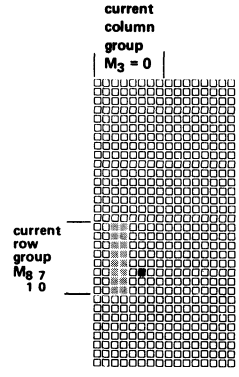
JCE
Jump Column/Enable



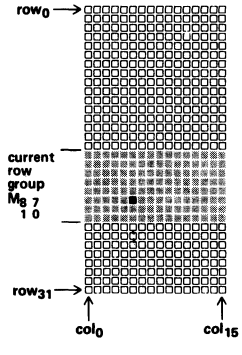
JFL
Jump/Test F-Latch



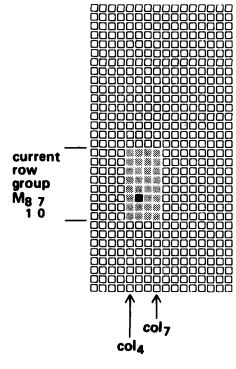
JCF, JZF
Jump/Test C-Flag
Jump/Test Z-Flag



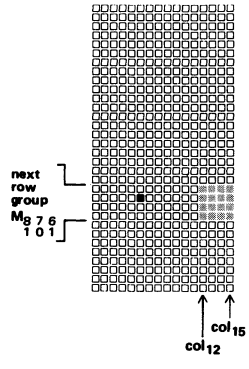
JPR
Jump/Test PR-Latch



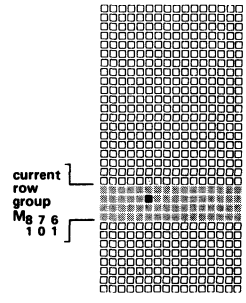
JLL
Jump/Test Left Latch



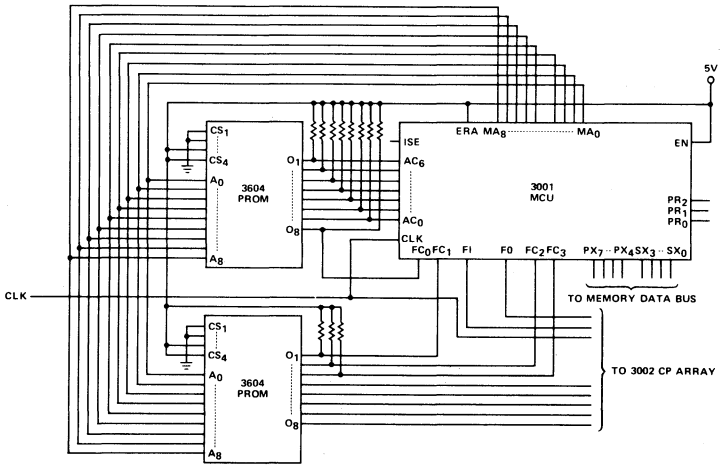
JRL
Jump/Test Right Latch



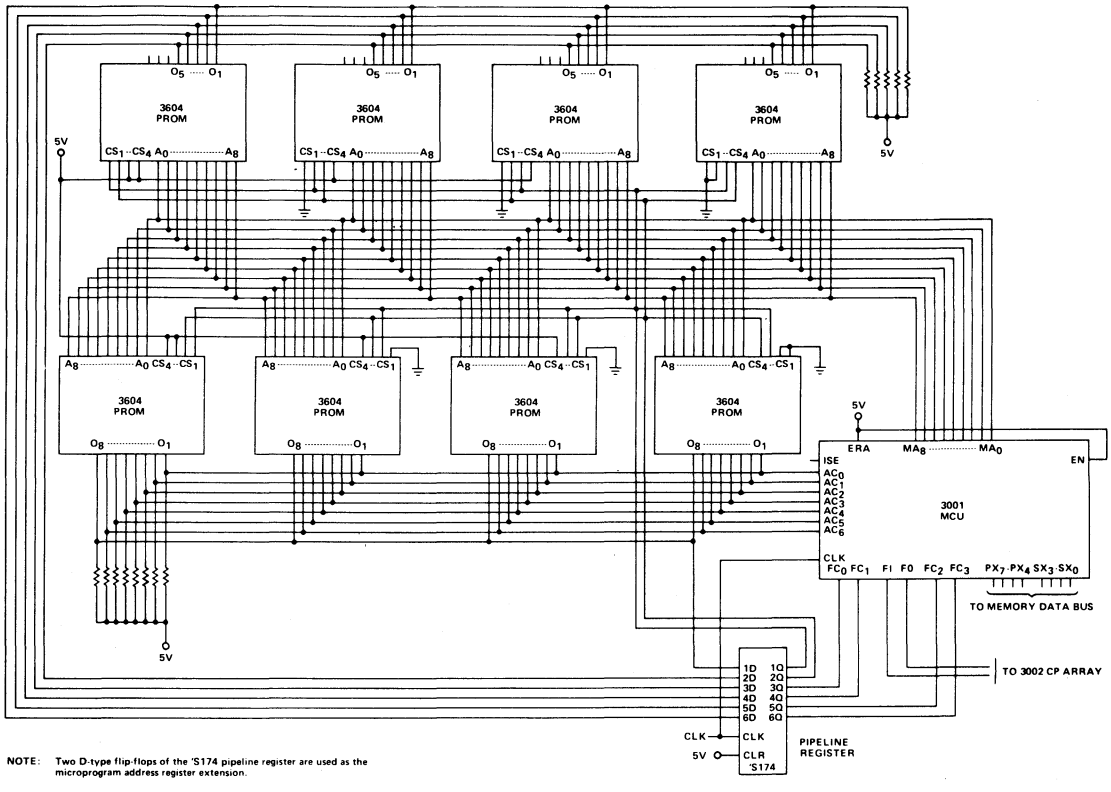
JPX
Jump/Test PX-Bus



TYPICAL CONFIGURATIONS



Non-Pipelined Configuration with
512 Microinstruction Addressability



NOTE: Two D-type flip-flops of the 'S174 pipeline register are used as the microprogram address register extension.

Pipelined Configuration with
2048 Microinstruction Addressability

SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

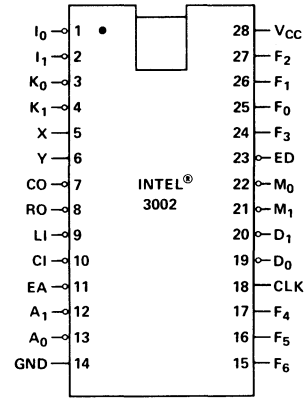
3002 CENTRAL PROCESSING ELEMENT

The INTEL® 3002 Central Processing Element contains all of the circuits that represent a 2-bit wide slice through the data processing section of a digital computer. To construct a complete central processor for a given word width N, it is simply necessary to connect an array of N/2 CPE's together. When wired together in such an array, a set of CPE's provide the following capabilities:

- 2's complement arithmetic
- Logical AND, OR, NOT and exclusive-OR
- Incrementing and decrementing
- Shifting left or right
- Bit testing and zero detection
- Carry look-ahead generation
- Multiple data and address busses

- High Performance – 100 ns Cycle Time
- TTL and DTL Compatible
- N-Bit Word Expandable Multi-Bus Organization
  - 3 Input Data Busses
  - 2 Three-State Fully Buffered Output Data Busses
- 11 General Purpose Registers
- Full Function Accumulator
- Independent Memory Address Register
- Cascade Outputs for Full Carry Look-Ahead
- Versatile Functional Capability
  - 8 Function Groups
  - Over 40 Useful Functions
  - Zero Detect and Bit Test
- Single Clock
- 28 Pin DIP

PACKAGE CONFIGURATION



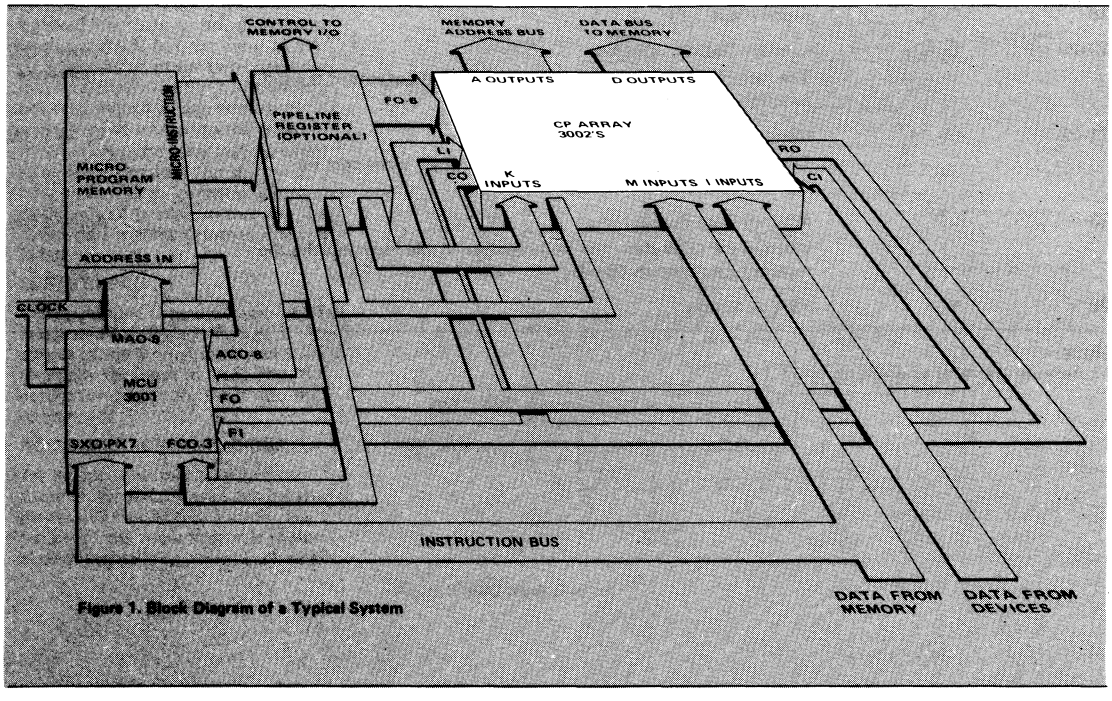


Figure 1. Block Diagram of a Typical System

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE[1]
1, 2	I_0-I_1	External Bus Inputs The external bus inputs provide a separate input port for external input devices.	Active LOW
3, 4	K_0-K_1	Mask Bus Inputs The mask bus inputs provide a separate input port for the microprogram memory, to allow mask or constant entry.	Active LOW
5, 6	X, Y	Standard Carry Look-Ahead Cascade Outputs The cascade outputs allow high speed arithmetic operations to be performed when they are used in conjunction with the INTEL 3003 Look-Ahead Carry Generator.	
7	CO	Ripple Carry Output The ripple carry output is only disabled during shift right operations.	Active LOW Three-state
8	RO	Shift Right Output The shift right output is only enabled during shift right operations.	Active LOW Three-state
9	LI	Shift Right Input	Active LOW
10	CI	Carry Input	Active LOW
11	EA	Memory Address Enable Input When in the LOW state, the memory address enable input enables the memory address outputs (A_0-A_1).	Active LOW
12-13	A_0-A_1	Memory Address Bus Outputs The memory address bus outputs are the buffered outputs of the memory address register (MAR).	Active LOW Three-state
14	GND	Ground	
15-17, 24-27,	F_0-F_6	Micro-Function Bus Inputs The micro-function bus inputs control ALU function and register selection.	
18	CLK	Clock Input	
19-20	D_0-D_1	Memory Data Bus Outputs The memory data bus outputs are the buffered outputs of the full function accumulator register (AC).	Active LOW Three-state
21-22	M_0-M_1	Memory Data Bus Inputs The memory data bus inputs provide a separate input port for memory data.	Active LOW
23	ED	Memory Data Enable Input When in the LOW state, the memory data enable input enables the memory data outputs (D_0-D_1)	Active LOW
28	V_{CC}	+5 Volt Supply	

NOTE:
1. Active HIGH, unless otherwise specified.

LOGICAL DESCRIPTION

The CPE provides the arithmetic, logic and register functions of a 2-bit wide slice through a microprogrammed central processor. Data from external sources such as main memory, is brought into the CPE on one of the three separate input busses. Data being sent out of the CPE to external devices is carried on either of the two output busses. Within the CPE, data is stored in one of eleven scratchpad registers or in the accumulator. Data from the input busses, the registers, or the accumulator is available to the arithmetic/logic section (ALS) under the control of two internal multiplexers. Additional inputs and outputs are included for carry propagation, shifting, and micro-function selection. The complete logical organization of the CPE is shown below.

MICRO-FUNCTION BUS AND DECODER

The seven micro-function bus input lines of the CPE, designated F_0-F_6, are decoded internally to select the ALS function, generate the scratchpad address, and control the A and B multiplexers.

M-BUS AND I-BUS INPUTS

The M-bus inputs are arranged to bring data from an external main memory into the CPE. Data on the M-bus is multiplexed internally for input to the ALS.

The I-bus inputs are arranged to bring data from an external I/O system into the CPE. Data on the I-bus is also multiplexed internally, although independently of the M-bus, for input to the ALS Separation of the two busses permits a relatively lightly loaded memory bus even though a large number of I/O devices are connected to the I-bus. Alternatively, the I-bus may be wired to perform a multiple bit shift (e.g., a byte exchange) by connecting it to one of the output busses. In this case, I/O device data is gated externally onto the M-bus.

SCRATCHPAD

The scratchpad contains eleven registers designated R_0 through R_9 and T. The output of the scratchpad is multiplexed internally for input to ALS. The ALS output is returned for input into the scratchpad.

ACCUMULATOR AND D-BUS

An independent register called the accumulator (AC) is available for storing the result of an ALS operation. The output of the accumulator is multiplexed internally for input back to the ALS and is also available via a three-state output buffer on the D-bus outputs. Conventional usage of the D-bus is for data being sent to the external main memory or to external I/O devices.

A AND B MULTIPLEXERS

The A and B multiplexers select the two inputs to the ALS specified on the micro-function bus. Inputs to the A-multiplexer include the M-bus, the scratchpad, and the accumulator. The B-multiplexer selects either the I-bus, the accumulator, or the K-bus. The selected B-multiplexer input is always logically ANDed with the data on the K-bus (see below) to provide a flexible masking and bit testing capability.

ALS AND K-BUS

The ALS is capable of a variety of arithmetic and logic operations, including 2's complement addition, incrementing, and decrementing, plus logical AND, inclusive-OR, exclusive-NOR, and logical complement. The result of an ALS operation may be stored in the accumulator or one of the scratchpad registers. Separate left input and right output lines, designated LI and RO, are available for use in right shift operations. Carry input and carry output lines, designated CI and CO are provided for normal ripple carry propagation. CO and RO data are brought out via two alternately enabled tri-state buffers. In addition, standard look ahead carry outputs, designated X and Y, are available for full carry look ahead across any word length.

The ability of the K-bus to mask inputs to the ALS greatly increases the versatility of the CPE. During non-arithmetic operations in which carry propagation has no meaning, the carry circuits are used to perform a word-wise inclusive-OR of the bits, masked by the K-bus, from the register or bus selected by the function decoder. Thus, the CPE provides a flexible bit testing capability. The K-bus is also used during arithmetic operations to mask portions of the field being operated upon. An additional function of the K-bus is that of supplying constants to the CPE from the microprogram.

MEMORY ADDRESS REGISTER AND A-BUS

A separate ALS output is also available to the memory address register (MAR) and to the A-bus via a three-state output buffer. Conventional usage of the MAR and A-bus is for sending addresses to an external main memory. The MAR and A-bus may also be used to select an external device when executing I/O operations.

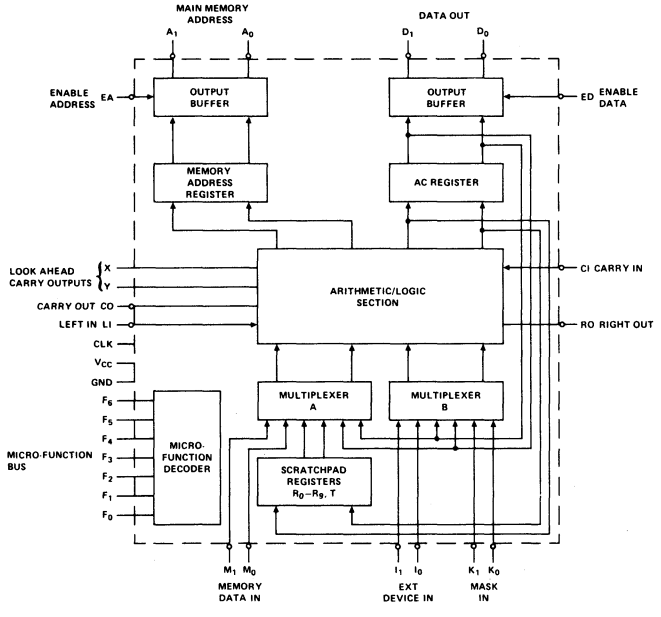


Figure 2. 3002 Block Diagram

FUNCTIONAL DESCRIPTION

During each micro-cycle, a micro-function is applied to F-bus inputs of the CPE. The micro-function is decoded, the operands are selected by the multiplexers, and the specified operation is performed by ALS. If a negative going clock edge is applied, the result of the ALS operation is either deposited in the accumulator or written into the selected scratchpad register. In addition, certain operations permit related address data to be deposited in the MAR. A new micro-function should only be applied following the rising edge of the clock.

By externally gating the clock input to CPE, referred to as conditional clocking, the clock pulse may be selectively omitted during a micro-cycle. Since the carry, shift, and look-ahead circuits are not clocked, their outputs may be used to perform a variety of non-destructive tests on data in the accumulator or in the scratchpad. No register contents are modified by the operation due to the absence of the clock pulse.

The micro-function to be performed is determined from the function group (F-Group) and register group (R-Group) selected by the data on the F-bus. The F-Group is specified by the upper three bits of data, F_4-F_6. The R-Group is specified by the lower four bits of data, F_0-F_3. R-Group I contains R_0 through R_9, T, and AC and is denoted by the symbol R_n. R-Group II and R-Group III contain only T and AC. F-Group and R-Group formats are summarized in Appendix A.

The following is a detailed explanation of each of the CPE micro-functions. A general functional description of each operation is given followed by two additional descriptions which explain the result of the micro-function with both K-bus inputs at logical 0 or both at logical 1. In most cases, the effect of placing the K-bus in the all-one or the all-zero state is to either select or deselect the accumulator in the operation, respectively. A micro-function mnemonic is included with each description for reference purposes and to assist in the design of micro-assembly languages. The micro-functions are summarized in Appendix A. The effective micro-functions for the all-zero and the all-one K-bus states are summarized in Appendix B.

F-GROUP 0 R-GROUP I

Logically AND the contents of AC with the data on the K-bus. Add the result to the contents of R_n and the value of the carry input (CI). Deposit the sum in AC and R_n.

ILR K-BUS = 00

Conditionally increment R_n and load the result in AC. Used to load AC from R_n or to increment R_n and load a copy of the result in AC.

ALR K-BUS= 11

Add AC and CI to R_n and load the result in AC. Used to add AC to a register. If R_n is AC, then AC is shifted left one bit position.

F-GROUP 0 R-GROUP II

Logically AND the contents of AC with the data on the K-bus. Add the result to CI and the data on the M-bus. Deposit the sum in AC or T, as specified.

ACM K-BUS = 00

Add CI to the data on the M-bus. Load the result in AC or T, as specified. Used to load memory data in the specified register, or to load incremented memory data in the specified register.

AMA K-BUS = 11

Add the data on the M-bus to AC and CI, and load the result in AC or T, as specified. Used to add memory data or incremented memory data to AC and store the sum in the specified register.

F-GROUP 0 R-GROUP III

(General description omitted, see Appendix A.)

SRA K-BUS = 00

Shift the contents of AC or T, as specified, right one bit position. Place the previous low order bit value on RO and fill the high order bit from the data on LI. Used to shift or rotate AC or T right one bit.

(K-bus = 11 description omitted, see Appendix B.)

F-GROUP 1 R-GROUP I

Logically OR the contents of R_n with the data on the K-bus. Deposit the result in MAR. Add the data on the K-bus to contents of R_n and CI. Deposit the result in R_n.

LMI K-BUS = 00

Load MAR from R_n. Conditionally increment R_n. Used to maintain a macro-instruction program counter.

DSM K-BUS = 11

Set MAR to all one's. Conditionally decrement R_n by one. Used to force MAR to its highest address and to decrement R_n.

F-GROUP 1 R-GROUP II

Logically OR the data on the M-bus with the data on the K-bus. Deposit the result in MAR. Add the data on the K-bus to the data on the M-bus and CI. Deposit the sum in AC or T, as specified.

LMM K-BUS = 00

Load MAR from the M-bus. Add CI to the data on the M-bus. Deposit the result in AC or T. Used to load the address register with memory data for macro-instructions using indirect addressing.

LDM K-BUS = 11

Set MAR to all ones. Subtract one from the data on the M-bus. Add CI to the difference and deposit the result in AC or T, as specified. Used to load decremented memory data in AC or T.

F-GROUP 1 R-GROUP III

Logically OR the data on the K-bus with the complement of the contents of AC or T, as specified. Add the result to the logical AND of the contents of specified register with the data on the K-bus. Add the sum to CI. Deposit the result in the specified register.

CIA K-BUS = 00

Add CI to the complement of the contents of AC or T, as specified. Deposit the result in the specified register. Used to form the 1's or 2's complement of AC or T.

DCA K-BUS = 11

Subtract one from the contents of AC or T, as specified. Add CI to the difference and deposit the sum in the specified register. Used to decrement AC or T.

FUNCTIONAL DESCRIPTION (con't)

F-GROUP 2 R-GROUP I

Logically AND the data on the K-bus with the contents of AC. Subtract one from the result and add the difference to CI. Deposit the sum in R_n.

CSR K-BUS = 00

Subtract one from CI and deposit the difference in R_n. Used to conditionally clear or set R_n to all 0's or 1's, respectively.

SDR K-BUS = 11

Subtract one from AC and add the difference to CI. Deposit the sum in R_n. Used to store AC in R_n or to store the decremented value of AC in R_n.

F-GROUP 2 R-GROUP II

Logically AND the data on the K-bus with the contents of AC. Subtract one from the result and add the difference to CI. Deposit the sum in AC or T, as specified.

CSA K-BUS = 00

Subtract one from CI and deposit the difference in AC or T, as specified. Used to conditionally clear or set AC or T.

SDA K-BUS = 11

Subtract one from AC and add the difference to CI. Deposit the sum in AC or T, as specified. Used to store AC in T, or decrement AC, or store the decremented value of AC in T.

F-GROUP 2 R-GROUP III

Logically AND the data of the K-bus with the data on the I-bus. Subtract one from the result and add the difference to CI. Deposit the sum in AC or T, as specified.

(K-bus = 00 description omitted, see CSA above.)

LDI K-BUS = 11

Subtract one from the data on the I-bus and add the difference to CI. Deposit the sum in AC or T, as specified. Used to load input bus data or decremented input bus data in the specified register.

F-GROUP 3 R-GROUP I

Logically AND the contents of AC with the data on the K-bus. Add the contents of R_n and CI to the result. Deposit the sum in R_n.

INR K-BUS = 00

Add CI to the contents of R_n and deposit the sum in R_n. Used to increment R_n.

ADR K-BUS = 11

Add the contents of AC to R_n. Add the result to CI and deposit the sum in R_n. Used to add the accumulator to a register or to add the incremented value of the accumulator to a register.

F-GROUP 3 R-GROUP II

(All descriptions omitted, identical to F-Group 0/R-Group II described above.)

F-GROUP 3 R-GROUP III

Logically AND the data on the K-bus with the data on the I-bus. Add CI and the contents of AC or T, as specified, to the result. Deposit the sum in the specified register.

INA K-BUS = 00

Conditionally increment the contents of AC or T, as specified. Used to increment AC or T.

AIA K-BUS = 11

Add the data on the I-bus to the contents of AC or T, as specified. Add CI to the result and deposit the sum in the specified register. Used to add input data or incremented input data to the specified register.

F-GROUP 4 R-GROUP I

Logically AND the data on the K-bus with the contents of AC. Logically AND the result with the contents of R_n. Deposit the final result in R_n. Logically OR the value of CI with the word-wise OR of the bits of the final result. Place the value of the carry OR on the carry output (CO) line.

CLR K-BUS = 00

Clear R_n to all 0's. Force CO to CI. Used to clear a register and force CO to CI.

ANR K-BUS = 11

Logically AND AC with R_n. Deposit the result in R_n. Force CO to one if the result is non-zero. Used to AND the accumulator with a register and test for a zero result.

F-GROUP 4 R-GROUP II

Logically AND the data on the K-bus with the contents of AC. Logically AND the result with the data on the M-bus. Deposit the final result in AC or T, as specified. Logically OR the value of CI with the word-wise OR of the bits of the final result. Place the value of the carry OR on CO.

CLA K-BUS = 00

Clear AC or T, as specified, to all 0's. Force CO to CI. Used to clear the specified register and force CO to CI.

ANM K-BUS = 11

Logically AND the data on the M-bus with the contents of AC. Deposit the result in AC or T, as specified. Force CO to one if the result is non-zero. Used to AND M-bus data to the accumulator and test for a zero result.

F-GROUP 4 R-GROUP III

Logically AND the data on I-bus with the data on the K-bus. Logically AND the result with the contents of AC or T, as specified. Deposit the final result in the specified register. Logically OR CI with the word-wise OR of the bits of the final result. Place the value of the carry OR on CO.

(K-bus = 00 description omitted, see CLA above.)

ANI K-BUS = 11

Logically AND the data on the I-bus with the contents of AC or T, as specified. Deposit the result in the specified register. Force CO to one if the result is non-zero. Used to AND the I-bus to the accumulator and test for a zero result.

F-GROUP 5 R-GROUP I

Logically AND the data on the K-bus with the contents of R_n. Deposit the result in R_n. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO.

(K-bus = 00 description omitted, see CLR above.)

TZR K-BUS = 11

Force CO to one if R_n is non-zero. Used to test a register for zero. Also used to AND K-bus data with a register (see general description) for masking and, optionally, testing for a zero result.

FUNCTIONAL DESCRIPTION (con't)

F-GROUP 5 R-GROUP II

Logically AND the data on the K-bus with the data on the M-bus. Deposit the result in AC or T, as specified. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO.

(K-bus = 00 description omitted, see CLA above.)

LTM K-BUS = 11

Load AC or T, as specified, with data from the M-bus. Force CO to one if the result is non-zero. Used to load the specified register from memory and test for a zero result. Also used to AND K-bus data with M-bus data (see general description) for masking and, optionally, testing for a zero result.

F-GROUP 5 R-GROUP III

Logically AND the data on K-bus with contents of AC or T, as specified. Deposit the result in the specified register. Logically OR CI with the word-wise OR of the result. Place the value of the carry OR on CO.

(K-bus = 00 description omitted, see CLA above.)

TZA K-BUS = 11

Force CO to one if AC or T, as specified, is non-zero. Used to test the specified register for zero. Also used to AND K-bus data to the specified register (see general description) for masking and, optionally, testing for a zero result.

F-GROUP 6 R-GROUP I

Logically OR CI with the word-wise OR of the logical AND of AC and the data on the K-bus. Place the result of the carry OR on CO. Logically OR the contents of R_n with the logical AND of AC and the data on the K-bus. Deposit the result in R_n.

NOP K-BUS = 00

Force CO to CI. Used as a null operation or to force CO to CI.

ORR K-BUS = 11

Force CO to one if AC is non-zero. Logically OR the contents of the accumulator to the contents of R_n. Deposit the result in R_n. Used to OR the accumulator to a register and, optionally, test the previous accumulator value for zero.

F-GROUP 6 R-GROUP II

Logically OR CI with the word-wise OR of the logical AND of AC and the data on the K-bus. Place the value of the carry OR on CO. Logically OR the data on the M-bus, with the logical AND of AC and the data on the K-bus. Deposit the final result in AC or T, as specified.

LMF K-BUS = 00

Load AC or T, as specified, from the M-bus. Force CO to CI. Used to load the specified register with memory data and force CO to CI.

ORM K-BUS = 11

Force CO to one if AC is non-zero. Logically OR the data on the M-bus with the contents of AC. Deposit the result in AC or T, as specified. Used to OR memory data with the accumulator and, optionally, test the previous value of the accumulator for zero.

F-GROUP 6 R-GROUP III

Logically OR CI with the word-wise OR of the logical AND of the data on the I-bus and the data on the K-bus. Place the value of the carry OR on CO. Logically AND the data on the K-bus with the data on the I-bus. Logically OR the result with the contents of AC or T, as specified. Deposit the final result in the specified register.

(K-bus = 00 description omitted, see NOP above.)

ORI K-BUS = 11

Force CO to one if the data on the I-bus is non-zero. Logically OR the data on the I-bus to the contents of AC or T, as specified. Deposit the result in the specified register. Used to OR I-bus data with the specified register and, optionally, test the I-bus data for zero.

F-GROUP 7 R-GROUP I

Logically OR CI with the word-wise OR of the logical AND of the contents of R_n and AC and the data on the K-bus. Place the value of the carry OR on CO. Logically AND the data on the K-bus with the contents of AC. Exclusive-NOR the result with the contents of R_n. Deposit the final result in R_n.

CMR K-BUS = 00

Complement the contents of R_n. Force CO to CI.

XNR K-BUS = 11

Force CO to one if the logical AND of AC and R_n is non-zero. Exclusive-NOR the contents of AC with the contents of R_n. Deposit the result in R_n. Used to exclusive-NOR the accumulator with a register.

F-GROUP 7 R-GROUP II

Logically OR CI with the word-wise OR of the logical AND of the contents of AC and the data on the K-bus and M-bus. Place the value of the carry OR on CO. Logically AND the data on the K-bus with the contents of AC. Exclusive-NOR the result with the data on the M-bus. Deposit the final result in AC or T, as specified.

LCM K–BUS = 00

Load the complement of the data on the M-bus into AC or T, as specified. Force CO to CI.

XNM K-BUS = 11

Force CO to one if the logical AND of AC and the M-bus data is non-zero. Exclusive-NOR the contents of AC with the data on the M-bus. Deposit the result in AC or T, as specified. Used to exclusive–NOR memory data with the accumulator.

F-GROUP 7 R-GROUP III

Logically OR CI with the word-wise OR of the logical AND of the contents of the specified register and the data on the I-bus and K-bus. Place the value of the carry OR on CO. Logically AND the data on the K-bus with the data on the I-bus. Exclusive-NOR the result with the contents of AC or T, as specified. Deposit the final result in the specified register.

CMA K-BUS = 00

Complement AC or T, as specified. Force CO to CI.

XNI K-BUS = 11

Force CO to one if the logical AND of the contents of AC or T, as specified, and the I-bus data is non-zero. Exclusive-NOR the contents of the specified register with the data on the I-bus. Deposit the result in AC or T, as specified. Used to exclusive-NOR input data with the accumulator.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . −65°C to +160°C

All Output and Supply Voltages . . . . . −0.5V to +7V

All Input Voltages . . . . . −1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may effect device reliability.

T_A = 0°C to +70°C, V_{CC} = 5.0V ±5%

		LIMITS				
SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		−0.8	−1.0	V	I_C = −5 mA
I_F	Input Load Current:					
	F_0–F_6, CLK, K_0, K_1, EA, ED		−0.05	−0.25	mA	V_F = 0.45V
	I_0, I_1, M_0, M_1, LI		−0.85	−1.5	mA	
	CI		−2.3	−4.0	mA	
I_R	Input Leakage Current:					
	F_0–F_6, CLK, K_0, K_1, EA, ED			40	μA	V_R = 5.25V
	I_0, I_1, M_0, M_1, LI			60	μA	
	CI			180	μA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	
I_{CC}	Power Supply Current(2)		145	190	mA	
V_{OL}	Output Low Voltage (All Output Pins)		0.3	0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage (All Output Pins)	2.4	3.0		V	I_{OH} = −1 mA
I_{OS}	Short Circuit Output Current (All Output Pins)	−15	−25	−60	mA	V_{CC} = 5.0V
$I_{O(off)}$	Off State Output Current			−100	μA	V_O = 0.45V
	A_0, A_1, D_0, D_1, CO and RO			100	μA	V_O = 5.25V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage
(2) CLK input grounded, other inputs open.

A.C. CHARACTERISTICS AND WAVEFORMS

T_A = 0°C to 70°C, V_{CC} = 5V ±5%

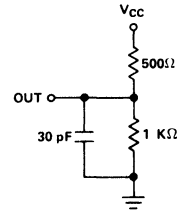
SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT
t_{CY}	Clock Cycle Time (2)	100	70		ns
t_{WP}	Clock Pulse Width	33	20		ns
t_{FS}	Function Input Set-Up Time (F_0 through F_6)	60	40		ns
	Data Set-Up Time:				
t_{DS}	I_0, I_1, M_0, M_1, K_0, K_1	50	30		ns
t_{SS}	LI, CI	27	13		ns
	Data and Function Hold Time:				
t_{FH}	F_0 through F_6	5	-2		ns
t_{DH}	I_0, I_1, M_0, M_1, K_0, K_1	5	-4		ns
t_{SH}	LI, CI	15	2		ns
	Propagation Delay to X, Y, RO from:				
t_{XF}	Any Function Input		37	52	ns
t_{XD}	Any Data Input		29	42	ns
t_{XT}	Trailing Edge of CLK		40	60	ns
t_{XL}	Leading Edge of CLK	20			ns
	Propagation Delay to CO from:				
t_{CL}	Leading Edge of CLK	20			ns
t_{CT}	Trailing Edge of CLK		48	70	ns
t_{CF}	Any Function Input		43	65	ns
t_{CD}	Any Data Input		30	55	ns
t_{CC}	CI (Ripple Carry)		14	25	ns
	Propagation Delay to A_0, A_1, D_0, D_1 from:				
t_{DL}	Leading Edge of CLK	5	32	50	ns
t_{DE}	Enable Input ED, EA		12	25	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) $t_{CY} = t_{DS} + t_{DL}$.

TEST CONDITIONS:

Input pulse amplitude: 2.5 V
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 10 mA and 30 pF.
Speed measurements are made at 1.5 volt levels.

TEST LOAD CIRCUIT:



CAPACITANCE(2) T_A = 25°C

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
C_{IN}	Input Capacitance		5	10	pF
C_{OUT}	Output Capacitance		6	12	pF

NOTE:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5.0V and T_A = 25°C.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . -55°C to +125°C

Storage Temperature . . . . . −65°C to +160°C

All Output and Supply Voltages . . . . . −0.5V to +7V

All Input Voltages . . . . . −1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may effect device reliability.

T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%.

		LIMITS				
SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		−0.8	−1.2	V	I_C = −5 mA
I_F	Input Load Current:					
	F_0–F_6, CLK, K_0, K_1, EA, ED		−0.05	−0.25	mA	V_F = 0.45V
	I_0, I_1, M_0, M_1, LI		−0.85	−1.5	mA	
	CI		−2.3	−4.0	mA	
I_R	Input Leakage Current:					
	F_0–F_6, CLK, K_0, K_1, EA, ED			40	μA	V_R = 5.5V
	I_0, I_1, M_0, M_1, LI			100	μA	
	CI			250	μA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	
I_{CC}	Power Supply Current		145	210	mA	
V_{OL}	Output Low Voltage (All Output Pins)		0.3	0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage (All Output Pins)	2.4	3.0		V	I_{OH} = −1 mA
I_{OS}	Short Circuit Output Current (All Output Pins)	−15	−25	−60	mA	V_{CC} = 5.0V
$I_{O(off)}$	Off State Output Current			−100	μA	V_O = 0.45V
	A_0, A_1, D_0, D_1, CO and RO			100	μA	V_O = 5.5V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage
(2) CLK input grounded, other inputs open.

A.C. CHARACTERISTICS AND WAVEFORMS

T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%.

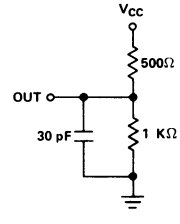
SYMBOL	PARAMETER	MIN	TYP[1]	MAX	UNIT
t_{CY}	Clock Cycle Time [2]	120	70		ns
t_{WP}	Clock Pulse Width	42	20		ns
t_{FS}	Function Input Set-Up Time (F_0 through F_6)	70	40		ns
	Data Set-Up Time:				
t_{DS}	I_0, I_1, M_0, M_1, K_0, K_1	60	30		ns
t_{SS}	LI, CI	30	13		ns
	Data and Function Hold Time:				
t_{FH}	F_0 through F_6	5	-2		ns
t_{DH}	I_0, I_1, M_0, M_1, K_0, K_1	5	-4		ns
t_{SH}	LI, CI	15	2		ns
	Propagation Delay to X, Y, RO from:				
t_{XF}	Any Function Input		37	65	ns
t_{XD}	Any Data Input		29	55	ns
t_{XT}	Trailing Edge of CLK		40	75	ns
t_{XL}	Leading Edge of CLK	22			ns
	Propagation Delay to CO from:				
t_{CL}	Leading Edge of CLK	22			ns
t_{CT}	Trailing Edge of CLK		48	85	ns
t_{CF}	Any Function Input		43	75	ns
t_{CD}	Any Data Input		30	65	ns
t_{CC}	CI (Ripple Carry)		14	30	ns
	Propagation Delay to A_0, A_1, D_0, D_1 from:				
t_{DL}	Leading Edge of CLK	5	32	60	ns
t_{DE}	Enable Input ED, EA		12	35	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) $t_{CY} = t_{DS} + t_{DL}$

TEST CONDITIONS:

Input pulse amplitude: 2.5 V
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 10 mA and 30 pF.
Speed measurements are made at 1.5 volt levels.

TEST LOAD CIRCUIT:

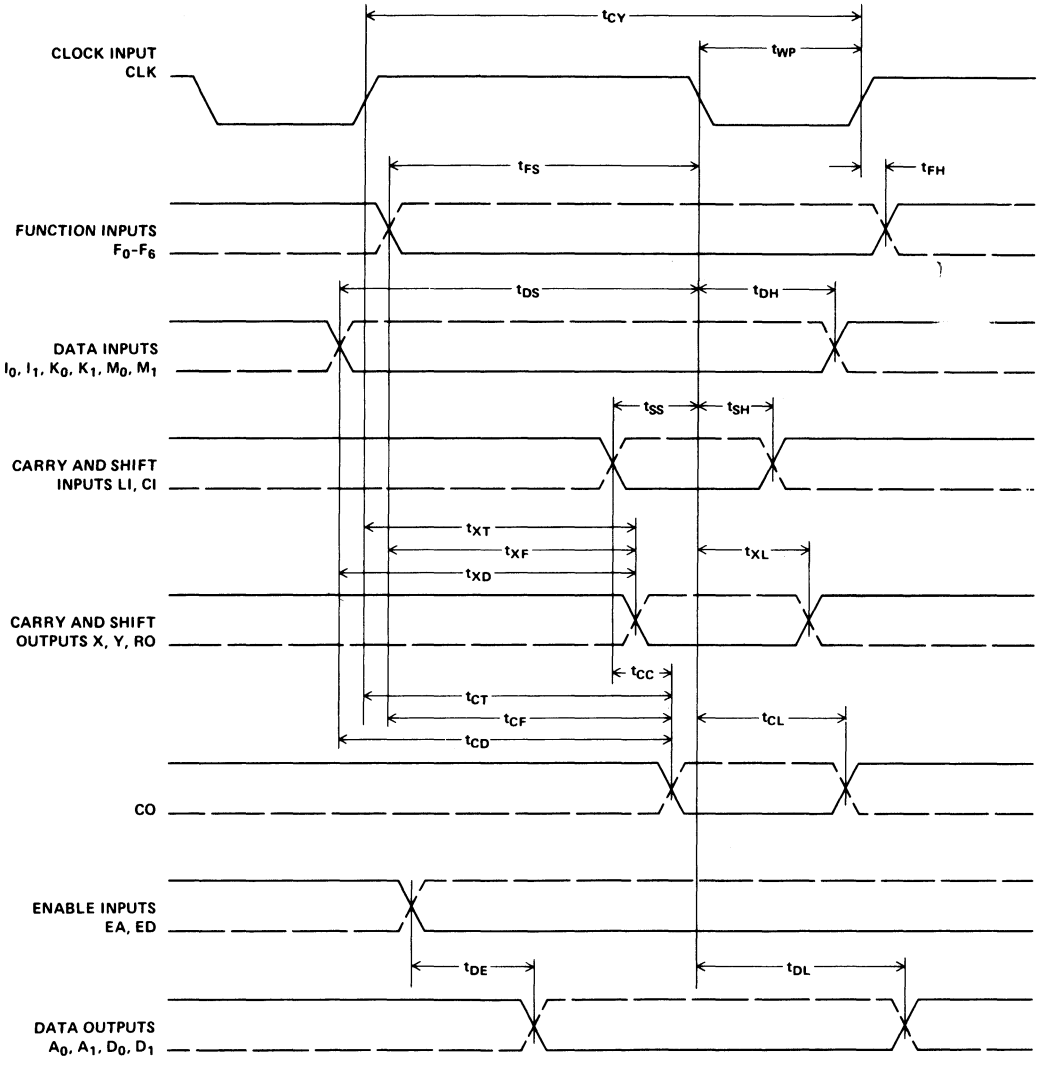


CAPACITANCE[2] T_A = 25°C

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
C_{IN}	Input Capacitance		5	10	pF
C_{OUT}	Output Capacitance		6	12	pF

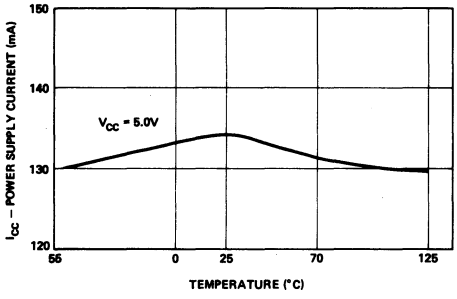
NOTE:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5.0V and T_A = 25°C.

3002 WAVEFORMS

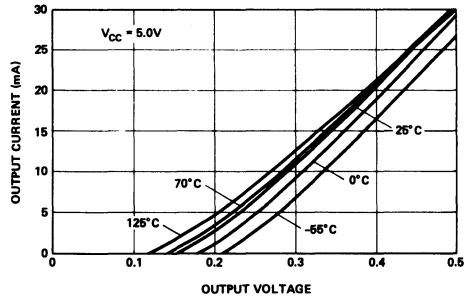


TYPICAL AC AND DC CHARACTERISTICS

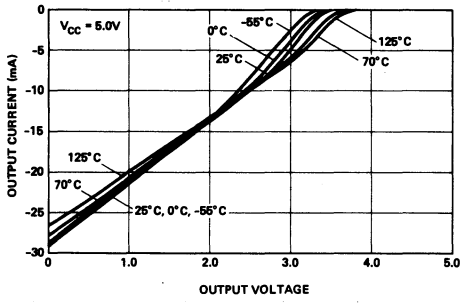
POWER SUPPLY CURRENT VS. TEMPERATURE



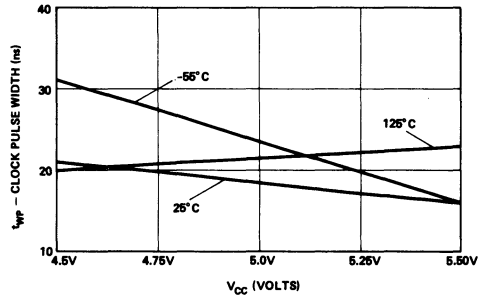
OUTPUT CURRENT VS. OUTPUT LOW VOLTAGE



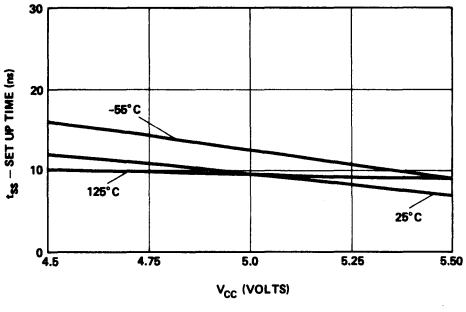
OUTPUT CURRENT VS. OUTPUT VOLTAGE



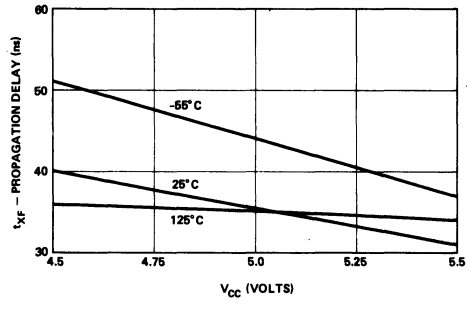
CLOCK PULSE WIDE VS. V_{CC} AND TEMPERATURE



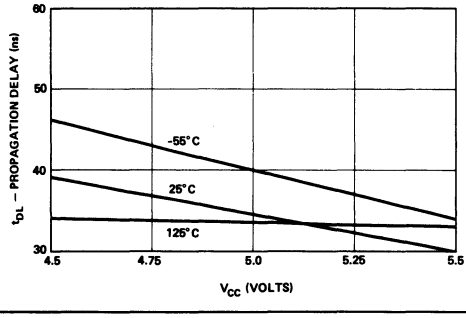
CARRY IN SET UP TIME VS. V_{CC} AND TEMPERATURE



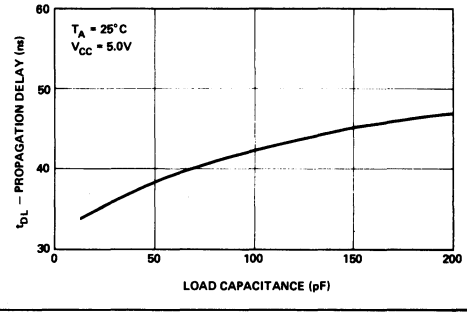
PROPAGATION DELAY FROM FUNCTION INPUTS TO CASCADE OUTPUTS VS. V_{CC} AND TEMPERATURE



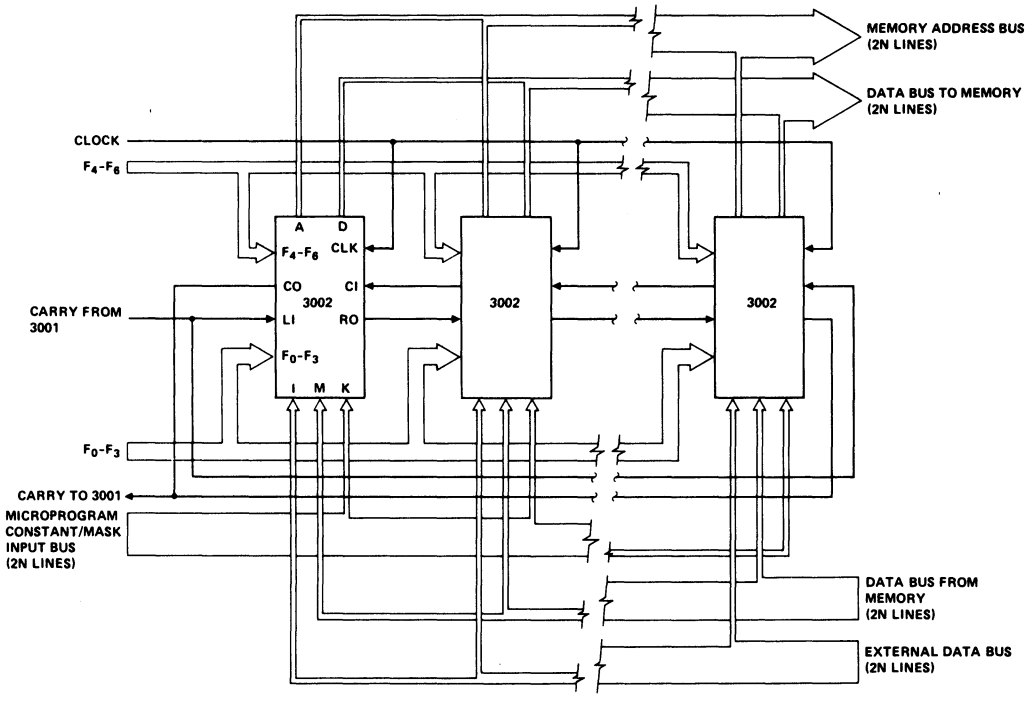
PROPAGATION DELAY – CLOCK TO "A" AND "D" DATA OUTPUT VS. V_{CC} AND TEMPERATURE



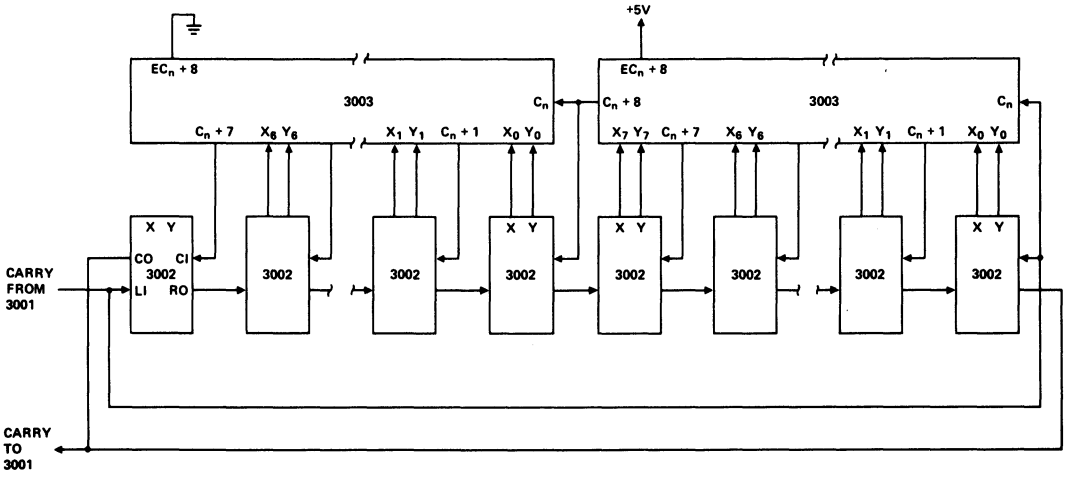
PROPAGATION DELAY – CLOCK TO "A" AND "D" DATA OUTPUT VS. LOAD CAPACITANCE



TYPICAL CONFIGURATIONS



Ripple-Carry Configuration
(N 3002 CPE's)



Carry Look-Ahead Configuration
With Ripple Through the Left Slice
(32 Bit Array)

APPENDIX A MICRO-FUNCTION SUMMARY

F-GROUP	R-GROUP	MICRO-FUNCTION	
0	I	$R_n + (AC \wedge K) + CI \rightarrow R_n, AC$	
	II	$M + (AC \wedge K) + CI \rightarrow AT$	
	III	$AT_L \wedge (\overline{I_L \wedge K_L}) \rightarrow RO$	$LI \vee [(I_H \wedge K_H) \wedge AT_H] \rightarrow AT_H$ $[AT_L \wedge (I_L \wedge K_L)] \vee [AT_H \vee (I_H \wedge K_H)] \rightarrow AT_L$
1	I	$K \vee R_n \rightarrow MAR$	$R_n + K + CI \rightarrow R_n$
	II	$K \vee M \rightarrow MAR$	$M + K + CI \rightarrow AT$
	III	$(\overline{AT} \vee K) + (AT \wedge K) + CI \rightarrow AT$	
2	I	$(AC \wedge K) - 1 + CI \rightarrow R_n$	(see Note 1)
	II	$(AC \wedge K) - 1 + CI \rightarrow AT$	
	III	$(I \wedge K) - 1 + CI \rightarrow AT$	
3	I	$R_n + (AC \wedge K) + CI \rightarrow R_n$	
	II	$M + (AC \wedge K) + CI \rightarrow AT$	
	III	$AT + (I \wedge K) + CI \rightarrow AT$	
4	I	$CI \vee (R_n \wedge AC \wedge K) \rightarrow CO$	$R_n \wedge (AC \wedge K) \rightarrow R_n$
	II	$CI \vee (M \wedge AC \wedge K) \rightarrow CO$	$M \wedge (AC \wedge K) \rightarrow AT$
	III	$CI \vee (AT \wedge I \wedge K) \rightarrow CO$	$AT \wedge (I \wedge K) \rightarrow AT$
5	I	$CI \vee (R_n \wedge K) \rightarrow CO$	$K \wedge R_n \rightarrow R_n$
	II	$CI \vee (M \wedge K) \rightarrow CO$	$K \wedge M \rightarrow AT$
	III	$CI \vee (AT \wedge K) \rightarrow CO$	$K \wedge AT \rightarrow AT$
6	I	$CI \vee (AC \wedge K) \rightarrow CO$	$R_n \vee (AC \wedge K) \rightarrow R_n$
	II	$CI \vee (AC \wedge K) \rightarrow CO$	$M \vee (AC \wedge K) \rightarrow AT$
	III	$CI \vee (I \wedge K) \rightarrow CO$	$AT \vee (I \wedge K) \rightarrow AT$
7	I	$CI \vee (R_n \wedge AC \wedge K) \rightarrow CO$	$R_n \bar{\oplus} (AC \wedge K) \rightarrow R_n$
	II	$CI \vee (M \wedge AC \wedge K) \rightarrow CO$	$M \bar{\oplus} (AC \wedge K) \rightarrow AT$
	III	$CI \vee (AT \wedge I \wedge K) \rightarrow CO$	$AT \bar{\oplus} (I \wedge K) \rightarrow AT$

NOTES:

1. 2's complement arithmetic adds 111 . . . 11 to perform subtraction of 000 . . . 01.
2. R_n includes T and AC as source and destination registers in R-group 1 micro-functions.
3. Standard arithmetic carry output values are generated in F-group 0, 1, 2 and 3 instructions.

SYMBOL	MEANING
I, K, M	Data on the I, K, and M busses, respectively
CI, LI	Data on the carry input and left input, respectively
CO, RO	Data on the carry output and right output, respectively
R_n	Contents of register n including T and AC (R-Group I)
AC	Contents of the accumulator
AT	Contents of AC or T, as specified
MAR	Contents of the memory address register
L, H	As subscripts, designate low and high order bit, respectively
+	2's complement addition
–	2's complement subtraction
$\wedge$	Logical AND
$\vee$	Logical OR
$\bar{\oplus}$	Exclusive-NOR
$\rightarrow$	Deposit into

APPENDIX B ALL-ZERO AND ALL-ONE K-BUS MICRO-FUNCTIONS

K-BUS = 00 MICRO-FUNCTION	MNEMONIC	K-BUS = 11 MICRO-FUNCTION	MNEMONIC
$R_n + CI \rightarrow R_n$, AC	ILR	$AC + R_n + CI \rightarrow R_n$, AC	ALR
$M + CI \rightarrow AT$	ACM	$M + AC + CI \rightarrow AT$	AMA
$AT_L \rightarrow RO$ $AT_H \rightarrow AT_L$ $LI \rightarrow AT_H$	SRA	(See Appendix A)	–
$R_n \rightarrow MAR$ $R_n + CI \rightarrow R_n$	LMI	$11 \rightarrow MAR$ $R_n - 1 + CI \rightarrow R_n$	DSM
$M \rightarrow MAR$ $M + CI \rightarrow AT$	LMM	$11 \rightarrow MAR$ $M - 1 + CI \rightarrow AT$	LDM
$\overline{AT} + CI \rightarrow AT$	CIA	$AT - 1 + CI \rightarrow AT$	DCA
$CI - 1 \rightarrow R_n$ See Note 1	CSR	$AC - 1 + CI \rightarrow R_n$ See Note 1	SDR
$CI - 1 \rightarrow AT$ See Notes 1,4	CSA	$AC - 1 + CI \rightarrow AT$ See Notes 1,4	SDA
(See CSA above)	–	$I - 1 + CI \rightarrow AT$	LDI
$R_n + CI \rightarrow R_n$	INR	$AC + R_n + CI \rightarrow R_n$	ADR
(See ACM above)	–	(See AMA above)	–
$AT + CI \rightarrow AT$	INA	$I + AT + CI \rightarrow AT$	AIA
$CI \rightarrow CO$ $0 \rightarrow R_n$	CLR	$CI \vee (R_n \wedge AC) \rightarrow CO$ $R_n \wedge AC \rightarrow R_n$	ANR
$CI \rightarrow CO$ $0 \rightarrow AT$	CLA	$CI \vee (M \wedge AC) \rightarrow CO$ $M \wedge AC \rightarrow AT$	ANM
(See CLA above)	–	$CI \vee (AT \wedge I) \rightarrow CO$ $AT \wedge I \rightarrow AT$	ANI
(See CLR above)	–	$CI \vee R_n \rightarrow CO$ $R_n \rightarrow R_n$	TZR
(See CLA above)	–	$CI \vee M \rightarrow CO$ $M \rightarrow AT$	LTM
(See CLA above)	–	$CI \vee AT \rightarrow CO$ $AT \rightarrow AT$	TZA
$CI \rightarrow CO$ $R_n \rightarrow R_n$	NOP	$CI \vee AC \rightarrow CO$ $R_n \vee AC \rightarrow R_n$	ORR
$CI \rightarrow CO$ $M \rightarrow AT$	LMF	$CI \vee AC \rightarrow CO$ $M \vee AC \rightarrow AT$	ORM
(See NOP above)	–	$CI \vee I \rightarrow CO$ $I \vee AT \rightarrow AT$	ORI
$CI \rightarrow CO$ $\overline{R_n} \rightarrow R_n$	CMR	$CI \vee (R_n \quad AC) \rightarrow CO$ $R_n \oplus AC \rightarrow R_n$	XNR
$CI \rightarrow CO$ $\overline{M} \rightarrow AT$	LCM	$CI \vee (M \quad AC) \rightarrow CO$ $M \oplus AC \rightarrow AT$	XNM
$CI \rightarrow CO$ $\overline{AT} \rightarrow AT$	CMA	$CI \vee (AT \quad I) \rightarrow CO$ $I \oplus AT \rightarrow AT$	XNI

4. The more general operations, CSR and SDR, should be used in place of the CSA and SDA operations, respectively.

APPENDIX C FUNCTION AND REGISTER GROUP FORMATS

FUNCTION GROUP	F_6	5	4
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

REGISTER GROUP	REGISTER	F_3	2	1	0
	R_0	0	0	0	0
	R_1	0	0	0	1
	R_2	0	0	1	0
	R_3	0	0	1	1
	R_4	0	1	0	0
	R_5	0	1	0	1
	R_6	0	1	1	0
	R_7	0	1	1	1
	R_8	1	0	0	0
	R_9	1	0	0	1
	T	1	1	0	0
	AC	1	1	0	1
II	T	1	0	1	0
	AC	1	0	1	1
III	T	1	1	1	0
	AC	1	1	1	1

SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

3003 LOOK-AHEAD CARRY GENERATOR

The INTEL® 3003 Look-Ahead Carry Generator (LCG) is a high speed circuit capable of anticipating a carry across a full 16-bit 3002 Central Processing Array. When used with a larger 3002 CP Array multiple 3003 carry generators provide high speed carry look-ahead capability for any word length.

The LCG accepts eight pairs of active high cascade inputs (X,Y) and an active low carry input and generates active low carries for up to eight groups of binary adders.

High Performance – 10 ns typical propagation delay

Compatible with INTEL 3001 MCU and 3002 CPE

DTL and TTL compatible

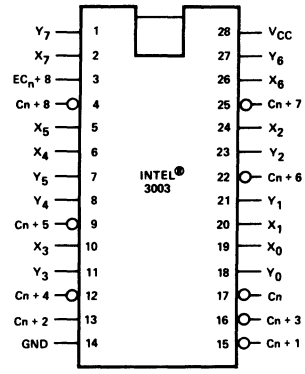
Full look-ahead across 8 adders

Low voltage diode input clamp

Expandable

28-pin DIP

PACKAGE CONFIGURATION



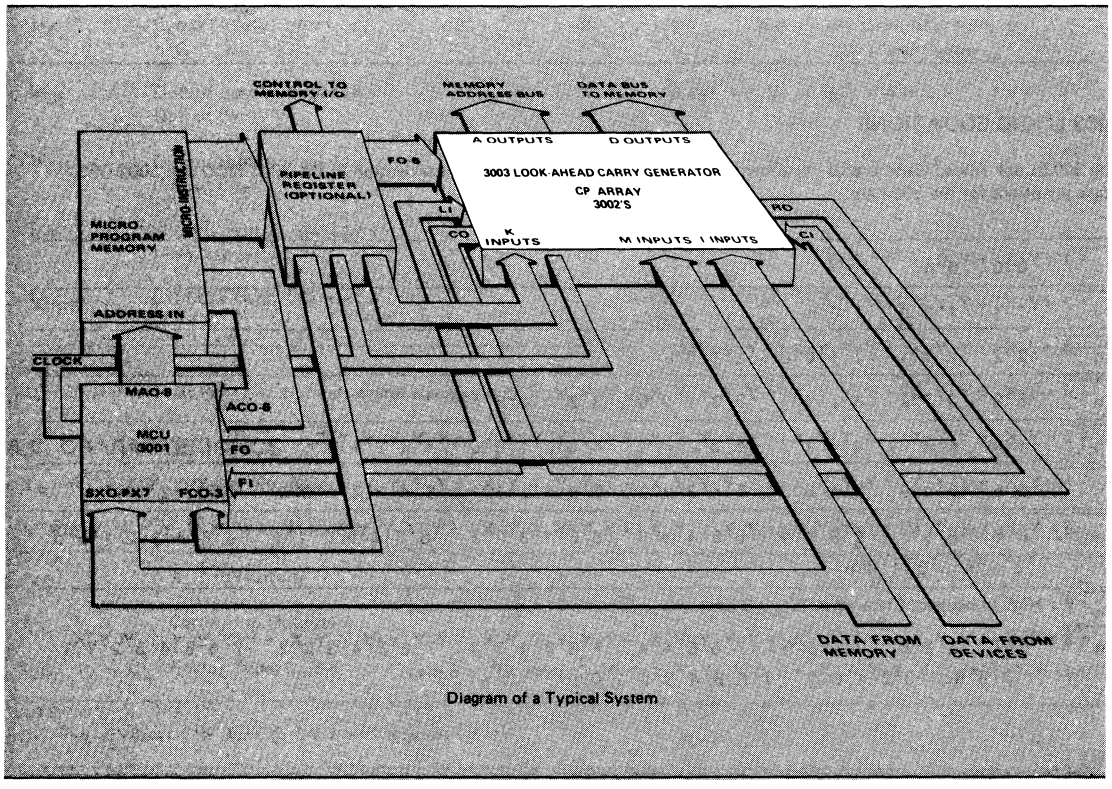
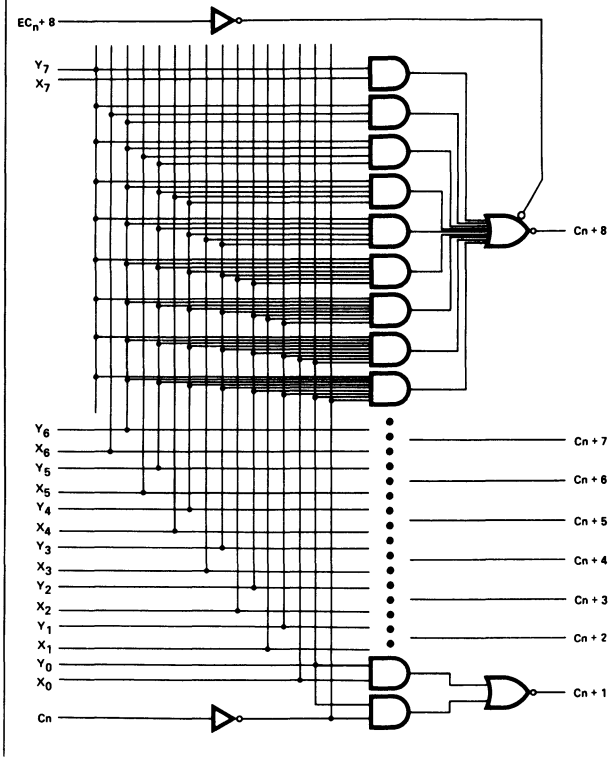


Diagram of a Typical System

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE
1,7,8,11 18,21,23 27	Y_0-Y_7	Standard carry look-ahead inputs	Active HIGH
2,5,6,10 19,20,24 26	X_0-X_7	Standard carry look-ahead inputs	Active HIGH
17	C_n	Carry input	Active LOW
4,9,12 13,15,16	C_{n+1}- C_{n+8}	Carry outputs	Active LOW
3	EC_{n+8}	C_{n+8} carry output enable	Active HIGH
28	V_{CC}	+5 volt supply	
14	GND	Ground	

LOGIC DIAGRAM



3003 LOGIC EQUATIONS

The 3003 Look-Ahead Generator is implemented in a compatible form for direct connection to the 3001 MCU and 3002 CPE. Logic equations for the 3003 are:

$$\overline{C_{n+1}} = Y_0X_0 + Y_0\overline{C}_n$$

$$\overline{C_{n+2}} = Y_1X_1 + Y_1Y_0X_0 + Y_1Y_0\overline{C}_n$$

$$\overline{C_{n+3}} = Y_2X_2 + Y_2Y_1X_1 + Y_2Y_1Y_0X_0 + Y_2Y_1Y_0\overline{C}_n$$

$$\overline{C_{n+4}} = Y_3X_3 + Y_3Y_2X_2 + Y_3Y_2Y_1X_1 + Y_3Y_2Y_1Y_0X_0 + Y_3Y_2Y_1Y_0\overline{C}_n$$

$$\overline{C_{n+5}} = Y_4X_4 + Y_4Y_3X_3 + Y_4Y_3Y_2X_2 + Y_4Y_3Y_2Y_1X_1 + Y_4Y_3Y_2Y_1Y_0X_0 + Y_4Y_3Y_2Y_1Y_0\overline{C}_n$$

$$\overline{C_{n+6}} = Y_5X_5 + Y_5Y_4X_4 + Y_5Y_4Y_3X_3 + Y_5Y_4Y_3Y_2X_2 + Y_5Y_4Y_3Y_2Y_1X_1 + Y_5Y_4Y_3Y_2Y_1Y_0X_0 + Y_5Y_4Y_3Y_2Y_1Y_0\overline{C}_n$$

$$\overline{C_{n+7}} = Y_6X_6 + Y_6Y_5X_5 + Y_6Y_5Y_4X_4 + Y_6Y_5Y_4Y_3X_3 + Y_6Y_5Y_4Y_3Y_2X_2 + Y_6Y_5Y_4Y_3Y_2Y_1X_1 + Y_6Y_5Y_4Y_3Y_2Y_1Y_0X_0 + Y_6Y_5Y_4Y_3Y_2Y_1Y_0\overline{C}_n$$

$\overline{C_{n+8}}$ = High Impedance State when $EC_n + 8$ Low

$$\overline{C_{n+8}} = Y_7X_7 + Y_7Y_6X_6 + Y_7Y_6Y_5X_5 + Y_7Y_6Y_5Y_4X_4 + Y_7Y_6Y_5Y_4Y_3X_3 + Y_7Y_6Y_5Y_4Y_3Y_2X_2 + Y_7Y_6Y_5Y_4Y_3Y_2Y_1X_1 + Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0X_0 + Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0\overline{C}_n \text{ when } EC_n + 8 \text{ high}$$

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Current . . . . . 100 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

T_A = 0°C to +70°C V_{CC} = 5.0V ±5%

SYMBOL	PARAMETER	MIN.	TYP.(1)	MAX.	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		-0.8	-1.0	V	I_C = -5 mA
I_F	Input Load Current:					
	X_6,X_7Cn,EC_n + 8		-0.07	-0.25	mA	V_F = 0.45V
	Y_7,X_0-X_5,		-0.200	-0.500	mA	
	Y_0-Y_6		-0.6	-1.5	mA	
I_R	Input Leakage Current:					
	C_n and EC_n + 8			40	µA	V_R = 5.25V
	All Other Inputs			100	µA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	V_{CC} = 5.0V
I_{CC}	Power Supply Current		80	130	mA	All Y and EC_n + 8 high, All X and C_n low
V_{OL}	Output Low Voltage (All Output Pins)		0.35	0.45	V	I_{OL} = 4 mA
V_{OH}	Output High Voltage (All Output Pins)	2.4	3		V	I_{OH} = -1 mA
I_{OS}	Short Circuit Output Current (All Output Pins)	-15	-40	-65	mA	V_{CC} = 5V
$I_{O(off)}$	Off-State Output Current (C_n + 8)			-100	µA	V_O = 0.45V
				+100	µA	V_O = 5.25V

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.

A.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ± 5%

SYMBOL	PARAMETER	MIN.	TYP.(1)	MAX.	UNIT
t_{XC}	X, Y to Outputs	3	10	20	ns
t_{CC}	Carry In to Outputs		13	30	ns
t_{EN}	Enable Time, C_n + 8		20	40	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . -55°C to +125°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Current . . . . . 100 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%.

SYMBOL	PARAMETER	MIN.	TYP.(1)	MAX.	UNIT	CONDITIONS
V_C	Input Clamp Voltage (All Input Pins)		-0.8	-1.2	V	I_C = -5 mA
I_F	Input Load Current:					
	X_6,X_7,Cn,EC_{n+8}		-0.07	-0.25	mA	V_F = 0.45V
	Y_7,X_0-X_5,		-0.200	-0.500	mA	
	Y_0-Y_6		-0.6	-1.5	mA	
I_R	Input Leakage Current:					
	C_n and EC_n + 8			40	µA	V_{CC} = 5.25V, V_R = 5.5V
	All Other Inputs			100	µA	
V_{IL}	Input Low Voltage			0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.1			V	V_{CC} = 5.0V
I_{CC}	Power Supply Current		80	130	mA	All Y and EC_n + 8 high, All X and C_n low
V_{OL}	Output Low Voltage (All Output Pins)		0.35	0.45	V	I_{OL} = 4 mA
V_{OH}	Output High Voltage (All Output Pins)	2.4	3		V	I_{OH} = -1 mA
I_{OS}	Short Circuit Output Current (All Output Pins)	-15	-40	-65	mA	V_{CC} = 5V
$I_{O(off)}$	Off-State Output Current (C_n + 8)			-100	µA	V_O = 0.45V
				+100	µA	V_O = 5.5V

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.

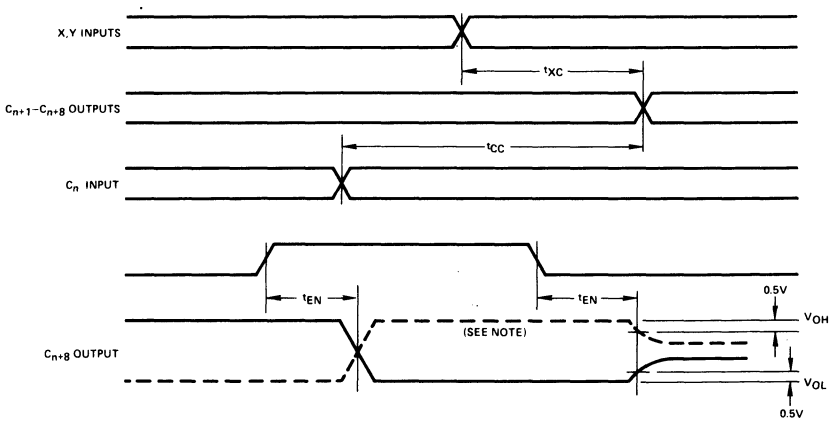
A.C. CHARACTERISTICS

T_A = -55°C to +125°C, V_{CC} = +5.0V ±10%

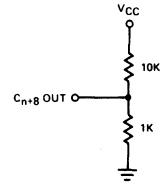
SYMBOL	PARAMETER	MIN.	TYP.(1)	MAX.	UNIT
t_{XC}	X, Y to Outputs	3	10	25	ns
t_{CC}	Carry In to Outputs		13	40	ns
t_{EN}	Enable Time, C_n + 8		20	50	ns

NOTE:
(1) Typical values are for T_A = 25°C and nominal supply voltage.

WAVEFORMS



NOTE: ALTERNATE TEST LOAD:



CAPACITANCE[2] T_A = 25°C

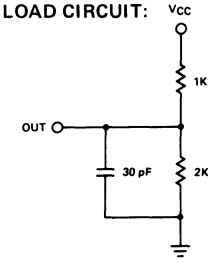
SYMBOL	PARAMETER		MIN	TYP	MAX	UNIT
C_{IN}	Input Capacitance	All inputs		12	20	pF
C_{OUT}	Output Capacitance	$C_n + 8$		7	12	pF

NOTE:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 5.0V, V_{CC} = 5.0V and T_A = 25°C.

TEST CONDITIONS:

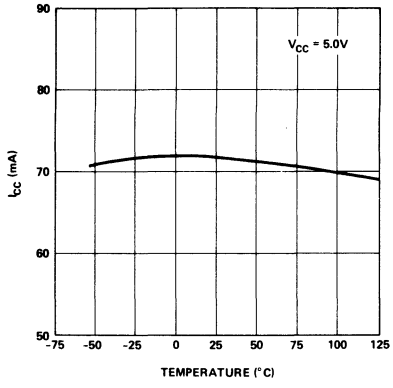
Input pulse amplitude of 2.5V.
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 5 mA and 30 pF.
Speed measurements are made at 1.5 volt levels.

TEST LOAD CIRCUIT:

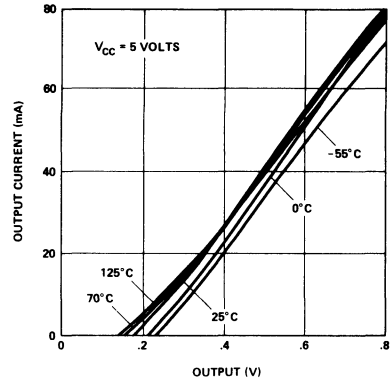


TYPICAL A.C. AND D.C. CHARACTERISTICS

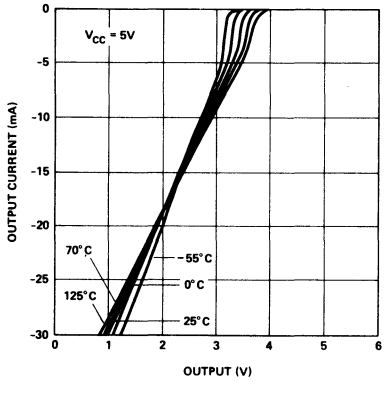
I_{CC} VS. TEMPERATURE



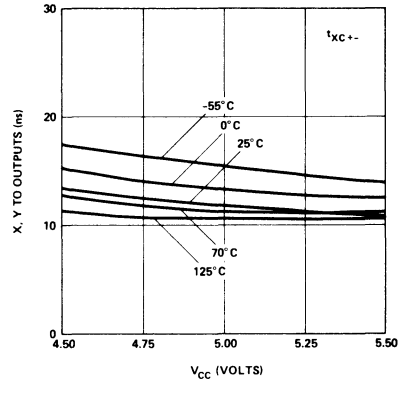
OUTPUT CURRENT VS. OUTPUT LOW VOLTAGE



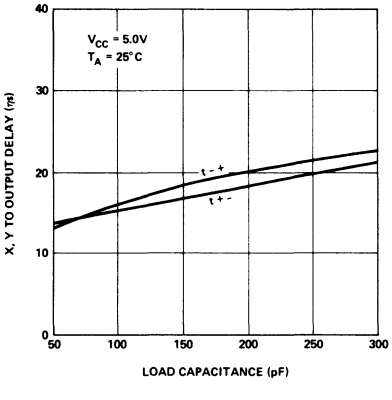
OUTPUT CURRENT VS. OUTPUT HIGH VOLTAGE



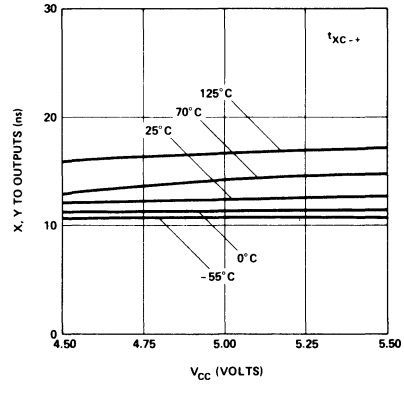
X, Y TO OUTPUTS VS. V_{CC} AND TEMPERATURE



X, Y TO OUTPUT DELAY VS. LOAD CAPACITANCE



X, Y TO OUTPUTS VS. V_{CC} AND TEMPERATURE



TYPICAL CONFIGURATIONS

The 3003 LCG can be directly tied to the 3001 MCU and a 3002 CP array of any word length. The following figures represent typical configurations of 16- and 32-bit CP arrays. Figures 1 and 2 illustrate use of the 3003 in a system where the carry output (CO) to the 3001 MCU is rippled through the high order CPE slice. Figure 3 illustrates use of the 3003 in a system where tri-state output C_{n+8} is connected directly to the flag input on the 3001 MCU. C_{n+8} is disabled during shift right by decoding that instruction externally, thus multiplexing C_{n+8} with the shift right (RO) output of the low order CPE slice.

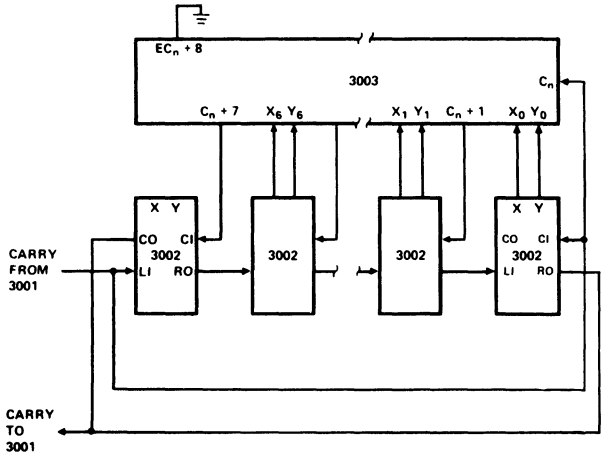


Figure 1. Carry Look-Ahead Configuration with Ripple through the Left Slice (16-Bit Array)

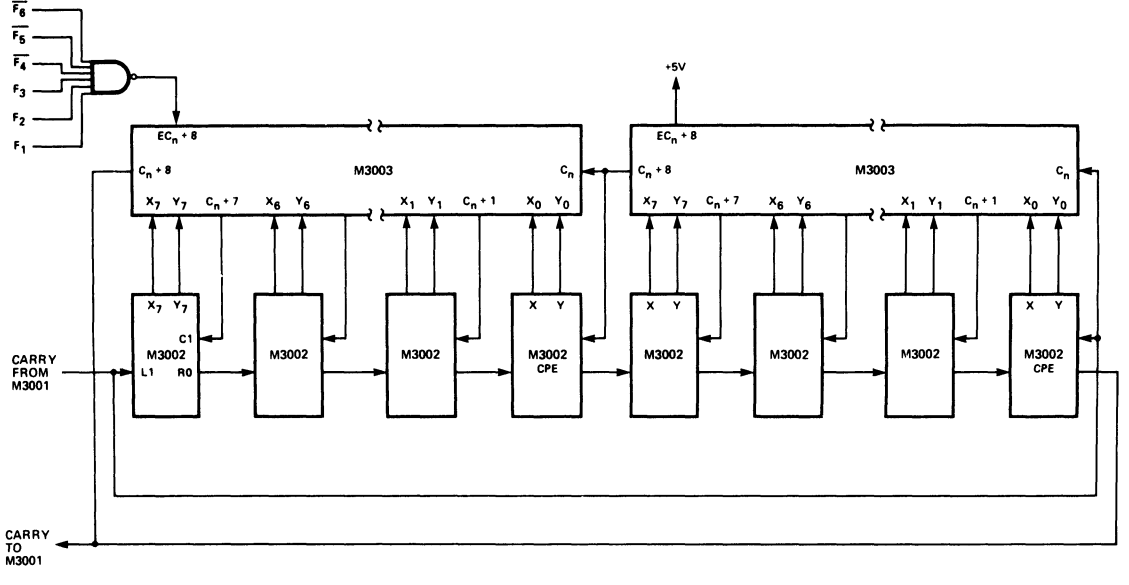


Figure 2. Carry Look-Ahead Configuration with No Carry Ripple through the Left Slice (32-Bit Array)

SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

3212 MULTI–MODE LATCH BUFFER

The INTEL® 3212 Multi-Mode Latch Buffer is a versatile 8-bit latch with three-state output buffers and built-in device select logic. It also contains an independent service request flip-flop for the generation of central processor interrupts. Because of its multi-mode capabilities, one or more 3212's can be used to implement many types of interface and support systems for Series 3000 computing elements including:

- Simple data latches
- Gated data buffers
- Multiplexers
- Bi-directional bus drivers
- Interrupting input/output ports

High Performance – 50 ns Write Cycle Time

Low Input Load Current – 250 μA Maximum

Three-State Fully Buffered Outputs

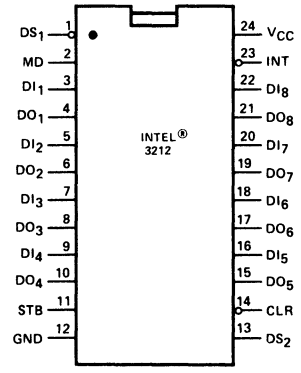
High Output Drive Capability

Independent Service Request Flip-Flop

Asynchronous Data Latch Clear

24 Pin DIP

PACKAGE CONFIGURATION



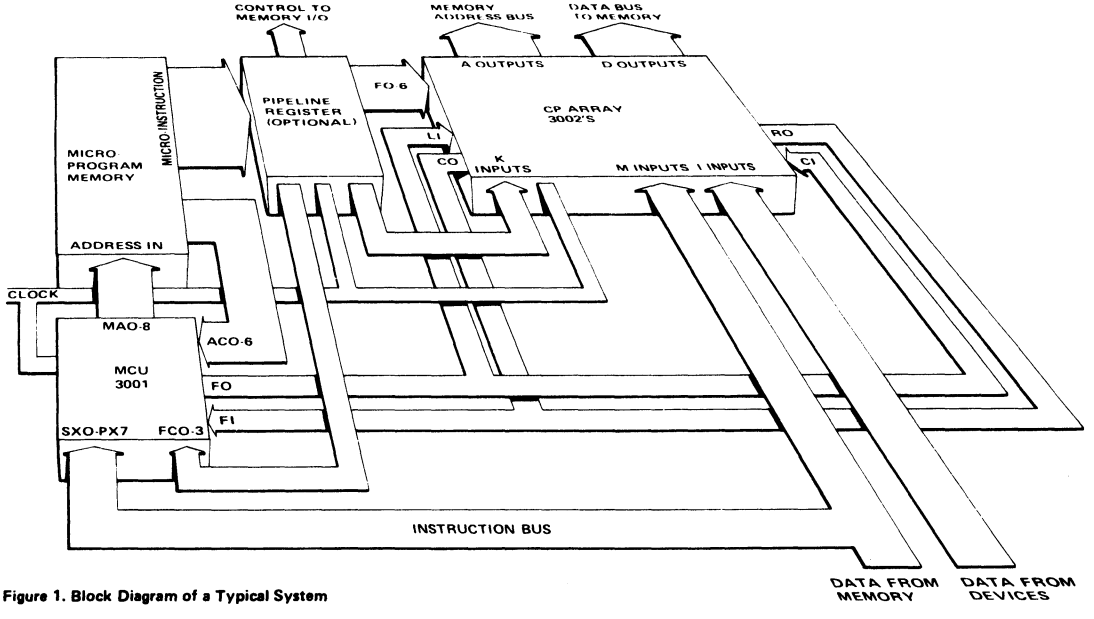


Figure 1. Block Diagram of a Typical System

PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE[1]
1	DS_1	Device Select Input 1	active LOW
2	MD	Mode Input When MD is high (output mode) the output buffers are enabled and the write signal to the data latches is obtained from the device select logic. When MD is low (input mode) the output buffer state is determined by the device select logic and the write signal is obtained from the strobe (STB) input.	
3, 5, 7, 9, 16, 18, 20, 22	DI_1–DI_8	Data Inputs The data inputs are connected to the D-inputs of the data latches.	
4, 6, 8, 10, 15, 17, 19, 21	DO_1–DO_8	Data Outputs The data outputs are the buffered outputs of the eight data latches.	three-state
11	STB	Strobe Input When MD is in the LOW state, the STB input provides the clock input to the data latch.	
12	GND	Ground	
13	DS_2	Device Select Input 2 When DS_1 is low and DS_2 is high, the device is selected.	
14	CLR	Clear	active LOW
23	INT	Interrupt Output The interrupt output will be active LOW (interrupting state) when either the service request flip-flop is low or the device is selected.	active LOW

NOTE:
(1) Active HIGH, unless otherwise specified.

FUNCTIONAL DESCRIPTION

The 3212 contains eight D-type data latches, eight three-state output buffers, a separate D-type service request flip-flop, and a flexible device select/mode control section.

DATA LATCHES

The Q-output of each data latch will follow the data on its corresponding date input line (DI_1–DI_8) while its clock input is high. Data will be latched when the internal write line WR is brought low. The output of each data latch is connected to a three-state, non-inverting output buffer. The internal enable line EN is bussed to each buffer. When the EN is high, the buffers are enabled and the data in each latch is available on its corresponding data output line (DO_0–DO_8).

DEVICE SELECT LOGIC

Two input lines DS_1 and $\overline{DS}_2$ are provided for device selection. When DS_1 is low and $\overline{DS}_2$ is high, the 3212 is selected.

MODE CONTROL SECTION

The 3212 may be operated in two modes. When the mode input line MD is low, the device is in the input mode. In this mode, the output buffers are enabled whenever the 3212 is selected; the internal WR line follows the STB input line.

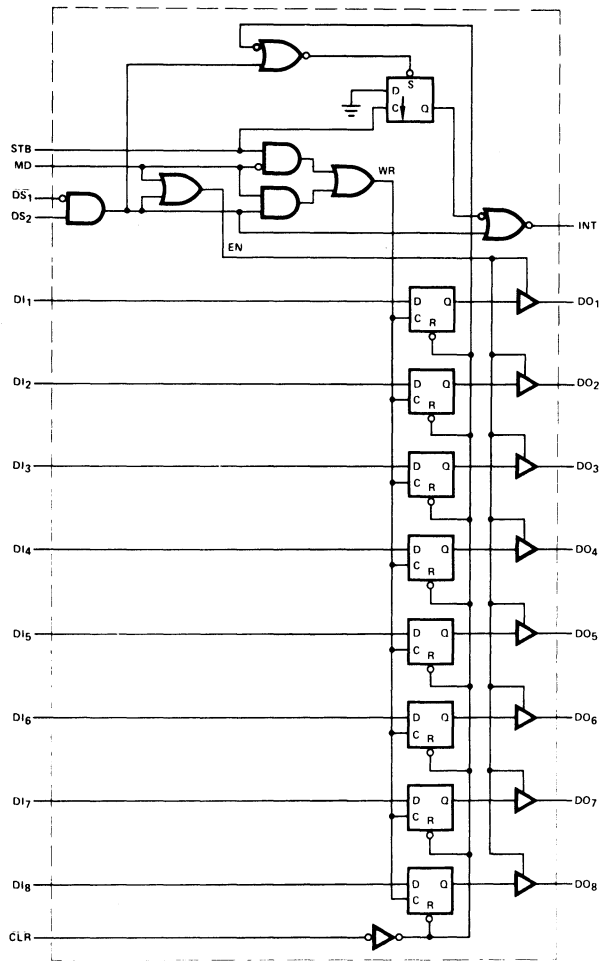
When MD is high, the device is in the output mode and, as a result, the output buffers are enabled. In this mode, the write signal for the data latch is obtained from the device select logic.

SERVICE REQUEST FLIP-FLOP AND STROBE

The service request flip-flop SR is used to generate and control central processor interrupt signals. For system reset, the SR flip-flop is placed in the non-interrupting state (i.e., SR is set) by bringing the CLR line low. This simultaneously clears (resets) the 8-bit data latch.

The Q output of the SR flip-flop is logically ORed with the output of device select logic and then inverted to provide the interrupt output INT. The 3212 is considered to be in the interrupting state when the INT output is low. This allows direct connection to the active LOW priority request inputs of the INTEL®3214 Interrupt Control Unit.

When operated in the input mode (i.e., MD low) the strobe input STB is used to synchronously write data into the data latch and place the SR flip-flop in the interrupting (reset) state. The interrupt is removed by the central processor when the interrupting 3212 is selected.



M3212 Logic Diagram

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may affect device reliability.

T_A = 0°C to +75°C V_{CC} = +5V ±5%

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
I_F	Input Load Current STB, DS_2, CLR, DI_1–DI_8 Inputs			-.25	mA	V_F = .45V
I_F	Input Load Current MD Input			-.75	mA	V_F = .45V
I_F	Input Load Current DS_1 Input			-1.0	mA	V_F = .45V
I_R	Input Leakage Current STB, DS, CLR, DI_1–DI_8 Inputs			10	µA	V_R = 5.25V
I_R	Input Leakage Current MD Input			30	µA	V_R = 5.25V
I_R	Input Leakage Current DS_1 Input			40	µA	V_R = 5.25V
V_C	Input Forward Voltage Clamp			-1	V	I_C = -5 mA
V_{IL}	Input "Low" Voltage			.85	V	
V_{IH}	Input "High" Voltage	2.0			V	
V_{OL}	Output "Low" Voltage			.45	V	I_{OL} = 15 mA
V_{OH}	Output "High" Voltage	3.65	4.0		V	I_{OH} = -1 mA
I_{SC}	Short Circuit Output Current	-15		-75	mA	V_{CC} = 5.0V
$\|I_O\|$	Output Leakage Current High Impedance State			20	µA	V_O = .45V/5.25V
I_{CC}	Power Supply Current		90	130	mA	

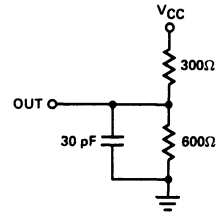
A.C. CHARACTERISTICS T_A = 0°C to 75°C, V_{CC} = +5.0V ± 5%

Symbol	Parameter	Min.	Typ.	Max.	Unit	
t_{PW}	Pulse Width	25			ns	
t_{PD}	Data To Output Delay			30	ns	
t_{WE}	Write Enable To Output Delay			40	ns	
t_{SET}	Data Setup Time	15			ns	
t_H	Data Hold Time	20			ns	
t_R	Reset To Output Delay			40	ns	
t_S	Set To Output Delay			30	ns	
t_E	Output Enable Time			45	ns	C_L = 30 pf
t_C	Clear To Output Display			45	ns	

TEST CONDITIONS:

Input pulse amplitude of 2.5 volts.
Input rise and fall times of 5 ns between 1 volt and 2 volts.
Output load of 15 mA and 30 pF.
Speed measurements are taken at the 1.5 volt level.

TEST LOAD CIRCUIT:



CAPACITANCE[1]

Symbol	Test	LIMITS Min.	Typ.	Max.	Units
C_{IN}	DS_1, MD Input Capacitance		9	12	pf
C_{IN}	DS_2, CLR, STB, DI_1–DI_8 Input Capacitance		5	9	pf
C_{OUT}	DO_1–DO_8 Output Capacitance		8	12	pf

NOTE:
(1) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5V and T_A = 25°C.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . -55°C to +125°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: 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 ratings for extended periods may affect device reliability.

T_A = -55°C to +125°C; V_{CC} = 5.0V ±10%

Symbol	Parameter	Min.	Typ.	Max.	Unit	Conditions
I_F	Input Load Current STB, DS_2, CLR, DI_1–DI_8 Inputs			-.25	mA	V_F = .45V
I_F	Input Load Current MD Input			-.75	mA	V_F = .45V
I_F	Input Load Current DS_1 Input			-1.0	mA	V_F = .45V
I_R	Input Leakage Current STB, DS, CLR, DI_1–DI_8 Inputs			10	µA	V_R = 5.5V
I_R	Input Leakage Current MD Input			30	µA	V_R = 5.5V
I_R	Input Leakage Current DS_1 Input			40	µA	V_R = 5.5V
V_C	Input Forward Voltage Clamp			1.2	V	I_C = -5 mA
V_{IL}	Input "Low" Voltage			.80	V	
V_{IH}	Input "High" Voltage	2.0			V	
V_{OL}	Output "Low" Voltage			.45	V	I_{OL} = 10 mA
V_{OH}	Output "High" Voltage	3.5	4.0		V	I_{OH} = .5 mA
I_{SC}	Short Circuit Output Current	-15		-75	mA	V_{CC} = 5.0V
$\|I_O\|$	Output Leakage Current High Impedance State			20	µA	V_O = .45V/5.5V
I_{CC}	Power Supply Current		90	145	mA	

MILITARY TEMP.

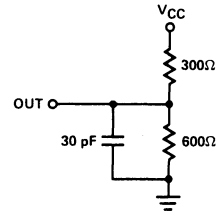
A.C. CHARACTERISTICS T_A = -55°C to +125°C, V_{CC} = 5.0V ± 10%

Symbol	Parameter	Min.	Typ.	Max.	Unit	
t_{PW}	Pulse Width	40			ns	
t_{PD}	Data To Output Delay			30	ns	
t_{WE}	Write Enable To Output Delay			50	ns	
t_{SET}	Data Setup Time	20			ns	
t_H	Data Hold Time	30			ns	
t_R	Reset To Output Delay			55	ns	
t_S	Set To Output Delay			35	ns	
t_E	Output Enable Time			50	ns	C_L = 30 pf
t_C	Clear To Output Display			55	ns	

TEST CONDITIONS:

Input pulse amplitude of 2.5 volts.
Input rise and fall times of 5 ns between 1 volt and 2 volts.
Output load of 15 mA and 30 pF.
Speed measurements are taken at the 1.5 volt level.

TEST LOAD CIRCUIT:

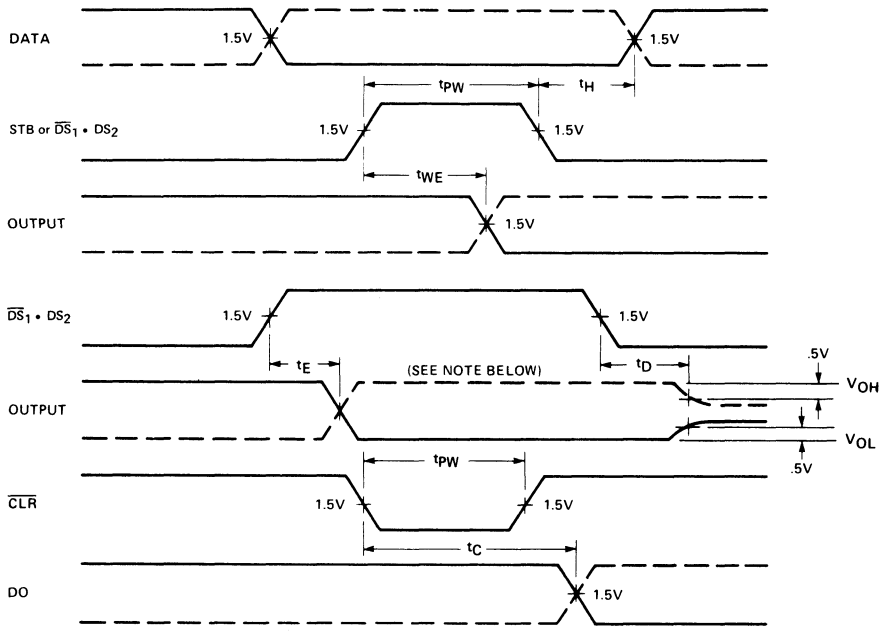


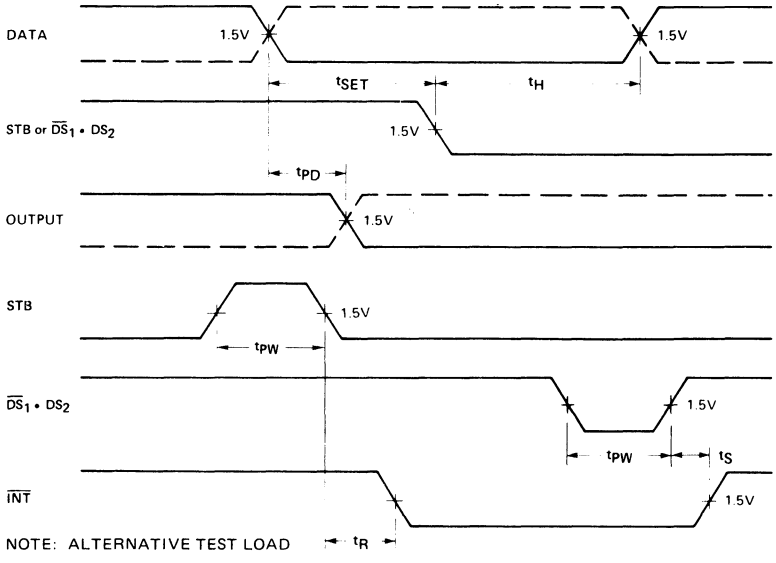
CAPACITANCE[1]

Symbol	Test	LIMITS			
		Min.	Typ.	Max.	Units
C_{IN}	DS_1, MD Input Capacitance		9	12	pf
C_{IN}	DS_2, CLR, STB, DI_1–DI_8 Input Capacitance		5	9	pf
C_{OUT}	DO_1–DO_8 Output Capacitance		8	12	pf

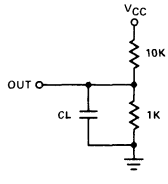
NOTE:
(1) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1 MHz, V_{BIAS} = 2.5V, V_{CC} = 5V and T_A = 25°C.

WAVEFORMS



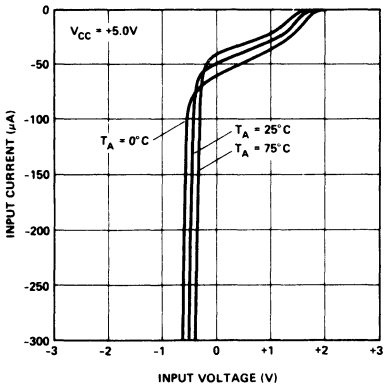


NOTE: ALTERNATIVE TEST LOAD tR

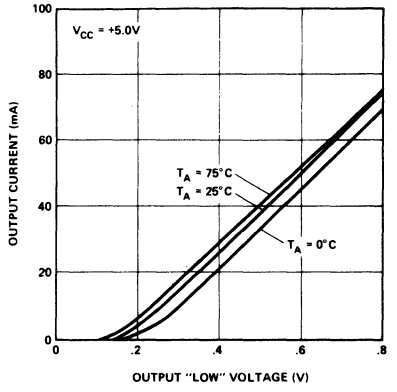


TYPICAL A.C. AND D.C. CHARACTERISTICS

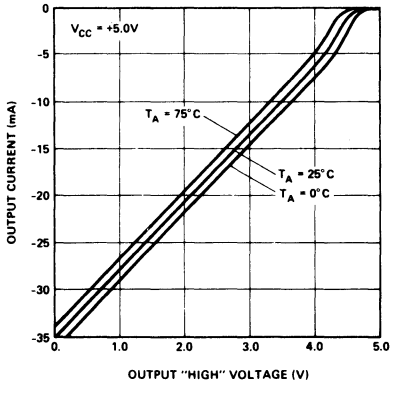
INPUT CURRENT VS. INPUT VOLTAGE



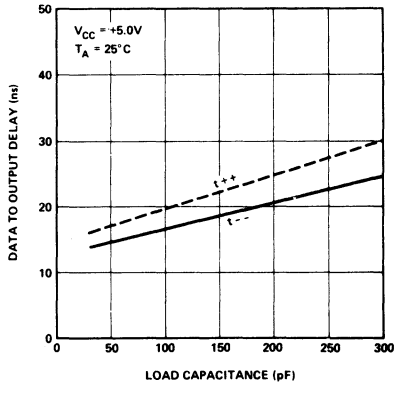
OUTPUT CURRENT VS. OUTPUT "LOW" VOLTAGE



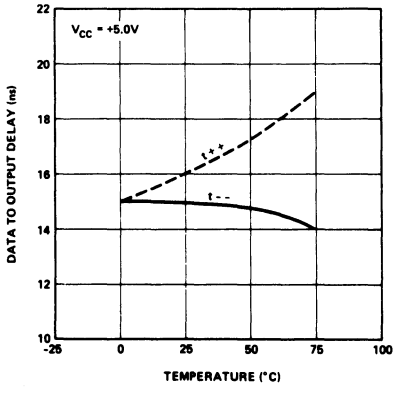
OUTPUT CURRENT VS. OUTPUT "HIGH" VOLTAGE



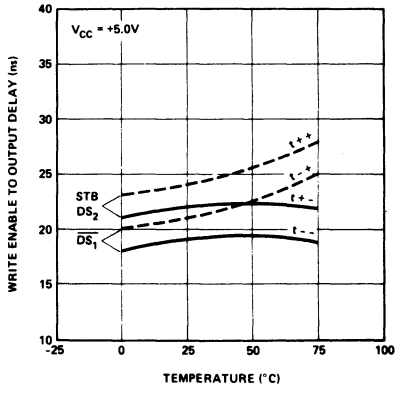
DATA TO OUTPUT DELAY VS. LOAD CAPACITANCE



DATA TO OUTPUT DELAY VS. TEMPERATURE

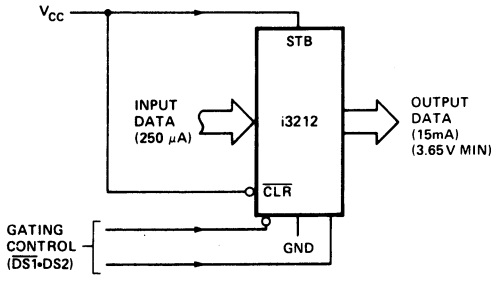


WRITE ENABLE TO OUTPUT DELAY VS. TEMPERATURE

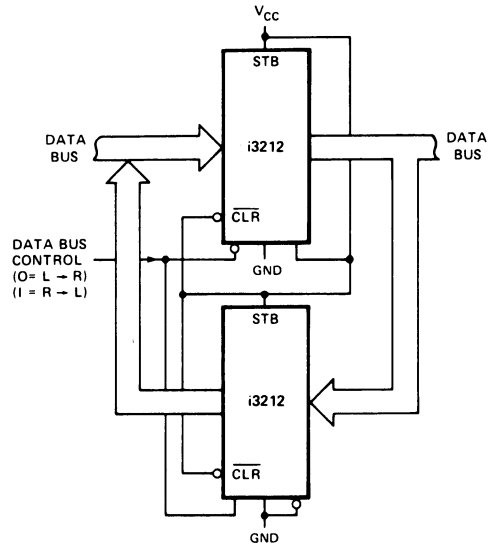


TYPICAL CONFIGURATIONS

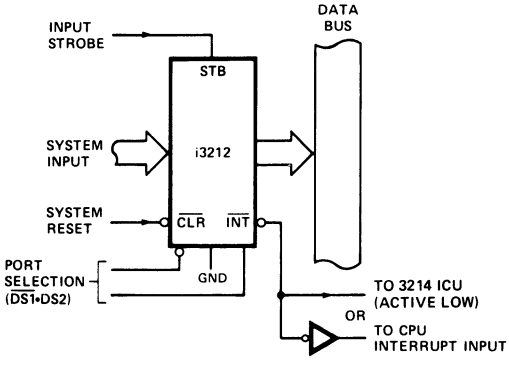
GATED BUFFER (TRI-STATE)



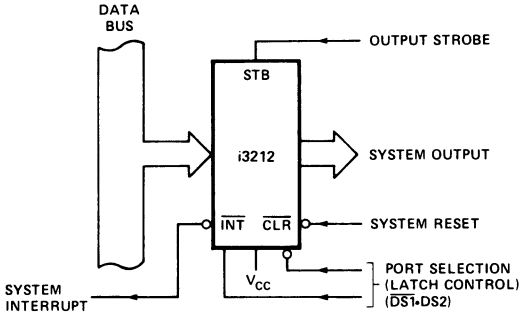
BI-DIRECTIONAL BUS DRIVER



INTERRUPTING INPUT PORT



OUTPUT PORT (WITH HAND-SHAKING)



SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

3214 INTERRUPT CONTROL UNIT

The Intel®3214 Interrupt Control Unit (ICU) implements multi-level interrupt capability for systems designed with Series 3000 computing elements.

The ICU accepts an asynchronous interrupt strobe from the 3001 Microprogram Control Unit or a bit in microprogram memory and generates a synchronous interrupt acknowledge and an interrupt vector which may be directed to the MCU or CP Array to uniquely identify the interrupt source.

The ICU is fully expandable in 8-level increments and provides the following system capabilities:

- Eight unique priority levels per ICU
- Automatic Priority Determination
- Programmable Status
- N-level expansion capability
- Automatic interrupt vector generation

High Performance – 80 ns Cycle Time

Compatible with Intel 3001 MCU and 3002 CPE

8-Bit Priority Interrupt Request Latch

4-Bit Priority Status Latch

3-Bit Priority Encoder with Open Collector Outputs

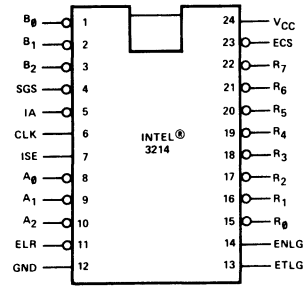
DTL and TTL Compatible

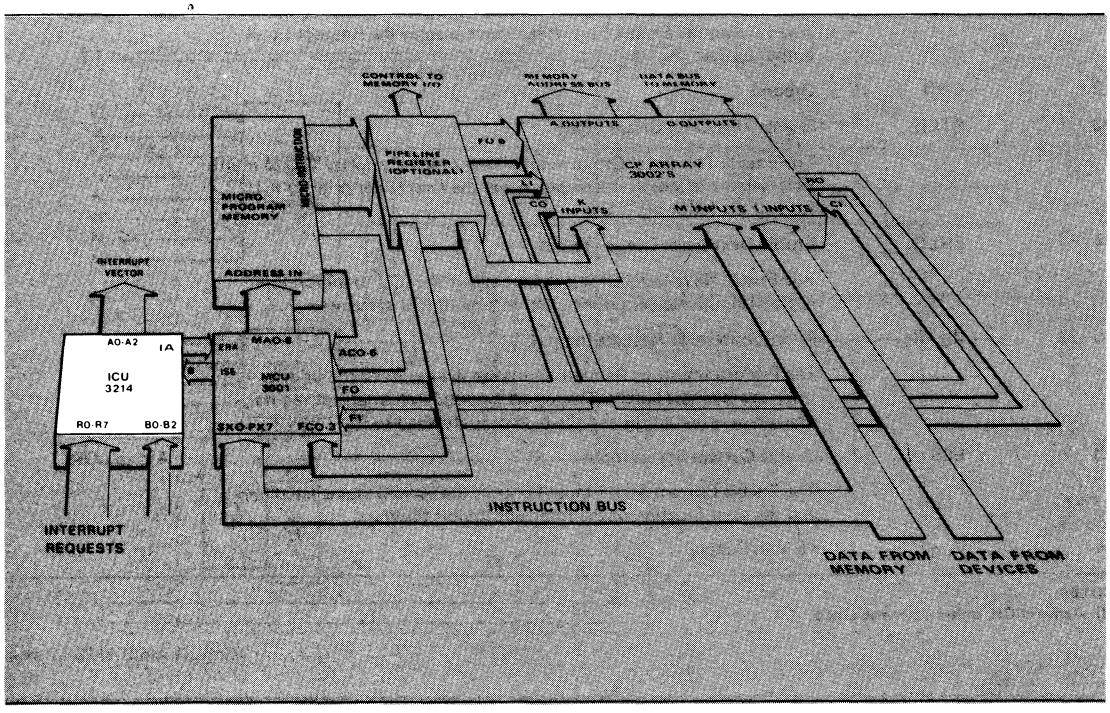
8-Level Priority Comparator

Fully Expandable

24-Pin DIP

PACKAGE CONFIGURATION





PIN DESCRIPTION

PIN	SYMBOL	NAME AND FUNCTION	TYPE[1]
1–3	B_0–B_2	Current Status Inputs The Current Status inputs carry the binary value modulo 8 of the current priority level to the current status latch.	Active LOW
4	SGS	Status Group Select Input The Status Group Select input informs the ICU that the current priority level does belong to the group level assigned to the ICU.	Active LOW
5	IA	Interrupt Acknowledge The Interrupt Acknowledge Output will only be active from the ICU (multi-ICU system) which has received a priority request at a level superior to the current status. It signals the controlled device (usually the processor) and the other ICUs OR-tied on the Interrupt Acknowledge line that an interrupt request has been recognized. The IA signal also sets the Interrupt Disable flip-flop (it overrides the clear function of the ECS input).	Active LOW Open-Collector Output
6	CLK	Clock Input The Clock input is used to synchronize the interrupt acknowledge with the operation of the device which it controls.	
7	ISE	Interrupt Strobe Enable Input The Interrupt Strobe Enable input informs the ICU that it is authorized to enter the interrupt mode.	
8–10	A_0–A_2	Request Level Outputs When valid, the Request Level outputs carry the binary value (modulo 8) of the highest priority request present at the priority request inputs or stored in the priority request latch. The request level outputs can become active only with the ICU which has received the highest priority request with a level superior to the current status.	Active LOW Open-Collector
11	ELR	Enable Level Read Input When active, the Enable Level Read input enables the Request Level output buffers (A_0–A_2).	Active LOW
12	GND	Ground	
13	ETLG	Enable This Level Group Input The Enable This Level Group input allows a higher priority ICU in multi-ICU systems to inhibit interrupts within the next lower priority ICU (and all the following ICUs).	
14	ENLG	Enable Next Level Group Output The Enable Next Level Group output allows the ICU to inhibit interrupts within the lower priority ICU in a multi-ICU system.	
15–22	R_0–R_7	Priority Interrupt Request Inputs The Priority Interrupt Request inputs are the inputs of the priority Interrupt Request Latch. The lowest priority level interrupt request signal is attached to R_0 and the highest is attached to R_7.	Active LOW
23	ECS	Enable Current Status Input The Enable Current Status input controls the current status latch and the clear function of the Interrupt Inhibit flip-flop.	Active LOW
24	V_{CC}	+5 Volt Supply	

NOTE:
(1) Active HIGH, unless otherwise noted.

FUNCTIONAL AND LOGICAL DESCRIPTION

The ICU adds interrupt capability to suitably microprogrammed processors or controllers. One or more of these units allows external signals called interrupt requests to cause the processor/controller to suspend execution of the active process, save its status, and initiate execution of a new task as requested by the interrupt signal.

It is customary to strobe the ICU at the end of each instruction execution. At that time, if an interrupt request is acknowledged by the ICU, the MCU is forced to follow the interrupt microprogram sequence.

Figure 1 shows the block diagram of the ICU. Interrupt requests pass through the interrupt request latch and priority encoder to the magnitude comparator. The output of the priority encoder is the binary equivalent of the highest active priority request. At the comparator, this value is compared with the Current Status (currently active priority level) contained in the current status latch. A request, if acknowledged at interrupt strobe time, will cause the interrupt flip-flop to enter the "interrupt active" state for one microinstruction cycle. This action causes the interrupt acknowledge (IA) signal to go low and sets the interrupt disable flip-flop.

The IA signal constitutes the interrupt command to the processor. It can directly force entry into the interrupt service routine as demonstrated in the appendix. As part of this routine, the microprogram normally reads the requesting level via the request level output bus. This information which is saved in the request latch can be enabled onto one of the processor input data buses using the enable level read input. Once the interrupt handler has determined the requesting level, it normally writes this level back into the current status register of the ICU. This action resets the interrupt disable flip-flop and acts to block any further request at this level or lower levels.

Entry into a macro level interrupt service routine may be vectored using the request level information to generate a subroutine address which corresponds to the level. Exit from such a macroprogram should normally restore the prior status in the current status latch.

The Enable This Level Group (ETLG) input and the Enable Next Level Group (ENLG) output can be used in a daisy chain fashion, as each ICU is capable of inhibiting interrupts from all of the following ICUs in a multiple ICU configuration.

The interrupt acknowledge flip-flop is set to the active LOW state on the rising edge of the clock when the following conditions are met:

- An active request level (R_0–R_7) is greater than the current status B_0–B_2
- The interrupt mode (ISE) is active
- ETLG is enabled
- The interrupt disable flip-flop is reset

When active, the IA signal asynchronously sets the disable flip-flop and holds the requests in the request latch until new current status information (B_0–B_2, SGS) is enabled (ECS) into the current status latch. The disable flip-flop is reset at the completion of this load operation.

During this process, ENLG will be enabled only if the following conditions are met:

- ETLG is enabled
- The current status (SGS) does not belong to this level group
- There is no active request at this level

The request level outputs A_0–A_2 and the IA output are open-collector to permit bussing of these lines in multi-ICU configuration.

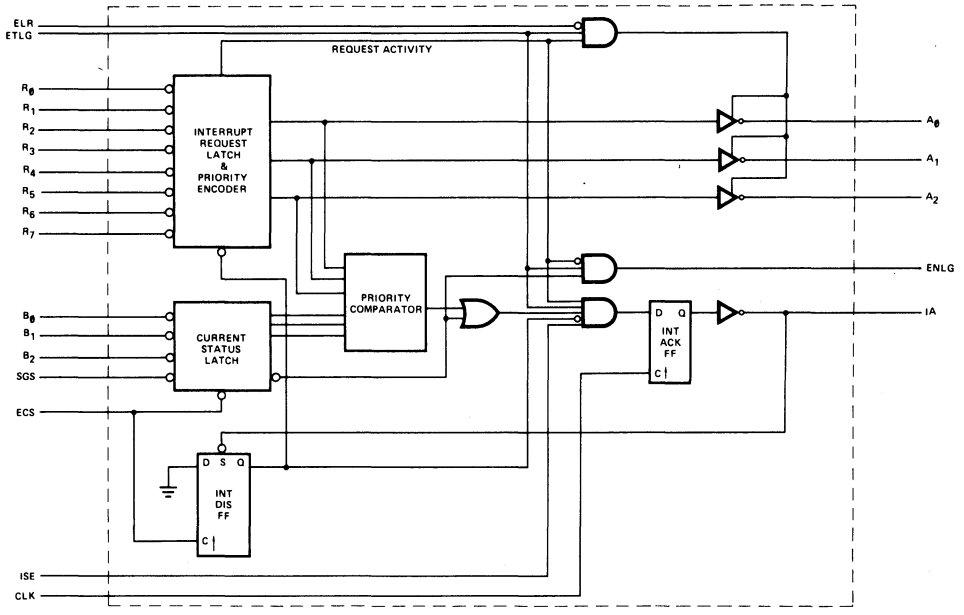


Figure 1. 3214 Block Diagram.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias
- Ceramic . . . . . -65°C to +75°C
- Plastic . . . . . 0°C to +75°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Currents . . . . . 100 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

T_A = 0°C to +75°C, V_{CC} = 5.0V ±5%

SYMBOL	PARAMETER		LIMITS MIN	LIMITS TYP(1)	LIMITS MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (all inputs)				-1.0	V	I_C = -5 mA
I_F	Input Forward Current:	ETLG input		-.15	-0.5	mA	V_F = 0.45V
		all other inputs		-.08	-0.25	mA	
I_R	Input Reverse Current:	ETLG input			80	μA	V_R = 5.25V
		all other inputs			40	μA	
V_{IL}	Input LOW Voltage:	all inputs			0.8	V	V_{CC} = 5.0V
V_{IH}	Input HIGH Voltage:	all inputs	2.0			V	V_{CC} = 5.0V
I_{CC}	Power Supply Current(2)			90	130	mA	
V_{OL}	Output LOW Voltage:	all outputs		.3	.45	V	I_{OL} = 15 mA
V_{OH}	Output HIGH Voltage:	ENLG output	2.4	3.0		V	I_{OH} = -1 mA
I_{OS}	Short Circuit Output Current:	ENLG output	-20	-35	-55	mA	V_{CC} = 5.0V
I_{CEX}	Output Leakage Current:	IA and A_0–A_2 outputs			100	μA	V_{CEX} = 5.25V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) B_0-B_2, SGS, CLK, R_0-R_4 grounded, all other inputs and all outputs open.

A.C. CHARACTERISTICS

T_A = 0°C to +75°C, V_{CC} = +5V ± 5%

SYMBOL	PARAMETER	LIMITS MIN	LIMITS TYP(1)	LIMITS MAX	UNIT
t_{CY}	CLK Cycle Time	80			ns
t_{PW}	CLK, ECS, IA Pulse Width	25	15		ns
	Interrupt Flip-Flop Next State Determination:				
t_{ISS}	ISE Set-Up Time to CLK	16	12		ns
t_{ISH}	ISE Hold Time After CLK	20	10		ns
t_{ETCS}[2]	ETLG Set-Up Time to CLK	25	12		ns
t_{ETCH}[2]	ETLG Hold Time After CLK	20	10		ns
t_{ECCS}[3]	ECS Set-Up Time to CLK (to clear interrupt inhibit prior to CLK)	80	25		ns
t_{ECCH}[3]	ECS Hold Time After CLK (to hold interrupt inhibit)	0			ns
t_{ECRS}[3]	ECS Set-Up Time to CLK (to enable new requests through the request latch)	110	70		ns
t_{ECRH}[3]	ECS Hold Time After CLK (to hold requests in request latch)	0			
t_{ECSS}[2]	ECS Set-Up Time to CLK (to enable new status through the status latch)	75	70		ns
t_{ECSH}[2]	ECS Hold Time After CLK (to hold status in status latch)	0			ns
t_{DCS}[2]	SGS and $B_\emptyset$-B_2 Set-Up Time to CLK (current status latch enabled)	70	50		ns
t_{DCH}[2]	SGS and $B_\emptyset$-B_2 Hold Time After CLK (current status latch enabled)	0			ns
t_{RCS}[3]	$R_\emptyset$-R_7 Set-Up Time to CLK (request latch enabled)	90	55		ns
t_{RCH}[3]	$R_\emptyset$-R_7 Hold Time After CLK (request latch enabled)	0			ns
t_{ICS}	IA Set-Up Time to CLK (to set interrupt inhibit F.F. before CLK)	55	35		ns
t_{CI}	CLK to IA Propagation Delay		15	25	ns
	Contents of Request Latch and Request Level Output Status Determination:				
t_{RIS}[4]	$R_\emptyset$-R_7 Set-Up Time to IA	10	0		ns
t_{RIH}[4]	$R_\emptyset$-R_7 Hold Time After IA	35	20		ns
t_{RA}	$R_\emptyset$-R_7 to $A_\emptyset$-A_2 Propagation Delay (request latch enabled)		80	100	ns
t_{ELA}	ELR to $A_\emptyset$-A_2 Propagation Delay		40	55	ns
t_{ECA}	ECS to $A_\emptyset$-A_2 Propagation Delay (to enable new requests through request latch)		100	120	ns
t_{ETA}	ETLG to $A_\emptyset$-A_2 Propagation Delay		35	70	ns

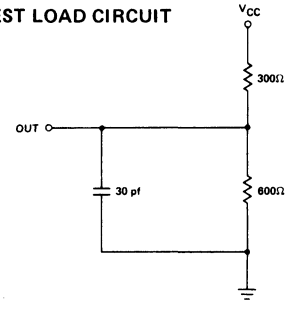
A.C. CHARACTERISTICS (CON'T)

SYMBOL	PARAMETER	LIMITS MIN	TYP(1)	MAX	UNIT
	Contents of Current Priority Status Latch Determination:				
t_{DECS}[4]	SGS and $B_{\emptyset}$-B_2 Set-Up Time to ECS	15	10		ns
t_{DECH}[4]	SGS and $B_{\emptyset}$-B_2 Hold Time After ECS	15	10		ns
	Enable Next Level Group Determination:				
t_{REN}	$R_{\emptyset}$-R_7 to ENLG Propagation Delay		45	70	ns
t_{ETEN}	ETLG to ENLG Propagation Delay		20	25	ns
t_{ECRN}	ECS to ENLG Propagation Delay (enabling new request through the request latch)		85	90	ns
t_{ECSN}	ECS to ENLG Propagation Delay (enabling new SGS through status latch)		35	55	ns

NOTES:
(1) Typical values are for $T_A = 25°C$ and nominal supply voltage.
(2) Required for proper operation if ISE is enabled during next clock pulse.
(3) These times are not required for proper operation but for desired change in interrupt flip-flop.
(4) Required for new request or status to be properly loaded.
(5) $t_{CY} = t_{ICS} + t_{CI}$

TEST CONDITIONS:
Input pulse amplitude: 2.5 volts.
Input rise and fall times: 5 ns between 1 and 2 volts.
Output loading of 15 mA and 30 pf.
Speed measurements taken at the 1.5V levels.

TEST LOAD CIRCUIT



CAPACITANCE(5)

$T_A = 25°C$

SYMBOL	PARAMETER	MIN	LIMITS TYP(1)	MAX	UNIT
C_{IN}	Input Capacitance		5	10	pf
C_{OUT}	Output Capacitance		7	12	pf

TEST CONDITIONS:
$V_{BIAS} = 2.5V$, $V_{CC} = 5V$, $T_A = 25°C$, f = 1 MHz

NOTE:
(5) This parameter is periodically sampled and not 100% tested.

D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias
CerDip . . . . . -55°C to +125°C
Storage Temperature . . . . . -65°C to +160°C
All Output and Supply Voltages . . . . . -0.5V to +7V
All Input Voltages . . . . . -1.0V to +5.5V
Output Currents . . . . . 100 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

T_A = -55°C to +125°C; V_{CC} = 5.0V ± 10%

SYMBOL	PARAMETER		LIMITS MIN	TYP(1)	MAX	UNIT	CONDITIONS
V_C	Input Clamp Voltage (all inputs)				-1.2	V	I_C = −5 mA
I_F	Input Forward Current:	ETLG input		-.15	-0.5	mA	V_F = 0.45V
		all other inputs		-.08	-0.25	mA	
I_R	Input Reverse Current:	ETLG input			80	μA	V_R = 5.5V
		all other inputs			40	μA	
V_{IL}	Input LOW Voltage:	all inputs			0.8	V	V_{CC} = 5.0V
V_{IH}	Input HIGH Voltage:	all inputs	2.0			V	V_{CC} = 5.0V
I_{CC}	Power Supply Current(2)			90	130	mA	
V_{OL}	Output LOW Voltage:	all outputs		.3	.45	V	I_{OL} = 10 mA
V_{OH}	Output HIGH Voltage:	ENLG output	2.4	3.0		V	I_{OH} = -1 mA
I_{OS}	Short Circuit Output Current:	ENLG output	-15	-35	-55	mA	V_{CC} = 5.0V
I_{CEX}	Output Leakage Current:	IA and A_0-A_3 outputs			100	μA	V_{CEX} = 5.5V

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) B_0-B_2, SGS, CLK, R_0-R_4 grounded, all other inputs and all outputs open.

A.C. CHARACTERISTICS

T_A = -55°C to +125°C; V_{CC} = 5.0V ± 10%

SYMBOL	PARAMETER	LIMITS MIN	LIMITS TYP(1)	LIMITS MAX	UNIT
t_{CY}	CLK Cycle Time(5)	85			ns
t_{PW}	CLK, ECS, IA Pulse Width	25	15		ns
	Interrupt Flip-Flop Next State Determination:				
t_{ISS}	ISE Set-Up Time to CLK	16	12		ns
t_{ISH}	ISE Hold Time After CLK	20	10		ns
t_{ETCS}[2]	ETLG Set-Up Time to CLK	25	12		ns
t_{ETCH}[2]	ETLG Hold Time After CLK	20	10		ns
t_{ECCS}[3]	ECS Set-Up Time to CLK (to clear interrupt inhibit prior to CLK)	85	25		ns
t_{ECCH}[3]	ECS Hold Time After CLK (to hold interrupt inhibit)	0			ns
t_{ECRS}[3]	ECS Set-Up Time to CLK (to enable new requests through the request latch)	110	70		ns
t_{ECRH}[3]	ECS Hold Time After CLK (to hold requests in request latch)	0			
t_{ECSS}[2]	ECS Set-Up Time to CLK (to enable new status through the status latch)	85	70		ns
t_{ECSH}[2]	ECS Hold Time After CLK (to hold status in status latch)	0			ns
t_{DCS}[2]	SGS and $B_\emptyset$-B_2 Set-Up Time to CLK (current status latch enabled)	90	50		ns
t_{DCH}[2]	SGS and $B_\emptyset$-B_2 Hold Time After CLK (current status latch enabled)	0			ns
t_{RCS}[3]	$R_\emptyset$-R_7 Set-Up Time to CLK (request latch enabled)	100	55		ns
t_{RCH}[3]	$R_\emptyset$-R_7 Hold Time After CLK (request latch enabled)	0			ns
t_{ICS}	IA Set-Up Time to CLK (to set interrupt inhibit F.F. before CLK)	55	35		ns
t_{CI}	CLK to IA Propagation Delay		15	30	ns
	Contents of Request Latch and Request Level Output Status Determination:				
t_{RIS}[4]	$R_\emptyset$-R_7 Set-Up Time to IA	10	0		ns
t_{RIH}[4]	$R_\emptyset$-R_7 Hold Time After IA	35	20		ns
t_{RA}	$R_\emptyset$-R_7 to $A_\emptyset$-A_2 Propagation Delay (request latch enabled)		80	100	ns
t_{ELA}	ELR to $A_\emptyset$-A_2 Propagation Delay		40	55	ns
t_{ECA}	ECS to $A_\emptyset$-A_2 Propagation Delay (to enable new requests through request latch)		100	130	ns
t_{ETA}	ETLG to $A_\emptyset$-A_2 Propagation Delay		35	70	ns

MILITARY TEMP.

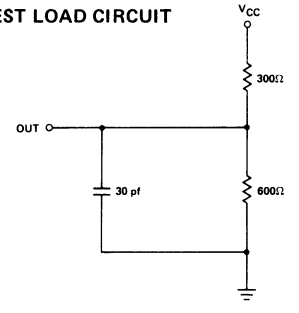
A.C. CHARACTERISTICS (CON'T)

SYMBOL	PARAMETER	LIMITS MIN	LIMITS TYP(1)	LIMITS MAX	UNIT
	Contents of Current Priority Status Latch Determination:				
t_{DECS}[4]	SGS and B_0-B_2 Set-Up Time to ECS	20	10		ns
t_{DECH}[4]	SGS and B_0-B_2 Hold Time After ECS	20	10		ns
	Enable Next Level Group Determination:				
t_{REN}	R_0-R_7 to ENLG Propagation Delay		45	70	ns
t_{ETEN}	ETLG to ENLG Propagation Delay		20	30	ns
t_{ECRN}	ECS to ENLG Propagation Delay (enabling new request through the request latch)		85	110	ns
t_{ECSN}	ECS to ENLG Propagation Delay (enabling new SGS through status latch)		35	55	ns

NOTES:
(1) Typical values are for T_A = 25°C and nominal supply voltage.
(2) Required for proper operation if ISE is enabled during next clock pulse.
(3) These times are not required for proper operation but for desired change in interrupt flip-flop.
(4) Required for new request or status to be properly loaded.
(5) $t_{CY} = t_{ICS} + t_{CI}$

TEST CONDITIONS:
Input pulse amplitude: 2.5 volts.
Input rise and fall times: 5 ns between 1 and 2 volts.
Output loading of 15 mA and 30 pf.
Speed measurements taken at the 1.5V levels.

TEST LOAD CIRCUIT



CAPACITANCE(5)

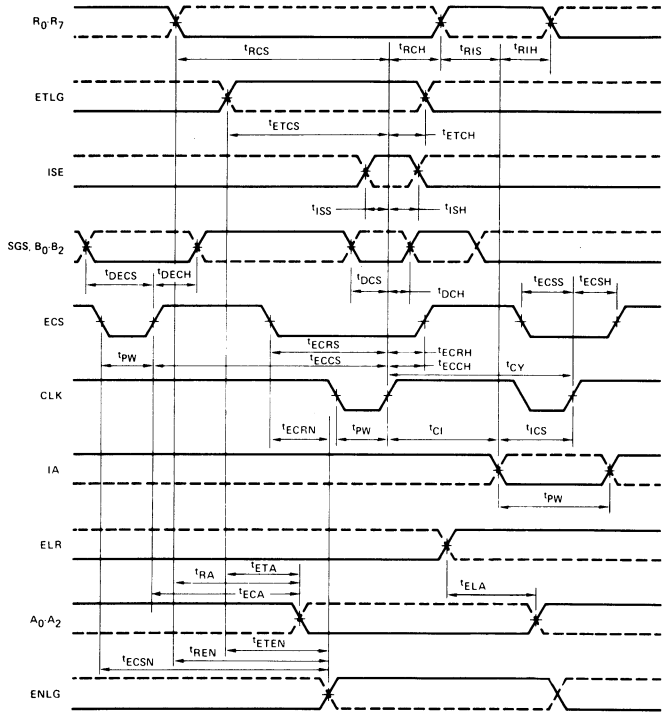
T_A = 25°C

SYMBOL	PARAMETER	MIN	TYP(1)	MAX	UNIT
C_{IN}	Input Capacitance		5	10	pf
C_{OUT}	Output Capacitance		7	12	pf

TEST CONDITIONS:
V_{BIAS} = 2.5V, V_{CC} = 5V, T_A = 25°C, f = 1 MHz

NOTE:
(5) This parameter is periodically sampled and not 100% tested.

WAVEFORMS



TYPICAL CONFIGURATIONS

The ICU has been designed for use with the INTEL Series 3000 Bipolar Microcomputer Set. It operates from the single common system clock and can accept an interrupt strobe (ISE) generated by the 3001 Micropgoram Control Unit or by a bit in microprogram memory as shown in Figures 2 and 3.

The ICU responds to interrupt requests of sufficient priority by entering the interrupt active mode. Its output (IA) can be tied to the row enable input (ERA) of the 3001 MCU. This gates an alternate row address onto the microprogram memory address bus which forces the system to execute an interrupt handling routine. Alternatively, the ICU output can be used to directly modify the MCU jump instruction (AC inputs) so that the next microprogram address corresponds to the start of the interrupt routine rather than the start of the macroinstruction fetch sequence. Of course, in the case of this particular implementation, the interrupt strobe must be generated one clock period earlier and the ISE output of the MCU should not be used.

As shown in Figure 4, when several ICUs are used together to provide a multiple of 8 priority levels, most control lines will be bussed. The Intel 3205 Decoder may be used to decode the high order bits of the request level, the information being derived from the daisy-chain group level signals.

As mentioned in the functional description, the request level information (A_0–A_2) may be sent to the 3001 MCU or the 3002 CP array as a constant through the Mask (K) bus or as data through the memory (M) or data (I) busses. Similarly, the status information can be generated by the CP array and carried to the ICU by the data (D) output bus of the CP array.

TYPICAL CONFIGURATIONS (CON'T)

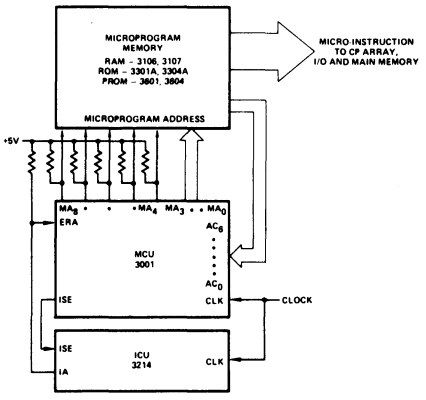


Figure 2. Interfacing 3214 with 3001.

Interrupt strobe generated by MCU.
Interrupt routine start address at column 15 row 31.
Macro-instruction fetch start address at column 15 row Ø.

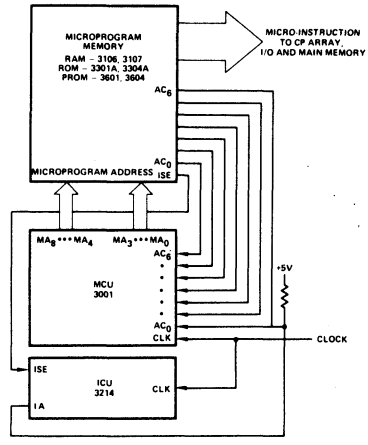


Figure 3. Interfacing 3214 with 3001.

Interrupt strobe generated by the microprogram memory.
Interrupt routine start address at column 14 row Ø.
Macro-instruction fetch start address at column 15 row Ø.

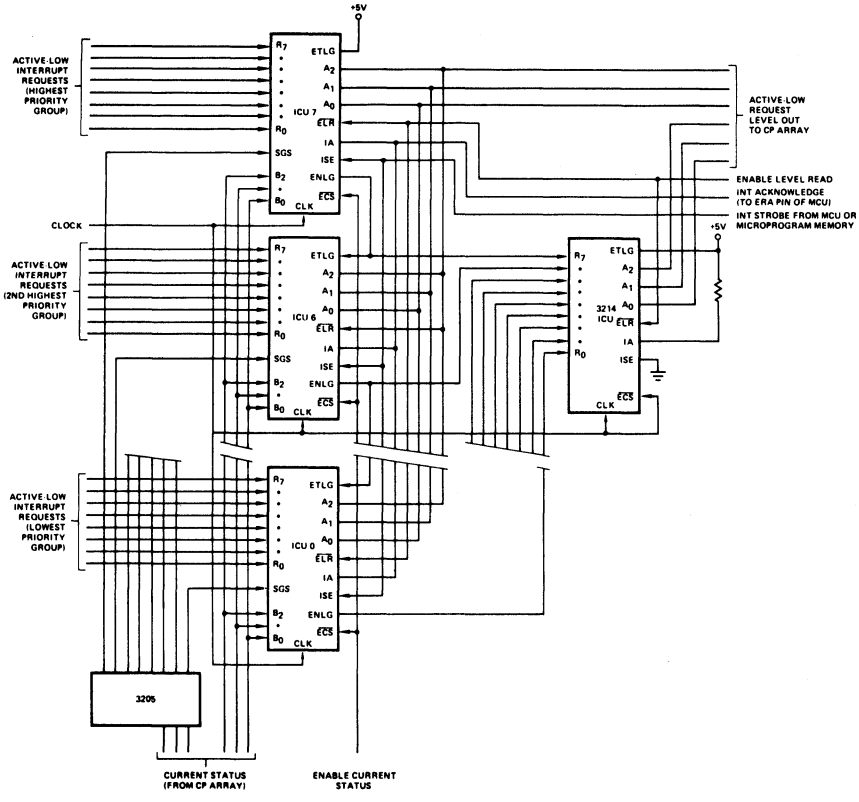


Figure 4. Using Several 3214 Interrupt Chips to Provide more than Eight Priority Levels. (The 3214 at the upper right is used to encode the high order bits of the requesting level)

SCHOTTKY BIPOLAR LSI MICROCOMPUTER SET

3216/3226 PARALLEL BIDIRECTIONAL BUS DRIVER

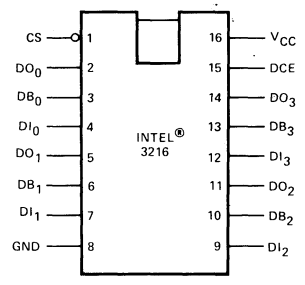
The INTEL® 3216 is a high-speed 4-bit Parallel, Bidirectional Bus Driver. Its three-state outputs enable it to isolate and drive external bus structures associated with Series 3000 systems

The INTEL 3226 is a high-speed 4-bit Parallel, Inverting Bidirectional Bus Driver. Its three-state outputs enable it to isolate and drive external bus structures associated with Series 3000 systems.

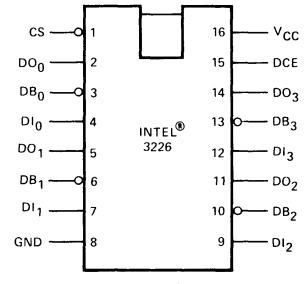
The 3216/3226 driver and receiver gates have three state outputs with PNP inputs. When the drivers or receivers are tri-stated the inputs are disabled, presenting a low current load, typically less than 40 μamps, to the system bus structure.

- **High Performance—25 ns typical propagation delay**
- **Low Input Load Current—0.25 mA maximum**
- **High Output Drive Capability for Driving System Data Busses**
- **Three-State Outputs**
- **TTL Compatible**
- **16-pin DIP**

PACKAGE CONFIGURATION

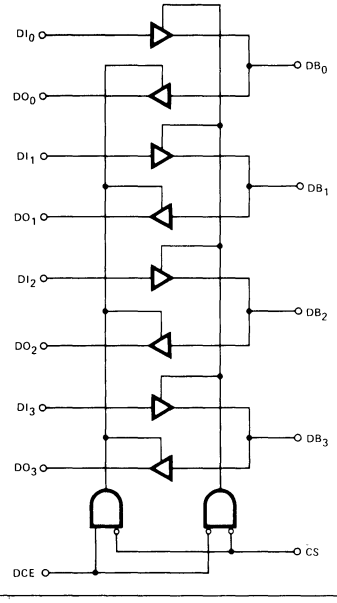


3216

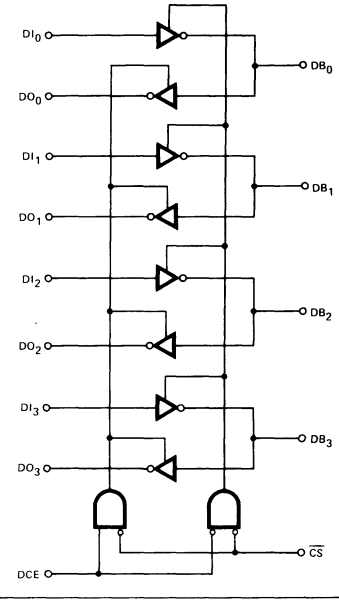


3226

LOGIC DIAGRAM 3216



LOGIC DIAGRAM 3226



D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias
- Ceramic . . . . . -65°C to +75°C
- Plastic . . . . . 0°C to +75°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Currents . . . . . 125 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

T_A = 0°C to +75°C, V_{CC} = +5.0V ±5%

Symbol	Parameter	Limit Min.	Typ.	Max.	Unit	Condition
I_F	Input Load Current					
	DCE, $\overline{CS}$ Inputs		-0.15	-0.5	mA	V_F = 0.45V
	All Other Inputs		-0.08	-0.25	mA	
I_R	Input Leakage Current					
	DCE, $\overline{CS}$ Inputs			80	μA	V_R = 5.25V
	DI Inputs			40	μA	
V_C	Input Clamp Voltage			-1	V	I_C = -5mA
V_{IL}	Input Low Voltage			0.95	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0			V	V_{CC} = 5.0V
V_{OL1}	Output Low Voltage		0.3	0.45	V	DO Outputs I_{OL}=15mA
	DO, DB Outputs					DB Outputs I_{OL}=25mA
V_{OL2}	Output Low Voltage		0.5	0.6	V	DB Outputs I_{OL} = 50mA
	DB Outputs Only					
V_{OH1}	Output High Voltage	3.65	4.0		V	I_{OH}=-1mA
	DO Outputs Only					
V_{OH2}	Output High Voltage	2.4	3.0		V	I_{OH}=-10mA
	DB Outputs Only					
I_{SC}	Output Short Circuit Current					
	DO Outputs	-15	-35	-65	mA	V_{CC} = 5.0V
	DB Outputs	-30	-75	-120	mA	
$\|I_O\|$	Output Leakage Current					
	High Impedance State					
	DO Outputs			20	μA	V_O=0.45V/5.25V
	DB Outputs			100	μA	
I_{CC}	Power Supply Current 3216		95	130	mA	
	3226		85	120	mA	

NOTE: Typical values are for T_A = 25°C

A.C. CHARACTERISTICS T_A = 0°C to +75°C, V_{CC} = +5.0V ±5%

Symbol	Parameter		Limit Min.	Typ.	Max.	Unit	Condition
T_{PD1}	Input to Output Delay	3216		15	25	ns	C_L=30pF, R_1=300Ω,
	DO Outputs	3226		14	25		R_2=600Ω
T_{PD2}	Input to Output Delay	3216		19	30	ns	C_L=300pF, R_1=90Ω,
	DB Outputs	3226		16	25		R_2=180Ω
T_E	Output Enable Time	3216		42	65	ns(2)	DO Outputs: C_L=30pF,
	DCE, CS	3226		36	54		R_1=300Ω/10KΩ,
							R_2=600Ω/1KΩ
							DB Outputs: C_L=300pF,
							R_1=90Ω/10KΩ,
							R_2=180Ω/1KΩ
T_D	Output Disable Time			16	35	ns(2)	DO Outputs: C_L=5pF,
	DCE, CS						R_1=300Ω/10KΩ,
							R_2=600Ω/1KΩ
							DB Outputs: C_L=5pF,
							R_1=90Ω/10KΩ,
							R_2=180Ω/1KΩ

NOTE: (1) Typical values are for T_A = 25°C and nominal supply voltage.

(2) The test load circuit is set for worst case source and sink loading on the outputs. The two resistor values for R1 and R2 correspond to worst case sink and source loading, respectively.

CAPACITANCE(2) T_A = 25°C

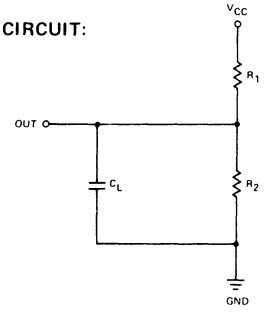
Symbol	Parameter	Limit Min.	Typ.	Max.	Unit
C_{IN}	Input Capacitance		4	6	pF
C_{OUT}	Output Capacitance				
	DO Outputs		6	10	pF
	DB Outputs		13	18	pF

Note:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1MHz, V_{BIAS} = 2.5V, V_{CC} = 5.0V and T_A = 25°C.

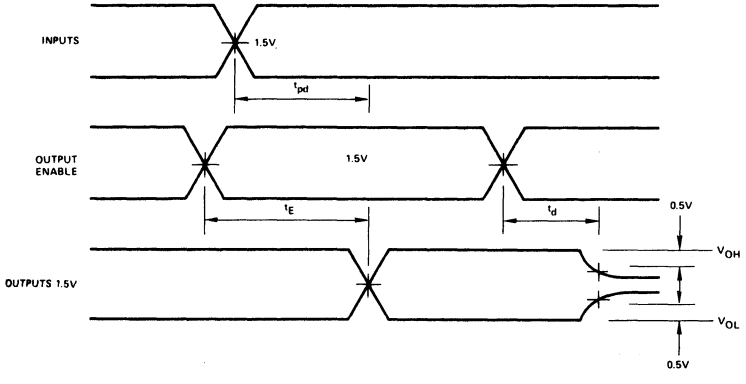
TEST CONDITIONS:

Input pulse amplitude of 2.5V.
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 5 mA and 10 pF.
Speed measurements are made at 1.5 volt levels.

TEST LOAD CIRCUIT:



WAVEFORMS



D.C. AND OPERATING CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias
Ceramic . . . . . -65°C to +75°C

Storage Temperature . . . . . -65°C to +160°C

All Output and Supply Voltages . . . . . -0.5V to +7V

All Input Voltages . . . . . -1.0V to +5.5V

Output Currents . . . . . 125 mA

*COMMENT: Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

T_A = -55°C to +125°C, V_{CC} = +5.0V ±10%

Symbol	Parameter	Min.	Limit Typ.	Max.	Unit	Condition
I_F	Input Load Current					
	DCE, $\overline{CS}$ Inputs		-0.15	-0.5	mA	V_F = 0.45V
	All Other Inputs		-0.08	-0.25	mA	
I_R	Input Leakage Current					
	DCE, $\overline{CS}$ Inputs			80	μA	V_R = 5.5V
	DI Inputs			40	μA	
V_C	Input Clamp Voltage			-1.2	V	I_C = -5mA
V_{IL}	Input Low Voltage M3216			0.95	V	V_{CC} = 5.0V
	M3226			0.90	V	
V_{IH}	Input High Voltage	2.0			V	V_{CC} = 5.0V
V_{OL1}	Output Low Voltage		0.3	0.45	V	DO Outputs I_{OL}=15mA
	DO, DB Outputs					DB Outputs I_{OL}=25mA
V_{OL2}	Output Low Voltage		0.5	0.6	V	DB Outputs I_{OL} = 45mA
	DB Outputs Only					
V_{OH1}	Output High Voltage	3.4	3.8		V	I_{OH}=-0.5mA
	DO Outputs Only					I_{OH}=-2.0mA
V_{OH2}	Output High Voltage	2.4	3.0		V	I_{OH}=-5mA
	DB Outputs Only					
I_{SC}	Output Short Circuit Current					
	DO Outputs	-15	-35	-65	mA	V_{CC} = 5.0V
	DB Outputs	-30	-75	-120	mA	
$\lvert I_O \rvert$	Output Leakage Current					
	High Impedance State					
	DO Outputs			20	μA	V_O=0.45V/5.5V
	DB Outputs			100	μA	
I_{CC}	Power Supply Current M3216		95	130	mA	
	M3226		85	120	mA	

NOTE: Typical values are for T_A = 25°C

A.C. CHARACTERISTICS T_A = -55°C to +125°C, V_{CC} = 5.0V ±10%

Symbol	Parameter		Limit Min.	Typ.	Max.	Unit	Condition
T_{PD1}	Input to Output Delay DO Outputs			15	25	ns	C_L=30pF, R_1=300Ω, R_2=600Ω
T_{PD2}	Input to Output Delay DB Outputs	M3216		19	33	ns	C_L=300pF, R_1=90Ω, R_2=180Ω
		M3226		16	25		
T_E	Output Enable Time	M3216		42	75	ns(2)	DO Outputs: C_L=30pF, R_1=300Ω/10KΩ, R_2=600Ω/1KΩ
		M3226		36	62		DB Outputs: C_L=300pF, R_1=90Ω/10KΩ, R_2=180Ω/1KΩ
T_D	Output Disable Time	M3216		16	40	ns(2)	DO Outputs: C_L=5pF, R_1=300Ω/10KΩ, R_2=600Ω/1KΩ
		M3226		16	38		DB Outputs: C_L=5pF, R_1=90Ω/10KΩ, R_2=180Ω/1KΩ

NOTE: (1) Typical values are for T_A = 25°C and nominal supply voltage.

(2) The test load circuit is set for worst case source and sink loading on the outputs. The two resistor values for R1 and R2 correspond to worst case sink and source loading, respectively.

CAPACITANCE(2) T_A = 25°C

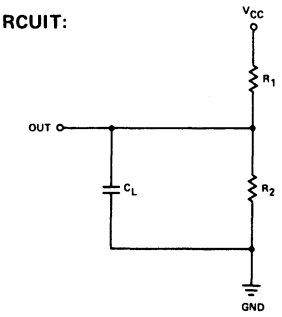
Symbol	Parameter	Limit Min.	Typ.	Max.	Unit
C_{IN}	Input Capacitance		4	6	pF
C_{OUT}	Output Capacitance				
	DO Outputs		6	10	pF
	DB Outputs		13	18	pF

Note:
(2) This parameter is periodically sampled and is not 100% tested. Condition of measurement is f = 1MHz, V_{BIAS} = 2.5V, V_{CC} = 5.0V and T_A = 25°C.

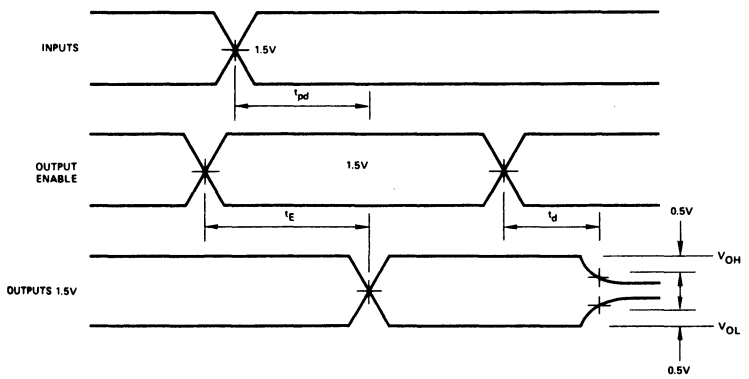
TEST CONDITIONS:

Input pulse amplitude of 2.5V.
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 5 mA and 10 pF.
Speed measurements are made at 1.5 volt levels.

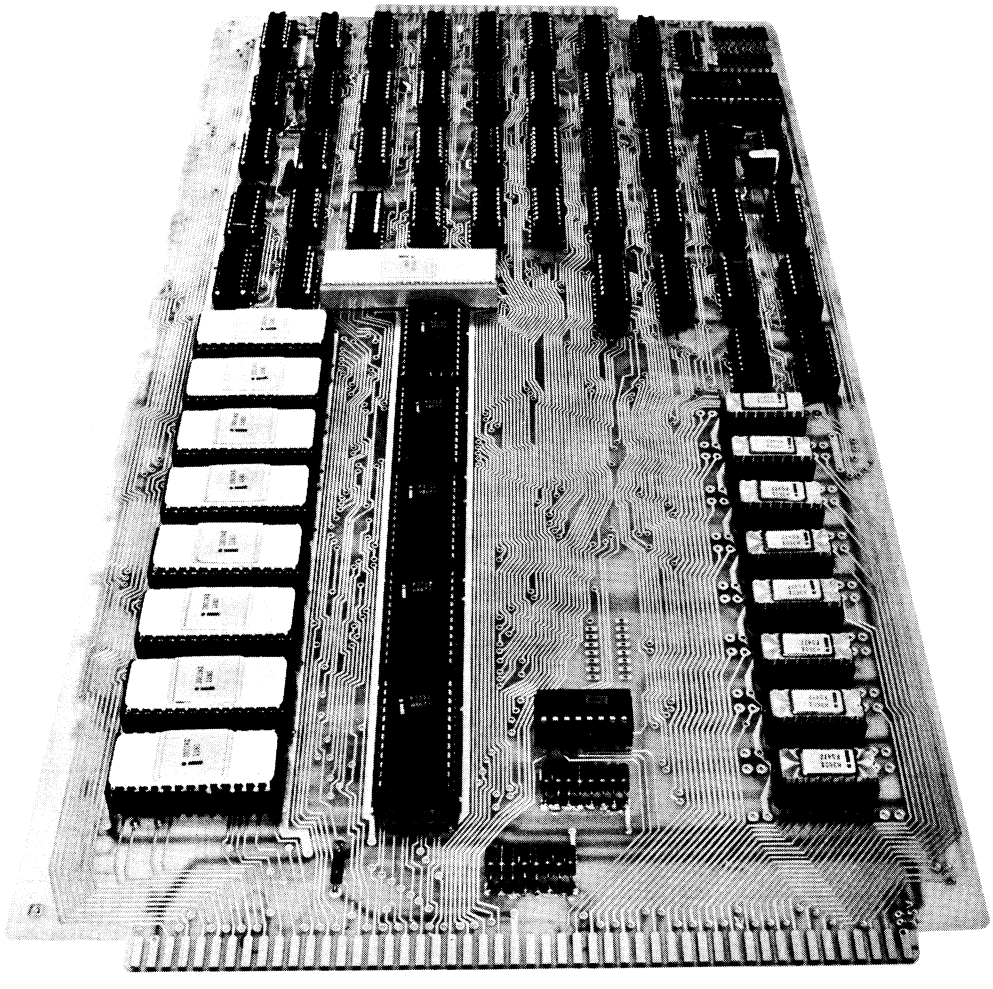
TEST LOAD CIRCUIT:



WAVEFORMS



APPLICATIONS



Series 3000 System Timing Considerations

by Gary Fielland

While the timing for each component in Intel's 3000 Series Schottky Bipolar Microcomputer Set is clearly specified, the composite system timing must be derived. This system timing is highly dependent on the particular configuration implemented, and hence, must be carefully considered for each implementation.

Though Intel cannot generate the system timing for every possible configuration, an effort has been made to study a few simple variations. By examining these examples and taking note of considerations given, it should be easier for the system designer to realize those times which are critical, and to generate the appropriate timing for his particular system.

The designer must consider many different factors in determining this "proper" system timing. Several simplifications are made to facilitate this discussion. Intel commercial grade parts are specified over a wide temperature range (0°C — 70°C) and so variations in timing due to temperature will not be considered, except for a short note at the end.

Whenever a signal must traverse a conductor between two points, there is a finite delay introduced into the signal path that is not accounted for by any data sheet. This is the delay due to such factors as the length of the conductor, its transmission properties, and the characteristics of the driver and receiver. When a TTL totempole output drives a TTL input a short distance away this delay is usually negligible compared to other delays in the signal path. However, if there are many loads (increasing the capacitance), or the driver is of the open-collector type (limiting the drive), or if the receiver is physically far removed, the designer should consider and allow for any possible deleterious effects of this delay. For this discussion, except in one special case, the delay introduced by interconnection is not considered.

Aside from these simplifications, it should be realized that this note is not an extensive study of the timing of any particular system, but rather a compendium of typical considerations which a designer might examine.

Consider the basic "data sheet" 16-bit processor configuration as shown in Figure 1. It utilizes pipeline registers, full carry look-ahead, and a priority interrupt mechanism. To implement any such system the designer must be very careful to provide the proper timing for all components under all possible operating conditions. Such a system is highly complex and the analysis is best approached in a piecemeal fashion.

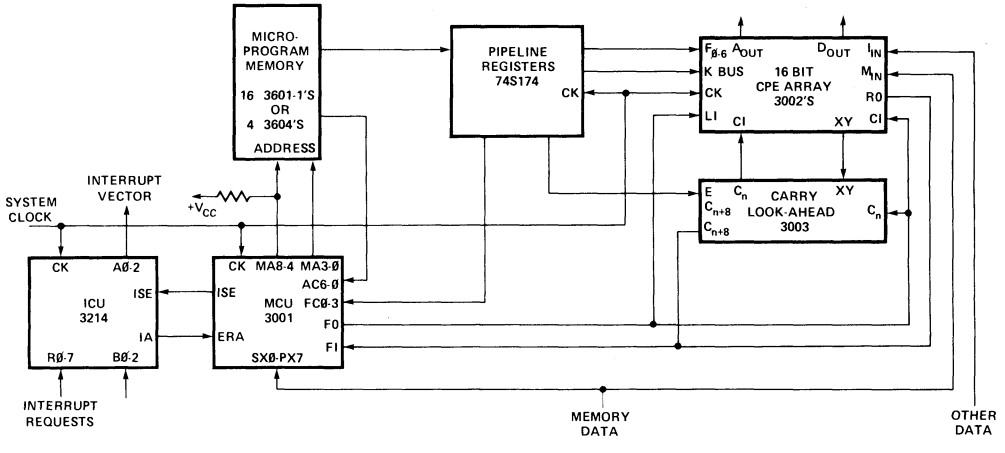


Figure 1. Basic 16-Bit Processor Configuration

ARITHMETIC DELAY PATHS

First an analysis will be made of the arithmetic paths and delays. Imagine cycles in which arithmetic is being done within the CPE array. The carrys must have time to propagate through the arithmetic portion and reach the MCU so that a conditional jump may be made based on that carry out bit. For the moment ignore other critical paths, and examine Figure 2 which illustrates these arithmetic cycles.

The cycle begins with the rising edge of the system clock as it clocks the pipeline registers. After the delay (t_{PLR}) introduced by the pipeline, the function is available at the CPE array. There is a delay (t_{XF}) while all the CPE's decode the function and generate their X and Y outputs for the operation. Once the X and Y outputs are stable, the Carry Look-Ahead circuit takes some time (t_{XC}) to simultaneously generate all the carry outputs, including the one which goes to the MCU flag input. Time must be provided to allow for the carry-input setup time of the CPE's (t_{SS}) and the MCU (t_{SI}). Finally, adding in enough time for the clock pulse, which acts as a write pulse for the CPE register array, the cycle time is determined. Note the time for the MCU flag output to stabilize (t_{KO}) was ignored as it is not a limiting specification for this configuration.

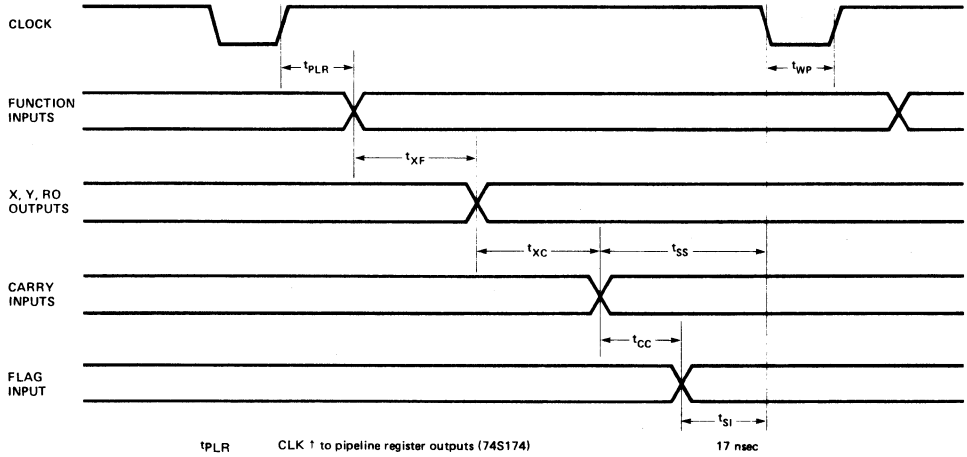
$$t_{CYCLE} = t_{PLR} + t_{XF} + t_{XC} + t_{SS} + t_{WP}$$

Keeping the same train of thought, consider individually the effects of variants from the configuration of Figure 1. If full carry look-ahead is not used and the carry is allowed to ripple through only the last slice, an additional delay path is introduced. After the 3003 has generated the carry outputs there is the CPE carry-in setup time (t_{SS}) which must be met as before. However, the carry-out of the last slice will not be available to the MCU flag input until it has rippled through (t_{CC}) that slice. Finally, the MCU flag input setup time (t_{SI}) must be satisfied.

$$t_{CYCLE} = t_{PLR} + t_{XF} + t_{XC} + t_{CC} + t_{SI} + t_{WP}$$

If the 3003 Look-Ahead Carry circuit is not used, there will be considerable delay added to the basic cycle due to ripple carry time. Once the CPE function-inputs are stable, the function must be decoded and the carry-out of the least significant slice generated (t_{CF}). The carry must ripple through six slices ($6* t_{CC}$) and meet the carry setup time (t_{SS}) of the most significant slice. However, it must also ripple through this last slice (t_{CC}) and meet the MCU flag input setup time which is a more severe restriction.

$$t_{CYCLE} = t_{PLR} + t_{CF} + (7* t_{CC}) + t_{SI} + t_{WP}$$



t_{PLR}	CLK ↑ to pipeline register outputs (74S174)		17 nsec
t_{XF}	Function inputs to X,Y,RO outputs		52
t_{XC}	Lookahead – X,Y inputs to carry outputs		20
t_{SS}	Data set-up time, LI & CI		27
t_{CC}	Ripple carry (CI to CO) delay		25
	NOTE: t_{CC} included only if carry ripples through last slice.		
t_{SI}	Flag input set-up time		15
t_{WP}	Clock pulse width		33
t_{CYCLE}	Full fast carry	$t_{PLR} + t_{XF} + t_{XC} + t_{SS} + t_{WP}$	149
	Last slice ripple	$t_{PLR} + t_{XF} + t_{XC} + t_{CC} + t_{SI} + t_{WP}$	162

If pipeline registers are not used; replace t_{PLR} with the sum of t_{CO} (CLK↑ to $MA_{\phi-8}$ outputs, 44 nsec) plus t_{ROM} (access time; 50 nsec for 3601-1, 70 nsec for 3604).

If 3003 fast carry is not used; replace t_{XF} with t_{CF} (function IN to CO output, 65 nsec); replace $t_{XC} + t_{CC}$ with $(N-1)*t_{CC}$, where "N" is the number of slices used.

Figure 2. Non-Interrupt 16-Bit Processor Cycle Timing

If pipeline registers are not used, there will be additional delay. It takes some time (t_{CO}) after the rising edge of the clock for the next address to propagate through the MCU address register and buffers. Then, when this address is stable the ROMs must be accessed and there will be a delay (t_{ROM}, access time) before their output and hence the CPE function-input is stable. Thus, the cycle time for a non-pipelined system with carry look-ahead is:

$$t_{CYCLE} = t_{CO} + t_{ROM} + t_{XF} + t_{XC} + t_{SS} + t_{WP}$$

In the previous discussion it was assumed that the operands in the arithmetic operations were internal registers and the K-bus as implemented. If one of the operands is the M-bus or the I-bus, additional consideration should be given. This situation will typically arise at the completion of a Memory-Read or Input cycle. Typically, these cycles are implemented such that the processor clock stops in its high state to wait for the data to be available, while the processor is in the midst of executing an LMM or similar instruction. Thus, it is often the case that the pipeline registers have long since been accessed and the function decoded.

Then, when the data becomes available a clock pulse is issued and normal operation continues. It is the time from the point the data becomes available until the clock pulse is issued (Data Input Setup Time) that is of concern here.

Consider first a special case. Namely, the data is input via an LTM instruction and no test will be made on the carry-output. This implies that for this specific instruction, carry propagation is unimportant and it is acceptable to have an erroneous carry-output. For such a case, it is sufficient to only allow for the CPE data setup time (t_{DS}).

$$t_{SETUP} = t_{DS}$$

For the more general case where arithmetic is done on input and the carry-output may be tested, the above analysis is incomplete. While the above condition must be met, it is no longer the determining factor. Time must be allowed for carry propagation. See Figure 3, which illustrates this case.

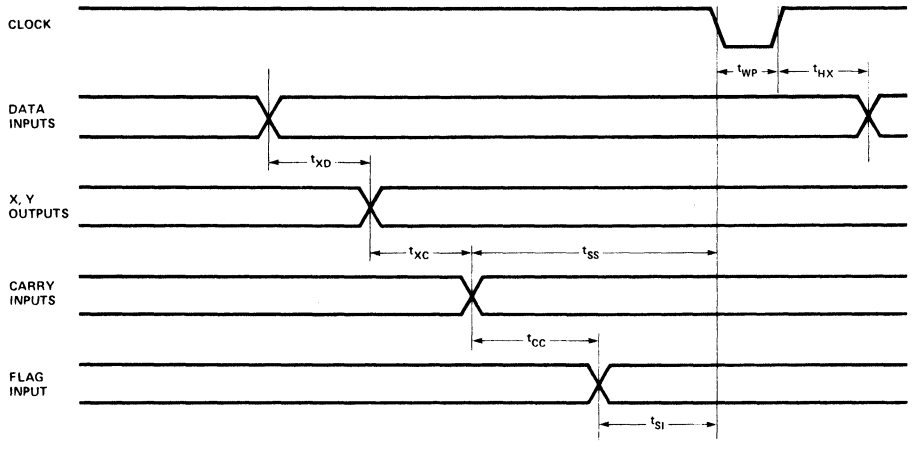
From the point in time when the data becomes stable at the CPE inputs, there is a delay (t_{XD}) while the CPE generates the X and Y outputs. If Ripple Carry is employed, the delay (t_{CD}) is in waiting for the carry-output of the least significant slice. After either of these delays the rest of the setup time is allocated analogously to that depicted in Figure 2 and discussed previously in relation to arithmetic cycle times.

t_{SETUP} (Basic) = $t_{XD} + t_{XC} + t_{SS}$

t_{SETUP} (Last Slice Ripple) = $t_{XD} + t_{XC} + t_{CC} + t_{SI}$

t_{SETUP} (Ripple Carry) = $t_{CD} + (7 * t_{CC}) + t_{SI}$

t_{SETUP} (No Pipeline) — Same as Basic



t_{XD}	Data inputs to X,Y outputs		42 nsec
t_{HX}	SX,PX input hold time		20
t_{SET-UP}	Full fast carry	$t_{XD} + t_{XC} + t_{SS}$	89
	Last slice ripple	$t_{XD} + t_{XC} + t_{CC} + t_{SI}$	102

If 3003 fast carry is not used; replace t_{XD} with t_{CD} (data input to CO output, 55 nsec); replace $t_{XC} + t_{CC}$ with (N-1)*t_{CC} then t_{SET-UP} (ripple carry). 245

NOTE: This diagram is usually of concern only in relation to memory-read, or input cycles.

Figure 3. 16-Bit Processor Data Input Set-Up Times

CONTROL DELAY PATHS

After carefully examining the arithmetic paths and delays it is appropriate to push all of this information onto your "mental stack" and begin again with a consideration of the control paths and delays. After this study the stack can be popped and information merged to yield overall system requirements.

Consider the MCU as it cycles in normal operation (see Figure 4). At the rising edge of the clock the new microprogram address is loaded into its holding register and through the output buffers. Thus, the new address reaches the ROM after a delay (t_{CO}). Then there is a wait (t_{ROM}) while the ROMs are accessed before the outputs are valid. At this time the MCU address control inputs (which are never pipelined) are valid and this must be early enough in the cycle to satisfy the MCU address control input setup time (t_{SF}). Adding the time for the clock pulse (t_{WP}) yields the cycle time requirement. Note this paragraph has ignored the generation of the ISE output.

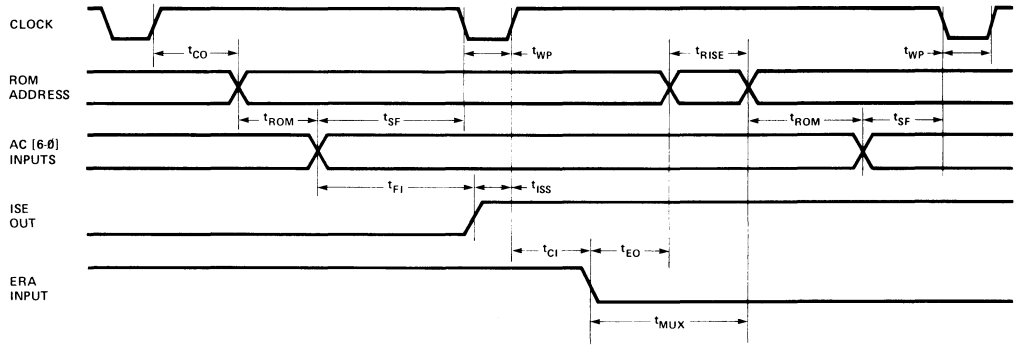
$$t_{CYCLE} = t_{CO} + t_{ROM} + t_{SF} + t_{WP}$$

In the basic configuration shown in Figure 1 the ISE output is used to strobe the 3214 Interrupt Control Unit each time a JZR 15 (usually a jump to macroinstruction fetch) is recognized at the MCU address control inputs. Some consideration must be given to the additional requirements on timing imposed by the use of this ISE output. After the ROM has been accessed and the MCU address control inputs are valid, it takes the MCU some time (t_{FI}) to decode the JZR 15 operation and raise the ISE output. This output is used as the 3214 ISE input and must be valid early enough to meet that input setup time (t_{ISS}). As this setup time is relative to the rising edge of the clock, the clock pulse width need not be added in.

$$t_{CYCLE} = t_{CO} + t_{ROM} + t_{FI} + t_{ISS}$$

Recalling the basic configuration depicted in Figure 1 and the situation described in the last paragraph, imagine that an interrupt request had been active long enough to meet the request setup time (t_{RCS}) of the ICU. Then since the ISE input went high and satisfied the input setup time, the Interrupt Acknowledge flip-flop within the 3214 will change state and lower the MCUs ERA input after a delay (t_{CI}). After the row address outputs are disabled (t_{EO}), the pull-up resistors will begin to pull these lines high and after the voltage on these lines rises to 2.0V (t_{RISE}) the ROM address will be valid. The remainder of this cycle is the same as previously described and usually will not be required to again generate an ISE pulse.

$$t_{CYCLE} = t_{CI} + t_{EO} + t_{RISE} + t_{ROM} + t_{SF} + t_{WP}$$



t_{CO}	CLK↑ to MA[8~ϕ] outputs	44 nsec
t_{ROM}	ROM access time (70 nsec for 3604) 3601-1	50
t_{SF}	AC[6~ϕ] input set-up time	10
t_{FI}	AC[6~ϕ] input to ISE output	40
t_{ISS}	ISE input (3214) set-up	16
t_{CI}	CLK↑ to IA output (3214)	25
t_{EO}	ERA input to MA[8~4] output	32
t_{RISE}	RISE time to 2.0 V with 1 KΩ pull-ups: (16*3601-1)	84
	(4*3604)	21
t_{MUX}	Multiplexer switch time (74S158)	12
t_{CYCLE}	Ignoring ISE output $t_{CO} + t_{ROM} + t_{SF} + t_{WP}$ (3601-1)	137
	(3604)	157
	Using ISE output $t_{CO} + t_{ROM} + t_{FI} + t_{ISS}$ (3601-1)	150
	Interrupt using pull-ups $t_{CI} + t_{EO} + t_{RISE} + t_{ROM} + t_{SF} + t_{WP}$ (3601-1)	234
	Interrupt using MUX* $t_{CI} + t_{MUX} + t_{ROM} + t_{SF} + t_{WP}$ (3601-1)	130

(*MUX adds $t_{MUX-PROP}$ [6 nsec] to t_{CO})

Figure 4. MCU & Interrupt Cycle Timing

Examining the times shown on Figure 4 for this case of an interrupt cycle using pull-up resistors, it is clear that unless something is done this will be the limiting cycle time requirement. There are several techniques which may be used to ease this requirement.

Since interrupt cycles are relatively infrequent in comparison with other cycles, one solution might be to extend just that cycle. In other words, the system cycle time would be determined by all considerations previously mentioned, but ignoring the abnormal interrupt cycle requirement. Then the clock circuit would be designed such that it could extend a cycle in response to a signal from the 3214 Interrupt Control Unit (see Figure 5).

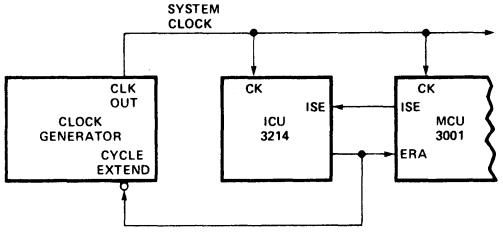


Figure 5. Interrupt Cycle Extension

The interrupt cycle would still be exactly as depicted in Figure 4, but the length of the interrupt cycle would be longer than a normal cycle, and in fact long enough to accommodate the interrupt cycle requirement.

It can be seen that a significant portion of the interrupt cycle is lost waiting for the pull-up resistors to charge the capacitance on the address lines. Thus, another method of easing the interrupt cycle requirement would be to reduce the address line rise time (t_{RISE}). Reducing the resistance of the pull-ups would help but this technique is limited by the available MCU address output fanout. Alternatively, the MCU row address outputs (MA8-4) could be connected to the ROM address lines through a multiplexer such as the 74S158 (see Figure 6). With such a connection the interrupt cycle time is reduced since the MCU enable time (t_{EO}) plus the address line rise time (t_{RISE}) may be replaced with simply the multiplexer select time (t_{MUX}) as shown in Figure 4. However, it should be noted that such a connection adds delay to the MCU address outputs, thus effectively lengthening this existing delay (t_{CO}) by the multiplexer propagate time ($t_{MUX\text{-}PROP}$) and hence lengthening any cycle which was dependent on the MCU delay (t_{CO}).

t_{CYCLE} (Interrupt with MUX) =

$$t_{CI} + t_{MUX} + t_{ROM} + t_{SF} + t_{WP}$$

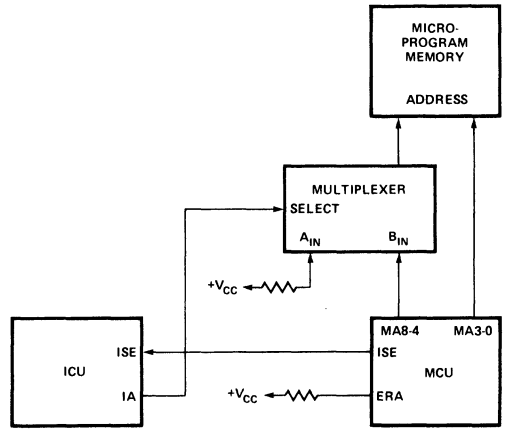


Figure 6. Multiplexer to Reduce Address Rise Time

A third alternative to solve the long interrupt cycle requirement is to implement the interrupts in quite a different way. Rather than changing the MCU address outputs, the MCU address control input least significant bit (ACϕ) may be altered (see Figure 7). Using this technique an extra ROM bit (Interrupt Strobe) is required to strobe the 3214 ICU since the MCUs ISE output occurs one cycle too late. Implementing the same mechanism (interrupt strobe on JZR 15) could be done by using the interrupt strobe bit to strobe the ICU (see Figure 7) the cycle before the JZR 15 code appears. An added benefit of this method is that the interrupt structure may be strobed at points other than the beginning of an instruction fetch cycle, facilitating PAUSE or WAIT instructions.

Examining the timing diagrams in Figure 7, it can be seen that this implementation of interrupts does not limit the system cycle time. Rather, this interrupt mechanism's timing is less restrictive than timing for a normal cycle. The only requirements are that the interrupt strobe bit from the ROM reaches the 3214 ICU ISE-input within its setup time (t_{ISS}). In the next cycle it is only necessary that the IA-output has gone low (t_{CI}) early enough to meet the MCU address control input setup time (t_{SF}). Thus, for the price of one bit of ROM interrupts can be implemented with no penalty in time.

At this point both major delay paths (arithmetic and control) have been examined for the implementation in question. After the designer has assured himself

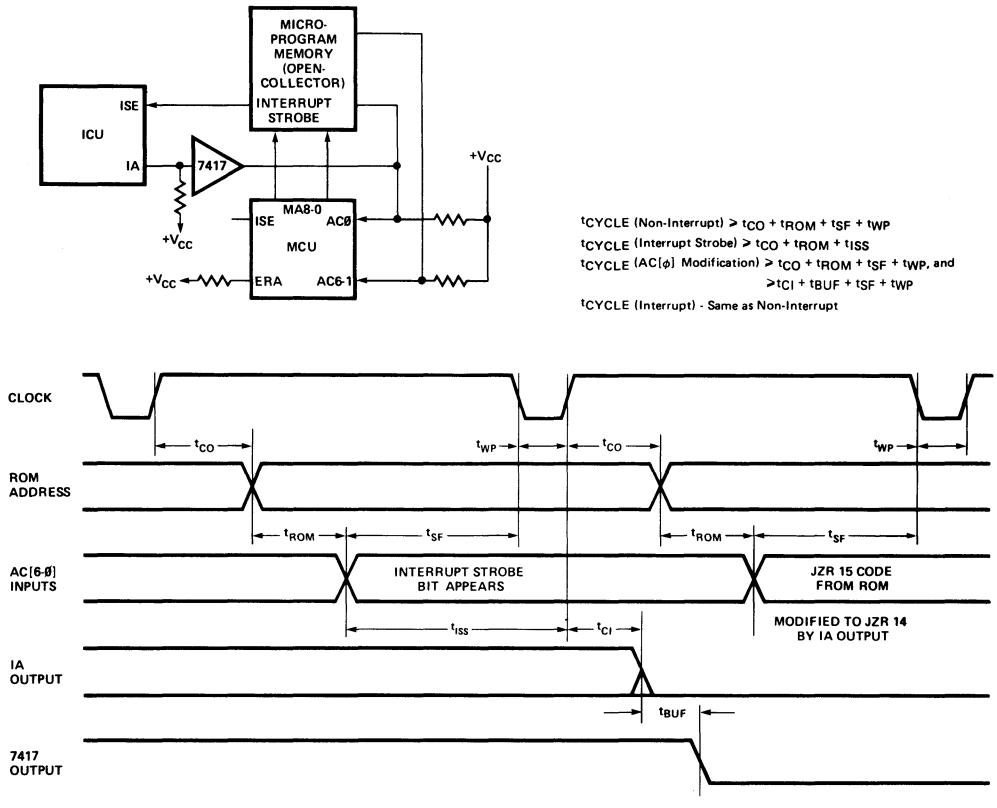


Figure 7. Interrupt Using AC [ϕ] Modification

there are no other delays which he may have overlooked, such as introducing external circuitry into the paths, he may merge the various requirements generated into a uniform set of system requirements. Any change introduced after these requirements have been generated must be closely examined that it does not subtly alter any system requirements. Delays that are negligible in one configuration may be dominant in a slightly different structure.

WHAT HASN'T BEEN MENTIONED?

1. In the introduction it was explained that temperature would not be considered in the examples since Intel specifies products over the 0°C to 70°C temperature range. This deserves further comment. A quick glance at an Intel Data Sheet will verify that Intel parts are specified and guaranteed over the 0°C to 70°C ambient temperature range and concurrently with a five percent tolerance power supply. This is a reasonable range and allows the designer to guarantee circuit operation.

   Unfortunately, the standard Schottky MSI line (74SXX) is only specified at 25°C ambient and V_{CC} = 5V. The variance of parameters over the allowable temperature and supply voltage is unspecified and left to the designer's experience. Due to this uncertainty the designer should "appropriately" modify any times attributable to non-Intel parts to allow for variations over temperature and supply voltage.

2. In the examples given it was always assumed that setup times would be honored. Though most of a computing system is synchronous, it typically has to interface with asynchronous events. It is at this interface that difficulty may be encountered. Consider a popular circuit (Figure 8) used to "synchronize" asynchronous signals.

   In this circuit the output delay is guaranteed only if the input setup time is met. But since the input is asynchronous, this setup time may not always be satisfied, as at the second event depicted in Figure 8. What happens? Though the results are highly dependent on the flip-flop circuit design,

some general observations may be made. Typically, the effect is to stretch out the flip-flop delay time as the event approaches arbitrarily close to the clock edge. Theoretically, the delay will go to infinity if the event falls precisely on the clock edge. Some flip-flops also exhibit a characteristic in which the output may change state and some time later return to the original state. This phenomena is known as "hang-up" and has been observed to last for twenty nanoseconds on a 74S74. It cannot be absolutely prevented when asynchronous signals are introduced into a synchronous system, but the probability of the "hung" flip-flop causing an error can be reduced without limit. The technique is simply to cascade these interfaces.

If two such flip-flops are cascaded there is some probability P_1 ($P_1 < 1$) that the asynchronous event will fall close enough to the clock edge to hang the first flip-flop. Given that it hangs, there is some probability P_2 that it will hang for very nearly an entire clock period and into the hang-up zone of the second flip-flop. Then, there is a probability P_3 that the second flip-flop will hang long enough to cause an error. Thus, the probability of error is $P_E = P_1 * P_2 * P_3$. Hence, by cascading flip-flops the probability of error can be reduced without limit.

Recall the 3214 Interrupt Control Unit and its request setup time (t_{RCS}) and its IA output delay (t_{CI}). This delay is in several critical system paths as shown by the examples. Of course, the IA output delay specified also presumes the IA flip-flop setup time was met. When deliberately violating the IA flip-flop setup time, a hang-up of 50 nsec has been observed. What is a designer to do?

Slowing down the system such that it could tolerate any expected hang-up would be the easiest solution. This may not always be as bad as it sounds. Recalling the situation depicted in Figure 7, note that some flip-flop hang-up is tolerable. [$t_{IA\text{-}HANG} = t_{CYCLE} - (t_{SF} + t_{WP})$]. An alternative would be to "synchronize" the asynchronous interrupt requests using the technique previously described. An octal D flip-flop such as the 74S374 would be suitable.

3. In the examples given it has been assumed that the system, including all the CPEs, the MCU, and the ICU, operates from a single clock. If a circuit, such as in Figure 9, that provides a separate clock for different components is used, the possible clock skew must also be considered when determining system timing.

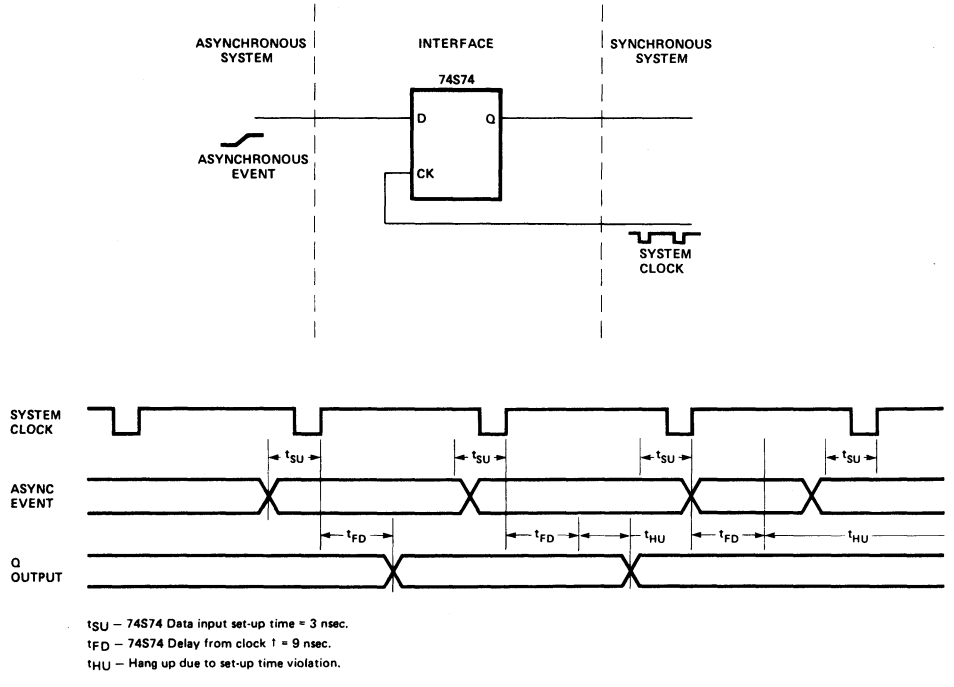


Figure 8. Synchronizing Circuit Exhibiting Hang-Up

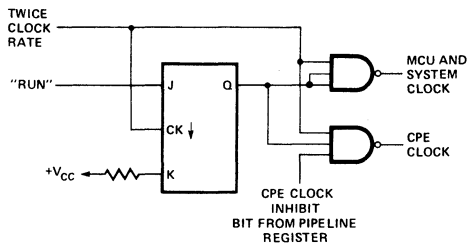


Figure 9. CPE Clock Inhibit Circuit

4. Though not explicitly mentioned, it has been assumed that all input hold times would be observed. Usually these times are satisfied with no conscious effort required of the system designer. However, parameters such as the MCU SX and PX input hold time must be carefully considered. These inputs are used for macro instruction decoding and typically are used at the end of an instruction fetch cycle. When using these inputs the designer must provide the necessary data hold time before allowing the data to change.

SUMMARY

Generating the correct timing for a complex system in which parameters may vary with temperature, power supply voltage, lead length and the like is no trivial task. Fortunately, large scale integration such as Intel's 3000 family is making the task much easier. With the 3000 family of compatible parts the designer need only worry about the interfaces and may be assured the internal timing is correct. Such a system is best analyzed by separately considering the various delay paths and later combining the sundry results. And of course, with Murphy on vacation the designer can be confident of a flawless design on the first pass.

Disk Controller Designed With Series 3000 Computing Elements

by Glenn Louie

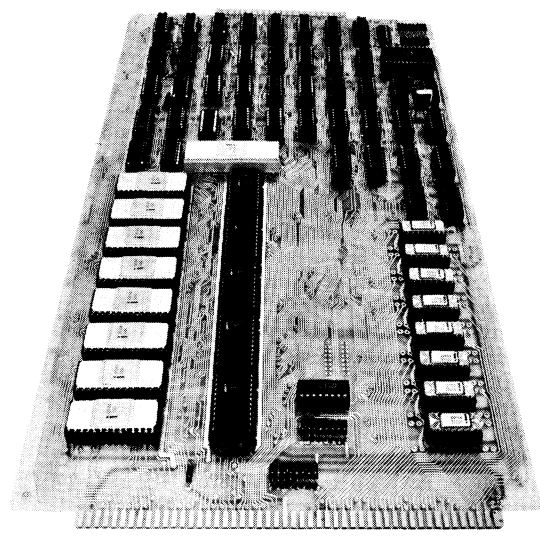


Figure 1. Bipolar Microprogrammed Disk Controller

With the introduction of the first microprocessor, digital designers began a massive switch to programmable LSI technology, away from hardwired random logic. Designers found that with these new LSI components and the availability of low cost ROMs they could easily implement structured designs which were both cost effective and flexible. However, not all digital designs were amenable to the microcomputer approach. One of the basic limitations was the speed at which a particular critical program sequence could be executed by a microprocessor. The early P-channel MOS microprocessors, such as Intel's 4004 and 8008, were able to solve a broad class of logic problems where speed was not essential. With the introduction of the more powerful n-channel MOS microprocessors, such as the Intel® 8080, the range of applications was significantly broadened, but there still existed a class of applications that even these newer devices were not fast enough to handle.

Recently, two new Schottky bipolar LSI computing elements, members of the Intel Bipolar Microcomputer Set, were introduced which expand the range of microcomputer applications to include high speed peripheral controllers and communication equipment. The new elements are the 3001 Microprogram Control Unit (MCU) and the 3002 Central Processing Element (CPE). These two components facilitate the design of specialized, high speed microprocessors that together with a minimum of external logic perform the intricate program sequences required by high speed peripheral controllers.

A multi-chip bipolar microprocessor differs from the single chip MOS microprocessor in that the bipolar microprocessor is programmed at the microinstruction level rather than at the macroinstruction level. This means that instead of specifying the action via a macroprogram using a fixed instruction set, a designer can specify the detailed action occurring inside the microprocessor hardware via a microprogram using his own customized microinstructions.

In general, microinstructions are wider than macroinstructions (e.g. 24 to 32 bits) and have a number of independent fields that specify simultaneous operations. In a single microcycle, an arithmetic operation can be executed while a constant is stored into external logic and a conditional jump is being performed.

A bipolar LSI microprocessor design is similar to a general MSI/SSI microprocessor design where the intricacies of the application are imbedded in the program patterns in ROM. However, the large amount of logic necessary to access the microcode has been replaced by the LSI MCU chip. Also,

the MSI logic required to provide the arithmetic and register capabilities has been replaced by the functionally denser LSI CPE slices. Because of these new LSI chips, microprogramming with all its advantages can now be applied to designs which previously were unable to justify microprogramming overhead.

The effectiveness of these new LSI components in a high speed peripheral controller design has been demonstrated by the Applications Research group at Intel with the design of a 2310/5440 moving head disk controller (BMDC). The BMDC has a total of 67 IC chips and is packaged on a printed circuit board measuring 8" x 15", as shown in Figure 1. Disk controllers of equivalent complexity realized with conventional components typically require between 150 and 250 I.C.'s. The BMDC performs all the operations required to interface up to four "daisy chained" moving head disk drives, with a combined storage capacity of 400 megabits, to a typical minicomputer. It is fast enough to keep up with the drive's 2.5 MHz bit serial data stream while performing the requisite data channel functions of incrementing an address register, decrementing a word count register, and terminating upon completion of a block transfer.

The BMDC interacts with the minicomputer's disk operating system (DOS) via I/O commands, interrupts and direct memory access (DMA) cycles. The I/O commands recognized by the BMDC's microprogram are:

Conditions In
Seek Cylinder
Write Data
Read Data
Verify Data
Format Data

The BMDC sends an interrupt to the minicomputer when either a command is successfully executed, a command is aborted, or a drive has finished seeking. The DOS then interrogates the BMDC with a Conditions In command. The following flags specify the conditions which the BMDC can detect:

Done flag
Malfunction flag
Not Ready flag
Change In Seek Status flag
Program Error flag
Address Error flag
Data Error flag
Data Overrun flag

Data transfers between the minicomputer and the disk BMDC occur during DMA cycles. DMA cycles are also used for passing command information from the minicomputer to the BMDC.

The bipolar LSI microcomputer in the BMDC performs the necessary command decoding, address checking, sector counting, overlap seeking, direct memory accessing, write protection, password protection, overrun detection, drive and read selection, and formatting. External hardware assists the microprocessor in updating the sector counter, performing parallel-to-serial and serial-to-parallel conversion, and generating the CRC data checking information. The BMDC uses a special purpose microprocessor, configured with the components listed in Table A. The LSI microprocessor uses an MCU, an 8:1 multiplexer, eight 3601 PROMs, a command latch, a data buffer, and an array of eight CPE slices (Fig. 2). The characteristics of this design, only one of many possible with the 3000 family, are as follows:

- 400 nsec system clock
- 16-bit wide CP array
- Ripple carry CPE configuration
- Non-pipelined architecture
- One level subroutining
- 230 32-bit microinstructions
- Word to 4-bit nibble serialization

The MCU controls the sequence in which microinstructions are executed. It has a set of unconditional and conditional jump instructions which is based on a 2-dimensional array for the microprogram address spece called the MCU Jump Map. [1]

PART #	DESCRIPTION	QUANTITY
3001	MCU	1
3002	CPE	8
3212	8 bit I/O Port	6
3205	1 of 8 Decoder	2
3601	1K PROM	8
3404	6 bit Latch	1
74173	4 bit Gated D F/F	1
74174	6 bit D F/F	1
74175	4 bit D F/F	1
74151	8:1 Multiplexer	1
8233	Dual 4:1 Multiplexer	2
9300	4 bit Shift Register	1
9316	4 bit Binary Counter	1
8503	CRC Generator	1
7474	Dual D F/F	5
7473	Dual J-K F/F	2
7451	And-Or-Invert Gate	1
7404	Hex Inverter	6
7400	Quad 2 Input Nand Gate	9
74H08	Quad 2 Input And Gate	1
7403	Quad 2 Input Nand O.C. Gate	2
7438	Quad O.C. Drivers	4
74H103	Dual J-K F/F	2
	Total	67 I.C. Packages

Table A. I.C. Component List for Disk Controller

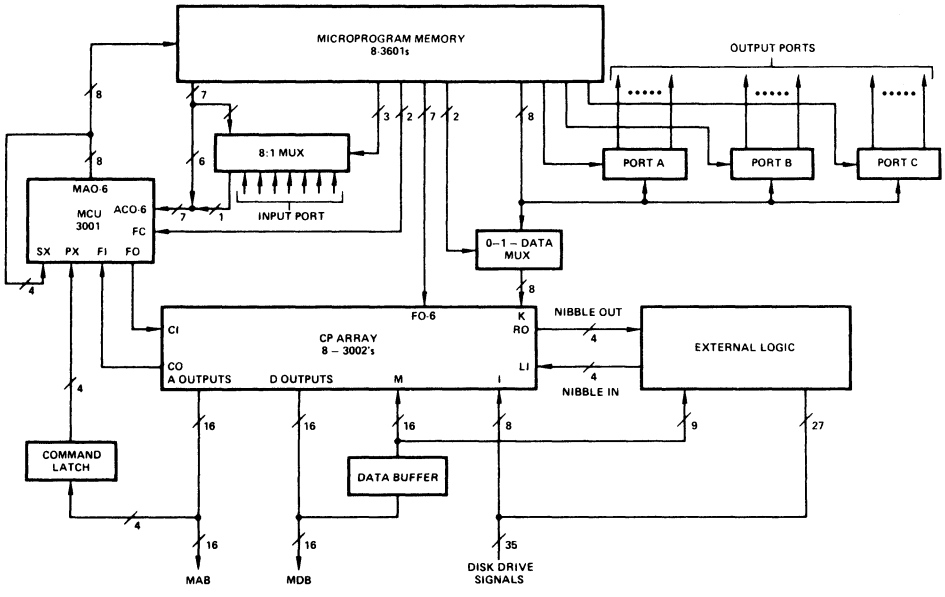


Figure 2. Disk Controller – The various elements of a specialized microprogrammed processor is shown with the external logic which together is the entire disk controller.

In addition, the MCU is connected in such a manner as to perform command decoding, external input testing, and one level subroutining.

Command decoding is achieved by connecting the command latch to the Primary Instruction (PX) bus inputs and using the JPX instruction (Fig. 3). The testing of external input signals is performed by routing the least significant bit (LSB) of the seven bit jump code through an eight-to-one multiplexer (Fig. 2). The multiplexer is controlled by a 3-bit Input Select Code which selects either the LSB of the jump code or one of 7 external input signals to be routed to the MCU. This technique has the effect of conditionally modifying an unconditional jump code so that the next address will either be an odd or even location (Fig. 3). A one instruction wait for external signal loop can be simply implemented in this fashion.

One level subroutining is achieved by feeding the four least significant bits of the address microprogram outputs back into the secondary instruction (SX) inputs. Enough program status information can then be saved in the internal PR latch when a subroutine is called with a JPX instruction so that upon exiting, a subroutine with a JPR instruction, control can be returned to the procedure which called it (Fig. 4). This technique saves a significant amount of microcode in the BMDC because some long sequences do not have to be repeated.

The microprogram control store is an array of eight 3601 PROMs organized to give 256 words x 32 bits (230 words were required for the BMDC). The 32-bit wide word is divided into the following subcontrol fields:

1. Jump Code field	7 bits
2. Flag Control field	2 bits
3. CPE Function field	7 bits
4. Input Select field	3 bits
5. Output Select field	3 bits
6. Mask or Data field	8 bits
7. Mask Control field	2 bits
TOTAL	32 bits

The command latch and data buffer retain command information from the computer so that the memory bus will not be held up if the BMDC should be busy performing an updating task. The data buffer also retains the next data word during a Write Data to disk operation.

The CP array is connected in a ripple carry configuration as shown in Figure 5. The eight CPE slices provide the BMDC with a 16-bit arithmetic, logic and register section. Word to nibble serialization is made possible by connecting the Shift Right Outputs (RO) of the first, third, fifth, and seventh CPE to the Nibble Out bus. By using only four shift right operations a word in a register can be converted into four 4-bit nibbles. The final serialization of these nibbles is done in the external logic. Similarly, the Shift Right Inputs (LI) of the second, fourth, sixth, and eighth CPE are connected to the Nibble In bus so that with only four shift right operations, a word can be assembled from four nibbles.

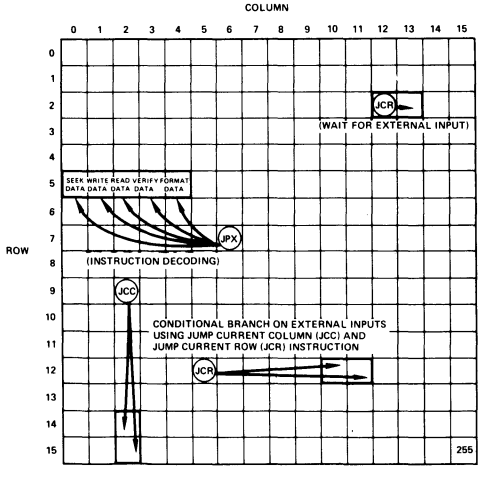


Figure 3. MCU Jump Map for instruction decoding and conditional branching on external inputs

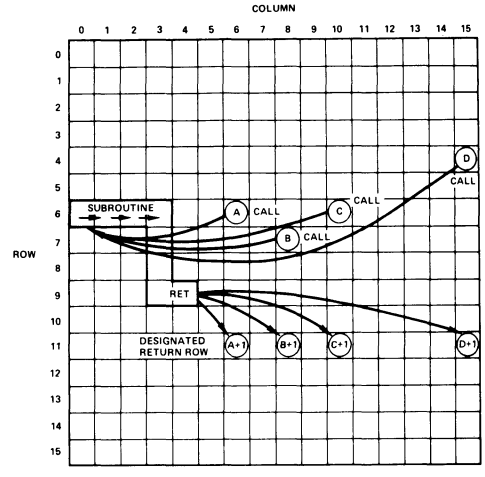


Figure 4. MCU Jump Map for one level subroutine call and return. A subroutine is called from four different places in the program each with a unique column number. Upon returning from the subroutine, control will be transferred back to the portion of program which called it. A subroutine may be called from a maximum of 16 different places.

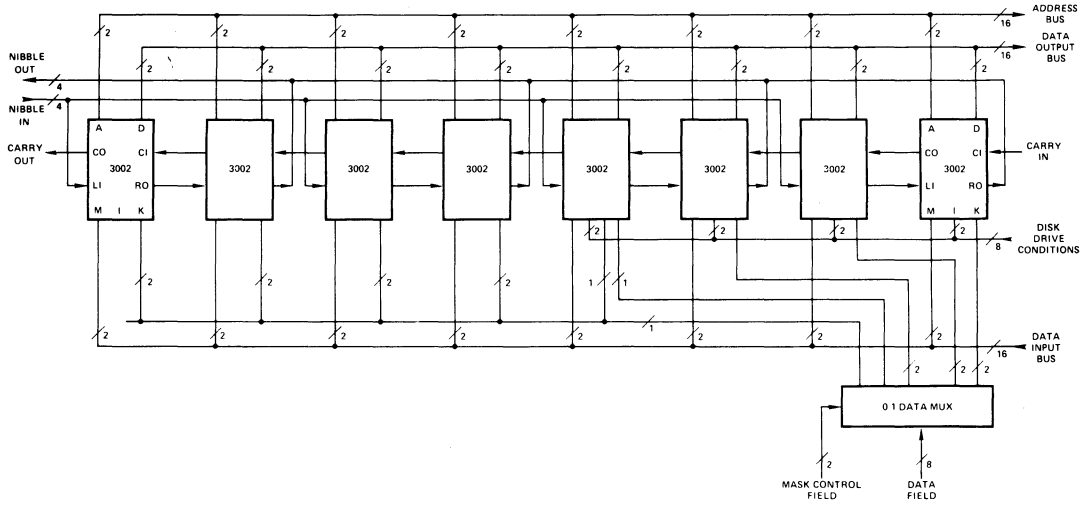


Figure 5. CPE Array – A 16-bit arithmetic, logic and register section is built up with 8 CPE slices connected in a ripple carry configuration. The K, I, and M bus is used for loading information into the CPE slices. The LI inputs and RO outputs are connected to make up the Nibble In and Nibble Out buses.

An eight bit mask bus is connected to the mask inputs of the least significant half of the array. The mask inputs of the most significant half of the CP array are all tied to the eighth mask bit. A constant with a value between +127 and −128 can therefore be loaded into the array from the microprogram. The mask bus comes from the data field of the microprogram via a 0-1 data multiplexer. When the CP array requires either an all one or all zero mask, the data field is freed to provide data to external logic.

The 3002 CPE is an extremely flexible component which makes it particularly attractive for controller designs. The Memory Address Register makes an ideal DMA address register.[1] The accumulator (AC) register, which also has its own output bus can be used as a data word buffer during a write DMA cycle. Concurrently, another word can be assembled in the T register using the shift right operation. The three separate input buses provide a multiplexing capability for routing different data into the CPE. In the BMDC, the I-bus is used for loading disk drive conditions, the K-bus for loading mask or constant information, and the M-bus for reading an external data buffer. The arithmetic logic section performs zero detection and bit testing with the result delivered to the MCU chip via the carry out line. Finally, the eleven scratchpad registers allow the controller to retain data and status for the processor.

The CP array in the microprocessor performs the following for the BMDC with its registers and arithmetic functions.

1. Sector counting
2. Word to nibble serialization
3. Drive seek status monitoring
4. Header checking
5. DMA address incrementing
6. Word counting
7. Multi-sector length counting
8. Automatic resynchronization of sector counter
9. Accessing of additional information from memory
10. Time delays

The organization of the microprocessor was chosen to maximize the use of the MCU and CPE in performing the various tasks required for disk control. However, there are some specialized tasks which are more economically performed by external logic. The microprocessor controls this external logic by output ports which are selected by the output select field in the microinstruction. The

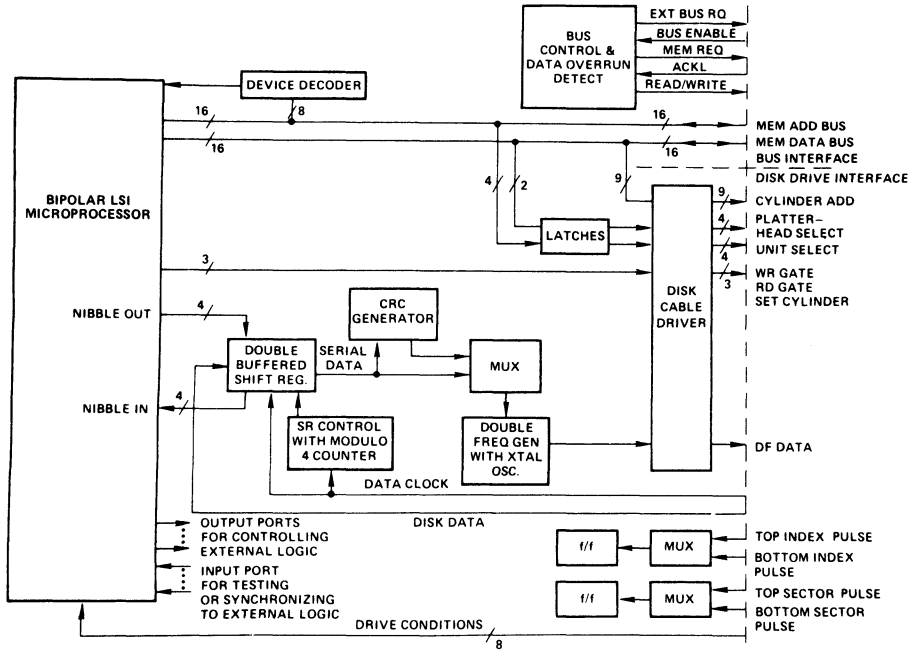


Figure 6. External Logic – Microprocessor monitors and controls external logic via input-output port to perform specialized disk controller functions.

data to these ports is delivered from the shared data field.

The external logic section of the BMDC (Fig. 6) has a double buffered 4-bit shift register which is used for initial packing and the final serialization of data. It is controlled by a modulo-4 counter circuit. During a write operation, serial data from the shift register is encoded by the clock controlled double frequency encoder and sent to the drive. As data is being transferred to a cyclic redundancy code (CRC) is generated and then appended to the end of the data stream to be recorded on the disk. The external logic also contains addressing latches and flag flip-flops to capture sector and index pulses. It also contains main memory bus control circuitry for performing bus protocol, bus acquisition, and data overrun detection.

The microprogram for the BMDC microprocessor directly implements the six I/O commands. The program controls the sequential action of the various elements of the microprocessor and of the external logic needed to decode and execute the commands. In Figure 7, the flow chart of the Read command shows the actions required to read a file off the disc. The BMDC first selects the drive specified by the command and checks its ready status. It then uses a memory pointer passed to it by the command to access four more words from the main memory using DMA cycles. The first word is the Header, which contains the track address and sector address information. The second word is the Starting Address specifying the first location in memory where the data is to be stored. The third word is the Block Length of the file to be retrieved. All of the address information and the Block Length are stored in several CPE registers for further processing. The fourth word is the Password which is compared against a microprogram word to insure that the command from the computer is a valid one and not a program error. The password can prevent an erroneous command, due to a user programming error, from destroying important files on the disc.

After the password check, the BMDC resynchronizes the sector counter if necessary and waits for the desired sector by monitoring the sector pulse flag. When the desired sector arrives, the BMDC synchronizes itself to a start nibble and reads the header which it compares to the desired header to insure that the head is positioned properly. It then reads and stores 128 words of data at sequential locations in memory. A cyclic redundancy code is compiled during the read oper-

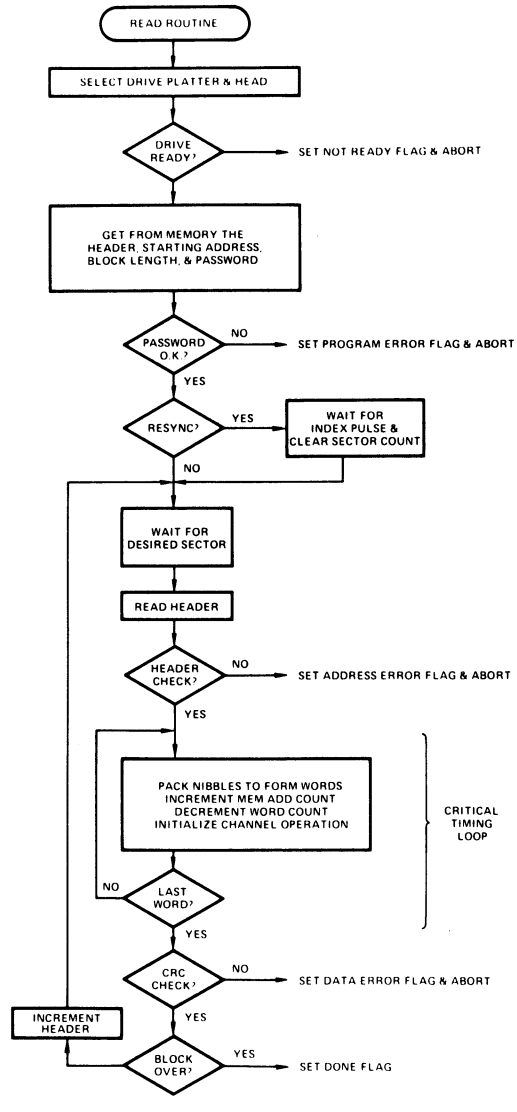


Figure 7. Read Command Flowchart – This flowchart is coded in the microprogram which when executed performs the disk Read operation.

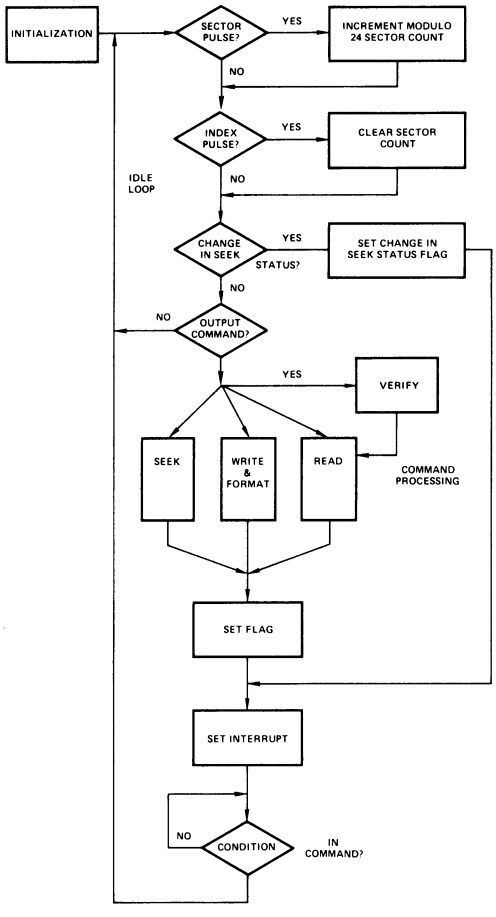


Figure 8. BMDC Flowchart – The BMDC runs in the idle loop when it is not busy doing command processing.

ation and compared against the CRC word read in after the data. At the end of each sector the block length is decremented to see if it is the last sector. If it is not, the sector address is incremented and another sector is read.

In addition to the command routines, the microprogram has an idle loop routine (Fig. 8) which the BMDC executes when it is not busy with a command. While in the loop, the BMDC updates the sector count, monitors the drives seeks status lines and decodes any disc commands from the disc operating system in the minicomputer.

The design process for the BMDC began with an evaluation of what disc controller operations could effectively be handled by the microprocessor. This also determined what had to be performed by external logic. A microprocessor configuration was then established and certain critical sequences were programmed to verify that the configuration was fast enough. A flow chart was produced and the microprogram coded directly from it. All attempts were made to use the MCU and CPE slices effectively and keep the microprogram within 256 words. The assignment of MCU addresses which initially appeared difficult, was, with a little experience, quite straight forward and less restrictive than a state counter design. After the coding, the microprogram was assembled and loaded into the microprocessor's control memory.

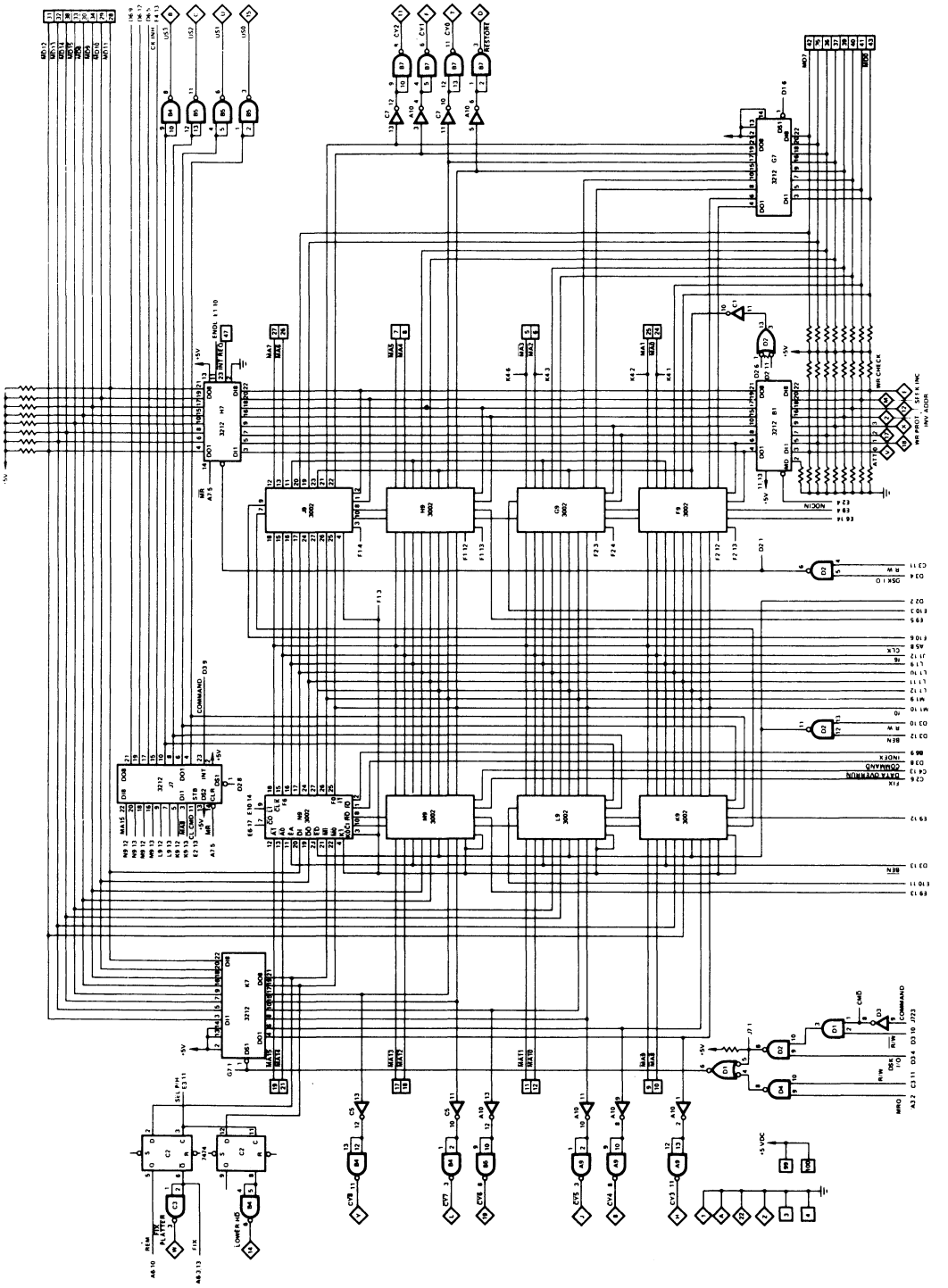
The BMDC design demonstrates how a specialized high speed microprocessor can be designed using standard bipolar LSI devices and microprogrammed to perform disc control functions with the addition of a small amount of external logic. The flexibility of Series 3000 allows a designer to optimize the configuration for his application. For extremely high speed applications, the designer can add fast carry logic and microinstruction pipelining to his microprocessor to achieve a 150 nsec 16-bit microprocessor.

At Intel, our design experience with the BMDC design exercise has shown that the use of the MCU and CPE results in a clean, well structured design. The complexity of the design resides primarily in the microprogram leaving the external logic relatively simple. During debugging, most of the problems encountered were restricted to the microprogram which was easily modified and debugged using bipolar RAM for the control memory.

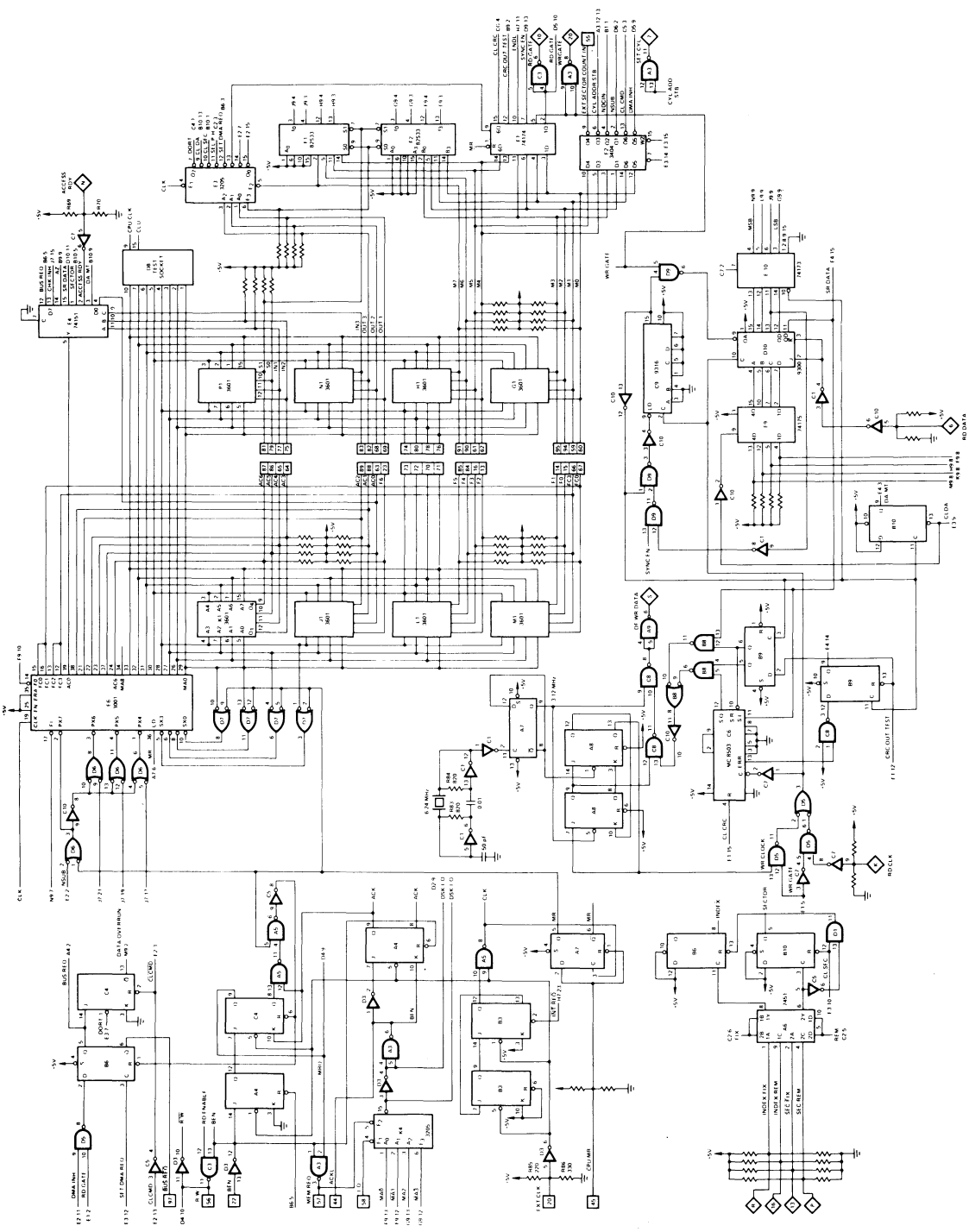
References

1. J. Rattner, J. Cornet, M. E. Hoff, Jr., "Bipolar LSI Computing Elements Usher In New Era of Digital Design," ELECTRONICS, September 5, 1974, pp 89–96.

APPENDIX — DISK CONTROLLER SCHEMATIC



APPENDIX — DISK CONTROLLER SCHEMATIC



Central Processor Designs Using The Intel® Series 3000 Computing Elements

by
M.E. Hoff, Jr.,
James Sugg,
Ron Yara

Contents

INTRODUCTION

Until recently, the area of high performance, general purpose and special purpose central processors was unaffected by the microprocessor revolution. Although they covered a broad range of applications, the P-channel and N-channel microprocessors' performance limitation prevented their use in applications where high speed was necessary.

The introduction of the Series 3000 Computing Elements has expanded the spectrum of microprocessor applications to include both high performance central processors and controllers. Utilizing Intel's Schottky bipolar technology, the Series 3000 components realized a level of performance that was not possible with MOS microprocessors. For example, a 16-bit processor with a microinstruction cycle time of 150 nanoseconds can be built with the 3000 components. In addition, the components of the family can be arranged into a number of different configurations and microprogrammed by the system designer to perform in a variety of processing environments from front end processing to arithmetic intensive computation.[1]

This application note describes a systematic procedure for designing central processors with the Series 3000. Using a CPU design example, simple guidelines are given for tasks such as macro-instruction opcode assignment, macro-instruction decoding and execution and microprogram memory assignment.

THE SERIES 3000 FAMILY

The Intel® Series 3000 Bipolar Microcomputer Set is a family of Schottky bipolar LSI computing elements which simplify the construction of microprogrammed central processors and device controllers. These processors and controllers are truly microprogrammed in the sense that their control functions are determined by the contents of a control memory. This control memory may be realized with standard read-only (ROM) memory, read/write (RAM) memory or programmable read-only memory (PROM) elements.

The two most important computing elements in the family are the 3001 Microprogram Control Unit (MCU) and the 3002 Central Processing Element (CPE). The MCU determines the sequence of micro-instruction execution and controls carry/shift data to and from the CPE array. The CPE provides a complete two-bit wide slice through the data processing section of a central processing unit. CPEs may be arrayed in parallel to form a processor of any desired word length. For example, to produce a 16-bit wide data path, eight CPEs would be used.

All of the above components use standard TTL logic levels, as some designers may wish to utilize SSI and MSI TTL logic to control external circuitry, or to add functions not included in the basic set to increase the speed of certain operations.

Other members of the family currently include the following computing elements:

- 3003 Look-Ahead Carry Generator
- 3212 Multi-Mode Latch Buffer
- 3214 Interrupt Control Unit
- 3216 Bidirectional Bus Driver
- 3226 Inverting Bidirectional Bus Driver

The control and main memory portion of the central processor may be implemented with any of the standard bipolar or MOS memory components shown on page 2.

AN INTRODUCTION TO MICROPROGAMMING

The central processing unit of a general purpose computer usually consists of two portions: an arithmetic portion and a control portion. The control portion determines the sequence of instructions to be executed and presides over their fetching and execution while the arithmetic portion performs arithmetic and logical operations.

The basic operation of the control portion consists of selecting the next instruction from memory, then executing a series of states based upon the instruction fetched. This sequence may be implemented via a combination of flip-flop and random logic, or by the use of tables in control memory.

[1] J. Rattner, J. Cornet, and M. E. Hoff, Jr., "Bipolar LSI Computing Elements Usher In New Era of Digital Design," ELECTRONICS, September 5, 1974, pp 89–96.

Standard Bipolar and MOS Memory Components

PART NUMBER	NUMBER OF PINS	TECHNOLOGY	DATA ORGANIZATION	ACCESS TIME
		CONTROL MEMORY		
3601	16	Bipolar PROM	256X4	70 nS*
3602	16	Bipolar PROM	512X4	70 nS
3604	24	Bipolar PROM	512X8	70 nS
3624	24	Bipolar PROM	512X8	70 nS
3301A	16	Bipolar ROM	256X4	45 nS
3302	16	Bipolar ROM	512X4	70 nS
3304A	24	Bipolar ROM	512X8	70 nS
3324A	24	Bipolar ROM	512X8	70 nS
3106A	16	Bipolar RAM	256X1	60 nS
3107A	16	Bipolar RAM	256X1	60 nS
		MAIN MEMORY		
1702A	24	Static MOS EPROM	256X8	1000 nS
2704	24	Static MOS EPROM	512X8	500 nS
2708	24	Static MOS EPROM	1024X8	500 nS
1302	24	Static MOS ROM	256X8	1000 nS
2308	24	Static MOS ROM	1024X8	500 nS
2316	24	Static MOS ROM	2048X8	850 nS
2101	22	Static MOS RAM	256X4	1000 nS*
2102	16	Static MOS RAM	1024X1	1000 nS*
2111	18	Static MOS RAM	256X4	1000 nS*
2112	16	Static MOS RAM	256X4	1000 nS*
2104	16	Dynamic MOS RAM	4096X1	
2107B	22	Dynamic MOS RAM	4096X1	200 nS
5101	22	Static CMOS RAM	256X4	650 nS

*Higher speed versions of these devices are available. Consult the Intel Data Catalog.

When the latter technique is used, the central processor is said to be microprogrammed.

The functions of the control portion of a microprogrammed central processing unit are very similar to that of a central processing unit itself. To avoid confusion, the terms "micro" and "macro" are used to distinguish those operations in the control unit from those of the realized central processor. For example, the central processor, under the direction of micro-instructions read from its control memory, fetches macro-instructions from main memory. Each macro-instruction is then executed as a series of micro-instructions. Main memory contains macroprograms, while control memory contains microprograms which define the realized central processor.

Figure 1 shows a block diagram of a microprogrammed central processing unit (defined by the dotted boundary). The control unit issues addresses to the control memory and fetches micro-instructions. This control unit uses the contents of control memory (micro-instructions) to drive the data processing unit, external circuits, and to select the

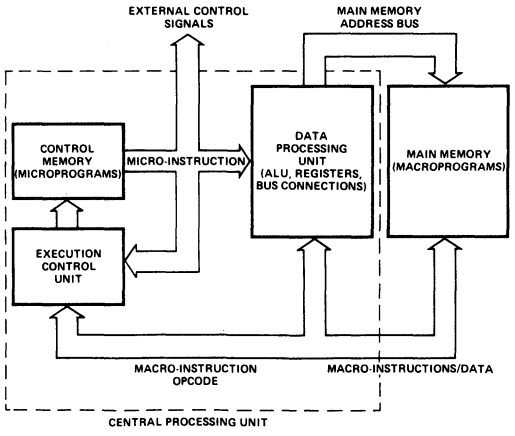


Figure 1. Block Diagram – Microprogrammed Computer

next micro-instruction. The data processing unit performs the actual computations, logical operations, etc.

In the Intel® Bipolar Microcomputer set, the 3001 MCU performs the control unit function, while the 3002 CPE is the basic building block for the data processing section.

Thus, within a microprogrammed machine, there are at least two levels of control and two levels of programming to be considered. The designer of a central processor is usually concerned with the definition of the macro-instruction set and its realization as a microprogram. The Intel® Series 3000 Bipolar Microcomputer Set establishes a micro-instruction set which is used as a base for the microprograms which generate macro-instruction sets.

The reason for using this microprogrammed approach is that very complex macro-instruction sets can be realized as sequences of relatively primitive micro-instructions. The logic of the final macro-machine remains relatively simple, with most of the design complexity residing in the micro-instruction sequences contained in control memory.

The final user of the computer seldom needs to be aware that the CPU was realized with microprograms rather than hardwired logic. A functional description of the macro-instruction set is usually sufficient for his purposes. However, the user will benefit from the microprogrammed approach if he finds it necessary to alter or enhance the basic macro-instruction set in some fashion. The tabular or programming approach offered by the microprogrammed architecture makes such changes far easier than would be possible in a processor realized via hardwired logic.

CONSTRUCTING CENTRAL PROCESSING UNITS

Basic Design Steps

To realize a central processor with the Series 3000 computing elements, several steps are necessary:

1. Definition of hardware organization.
2. Definition of the central processor macro-instruction set.
3. Implementation of microprograms which realize the desired macro-instruction set.

Hardware Organization

A typical CPU constructed utilizing the Series 3000 computing elements will consist of an array of CPE chips, one MCU, and a control memory. The array of CPE chips realizes the arithmetic, logical functions and registers of the CPU, while the combination of the MCU and control memory realizes the control portion. The microprogram contained in control memory initializes the machine when power is first turned on and supervises the fetching and execution of macro-level instructions. In addition, routines to handle such special functions as interrupts will also be contained within the control memory.

The 3002 CPE array contains six buses for communication with external circuitry. Four of these buses are used primarily to communicate with memory and I/O devices while the remaining two, the function control bus (F-Bus) and the control memory data bus (K-Bus), enable the control portion of the processor to drive the CPE array. The function control bus is driven by control memory outputs which direct the CPE array to execute the desired operation. The K-Bus allows the control memory to supply various constants and/or masks to the CPE array.

Because 8 bits of operation code information can be passed directly to the MCU, the set is best adapted to macro-instruction sets in which all of the operation code information is defined by 8 bits (256 unique macro-instructions). However, larger macro-instruction sets can be realized by saving any remaining bits of the operation code in the CPE array or in an external register. The saved bits can

then be tested later by routing them to the MCU, through its 8-bit input port.

A "pipelined" mode of operation may be implemented by placing a register of edge triggered D flip-flops between control memory outputs and the circuitry controlled by those outputs. This register causes the execution of a micro-instruction to overlap the fetching of the next micro-instruction. The control lines which issue micro-instruction sequence information to the MCU are not routed through the pipeline register when the pipelined mode is used; they are routed directly from the microprogram memory outputs to the AC0–AC6 inputs of the MCU.

Microprograms written to realize a given macro-instruction set will differ for pipelined and non-pipelined machines. The major differences are associated with conditional jumps in the microprogram which test the results of arithmetic or logical operations executed by the CPE array. In a pipelined machine, these results are delayed by one micro-instruction, so that conditional jumps must be delayed by at least one micro-instruction before execution. More detailed information concerning these differences is contained in the microprogramming section of this application note.

Figure 2 shows block diagrams illustrating the organization of standard and pipelined central processing units. The block diagrams show the basic modules of standard and pipelined CPUs: the MCU, CPE array, microprogram memory and the pipeline register. The six buses associated with the CPE array are shown:

- The address bus (A-Bus) to main memory
- The data bus (D-Bus) to memory
- The data bus (M-Bus) from memory with its path for operation code data to the MCU
- The external device input bus (I-Bus), not shown
- The micro-function bus (F-Bus) from the pipeline register
- The constant bus (K-Bus) from the pipeline register

In addition, the carry logic bus to and from the MCU and the micro-instruction sequence logic bus from control memory to the MCU are shown. Additional control fields to such external logic as memory and I/O control are shown as an output bus from control memory.

The number of bits required for each word of control memory, i.e., each micro-instruction, is determined by the number of logical functions the micro-instruction controls. A minimum of 18 bits is usually required for basic hardware control: 7 bits of micro-instruction sequence control to the MCU, (AC0–AC6), 4 bits of carry control to the MCU, (FC0–FC3), and 7 bits of micro-function selection to the CPE array, (F0–F6). That is, the basic hardware requires at least three control word fields of 7 bits, 4 bits, and 7 bits width respectively. Almost every processor will require additional fields to control other logical functions such as main memory control, I/O control, and constant generation. Figure 3 illustrates a typical micro-instruction word format with several typical user defined control fields added.

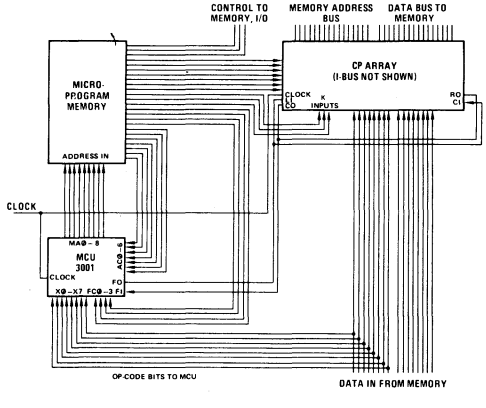


Figure 2. Bipolar Microcomputer Non-Pipelined Organization

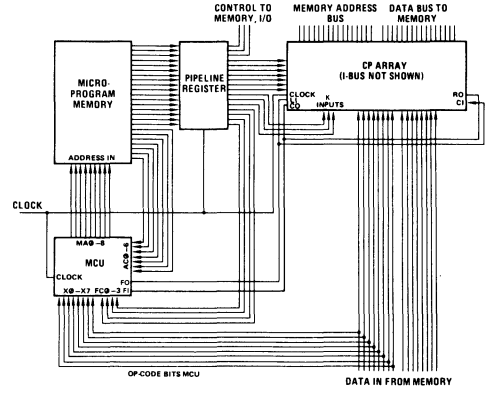


Figure 2. Bipolar Microcomputer Pipelined Organization

The constant bus to the CPE array seldom needs to be as wide as the data buses. For example, consider a 16-bit machine where an array of eight CPEs is used. While the constant bus is nominally 16 bits wide, if a limited set of masking operations are used, the number of bits can be reduced significantly. Figure 4 shows how 4 bits can be used to generate the masks for such a machine where the only masks needed are for separating high and low order data bytes, for testing the sign and magnitude of the data word, and for testing the least significant bit of the word.

As an example of the use of additional logic to enhance the set, consider the use of a control field (1-bit width) to inhibit the CPE clock. This operation allows non-destructive testing of CPE registers via the MCU carry logic. The carry logic in the MCU responds just as if the micro-instruction were executed, but the fact that the CPE clock was inhibited leaves the CPE registers unaltered. An example of conditional clocking is given in a later section called "Programming Techniques."

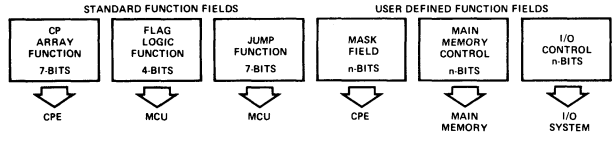
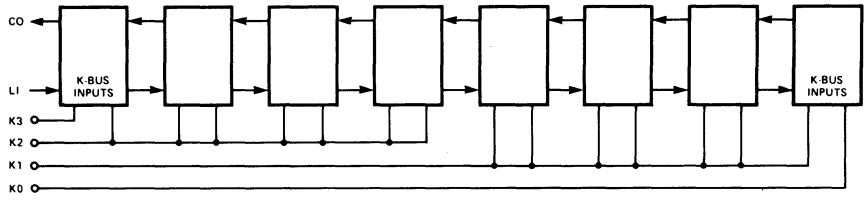


Figure 3. General Micro-Instruction Format



K3	K2	K1	K0 (BINARY, LOW TRUE)	K-BUS (HEXADECIMAL)	MASK FUNCTION
1	1	1	1	0000	
1	1	1	0	0001	SELECT LSB
1	1	0	0	00FF	SELECT LOW ORDER BYTE
0	0	1	1	FF00	SELECT HIGH ORDER BYTE
1	0	0	0	7FFF	SELECT WORD MAGNITUDE
0	1	1	1	8000	SELECT WORD SIGN
0	0	0	0	FFFF	SELECT ENTIRE WORD

Figure 4. Wiring the K-Bus Using 4-Bits

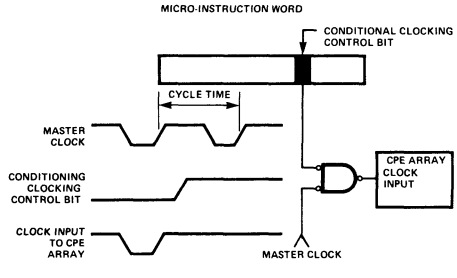


Figure 5. Conditional Clocking

Writing of Microprograms

Once the hardware design is established and the macro-instruction set chosen, the designer should proceed to implement the microprograms for the system. To assist in the writing of these microprograms, Intel has developed CROMIS, a complete microprogramming system for Series 3000 computing elements.

CROMIS consists of two major software subsystems, XMAS and XMAP. XMAS is a symbolic microassembler which is extensible in both micro-instruction length and memory address space. XMAP is a complementary subsystem which maps the micro-instruction bit patterns produced by XMAS into compatible ROM/PROM programming files for use with standard memory components.

Programs written in the microassembly language have two main parts, a declaration part in which various aspects of the micro-instruction word are defined and a specification part in which micro-instruction contents are symbolically declared. Provision is made for comment statements throughout the program so that the programmer may explain the functions being performed.

The main body of the program, the specification part, defines the sequences of states to be executed, and the operations which take place for each state. The main effort in writing a microprogram will be expended in developing this section.

Each statement of the specification part of the program defines the action (and location) of one micro-instruction, i.e., one word of control memory. The statement will declare, either directly or by default, the contents of each control field for the specified micro-instruction. Furthermore, the statement will include assignment information designating the address in control memory where the statement is located.

A specification statement consists of one or more labels followed by a series of control field specifications. A colon after an entry indicates that it is a label. The contents of the control fields are indicated symbolically, using either standard MCU or CPE symbols or user-defined symbols, or by an equation of the type

```
FNM = 101B
```

where FNM is a name associated with the field. The entry 101B implies the binary value 101. Each symbol is associated with only one field, so that the various symbols can be uniquely interpreted by the assembler. A number of symbols are predefined for the assembler, and are not to be used except as provided by the assembler. These reserved symbols include the standard symbols for the MCU and CPE functions, and a number of directives to the assembler.

DEFINITION OF CONTROL FIELDS

Each control field added by the hardware designer must be declared to the microprogram assembler. In addition, each bit pattern to be assembled into a word in the control field may be symbolically designated. A FIELD definition statement in the declaration part of the microprogram is used to declare the field by name and define any states.

As an example, let a 2-bit field be defined for memory control. If the programmer wishes to name this field MEMC, and define symbols for the states with 01 corresponding to READ, 10 corresponding to WRITE, and 11 signalling RMW (read-modify-write) and default to 00 if READ, WRITE or RMW is not specified, the statement:

```
MEMC  FIELD  MICROPS (READ=01B, WRITE=10B, RMW=11B)
LENGTH=2  DEFAULT=00B;
```

would perform the definition. The words FIELD, MICROPS, LENGTH, and DEFAULT are directives to the microprogram assembler.

Additional directives include IMPLY, STRING, KBUS, and ADDRESS. The use of these words, and other features of CROMIS are covered in the Series 3000 Cross Microprogramming System Specification.

A typical statement of the specification section might take the form:

```
7BH:  LAB:  ILR(R3)  FF0  STZ  JFL(NC TC);
```

The number 7BH (hexadecimal) followed by a colon tells the assembler that the micro-instruction is assigned to row 7 column 11 of control memory (when control memory is treated as an array of 32 rows and 16 columns). The symbolic label LAB

(the colon indicates a label) is also associated with this location. ILR(R3) indicates that the contents of register 3 are to be conditionally incremented and copied to the AC register, while FF0 forces the carry input to a logic zero, so that the increment operation does not take place. STZ indicates that the Z flip-flop is to be set by the results, so that, as no carry can result, the Z flip-flop will be set to a logic zero. These symbols are standard symbols, with ILR associated with the CPE and FF0 and STZ associated with the MCU carry logic. The JFL tests the carry output line for a conditional jump to either the statement labeled NC or to the statement labeled TC. JFL is also a standard symbol. Note that, if the machine is pipelined, the conditional jump tests the results of the previous instruction, not of the present one. The semicolon indicates the end of the statement.

In the statement above, no information was provided for the K-Bus. It is assumed the assembler will provide the appropriate default value associated with the ILR operation, i.e., the K-Bus at all zeros.

The reader is referred to the Intel® Series 3000 Cross Microprogramming System Specification for detailed information concerning CROMIS.

ASSIGNMENT TO CONTROL MEMORY

The nature of the MCU next state address control requires the programmer to assign control memory locations to each micro-instruction. While this may at first seem unfamiliar, it can usually be easily accomplished if the following sequence is followed:

1. The microprogram should be written without regard to address assignment. Then conditional jumps are assigned using the basic conditional jumps provided by the MCU (JFL, JCF, JZF, JPR, JLL, JRL, JPX), noting the number of possible destinations for the conditional jumps chosen. When a sequence of instructions is to be executed unconditionally and does not indicate what jump codes will be used to advance to the next state (unless the JCE enable feature is required), use the non-committal code JMP rather than selecting a JCC, JZR or JCR.
2. Prepare a state sequence flowchart for the program (see example, Figure 7). According to the programmer's preference, this may be done before, during or after the actual writing of the code. Label the conditional jump points on the flowchart.
3. Using the flowchart as a guide, perform the assignment. In general, conditional jumps should be assigned first, with clusters of conditional jumps assigned before isolated jumps. Leave long chains of unconditional sequences for last. The process of assignment can be assisted by using a diagram of the control memory showing the 32 rows and 16 columns. As each state is assigned, the control memory diagram is marked to show occupancy of that word and the flowchart marked to show the assignment of the state. With the assignment complete, the addresses are copied from the memory diagram.

One other procedure in microprogram memory assignment has been found to be useful. When the control memory diagram is marked as each state is assigned, it is helpful to include state linkage information in the diagram, i.e., memory location(s) that reference the current location and memory location(s) referenced by the current location. With the additional information, micro-instruction sequences can be easily traced on the control memory diagram.

The state linkage information can be quite useful when most of the microcode has been assigned and only a few locations are left to assign the remaining states. If reassignment of memory locations becomes necessary in order to assign the remaining microcode, or modify the existing microcode, the state linkage information will greatly simplify the task.

When reassignment becomes necessary, sequences of unconditional micro-instructions should be considered first since they are the easiest to move. Therefore, these types of states are useful to annotate.

In some cases, a particular sequence may be impossible to assign as written. For example, consider the following section of microprogram:

```
/* ENTER WITH INSTRUCTION DISPLACEMENT "D" IN AC, SAVE AT R9 */
  175:   SDR(9)   FF1 JPX(M0, M1, M2, M3, M4, M5, M6, M7, M8, M9, MA, MB, MC,
         MD, ME, MF);  /* ALSO TESTS HIGH 4 BITS OF MACRO-INSTRUCTION */
/* M0 – MACRO INSTRUCTION GROUP 1, FETCH R2 */
  128:   M0:    ILR(R2)     FF0    JMP(M1P);
  129:   M1:    ILR(R3)     FF0;
         M1P:   ADR(R9)     FF0;
```

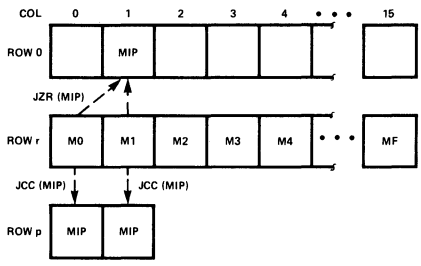


Figure 6. Operation MIP Can Be Reached From Both M0 and M1 by Locating MIP in Row 0 or Duplicating it in Both Column 0 and Column 1

In the above example, MIP follows both M0 and M1. Since the row in which M0 and M1 reside is completely filled, MIP must be located in row zero (because the JZR jump operation allows a location in row zero to be reached from anywhere in memory). If row zero were already fully occupied, the assignment could not be made. However, in this case the state represented by MIP might be duplicated so that it can be reached from state M0 and M1. No extra execution time is added by this modification, although one more memory location is used.

When assigning to memory, row zero locations should be used judiciously, but not sparingly, because only they can be reached from anywhere else in the program using a single JZR jump function.

Finally, in a 512-word microprogram memory there are 64 possible destination pairs for the JCF, JZF and JFL conditional jump functions, since all three use columns 2 and 3 or columns 10 and 11 as their jump target. It is therefore important to insure that enough destination pairs are available for the conditional jumps used in a microprogram.

PROGRAMMING TECHNIQUES

Because of the flexibility of both the micro-operations and the architecture of the Series 3000 computing elements, a number of programming "tricks" can be used to implement a desired operation. As the programmer becomes more familiar with the set, he will find new ways to perform different functions. The list of operations given here are intended as examples. In general, the labels indicating assignments to memory are not shown. In all of the examples, KB is the name associated with the K-Bus field of the micro-instruction. Statements bounded by /*. . .*/ are comments and do not affect the assembly.

1. Forcing a fixed address to access a predetermined location in memory or to select a specific I/O device. (Also may be used to load literals.)

```
CLR(N)              ;
LMI(N)   KB=DESAD ;
```

The first operation clears the register selected by N, while the second loads the logical OR of the contents of N and the contents of the K-Bus to the memory address register (MAR) of the CPE array and into register N. DESAD is a symbol for the desired address value previously defined by the programmer. The pair of micro-ops above may also be used to set any register to any desired constant, although the contents of the MAR are destroyed.

2. Any register may be set to all 1's by the operation

```
CSR(N)  FF0      ;
```

3. A value read from memory or I/O into the AC may be split into bytes and stored in another register as follows:

```
SDR(N)   FF1   KFF00;   /* STORE RIGHT BYTE IN REG N */
SDR(AC)  FF1   K00FF;   /* SET LEFT BYTE OF AC TO ZERO */
```

where KFF00 is a symbol which causes the K-Bus to be set to 1111 1111 0000 0000 in binary, and K00FF is a symbol for setting the K-Bus to 0000 0000 1111 1111 in binary. The high order byte is placed in the upper byte of register N while the low order byte remains in the low position of the AC. The low byte of register N and high byte of the AC are cleared.

4. *Sign Testing and Absolute Magnitude* – To test sign bits most effectively, an inhibit operation at the CPE clock is very desirable. In the following examples the symbol INH implies a signal from the control memory to inhibit the CPE clock. This prevents modification of the AC register.

The operations

```
        TZR(AC)  K8000  INH  JFL(AP,AN) ;
AN:     CIA(AC)                         ;
AP:     ...
```

generate the absolute magnitude of AC in AC for the non-pipelined case (note K8000 implies 1000 0000 0000 0000 on the K-Bus) while

```
        TZR(AC)  K8000  INH              ;
        NOP                   JFL(AP,AN) ;
AN:     CIA(AC)                          ;
AP:
```

performs the same operation for the pipelined case.

When two numbers in AC and T must be converted to positive numbers and the signs saved, as well as the sign of the product, the following routine may be used for a pipelined machine.

```
/* ENTER WITH VALUES IN T, AC */
/* FIRST CLEAR SIGN AREA - REGISTER 9 FOR THIS EXAMPLE */
   CLR(R9);
/* NEXT TEST SIGNS OF AC, THEN T */
         TZR(AC)   K8000  INH;   /* TEST AC SIGN BIT */
         TZR(T)    K8000  INH    JFL(AP,AN);   /* TEST T SIGN BIT */
   AP:   LMI(R9)   K8000  FF1    JFL(TP,TN);   /* SET HIGH AND LOW ORDER BIT */
   AN:   CIA(AC)                 JFL(TP,TN);   /* COMPLEMENT AC */
   TP:   LMI(R9)   K4000  FF1    JMP(NXOP);    /* SET BIT 15 */
   TN:   CIA(T);                               /* COMPLEMENT T */
NXOP:    ...
```

Upon reaching label NXOP, both AC and T will contain positive numbers (high order bit = 0) and register 9 will contain a 1 in the high order bit if and only if AC was originally positive, a 1 in the second bit from the top if and only if T was originally positive, and a zero in the low order bit if and only if the signs were the same. A one will appear in the second lowest order bit if and only if both numbers were originally positive. Execution of the sequence takes 5 micro-instruction cycles.

5. *Pipelined Multiply* – Assume that AC and T represent the partial product and multiplier respectively, while register 9 contains the multiplicand and register 8 will be used as a loop counter. Register 7 is used for temporary storage. It is assumed that both numbers are positive.

```
/* SET UP LOOP COUNTER */
   MCL:  CSR(R8)  K0000;   /* SET R8 TO FFFF HEX */
         TZR(R8)  KFFF0;   /* SET R8 TO FFF0 HEX */
/* CLEAR PARTIAL PRODUCT (AC) */
         CLR(AC);
/* FETCH AND TEST MULTIPLIER LOW ORDER BIT */
         SRA(T);
/* MAIN LOOP - EXECUTE MULTIPLIER BIT TEST, ADD IF NECESSARY */
   MLP:  LMI(R8)  FF1    STZ    JFL(MBZ,MB1);   /* INCREMENT LOOP COUNTER SAVE IN Z */
/* ADD SEQUENCE */
   MB1:  SDR(R7)  FF1;   /* SAVE AC IN REG7 */
         ILR(R9)  FF0;   /* PLACE MULTIPLICAND, R9, IN AC */
         ALR(R7)  FF0    /* ADD MULTIPLICAND TO PARTIAL PRODUCT */
/* NOW ROTATE, THEN TEST LOOP COUNT - SAVED IN Z */
/* NOTE - PIPELINE ALLOWS USE OF Z FOR SHIFT BIT PROPAGATION */
/* NOTE - THE SDR(R7), ILR(R9), AND ALR(R7) MICRO-INSTRUCTIONS CAN BE
   REPLACED WITH AN AMA MICRO-INSTRUCTION ELIMINATING Z INSTRUCTIONS
   FROM THE INNER LOOP IF DATA IS LATCHED ON THE M-BUS */
   MBZ:  SRA(AC)  FF0  STZ;              /* SHIFT PARTIAL PRODUCT, SAVE LSB */
         SRA(T)   FFZ  JZF(MLP,MEX);     /* Z TEST IF OF LOOP COUNT! */
   MEX:  ...
```

Note that the pipeline causes the JZF (or a JCF) to test the contents of the flip-flop as set two or more instructions earlier.

A state sequence flow diagram for the multiply sequence might be drawn as shown in Figure 7.

Note that in Figure 7, each symbolically labeled state is noted, and each conditional jump is indicated and the conditions corresponding to each jump are noted. A flowchart like that of Figure 7 contains sufficient information to perform the assignment to memory. An assignment might be as shown in Figure 8.

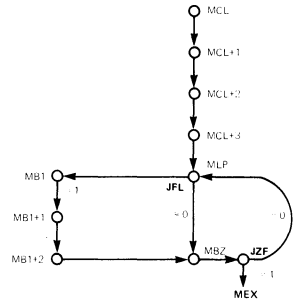


Figure 7. State Sequence Flow Diagram – Multiply Loop

	COL 0	COL 1	COL 2	COL 3	COL 4	COL 5
ROW 9	MCL	MCL +1	MLP	MEX	MCL +2	MCL +3
ROW 10	MB1 +1	MB1 +2	MBZ	MB1		

Figure 8. An Assignment of the Multiply Loop to Control Memory

Because MLP and MEX are the two destinations of a JZF jump function, they must be in the same row, in columns two and three respectively or in columns 10 and 11 respectively. Since MLP executes a JFL to MBZ, MB1, then MBZ and MB1 must be in the same pair of columns as MLP and MEX. For the example, rows 9 and 10 were chosen, and columns 2 and 3, and the four states MLP, MEX, MBZ, MB1 are assigned first. Next the states following MBL (indicated by MB1+1 and MB1+2) and MBZ are assigned. As all of these jumps are unconditional, the operations JCC, JCR, and JZR are used. As the JZR is usually reserved for entry to commonly used routines, only the JCC and JCR jumps are used here.

To demonstrate the techniques introduced above, a central processing unit design cycle will be carried through from initial specification to final microprogram memory assignment.

A DESIGN EXAMPLE

The following design example illustrates some of the basic techniques which may be used in developing a central processor with the Intel® Series 3000 Bipolar Microcomputer Set. The basic design sequence consists of stating the machine objectives, then designing the hardware configuration and microprograms. For this example, it is assumed that the designer has the freedom to specify operation code assignments, and to modify the instruction set to take greatest advantage of the chip set's capabilities.

Initial Specifications

Let the following list of design objectives represent the initial specifications for a central processor instruction set.

1. The machine should use a 16-bit data path, with instructions containing an opcode portion and a data or displacement portion.

2. Machine registers should include a program counter, P, a stack pointer, S, an accumulator, A, an index register, X, and two base registers, B and E. B is a base register for data and E is a base register for program. In addition, a carry flip-flop may be a bit in the status word, W.

3. References to memory for data should be relative to the B register, using the displacement portion of the instruction (designated D). Memory reference modes include direct (Address=B+D), indirect (address equals the contents of B+D), and indirect indexed (address equals the value given by the sum of X and the contents of the word at address B+D). Indirect and indirect-indexed modes should include both absolute and B relative (i.e., the address is relative to the contents of the B register) forms so that indirections may be computed both at time of assembly and during program execution.

4. Memory reference instructions include: load address to A, load data to A, AND data to A, OR data to A, XOR data to A, add data to A, subtract data from A, push address to stack, push data to stack, store A at computed address, pop stack to computed address, load address to X, load data to X, add data to X, subtract data from X, store X at computed address (operations involving X may not need to implement indirect-indexed modes).

5. Immediate instructions using the displacement portion of the instruction as the data, include, load A, load X, add to A, add to X. A two word "load immediate" instruction may also be implemented.

6. Jump instructions include a short relative jump (Address=P+D–K, where K is a constant), an indirect jump to an address relative to the E base register, and an indirect call operation.

7. The call (to a subroutine) operation saves the P, E, B, and W registers (global call), or the P register (local call) on the stack and loads the P register with the starting address of the routine. Similarly, a return instruction restores the appropriate registers. Some jumps may also be conditional, checking the status of the C flip-flop, or the sign or magnitude of the A register.

8. Additional operations may involve manipulations of data in the A and X registers and the ability to move data between the X and the W, B, E or S registers.

9. Byte load and store operations should include automatic packing and unpacking of bytes in a 16-bit memory location.

10. Input/output instructions should use either the displacement or the X register to specify the I/O device address.

In addition to the definition of the macro-instruction set, the designer should also prepare descriptions of the initialization operations (i.e., at "power on") and interrupt handling to be used. For this machine, let it be considered necessary for the machine to start at power up with W, A, and X cleared and for S to be set to the contents of word 0, B to be set to the contents of word 1 of memory, E set to the contents of word 2, and P set to the contents of the memory location pointed to by E.

Let I/O device 0 represent a source of interrupt level information (level requesting in) and a destination for current level out, consistent with the use of the 3214 Interrupt Control chip. In addition, let the low order bits of W contain current interrupt level information.

When servicing an interrupt, the processor will execute a jump to subroutine which will reload P and E while saving all registers except S on the stack. The service routine will interrogate the interrupt hardware to determine the level of the request and will restore former status upon exit from the interrupt program. For this purpose, a return and restore status instruction will be provided.

In parallel with the specification of the design objectives, a first pass at the CPU's architecture can be made. The block diagram in Figure 9 shows a general CPU architecture as defined in the initial specification above.

The design example machine uses a pipelined architecture and includes a control structure which implements eight basic memory bus and clock operations. A 3-bit field is used to control this structure. The states for this field are designated

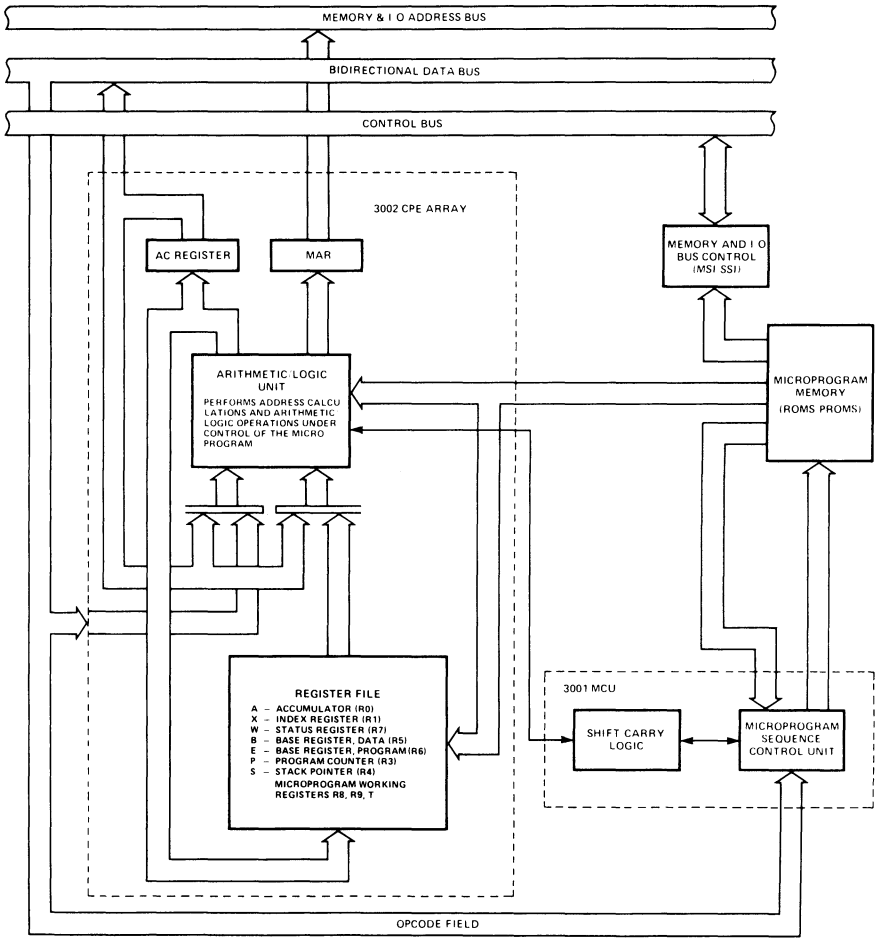


Figure 9. Block Diagram of CPU Architecture

NBO (No Bus Operation), INH (Inhibit CPE Clock), CNB (CPE uses bus), RMW (read modify write signal to memory – starts a read cycle and prevents release of bus until the CPU executes a write cycle), RRM (Request read cycle from memory), RWM (Request write to memory), RIN (Request input from an I/O device), and ROT (Request an output to an I/O device).

The stack has been designed to run "backwards" through memory, with a pop incrementing the stack pointer and a push decrementing it. This direction is preferred, as it leaves the stack pointer pointing at the topmost entry in the stack. In addition, pops usually appear more often than pushes (pushes share code), and the increment operation requires fewer micro-instructions.

The designer must select the actual instructions to be used. Let the instructions and their associated mnemonics shown in Table I be selected in the first design pass.

Table I. Proposed Instruction Set

MEMORY REFERENCE GROUP

MNEMONIC	FUNCTION
LAA	Load address to A
LDA	Load data to A
ADA	Add data to A
SDA	Subtract data from A
NDA	AND data to A
ODA	OR data to A
XDA	Exclusive OR data to A
PAS	Push address to stack
PDS	Push data to stack
SAM	Store A into memory
PSM	Pop stack into memory
LAX	Load address to X
LDX	Load data to X
ADX	Add data to X
SDX	Subtract data from X
SXM	Store X in memory

IMMEDIATE GROUP

MNEMONIC	FUNCTION
LAI	Load to A immediate
AAI	Add to A immediate
NAI	AND to A immediate
OAI	OR to A immediate
XAI	Exclusive OR to A immediate
PSI	Push to stack immediate
LXI	Load to X immediate
AXI	Add to X immediate

If D is equal to zero, the contents of the memory location following the instruction is used as the immediate value.

JUMP GROUP

MNEMONIC RELATIVE	MNEMONIC INDIRECT	FUNCTION
JRU	JIU	Jump unconditional
JRGE	JIGE	Jump if A≥0
JRLT	JILT	Jump if A<0
JRXG	JIXG	Jump if X>A
JREZ	JIEZ	Jump if A=0
JRNZ	JINZ	Jump if A≠0

JUMP GROUP (continued)

MNEMONIC RELATIVE	MNEMONIC INDIRECT	FUNCTION
JRCZ	JICZ	Jump if C=0
JRXL	JIXL	Jump if X≤A
JRLE	JILE	Jump if A≤0
JRGT	JIGT	Jump if A>0
JRCN	JICN	Jump if C≠0
JRXE	JIXE	Jump if X=A

Jump relative: P=P+D−128
Jump indirect: P=(E+D)+E

STACK PUSH AND POP GROUP

MNEMONIC	FUNCTION
PHAX	Push A and X onto stack
PPAX	Pop A and X from top of stack

SPECIAL MEMORY REFERENCE INSTRUCTION

MNEMONIC	FUNCTION
ISZ	Increment location B+D and skip if zero

SUBROUTINE CALL GROUP

MNEMONIC	FUNCTION
CLS	Call local subroutine, push P onto stack P=E+(E+D)
CVS	Call value subroutine, push W, B, E, P onto stack E=E+(E+D) P=E'+(E') where E'=E+(E+D)
CAS	Call absolute subroutine, push W, B, E, P onto stack P=(D)

SUBROUTINE RETURN GROUP

MNEMONIC	FUNCTION
RLS	Return from local subroutine, pop P from stack
RVS	Return from value subroutine, pop P, E, B, W from stack
RSA	Return from subroutine, restore all, pop A, X, P, E, B, W from stack

Table I. Proposed Instruction Set (continued)

BYTE LOAD AND STORE GROUP

MNEMONIC	FUNCTION
LBA	Load byte absolute
LBR	Load byte relative
SBA	Store byte absolute
SBR	Store byte relative

Absolute mode: Byte address = (B+D)+X/2
Relative mode: Byte address = (B+D)+B+X/2

The least significant bit of the X register is treated as the byte pointer in main memory as follows:

X Reg. LSB = 0 the left or high order byte is selected
= 1 the right or low order byte is selected

For load operations, the selected byte is loaded into the right byte position of the A register and the left byte is cleared. For store operations, the right byte of the A register is stored at the selected byte location leaving the unselected byte of the word unaltered.

REGISTER MANIPULATION GROUP

MNEMONIC	FUNCTION
RAR	Rotate A right, include CFF
RAX	Rotate A and X right, include CFF
SAX	Shift A and X right, preserve sign
SAL	Shift A left, fill with zeros

The shift count is given by D if D is non-zero or by the least significant seven bits of the X register if D is zero.

BASE AND STATUS REGISTER MOVE GROUP

MNEMONIC	FUNCTION
MSX	Move S to X, adjust
MBX	Move B to X, adjust
MEX	Move E to X, adjust
MWX	Move W to X, adjust
MXS	Move X to S, adjust
MXB	Move X to B, adjust
MXE	Move X to E, adjust
MXW	Move X to W, adjust

The destination register is adjusted by D–128 (i.e., D–128 is added to the destination register).

INPUT/OUTPUT GROUP

MNEMONIC	FUNCTION
IND	Input one word to the A register
OTD	Output one word from the A register

D serves as the address for the I/O port.

INX	Input one word to the A register
OTX	Output one word from the A register

The X register provides the address for the I/O port.

Given the basic design objectives, the next step is to write the sequences of micro-instructions to implement the macro-instruction described above. Each macro-instruction must be assigned a unique operation code. The operation code (opcode) will be used by the 3001 MCU to generate the appropriate address for the micro-instruction which executes that macro-instruction.

Macro-Instruction Decoding

To take full advantage of the 3001 MCU's eight input lines (SX0–3, PX4–7) for instruction decoding, all macro-instruction operations should be completely specified in an 8-bit opcode field and use the remaining 8 bits for displacement values. In Figure 10 the 8-bit opcode of a macro-instruction being read in on the memory data bus is gated directly to the 3001 MCU. While the displacement is being stored in the CPE array, a JPX operation is

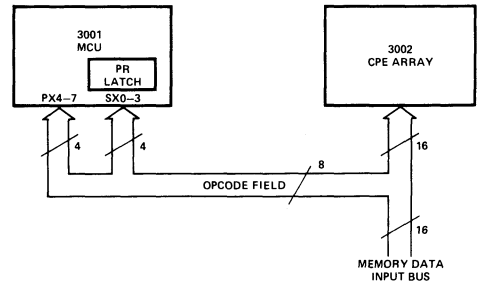


Figure 10. Macro-Instruction Decoding with the 3001

executed by the 3001. The JPX operation executes a 16 way branch based on the 4 bits of the PX lines and also stores the 4 bits on the SX lines in the PR latches for later decoding. For best microcode efficiency then, the opcode field should be arranged in such a manner that the first 4 bits tested (by the JPX operation) select the initial processing (usually an address calculation) of the macro-instruction. A possible instruction format is shown in Figure 11.

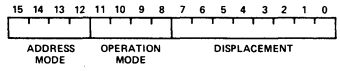


Figure 11. Possible Macro-Instruction Format

In the case of the CPU design example, the initial processing involves address calculations and/or operand fetching. Table II contains the initial processing modes for the design example.

Table II. Memory Modes

In the description below, the letters A, X, B, S, P, W, and E represent the contents of the respective registers. D represents the 8-bit displacement treated as a positive number ranging from 0 to 255. D' represents D–128. ( ) are used to designate contents of memory. For example, (B+D) means the contents of the memory location whose address is equal to the sum of the contents of B and the displacement D. It is assumed that, when the instruction is fetched, P is incremented prior to instruction execution.

MEMORY REFERENCE MODES
1. Direct: Address = B+D
2. Indirect: Address = (B+D)
3. Indirect relative: Address = (B+D)+B
4. Indirect indexed: Address = (B+D)+X
5. Indirect indexed relative: Address = (B+D)+B+X
IMMEDIATE MODES
6. If D≠0, Data = D–128 If D=0, Data = (P), P=P+1
JUMP MODES
7. Jump relative: P=P+D–128
8. Jump indirect: P=(E+D)+E
9. Call relative: P=(E+D)+E
10. Call indirect: P=E'+(E') where E'=E+(E+D)
REGISTER MODE
11. Fetch source register

Using the instruction format shown in Figure 11, the high order 4 bits (bits 12 to 15) will be used to select one of the modes listed in Table II. Thus, by executing a JPX operation, a 16 way branch on the PX0–PX3 bus can be performed to determine the address mode specified. At the same time the SX bus bits (the Operation Code field) will be stored in the PR latches for later use. A possible assignment of the first 4 bits (bits 12 through 15) might be as shown in Table III.

In addition to the initial address mode processing input/output, register to register, and other special function operations can be specified in the first 4 bits, as shown in Table III.

Microprogram Implementation

Having assigned the first 4 bits of the macro-instruction operation code, the next 4 may be tentatively assigned. These 4 bits will have different meanings for different instruction classes. To improve microcode efficiency it is desirable to share as much code as possible between different microprogram segments. For example, the ADA and AAI instructions might share the add operation once the data has been fetched.

MEMORY REFERENCE AND IMMEDIATE GROUP

The assignment shown in Table IV might be used for the memory reference and immediate group instructions. The clustering has been chosen in a way that should allow JPR and JLL and JRL micro-operations to be used effectively and to allow code sharing between the two groups.

An initial flowchart for the memory reference and immediate group instructions is shown in Figure 12. In the flowchart, the boxes indicate the operations performed. The appropriate jump operations (JPX, JLL and JRL) are indicated along with the bit patterns that select each box.

It is possible that when the actual code for the sequence is written, some improvements in efficiency may still be made. In addition, some of the boxes shown as dummies may be eliminated by suitable placement of the JLL and JRL instructions.

Knowledge of the MCU assignment restrictions may also influence some choices here. For example, the MCU provides twice as many possible JLL jump destinations as JRL jump destinations, while the sequence shown uses twice as many JRLs as JLLs. As a result, an easier assignment might be obtained if the JLLs and JRLs were exchanged, which is equivalent to a reassignment of the macro-operation codes.

Also, recognizing that the MCU's JCC type jump facilitates jumping from one JLL destination to another, it is desirable to assign the macro-operation codes so that operations which share final segments are aligned in columns. For example, the SDA instruction would typically be achieved by complementing the data, then adding it to A, which may share the code for ADA. As a result, a

Table III. Mode Bit Assignments

ADDRESS MODE BITS	MODE	INITIAL PROCESS	SUBSEQUENT PROCESSING
0000	No operation		
0001	Jump relative	P+D'	Condition testing
0010	Jumps (index, etc.)	(E+D)+E	
0011	Immediate	D' or (P)	LAI, AAI, etc.
0100	Direct memory reference	B+D	
0101	Indirect memory reference	(B+D)	
0110	Indirect index	(B+D)+X	LAA, LDA, etc.
0111	Indirect index relative	(B+D)+X+B	
1000	I/O input	D → MAR	
1001	I/O input	X → MAR	
1010	I/O output	D → MAR	
1011	I/O output	X → MAR	
1100	Move group		
1101	Special function group		Shift A
1110	Indirect relative memory reference	(B+D)+B	
1111	No operation		

Table IV. Memory Reference and Immediate Op Code Assignment

OP FIELD BITS	MEMORY REFERENCE FUNCTION	IMMEDIATE FUNCTION
0000	ADA	AAI
0001	ADX	AXI
0010	NDA	NAI
0011	ODA	OAI
0100	LDA	LAI
0101	LDX	LXI
0110	PDS	PSI
0111	XDA	XAI
1000	LAA	
1001	LAX	
1010	PAS	
1011	SDA	
1100	SAM	
1101	SXM	
1110	PSM	
1111	SDX	

better assignment of opcodes might be achieved by placing ADA and SDA in the same column. For example, see the assignment shown in Table V. Table V also assumes exchange of the JLL and JRL instructions.

Table V. Modified Memory Reference Op Code Assignments

0000 = NDA	0100 = ODA	1000 = XDA	1100 = ADA
0001 = LDA	0101 = LDX	1001 = PDS	1101 = ADX
0010 = LAA	0110 = LAX	1010 = PAS	1110 = SDA
0011 = SAM	0111 = SXM	1011 = PSM	1111 = SDX

Except for those considerations mentioned above, the code is most easily written without regard to memory assignment. Also, it is assumed that reassignments of macro-operations codes are made when efficiency can be improved.

Let the CPE register assignments be made as shown in Table VI.

The code which follows represents the specification portion of the microprogram in which the various fields are identified, and symbols defined.

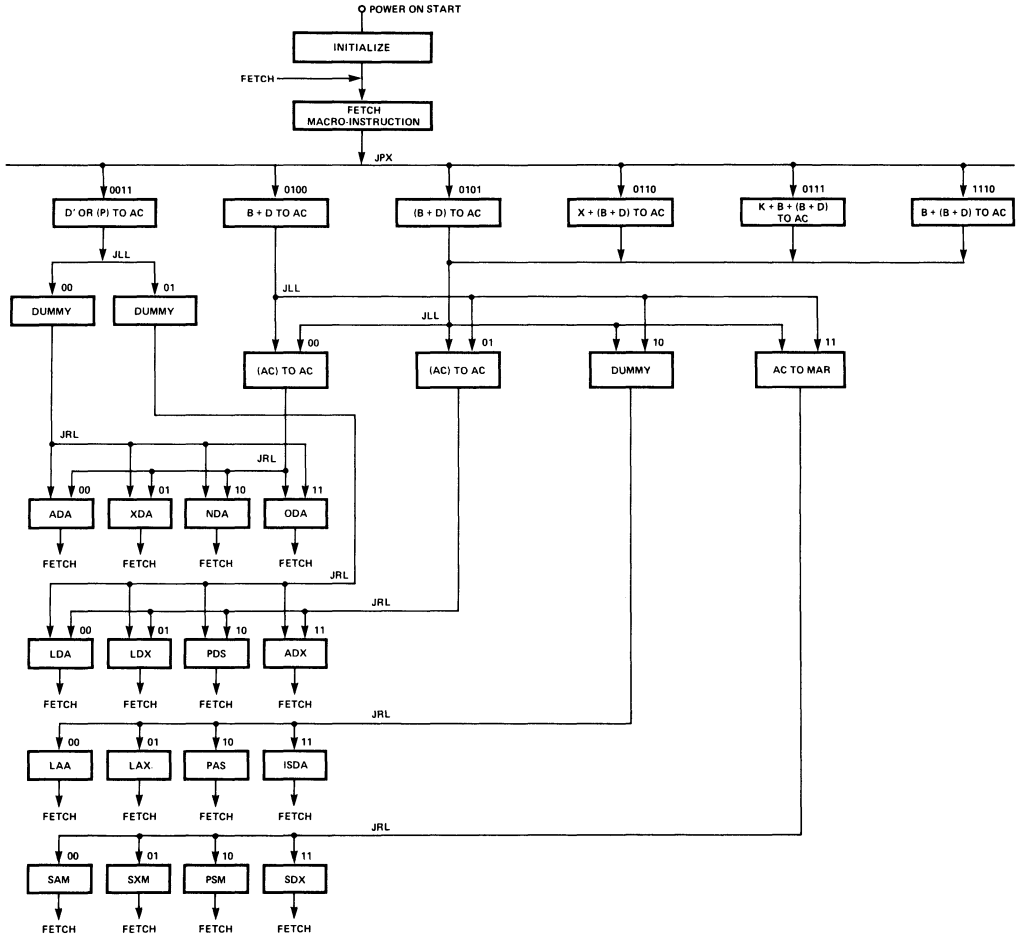


Figure 12. First Pass of Memory Reference Group Flowchart

```
/* BIPOLAR MICROCOMPUTER MACRO-MACHINE
   REGISTER MACHINE- -12/13/74
   UPDATED 3/18/75

   MACHINE HAS 7 REGISTERS AS FOLLOWS:
   A          ACCUMULATOR        R0
   X          INDEX REGISTER     R1
   P          PROGRAM COUNTER    R3
   S          STACK POINTER      R4
   B          DATA BASE REG      R5
   E          PROG. BASE REG.    R6
   W          STATUS WORD        R7

   C=CARRY,LINK FLIP-FLOP=HOB OF W

   DEFINITION OF KBUS FIELD */
```

```
   KB          FIELD    LENGTH=4          DEFAULT=0
               MICROPS(K0000=0     K007F=1      K000FF=3    K7FFF=7
                       K8000=8     KFF00=12   KFF80=14    KFFFF=15);
   KB          KBUS;

/* DEFINITION OF BUS CONTROL FIELD */

   MCF         FIELD    LENGTH=3          DEFAULT=0
               MICROPS(NBO=000B    INH=001B     RMW=010B     CNB=011B
                       RIN=100B    ROT=101B     RRM=110B     RWM=111B);

/*             NBO      NO BUS OPERATION
   INH         INHIBIT CPE ARRAY
   RMW         READ-MODIFY-WRITE
   CNB         CPU NEEDS BUS
   RIN         REQUEST INPUT
   ROT         REQUEST OUTPUT
   RRM         REQUEST READ MEM.
   RWM         REQUEST WRITE MEM.

   SET UP FOR SYMBOLIC REPRESENTATION OF REGISTER DESIGNATIONS */

   A           STRING   'R0';
   X           STRING   'R1';
   P           STRING   'R3';
   S           STRING   'R4';
   B           STRING   'R5';
   E           STRING   'R6';
   W           STRING   'R7';

/* SET UP A SPECIAL NO.OP STRING */

NO.OP          STRING   'NOP(R3)';

/* NEXT WE SPECIFY A DEFAULT TO FF1 IN THE FO FIELD FOR THE SDR
   MICROP IN THE CPE FIELD. SDR IS NORMALLY USED AS A STORE
   OPERATION. WHEN A DECREMENT OPERATION IS ALSO DESIRED, FF0
   WILL HAVE TO BE EXPLICITLY SPECIFIED */

   SDR         IMPLY    FO=11B;
```

Table VI. Register Assignments

R0	= A
R1	= X
R3	= P
R4	= S
R5	= B
R6	= E
R7	= W (C is high order bit of W)

The next portion of the code represents the machine initialization (in which registers are set to initial values during power up), and the memory reference and immediate group of instructions. The elementary flowchart followed is that of Figure 13, reflecting the reassignment shown in Table V.

A number of programming "tricks" can be found in the microcode. For example, the C flag of the MCU (not to be confused with the C flip-flop of the macro machine) is set each time the machine executes a fetch instruction by the SDR micro-operation. SDR adds 111...1 to the AC (as masked by the K-Bus) so that whenever the carry input of the CPE array is a 1, the masked AC register will be stored unchanged into the designated register, and the carry output of the CPE array will be 1. Similarly, a ILR micro-operation (KBUS = 0) with a carry-in of zero never generates a carry, so that it can be used to clear the C flag if so desired.

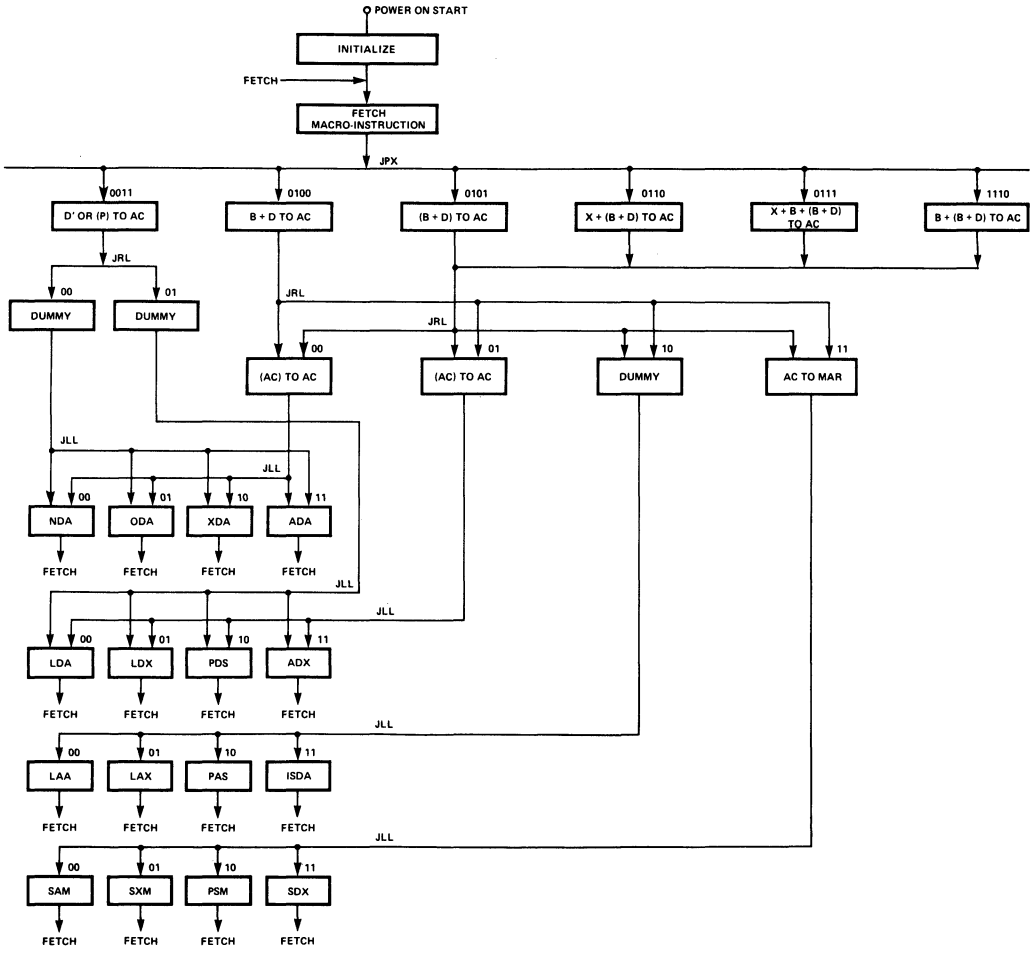


Figure 13. Second Pass of Memory Reference Group Flowchart

The C flag is used to implement a type of microcode subroutine where code is shared by two "calling" routines, one which leaves the C flag unchanged and the other which clears it. Upon exit from the shared code sequence, the C flag is tested giving a unique exit for each of the two calling routines (see Figure 14).

The inhibit operation, indicated by the "INH" micro-operation, inhibits the clock to the CPE array. For these operations the carry function and conditional jump results are the same as if the operation were executed. However, none of the CPE registers are altered when the clock is inhibited.

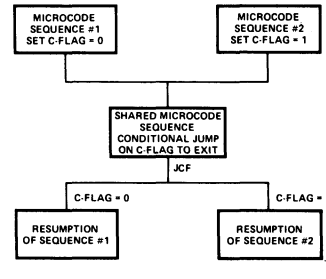


Figure 14. Microcode Subroutine Using the C-Flag to Determine Exit

The result is a number of "compare" or test micro-operations.

In general, row zero locations should be used sparingly because they are the only locations that can be reached from anywhere in microprogram memory using a single JZR micro-operation. During the first pass of the microprogram implementation, notes can be added to indicate where code might be saved if row zero locations are used.

A common case of such microcode saving follows the execution of a JPR or JPX micro-operation. If the datum being tested by the JPR or JPX represents a macro-instruction operation code in which less than 16 modes are used, there is always the possibility that an invalid code might be encountered. Rather than have the machine behave unpredictably, it is better to have the machine execute some designated sequence for invalid macro-operation codes. As a result, all 16 locations reached by the JPX or JPR micro-operation must be considered occupied. Therefore, when it is desirable to have a single state follow each of several states reached by a single JPX or JPR micro-operation, two possible methods can be used which do not require additional jump micro-operations:

1. Locate the single state in the row zero
2. Locate the single state in a column reached by a JCF or JZF micro-instruction and insure the corresponding (C or Z) flag is in the desired state.

As an example of this situation, consider the following sequence of micro-instructions (only labels and jumps shown):

```
TST:    JPR (D0, D1, D2, D, . . . D15)
D0:     JMP (D1A)
D1:
D1A:
```

In the sequence above, D0 through D15 occupy an entire row. The micro-instruction labeled D1A unconditionally follows both of those labeled D0 and D1. Since the row containing D0 through D15 is fully occupied, D1A cannot be assigned to that row. The only other unconditional jump which can reach a common location from more than one column is the JZR. However, such conditional jumps as JCF and JZF, where the condition is pre-set, can jump to a given location from up to eight sites in a given row, as illustrated in Figure 15.

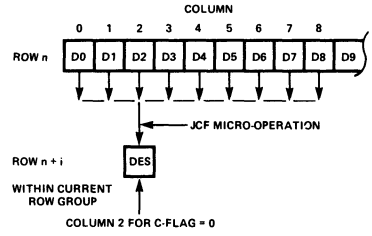


Figure 15. Special Use of the Conditional Jump Functions

```
/* INITIALIZATION SEQUENCE
   ZERO A, X, AND W */

   INIT:    CLR(A);
            CLR(X);
            CLR(W);

/* ZERO T AS TEMPORARY POINTER, WRITE W TO INTERRUPT STRUCTURE    */

            CLR(T);
            LMI(T);
            ILR(W) ROT;

/* SET S = (0), T = 1 FOR NEXT OPERATION   */

            LMI(T) FF1 RRM;
            ACM(AC) ;
            SDR(S);

/* SET B = (1), T = 2 FOR NEXT OPERATION   */

            LMI(T) FF1 RRM;
            ACM(AC);
            SDR(B) STC;          /* THIS SETS THE C FLAG TO INSURE
                                    A CORRECT JUMP TO XRTN   */
```

```
/* GET (2), JUMP TO XRTN TO SET E = (2), P = (E)  */

            LMI(T) RRM;
            ACM(AC)                 JCF (*,XRTN);

/* FETCH SEQUENCE & START OF MACRO-INSTRUCTION PROCESSING
   P IS ISSUED TO MAR AND INCREMENTED, MACRO-INSTRUCTION
   IS FETCHED AND TESTED BY JPX MICRO-OPERATOR. NOTE
   FETCH IS IN LOCATION 15 TO STROBE INTERRUPT ON ENTRY.  */

   FETCH:   LMI(P) FF1 RRM;

/* LOAD DISPLACEMENT AND TEST FOR ZERO USING Z FLAG  */

            LTM(AC) STZ K00FF;

/* SAVE DISPLACEMENT, TEST 4 BITS OF MACRO-OP.  TEST IS
   DELAYED TO ALLOW PIPELINE PROPAGATION.  ALSO C FLAG IS
   SET FOR LATER USE IN PSEUDO-SUBROUTINES.  */

            SDR(R9) STC             JPX(NA0,JREL,JIG,IMMD,DMRF,IMRF,IXMA,IXMB,IND,
                                        INX,OTD,OTX,MVGP,SPFG,IRBM,NA15);

/* UNASSIGNED OP-CODE GROUPS- -NOPS FOR THIS VERSION  */

   NA0:     NO.OP                  JZR(FETCH);
   NA15:    NO.OP                  JZR(FETCH);

/* IMMEDIATE GROUP OF MACRO-INSTRUCTIONS- -TEST FOR LONG OR SHORT
   FORM- -D IS IN AC AND R9- -ADJUST AC BY -128  */

   IMMD:    LMI(AC) KFF80          JZF(IMML,IMMS);

/* LONG FORM:  FETCH NEXT WORD TO AC  */

   IMML:    LMI(P) FF1 RRM;
            ACM(AC)                JRL(ILGA,ILPX,NAI1,NAI2);

/* SHORT FORM:  NO PROCESSING NEEDED  */

   IMMS:    NO.OP                  JRL(ILGA,ILPX,NAI1,NAI2);

/* PREPROCESSING FOR ARITHMETIC AND LOGIC ROUTINES?  NONE NEEDED  */

   ILGA:    NO.OP                  JLL(NDA,ODA,XDA,ADA);
   ILPX:    NO.OP                  JLL(LDA,LDX,PDS,ADX)

/* NOTE:  NAI1 AND NAI2 ARE NON-VALID INSTRUCTIONS!!  THEY ARE
   MADE INTO NO-OPS IN THIS VERSION OF THE MACRO-MACHINE  */

   NAI1:    NO.OP                  JZR(FETCH);
   NAI2:    NO.OP                  JZR(FETCH);

/* BASIC ARITHMETIC AND LOGIC PROCESSING- -UPDATE C FF OF MACRO-
   MACHINE FOR ADA- -TOGGLE IT ON CARRY FROM ADA  */

   ADA:     ADR(A);
   ADA1:    NO.OP                  JFL(NCY,SCY);
   NCY:     NO.OP                  JZR(FETCH);
   SCY:     LMI(W) K8000           JZR(FETCH);

/* LOGICALS  */

   NDA:     ANR(A)                 JZR(FETCH);
   ODA:     ORR(A)                 JZR(FETCH);
   XDA:     CMR(AC);
            XNR(A)                 JZR(FETCH);
```

```
/* LDA AND LDX OPERATIONS */

   LDA:    SDR(A)                    JZR(FETCH);
   LDX:    SDR(X)                    JZR(FETCH);

/* STACK PUSH- -ADVANCE STACK POINTER TO NEXT LOCATION (FOR THE
   REVERSE DIRECTION STACK- -A DECREMENT OF S), THEN WRITE */

   PDS:    DSM(S);
   PDS1:   LMI(S) RWM                JZR(FETCH);

/* ADX - SHARES CODE FOR ADA - ALSO TOGGLES C FF OF MACRO MACHINE */

   ADX:    ADR(X)                    JMP(ADA1);

/* MEMORY REFERENCE INSTRUCTION GROUPS
   DIRECT- -GET B+D INTO AC- -ALSO R9 */

   DMRF:   ILR(B);
           ALR(R9)                   JRL(MRV1,MRV2,MRAD,STPG);
/* INDIRECT-ABSOLUTE- -GET (B+D) INTO AC- -C FLAG USED FOR PSEUDO-SUBROUTINE */

   IMRF:   ILR(B);
   IMRF1:  ALR(R9);
           LMI(R9) RRM               JCF(MADD,MLOAD);
   MLOAD:  ACM(AC)                   JRL(MRV1,MRV2,MRAD,STPG);

/* NOTE: MADD WILL BE USED FOR OTHER INDIRECT OPERATIONS WHERE
   B, X, ETC. HAS BEEN LOADED TO R8 */

   MADD:   ACM(AC);
           ALR(R8)                   JRL(MRV1,MRV2,MRAD,STPG);
/* INDIRECT INDEXED ABSOLUTE - CLEAR C FLAG, MOVE X TO R8 */

   IXMA:   ILR(X) STC;
           SDR(R8);

/* NOTING THAT ASSIGNMENT RULES WOULD NOT ALLOW THE DESIRED
   JUMP TO IMRF UNLESS IXMA+1 WERE IN ROW ZERO- -AN EXTRA STATE
   IS ADDED HERE */

   IXMA2:  ILR(B)                    JMP(IMRF1);

/* INDIRECT INDEXED RELATIVE - CLEAR C FLAG, PUT B+X IN R8 */

   IXMB:   ILR(X) STC;
           SDR(R8);
           ILR(B);
           ADR(R8)                   JMP(IMRF);

/* INDIRECT RELATIVE (TO B) - CLEAR C FLAG, PUT B IN R8 */

   IRBM:   ILR(B);

/* AGAIN ASSIGNMENT RULES PREVENT JUMPING TO IXMA+1 UNLESS IT IS
   LOCATED IN ROW ZERO- -PLACEMENT THERE COULD FREE TWO WORDS */

           SDR(R8)                   JMP(IXMA2);

/* THE FOLLOWING PROCEDURES IMPLEMENT THE BASIC PREPROCESSING FOR
   VALUE AND ADDRESS LOADING.

   VALUE-GROUP 1: GET (AC) IN AC */

   MRV1:   LMI(AC) RRM;
           ACM(AC)                   JLL(NDA,ODA,XDA,ADA);
```

```
/* VALUE GROUP 2 */

   MRV2:   LMI(AC) RRM;
           ACM(AC)                    JLL(LDA,LDX,PDS,ADX);

/* MRAD GROUP INCLUDES ADDRESS LOADS AND SUBTRACT FROM A */

   MRAD:   NO.OP                      JLL(LAA,LAX,PAS,ISDA);

   LAA:    SDR(A)                     JZR(FETCH);
   LAX:    SDR(X)                     JZR(FETCH);
   PAS:    DSM(S)                     JMP(PDS1);

/* FOR SUBTRACT, ADD 1'S COMPLEMENT PLUS 1 */

   ISDA:   LMI(AC) RRM;
           LCM(AC);
           ADR(A) FF1                 JMP(ADA1);

/* STPG GROUP INCLUDES STORES AND SUBTRACT FROM X */

   STPG:   LMI(AC)                    JLL(SAM,SXM,PSM,SDX);

   SAM:    ILR(A) RWM                 JZR(FETCH);
   SXM:    ILR(X) RWM                 JZR(FETCH);

/* POP STACK TO MEMORY – SAVE ADDRESS, POP STACK */

   PSM:    SDR(T);
           LMI(S) FF1 RRM;
           ACM(AC);
           LMI(T) RWM                 JZR(FETCH);

/* SUBTRACT FROM X */

   SDX:    LMI(AC) RRM;
           LCM(AC);
           ADR(X) FF1                 JMP(ADA1);
```

Thus the initialization procedure requires 16 words of microcode, the fetch sequence 3, and the memory reference and immediate groups use a total of 57 words. In addition, two dummy locations (NAI1 and NAI2) are needed for unassigned macro-operation codes.

Sample execution times for some of the instructions may be estimated by counting the number of micro-instructions in the sequences and the number of read and write memory cycles. Allowing 150 nsec for each micro-instruction, and 400 nsec for each memory cycle, some representative execution times would be as shown in Table VII.

Table VII. Representative Execution Times

INSTRUCTION	MICROCYCLES	READ CYCLES	WRITE CYCLES	EXECUTION TIME
ADA, direct	10	2		2.3 μS
ADI, short	9	1		1.75 μS
LDA	8	2		2.0 μS
LAI, short	7	1		1.45 μS
LDA, indirect index relative	15	3		3.45 μS

JUMP GROUP

The next section shows the realization of the jump group instructions. Two basic classes, a jump relative to the program counter and an indirect jump through a table stored at the beginning of the program are represented. Conditional jumps include A>0, A≥0, A=0, A≠0, A≤0, A<0, X≠A, X>0, X≤A, C=0 and C≠0.

In addition, two classes of subroutine calls are provided; a local call which pushes P onto the stack, and jumps relative to E, and a global subroutine call which stores the W, B, E, and P registers on stack and computes new values for E, the program base register, and P. Also, included in this section of microcode is the operation that pushes both A and X onto the stack.

Table VIII shows the opcode assignments for the various jump operations implemented. Except for the conditional jumps, X>A, X≤A, X=A and X≠A which share a common subroutine and exit via a JLL jump, the opcode values were assigned arbitrarily.

A flowchart representing the jump coding is shown in Figure 16. During the microcoding of the sequence, two methods were evaluated. One used the JRL, JLL sequence of testing 2 bits of macro-operation code at a time, while the one actually selected uses a JPR macro-operation. The JPR test selected uses no more code than the JRL, JLL sequence method, and executes more rapidly. At one point (for the X=A, X≠A, X>A, X≤A tests), code is shared as if it were part of a subroutine, then a JLL instruction is used to resolve the exit. This method is another example of a pseudo-subroutine that saves microprogram memory. Use of this technique does put a constraint on the assignment of macro-operation codes.

Table VIII. Jump Instruction Group

MNEMONIC	FUNCTION	RELATIVE M	RELATIVE O	INDIRECT M	INDIRECT O
JRU, JIU	Jump unconditional	0001	0000	0010	0000
JRGE, JIGE	Jump if A≥0	0001	0001	0010	0001
JRLT, JILT	Jump if A<0	0001	0010	0010	0010
JRXG,JIXG	Jump if X>A	0001	0011	0010	0011
JREZ, JIEZ	Jump if A=0	0001	0100	0010	0100
JRNZ, JINZ	Jump if A≠0	0001	0101	0010	0101
JRCZ, JICZ	Jump if C=0	0001	0110	0010	0110
JRXL, JIXL	Jump if X≤A	0001	0111	0010	0111
JRLE, JILE	Jump if A≤0	0001	1000	0010	1000
JRGT, JIGT	Jump if A>0	0001	1001	0010	1001
JRCN, JICN	Jump if C≠0	0001	1010	0010	1010
JRXE,JIXE	Jump if X=A	0001	1011	0010	1011
CVS	Call subroutine, push W, B, E, P	N.A.		0010	1100
PHAX	Push A, X onto stack	0001	1101	0010	1101
CLS	Call subroutine, push P	N.A.		0010	1110
JRXN, JIXN	Jump if X≠A	0001	1111	0010	1111

Subroutine calls

Local: Push P to stack
P=E+(E+D)

Value: Push W, B, E, P to stack
E=E+(E+D)
P=E'+(E') where E'=E+(E+D)

Unconditional and conditional jumps

Relative: P=P+D' where D'=D−128
Indirect: P=E+(E+D)

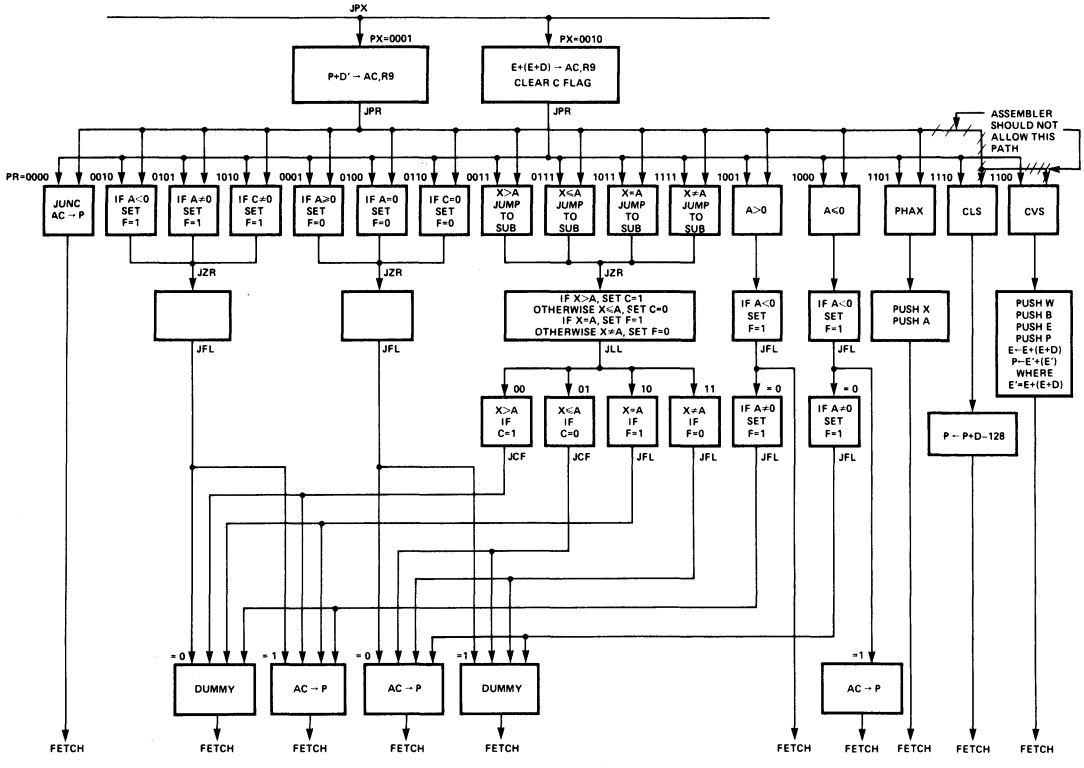


Figure 16. Jump Group Flowchart

```
/* JUMP GROUPS- -USE JPR MICRO-OPERATION TO RESOLVE CONDITION SELECTION
   DESTINATION ADDRESS IS COMPUTED FIRST- -PLACED IN AC AND R9
   JUMP RELATIVE TO P- -ADDRESS=P+D-128 */

   JREL:    ILR(P);
   JRDR:    LMI(AC) KFF80;
            ALR(R9)                    JPR(JUNC,JAGE,JALT,JXGA,JAEQ,JANE,JCEZ,JXLA,
                                           JALE,JAGT,JCNZ,JXEA,CPSS,PXA,CLOP,JXNA);
/* JUMP INDIRECT – GET E+(E+D) IN AC AND R9 */

   JIG:     ILR(E);
            ADR(R9);
            LMI(R9) RRM;
            AMA(AC);
            SDR(R9)                    JPR(JUNC,JAGE,JALT,JXGA,JAEQ,JANE,JCEZ,JXLA,
                                           JALE,JAGT,JCNZ,JXEA,CPSS,PXA,CLOP,JXNA);

/* UNCONDITIONAL JUMP */

   JUNC:    SDR(P)                     JZR(FETCH);

/* TESTS FOR A.GE.0, ETC. */

   JAGE:    TZR(A) K8000 INH           JMP(TTRU);
   JALT:    TZR(A) K8000 INH           JMP(TFAL);
   JAEQ:    TZR(A)                     JMP(TTRU);
   JANE:    TZR(A)                     JMP(TFAL);
```

```
JAGT:   TZR(A) K8000 INH;
        TZR(A)                          JFL(APRE,ANPE);

APRE:   NO.OP                           JFL(JNT2,JTR2);
ANPE:   NO.OP                           JZR(FETCH);

JALE:   TZR(A) K8000 INH;
        TZR(A)                          JFL(APE2,AN2);

APE2:   NO.OP                           JFL(JTR1,JNT1);
AN2:    SDR(P)                          JZR(FETCH);

/* TESTS OF C FLIP-FLOP (HIGH ORDER BIT OF W) */

JCEZ:   TZR(W) K8000 INH                JMP(TTRU);
JCNZ:   TZR(W) K8000 INH                JMP(TFAL);

/* TEST EXECUTION FOR ABOVE TESTS – ROW ZERO USED */

TTRU:   NO.OP                           JFL(JTR1,JNT1);

JTR1:   SDR(P)                          JZR(FETCH);
JNT1:   NO.OP                           JZR(FETCH);

TFAL:   NO.OP                           JFL(JNT2,JTR2,);

JNT2:   NO.OP                           JZR(FETCH);
JTR2:   SDR(P)                          JZR(FETCH);

/* TESTS FOR X.GT.A, X.LE.A, X.EQ.A, X.NE.A– –SHARED PSEUDO-
   SUBROUTINE USES JLL FOR AN EXIT TEST– –ROUTINE ENTRY IN ROW 0
   C FLAG IS SET FOR X.GT.A, FL TEST FOR X.EQ.A */

JXGA:   ILR(X)                          JMP(XATS);
JXLA:   ILR(X)                          JMP(XATS);
JXEA:   ILR(X)                          JMP(XATS);
JXNA:   ILR(X)                          JMP(XATS);

/* SAVE X AT T, FETCH AND COMPLEMENT A */

XATS:   SDR(T);
        ILR(A) STC;     /* CLEAR C FLAG */
        CMA(AC);

/* ADD HOB'S OF A' AND X – CARRY MEANS X NEG., A.GE.0 */

        ADR(T) K8000;

/* EXECUTE PREVIOUS TEST, SET UP TO TEST HOB OF RESULT– –IF 1,
   THE SIGNS OF A AND X WERE THE SAME */

        TZR(T) K8000 INH                JFL(TFEQ,TXNG);

/* TXNG IMPLIES X NEG AND A.GE.0– –I.E. X.NE.A AND X.LT.A– –DO A
   DUMMY OPERATION TO FORCE THE PROPER F FLAG */

TXNG:   ILR(A)                          JLL(JXGX,JXLX,JXEX,JXNX);

/* PERFORM A TEST ADDITION AND EXECUTE SIGN-EQUAL TEST
   C WILL BE SET IF SIGNS WERE THE SAME AND X.GT.A */

TFEQ:   ADR(T) STC K7FFF                JFL(SNEQ,SWEQ);

/* SNEQ IMPLIES SIGNS NOT EQUAL– –I.E. X.GE.0, A NEG– –X.GT.A */

SNEQ:   SDR(AC) STC;    /* DUMMY OP TO SET C FLAG */
        NO.OP                           JLL(JXGX,JXLX,JXEX,JXNX);
```

```
/* FOR SIGNS EQUAL, IF X=A RESULT WOULD BE 1111...1. INCREMENT
   WILL GENERATE A CARRY IF SO */

   SWEQ:    ILR(AC) FF1                    JLL(JXGX,JXLX,JXEX,JXNX);

/* EXECUTION OF JUMP TESTS */

   JXGX:    ILR(R9)                        JCF(JNT2,JTR2);
   JXLX:    ILR(R9)                        JCF(JTR1,JNT1);
   JXEX:    ILR(R9)                        JFL(JNT2,JTR2);
   JXNX:    ILR(R9)                        JFL(JTR1,JNT1);

/* SUBROUTINE CALLS
   CALL LOCAL AND PUSH W, B, E, P =CPSS
   CALL LOCAL AND PUSH P ONLY=CLOP
   CL FLAG IS USED FOR EXIT TEST AFTER PUSHING P */

   CPSS:    DSM(S);
            ILR(W);

            LMI(S) RWM;

   CPG2:    DSM(S);
            ILR(B);
            LMI(S) RWM;

            DSM(S);
            ILR(E);
            LMI(S) RWM;

            DSM(S);
            ILR(P);
   CLOP2:   LMI(S) RWM;

/* E+(E+D) INTO AC */

            ILR(R9)                        JCF(LRTN,XRTN);

   XRTN:    SDR(E);
            LMI(E) RRM;
            AMA(AC);

   LRTN:    SDR(P)                         JZR(FETCH);

   CLOP:    DSM(S);
            ILR(P) STC                     JMP(CLOP2);

/* PUSH INSTRUCTION */

   PXA:     DSM(S);
            ILR(X);
            LMI(S) RWM;

            DSM(S);
            ILR(A);
            LMI(S) RWM                     JZR(FETCH);
```

REGISTER MOVE AND SUBROUTINE RETURN GROUP

In this section of code, the Register Move and Subroutine Return group instructions are implemented. Both groups share the same JPX entry point, 1100B. Table X shows the opcode values assigned to the macro-instructions.

To simplify the decoding for register selection (S, B, E or W) in the Register Move group, the two low order bits of the PR latch are used to modify the micro-instruction as it is strobed into the pipeline register. By tying the two PR latch outputs of the 3001 to the two low order bits of the CPE control field, a JCE jump function (which enables the PR

latch outputs) can be used to provide a wire OR of PR0, PR1 and F0, F1 (see Figure 17).

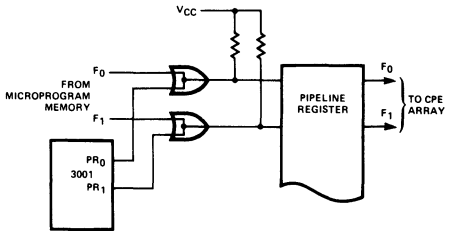


Figure 17. Wire-OR of PO_{0-1} and F_{0-1}

Thus, in the micro-instruction

```
SDR(R7)  JCE (MXRX)
```

the register group field F0–F3 is modified as shown in Table IX.

The microprogram sequence is shown in Figure 18.

Table X. Register Move and Subroutine Return Group

MNEMONIC	FUNCTION	M	O
RLS	Pop P	1100	1111
RVS	Pop P, E, B, W	1100	1101
RSA	Pop A, X, P, E, B, W	1100	1100
PPAX	Pop A, X	1100	1110
MSX	Move S to X, adjust	1100	0100
MBX	Move B to X, adjust	1100	0101
MEX	Move E to X, adjust	1100	0110
MWX	Move W to X, adjust	1100	0111
MXS	Move X to S, adjust	1100	0000
MXB	Move X to B, adjust	1100	0001
MXE	Move X to E, adjust	1100	0010
MXW	Move X to W, adjust	1100	0011
NO.OP	Nothing implemented	1100	10XX

Table IX. Register Group Field F0–F3 Modification

MICROPROGRAM MEMORY OUTPUT (F0–F3)	PR LATCH OUTPUT	RESULT STORED IN PIPELINE REGISTER	SELECTED REGISTER
0111	00	0100	S
0111	01	0101	B
0111	10	0110	E
0111	11	0111	W

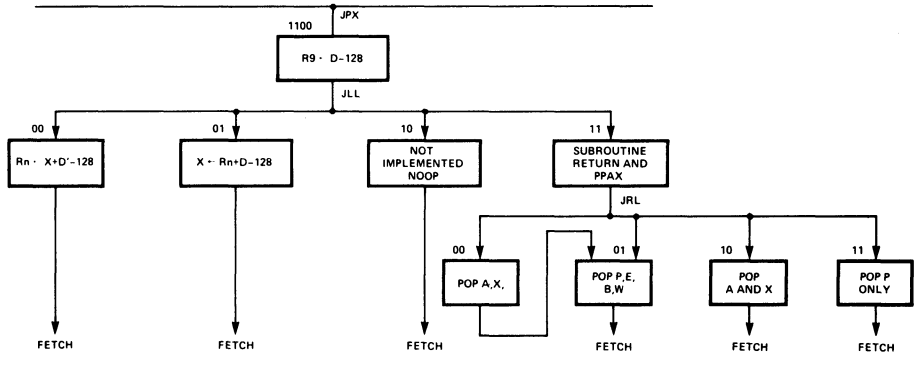


Figure 18. Register Move and Subroutine Return Group Flowchart

```
/* MOVE GROUP OF INSTRUCTIONS- -USES JCE TO SELECT REGISTER- -NOTE
   THAT REGISTER ASSIGNMENT BECOMES IMPORTANT
   FIRST MODIFY D TO GET D-128 */

   MVGP:   LMI(R9) KFF80                JLL(MVXR,MVRX,MOD,PGRP);

/* MOVE X TO REG. - GET X, MODIFY BY D'=D-128 */

   MVXR:   ILR(X);
           ALR(R9);
           SDR(R7)                      JCE(MXRX);      /* REGISTER OVERRIDE */
   MXRX:   NO.OP                        JZR(FETCH);

/* MOVE REG TO X - FETCH REG USING JCE OVERRIDE */

   MVRX:   ILR(R7)                      JCE(MRXX);
   MRXX:   ALR(R9)                      JMP(LDX);

/* MOD NOT IMPLEMENTED IN THIS VERSION */

   MOD:    NO.OP                        JZR(FETCH);

/* ADJUST STACK AND RETURN GROUP
   PPAL- -POPS A, X, P, E, B, AND W
   PPRA- -POPS P, E, B, AND W
   PPAX- -POPS ONLY A AND X
   POPP- -POPS ONLY P */

   PGRP:   ILR(R9);
           ADR(S)                       JRL(PPAL,PPRA,PPAX,POPP);

   PPAL:   LMI(S) FF1 RRM;
           ACM(AC);
           SDR(A);

           LMI(S) FF1 RRM;
           ACM(AC)                      JCF(PAXE,PAXC);
   PAXC:   SDR(X);

   PPRA:   LMI(S) FF1 RRM;
           ACM(AC);
           SDR(P);

           LMI(S) FF1 RRM;
           ACM(AC);
           SDR(E);

           LMI(S) FF1 RRM;
           ACM(AC);
           SDR(B);

           LMI(S) FF1 RRM;
           ACM(AC);
           SDR(W);

/* RESTORE INTERRUPT STRUCTURE */

           CLR(T);
           LMI(T) ROT                   JZR(FETCH);

   PAXE:   SDR(X)                       JZR(FETCH);

   PPAX:   ILR(AC) STC                  JMP(PPAL);

   POPP:   LMI(S) FF1 RRM;
           ACM(AC)                      JMP(JUNC);
```

SPECIAL FUNCTION GROUP

The JPX entry point 1101B is used as an entry point for the special function groups which include byte load and store, register manipulation, and the absolute subroutine call and increment and skip if zero instructions. Table XI lists the opcode values assigned to the instructions. A flowchart of the sequences is shown in Figure 19.

In order to execute a byte load or store operation efficiently, a byte swap capability (which exchanges the high and low order byte positions) is necessary. By wiring the data outputs of the high order byte to the I inputs of the low order byte, and the low order outputs to the high order I inputs, a byte swap operation can be performed (see Figure 20).

Note that with the configuration shown in Figure 20, a byte swap can be performed on either a memory word or the AC register of the CPE array by reading data in on the I-Bus inputs while performing a memory read or enabling the D-Bus, respectively.

Table XI. Special Function Groups

MNEMONIC	FUNCTION	M	O
LBA	Load byte absolute	1101	0000
LBR	Load byte relative	1101	0100
SBA	Store byte absolute	1101	1000
SBR	Store byte relative	1101	1100
RAR	Rotate A right, include CFF	1101	0001
RAX	Rotate A and X right, include CFF	1101	0101
SAX	Shift A and X right, preserve sign	1101	1001
SAL	Shift A left, fill with zeros	1101	1101
ISZ	Increment and skif ip zero	1101	XX10
CAS	Call absolute, push P, E, W, B P ← (D)	1101	XX11

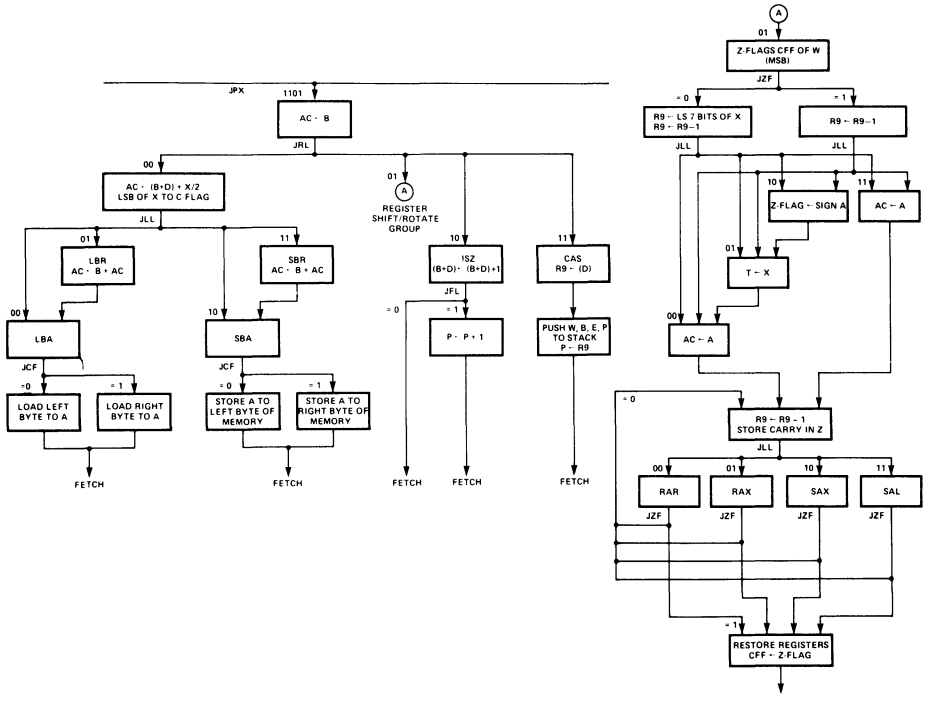


Figure 19. Special Function Groups Flowchart

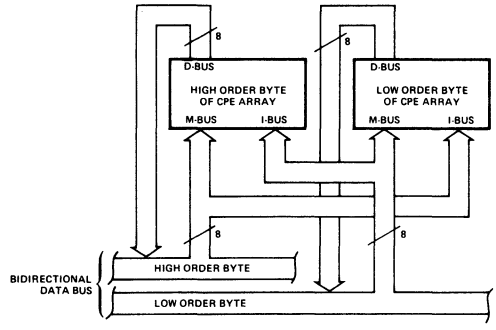


Figure 20. I-Bus Wired for Byte Swap

```
/* SPECIAL FUNCTION GROUP
   BYTE OPERATORS- -ADDR=(B+D)+B+X/2 OR (B+D)+X/2
   CALL TO (D) AND PUSH ALL
   SHIFT AND ROTATE GROUP
   INCREMENT AND SKIP
   FETCH B JUST IN CASE */

   SPFG:    ILR(B)                        JRL(BYTE,RSGP,SCJG,ISJG);

/* BYTE GROUP- -COMPUTE ADDR, STORE B IN CASE NEEDED */

   BYTE:    SDR(R8);
            ADR(R9);
            ILR(X);
            SRA(AC) STC;
            LMI(R9) RRM;
            ACM(AC)                       JLL(LBYA,LBYR,SBYA,SBYR);

   LBYR:    ALR(R8);
   LBYA:    LMI(AC) RRM                   JCF(LBYT,RBYT);
   LBYT:    LDI(AC) FF1 K00FF             JMP(DBIA);
   RBYT:    LTM(AC) K00FF;
   DBIA:    SDR(A)                        JZR(FETCH);

   SBYR:    ALR(R8);
   SBYA:    LMI(AC);                      /* LOAD MAR FOR LATER USE */
            ILR(A);
            TZR(AC) K00FF RRM             JCF(STLB,STRB);
   STRB:    LTM(T) KFF00;
   SRB1:    ALR(T) RWM                    JZR(FETCH);

   STLB:    LTM(T) K00FF;
            LDI(AC) FF1 CNB               JMP(SRB1);

/* ROTATE GROUP
   ROTATE A WITH C- -ROTATE A AND X WITH C- -SHIFT A, X RIGHT, FILL
   WITH SIGN- -SHIFT A LEFT, FILL WITH ZEROES
```

```
   AT ENTRY, Z FLAG IS ZERO IF D=0.  DUE TO PIPELINED OPERATION, IT IS
   THIS CONDITION THAT IS TESTED BY THE FIRST JZF  */

   RSGP:   TZR(W) STZ K8000 INH        JZF(SZDS,SNZD);
   SZDS:   ILR(X);
           SDR(R9) FF0 K007F           JLL(RACI, RAXI,SAXI,SLZI);
   SNZD:   DSM(R9)                     JLL(RACI,RAXI,SAXI,SLZI);

   RACI:   ILR(A)                      JMP(RUNR);
   RAXI:   ILR(X);
           SDR(T)                      JMP(RACI);
   SAXI:   TZR(A) STZ K8000 INH        JMP(RAXI);
   SLZI:   ILR(A)                      JMP(RUNR);

/* MAIN ROTATION LOOP  */

   RUNR:   DSM(R9) STC                 JLL(RACR,RAXR,SAXR,SLZR);

   RACR:   SRA(AC) FFZ STZ             JFL(RSEX,RUNR);
   RAXR:   SRA(AC) FFZ STZ;
           SRA(T) FFZ STZ              JCF(RSEX,RUNR);
   SAXR:   SRA(AC) FFZ STC;
           SRA(T) FFC                  JCF(RSEX,RUNR);
   SLZR:   ADR(AC) STZ                 JFL(RSEX,RUNR);

   RSEX:   SDR(A)                      JLL(RACF,RAXF,SAXF,SLZF);

   RACF:   TZR(W) K7FFF                JZF(SNCF,SSCF);
   SNCF:   NO.OP                       JZR(FETCH);
   SSCF:   LMI(W) K8000                JZR(FETCH);
   RAXF:   ILR(T);
   RXF1:   SDR(X)                      JMP(RACF);
   SAXF:   ILR(T)                      JMP(RXF1);
   SLZF:   TZR(W) K7FFF                JZF(SNCF,SSCF);

/* SPECIAL CALL AND JUMP GROUP- -CURRENTLY CONTAINS ONLY THE
   CALL TO (D) AND PUSH W,B,E,P- -ALL 4 OPCODES DO THE SAME THING  */

   SCJG:   LMI(R9) RRM;
           ACM(AC);
           SDR(R9)                     JMP(CPSS);

/* INCREMENT AND SKIP GROUP- -AGAIN 4 OPCODES ARE USED FOR ONE
   INSTRUCTION- -LOCATION AT B+D IS INCREMENTED  */

   ISJG:   ALR(R9);
           LMI(R9) RMW;
           ACM(AC) FF1 RWM;
           NO.OP                       JFL(NOSK,SKIP);
   NOSK:   NO.OP                       JZR(FETCH);
   SKIP:   LMI(P) FF1                  JZR(FETCH);
```

INPUT/OUTPUT GROUP

In this section of code, the input/output instructions are implemented. In conjunction with the memory address register, the bus control field generates a Request Input or Request Output to select an I/O port and specify the operation to be performed. Table XII lists the opcode values assigned to the macro-instructions. The flowchart in Figure 21 shows the microcode sequence used.

Table XII. Input/Output Group

MNEMONIC	FUNCTION	M	O
IND	Input one word A ← (D)	1000	XXXX
OTD	Output one word (D) ← A	1001	XXXX
INX	Input one word A ← (X)	1010	XXXX
OTX	Output one word (X) ← A	1011	XXXX

INTERRUPTS

A basic means for microcoding interrupts when using the 3214 Interrupt Control Circuit involves forcing an alternate microprogram address which then leads to an interrupt handling routine. The interrupt handling routine interrogates the interrupt structure to determine the interrupting level. This level is rewritten to the interrupt structure to block further interrupts at the interrupting priority level or lower levels while enabling interrupts at higher levels.

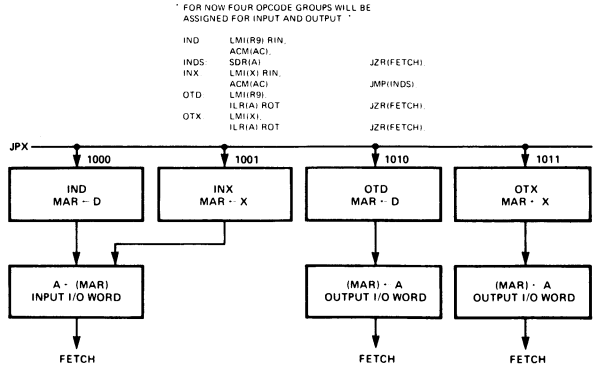
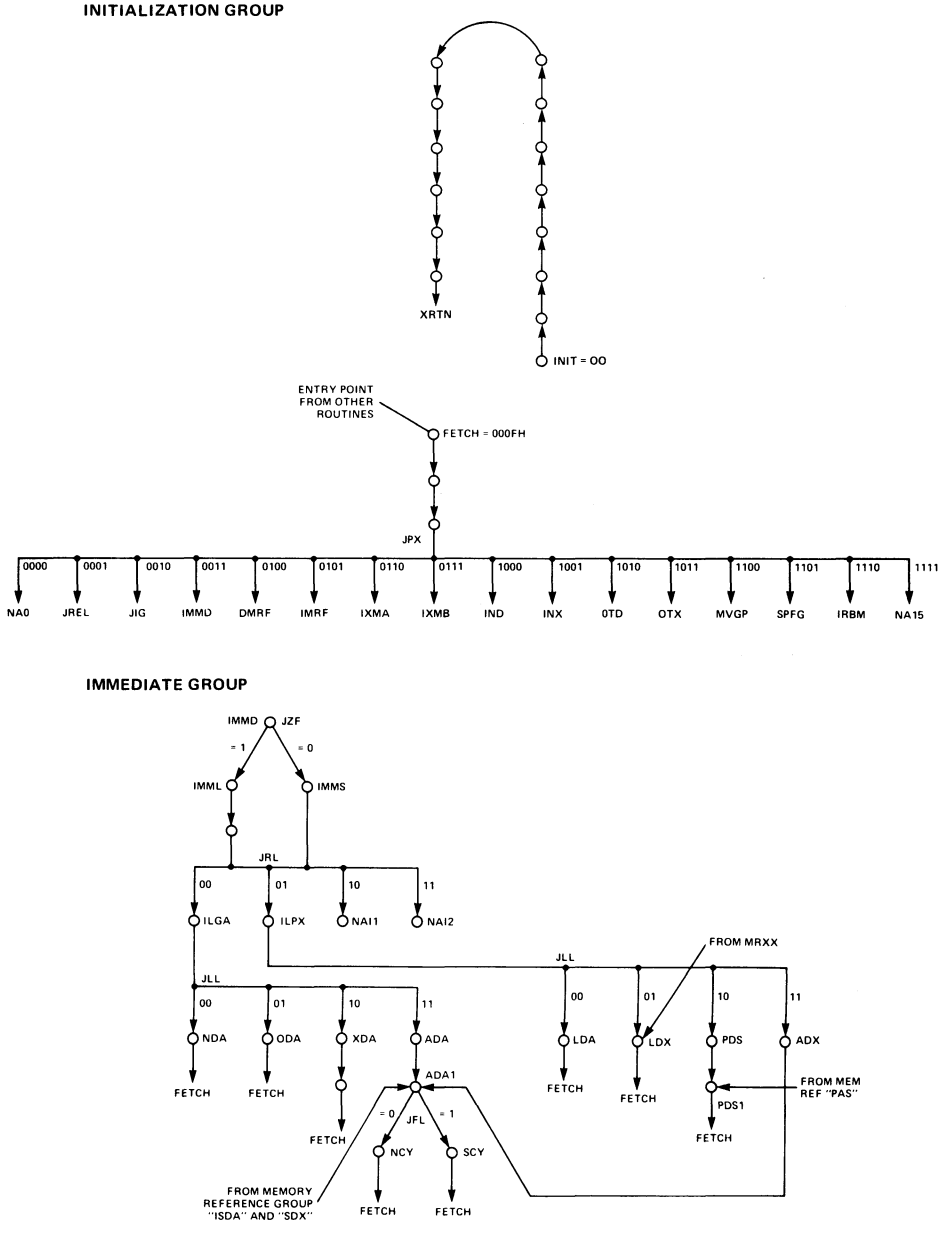


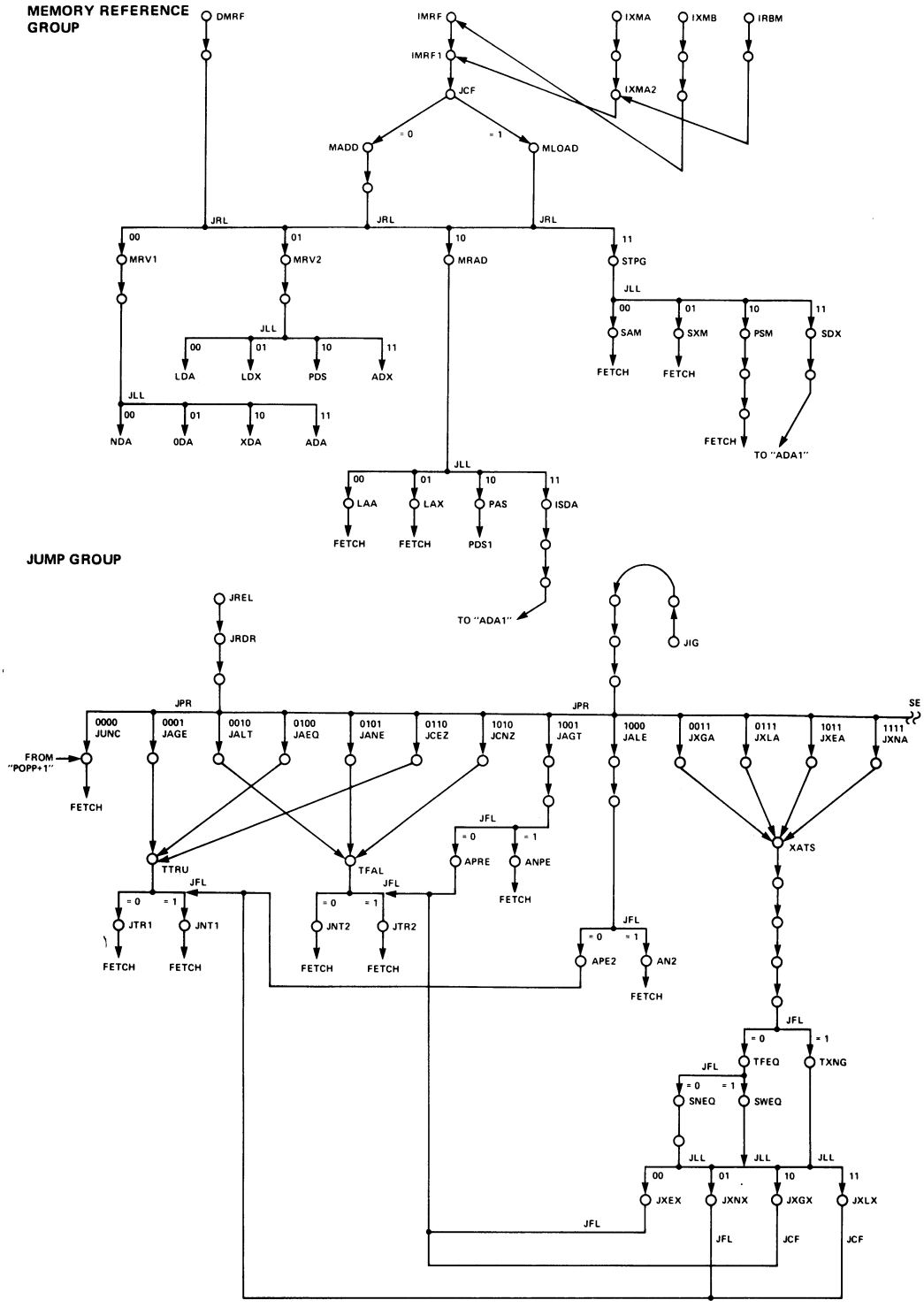
Figure 21. Input/Output Flowchart

```
/* INTERRUPT- -UTILIZED CALL ROUTINES FOR REGISTER SAVING
   I/O DEVICE #0 REPRESENTS EXTERNAL INTERRUPT STRUCTURE
   START BY PUSHING OLD VALUE OF STATUS */

   INTER:   DSM(S);
            ILR(W);
            LMI(S) RWM;

/* READ INTERRUPTING LEVEL FROM EXTERNAL STRUCTURE */

            CLR(T);
            LMI(T) RIN;
            LTM(AC) K00FF ROT; /* NOTE LEVEL REWRITTEN */

/* STORE PRIORITY IN W – SET C FLAG FOR PROPER LOADING OF REGISTERS */

            SDR(W) STC;

/* INTERRUPT ROUTINE STARTING ADDRESS IS COMPUTED IN R9 */

            LMI(W) RRM;
            ACM(AC);
            SDR(R9)                    JMP(CPG2);
```

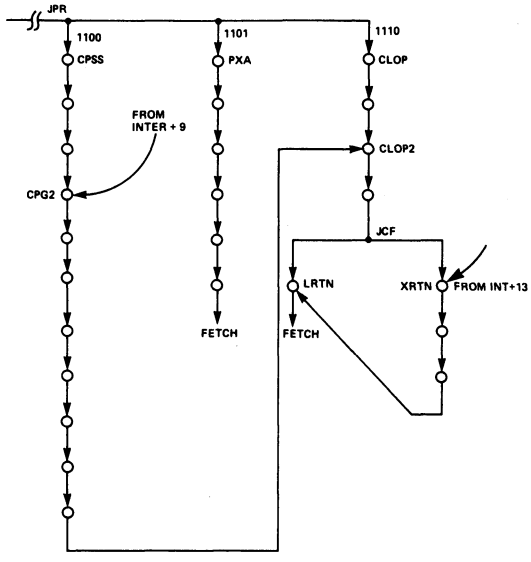
Microprogram Memory Assignment

Having written the actual code with minimal regard to memory assignment, the actual assignment to ROM must be performed. To assist in this function, a complete state (i.e., microcode instruction) flowchart should be prepared. Each machine state is represented by a dot in the state diagrams shown below. Conditional jumps should be labeled as to type and condition corresponding to each destination. This information will be necessary when performing an assignment. No other information is needed on the flowchart, but it is quite useful to show any symbolic label that may be associated with a state.

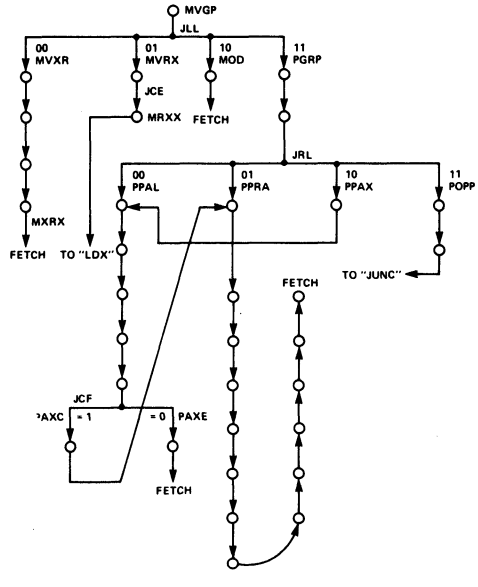



MEMORY REFERENCE GROUP
DMRF
IMRF
IMRF1
JCF
IXMA
IXMB
IRBM
IXMA2
MADD
MLOAD
= 0
= 1
JRL
00
01
10
11
MRV1
MRV2
MRAD
STPG
JLL
LDA
LDX
PDS
ADX
NDA
0DA
XDA
ADA
SAM
SXM
PSM
SDX
FETCH
TO "ADA1"
LAA
LAX
PAS
ISDA
PDS1
JUMP GROUP
JREL
JRDR
JIG
JPR
SE
0000 JUNC
0001 JAGE
0010 JALT
0100 JAEQ
0101 JANE
0110 JCE2
1010 JCNZ
1001 JAGT
1000 JALE
0011 JXGA
0111 JXLA
1011 JXEA
1111 JXNA
FROM "POPP+1"
TTRU
TFAL
JFL
APRE
ANPE
XATS
JTR1
JNT1
JNT2
JTR2
APE2
AN2
TFEQ
TXNG
SNEQ
SWEQ
JXEX
JXNX
JXGX
JXLX

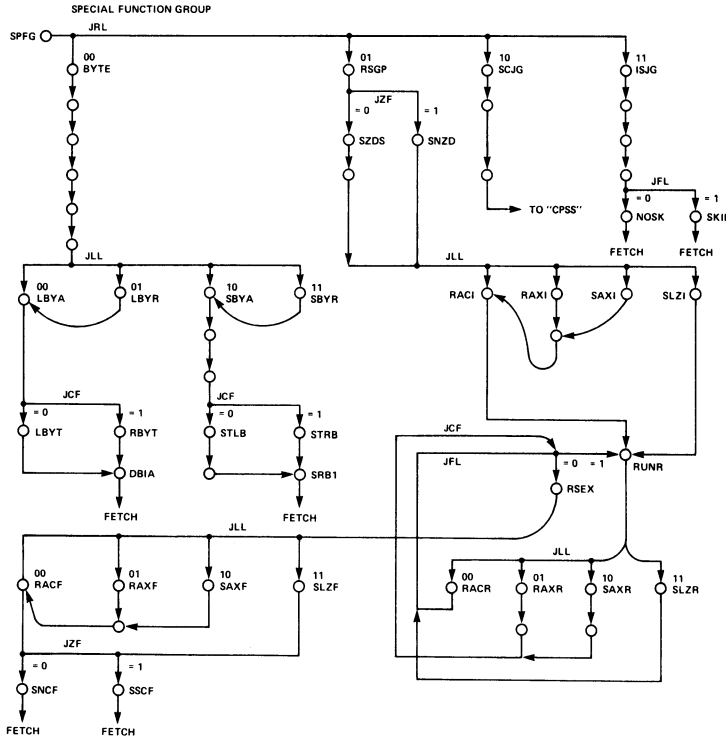
JUMP GROUP (CONTINUED)



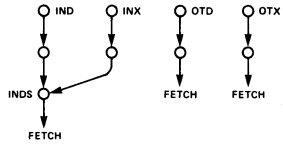
MOVE GROUP



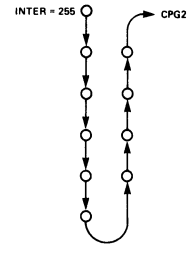
SPECIAL FUNCTION GROUP



I/O GROUP



INTERRUPT SERVICE ROUTINE



Once all of the state diagrams have been prepared, a number of steps may be followed to simplify the assignment procedure. First, the basic hardware characteristics dictate that INIT, FETCH, and INTER be located in microprogram memory locations 0, 15, and 255 (decimal), respectively. Then, note that each conditional jump has a limited range. As a result, when several conditional jumps follow one another in sequence, all may have to be located within a restricted range in microprogram memory. For JCF, JZF, JLL and JRL micro-instructions, the calling instruction must be in the same block of eight rows as the destinations.

To do the best assignment, the most restricted set of micro-instructions should be assigned first. The most restricted groups of micro-instructions are usually associated with clusters of conditional jumps which must be located within a given range of memory. It is therefore very useful to catalog all such clusters of conditional jumps. Table XIII lists the clusters associated with this machine. In each case the conditional jump is identified by the jump micro-operation and the first of its destinations. Thus in Table XIII the symbol JRL(MRV1) really refers to the code JRL(MRV1, MRV2, MRAD, STPG). For this machine, there are only five clusters.

Table XIII. Conditional Jump Clusters

1.	JPX (NAO) JRL (ILGA), JRL (BYTE) JLL (NDA), JLL (LDA), JLL (MVXR), JLL (RACI) JZF (IMML), JZF (SZDS)
2.	JRL (MRV1) JLL (SAM), JLL (LAA) JCF (MADD)
3.	JLL (JXEX) JFL (JTR1), JCF (JNT2)
4.	JRL (PPAL) JCF (PAXC)
5.	JLL (RACR), JLL (RACF) JCF (RSEX) JZF (SNCF)

An examination of the flowcharts indicates that a simpler code might result if clusters one and five were combined because of the coupling between JLL(RACI) of cluster one and the JCF(RSEX) of cluster five. The combination of these two clusters represents the greatest degree of restriction, as within the same block of rows there would be one JPX, six JLL, two JRL, one JCF and three JZF micro-operations. In addition, the JLL(MVXR) executes a JCE jump which uses an additional location within the JLL destination columns. However, the basic jump micro-operation characteristics do allow all of these conditional jumps to be placed within one block of eight rows.

To retain row zero, the conditional jumps of clusters one and five are placed in the last eight rows of the microprogram memory. In addition to the destinations, space must be reserved for the "calling" micro-instructions for each of the conditional jumps listed in the clusters.

Chart 1 shows an assignment of the conditional jumps of clusters one and five, together with some of the immediately related states. For the assignment procedure, a form like that of Chart 1 is used to show which microprogram memory locations are occupied and which are available. The format also aids visualization of valid jump micro-operations. As each state is assigned to its location in micro memory, the corresponding position on the state diagram is marked to show assignment. In this way, unassigned states are easily located on the state diagrams.

The information placed in the memory maps includes the state label or, for strings of states with no assigned label, the label of the nearest previously labeled state plus information to indicate how far from that labeled state the present state is. For example, INIT+2 is the second state after INIT.

The state assignment can proceed, with conditional jumps and short unconditional sequences being assigned before long unconditional sequences. Chart 2 shows the state assignment at a point when all states except those between INIT and FETCH, those between PPRA and FETCH, and those associated with IND, INX, OTD and OTX have been assigned.

For those states which have only one calling state (i.e., a state which has only one state jumping to it with a non-conditional jump) and only one target state (i.e., it makes a non-conditional jump to another state), two hexadecimal numbers are also written on the memory map. The number in the lower left-hand corner is the address of the calling state (first hex digit is the row, second hex digit is the column), and the number in the lower right-hand corner is the address of the target state. This information will tell the designer at a glance which states can be easily moved in the process of memory assignment, and to which locations they can be moved. For instance, a state with its calling state and target state in the same row (or column) can be moved anywhere in that row (or column), and a

state with its target state in the row zero can be moved anywhere in the same row or column as its calling state.

As an example of how this information can be used, note that in Chart 2 state RAXI+1 has been assigned to location 090H. However, when the INIT sequence is assigned, it becomes convenient to locate INIT+1 somewhere in column 0. Since there are no available spaces in column 0, the designer notes that state RAXI+1 has both its calling and target states in row 9, and so RAXI+1 can be moved anywhere in row 9. In Chart 3, RAXI+1 has been reassigned to location 098H, and INIT+1 has been assigned to location 090H. This moving process will typically be frequently necessary in the assignment procedure, and thus it is quite useful to have this information right on the working memory map.

The final state assignments consist mostly of the long unconditional sequences. Row zero locations may then be used freely. In those cases where extra states were used to avoid the use of row zero locations, the assignment may be reconsidered. For this machine, the operations IND, INX, OTD and OTX were rewritten to utilize row zero locations. Figure 22 shows the revised flow diagram for these four operations.

The final assignment is as shown in Chart 3. Two locations remain.

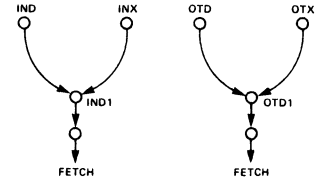


Figure 22. IND, INX, OTD and OTX Revised Flow Diagram

```
/* INPUT AND OUTPUT- -CURRENT VERSION DOES NOT DECODE INTO
   SUBGROUPS- -ALSO ROW ZERO IS USED TO SAVE CODE */

IND:    LMI(R9) RIN;
IND1:   ACM(AC);
        SDR(A)                  JZR(FETCH);
INX:    LMI(X) RIN              JMP(IND1);
OTD:    LMI(R9);
OTD1:   ILR(A) ROT              JZR(FETCH);
OTX:    LMI(X)                  JMP(OTD1);
```

CONCLUSION

In the central processor design example described above, the final definition of the central processor macro-instruction set evolved as the microprograms were being implemented. In many instances, it was necessary to modify the macro-instruction opcode assignment in order to take full advantage of the capabilities of the Series 3000 architecture. Macro-instruction operations were also redefined to add more flexibility as microprogramming techniques improved.

The microprograms were implemented without regard to memory assignment except in cases where code sharing between micro-instruction opcode assignments were critical. Actual assignment of the micro-instructions to memory involved a very small portion of the design cycle. The 3001 MCU's ability to decode macro-instruction opcodes and large repertoire of conditional and unconditional jump operations resulted in both efficient microprograms and complete memory utilization. Only two memory locations remained unused after the microcoding was complete.

The central processor developed in this application note is used as a design example only, and therefore does not represent a complete central processor or an instruction set designed for a specific application. However, because of the microprogrammability of the Series 3000 family, the same basic organization can be tailored to a wide range of operating environments from I/O processing to data processing and dedicated arithmetic computation.

Chart 1

			JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	f,c,z=1	JLL COLUMN RESTRICT						JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	f,c,z=1	JRL COLUMN RESTRICT			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	INIT															FETCH
01																
02																
03																
04																
05																
06																
07																
08	NA0	JREL	JIG	IMMD	DMRF	IMRF	IXMA	IXMB	IND	INX	OTD	OTX	MVGP	SPFG	IRBM	NA15
09	RAXI+1 95 94		RSEX	RUNR	RACI	RAXI	SAXI	SLZI		SZDS+1 9A	SZDS	SNZD				
0A	RAXR+1 A4	SAXR+1 A6	SNCF	SSCF	RACR	RAXR	SAXR	SLZR								
0B	RXFI	IMML+1 B2	IMML	IMMS	RACF	RAXF	SAXF	SLZF								
0C					MVXR	MVRX	MOD	PGRP					BYTE	RSGP	SCJG	ISJG
0D					NDA	ODA	XDA	ADA			PAXC	PAXE	PPAL	PPRA	PPAX	POPP
0E					LDA	LDX	PDS	ADX					ILGA	ILPX	NAI1	NAI2
0F						MRXX										INTER

Chart 2

			JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	f,c,z=1	JLL COLUMN RESTRICT						JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	f,c,z=1	JRL COLUMN RESTRICT			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	INIT	JAGT+1	JTR1	JNT1	TFAL	TTRU		JALE+1		MADD+1	IMRF1+1	XATS		CLOP2		FETCH
01	SNEQ+1 12		SNEQ	SWEQ	LBYA	LBYR	SBYA	SBYR	SBYA+1 16 79	SBYA+2 18	STLB	STRB	STLB+1 1A 1D	SRBI		FETCH+1 0F 9F
02	DBIA		LBYT	RBYT	JXGX	JXLX	JXEX	JXNX						PPRA+1 DD 2F	BYTE+5 AE	
03	JUNC	JAGE	JALT	JXGA	JAEQ	JANE	JCEZ	JXLA	JALE	JAGT	JCNZ	JXEA	CPSS	PXA	CLOP	JXNA
04	PXA+2 4D 70		APE2	AN2		IXMB+3 47 85	IXMA+1 86 56	IXMB+2 57 45				XATS+1 0B 5B	SCJG+2 4E 3C	PXA+1 3D 40	SCJG+1 CE 4C	
05	JIG+3 E0 60	XATS+4 5A	TFEQ	TXNG	DMRF+1 84	IMRF1	IXMA2	IXMB+1 87 47	INTER+3 5F 59	INTER+4 58 69	XATS+3 5B 51	XATS+2 4B 5A	CPSS+1 3C 5D	CPSS+2 5C AD	IRBM+1 8E 56	INTER+2 BF 58
06	JIG+4 50	JRDR+1 71	JNT2	JTR1	LAA	LAX	PAS	ISDA	ISDA+1 67 F8	INTER+5 59 6C	MADD	MLOAD	INTER+6 69 6D	INTER+7 6C 6F	CLOP+1 3E 0D	INTER+8 6D AF
07	PXA+3 40 F0	JRDR 81 61	APRE	ANPE	SAM	SXM	PSM	SDX	PSM+1 76 A8	SDX+1 77 F9	CPG2+2 AA 7B	CPG2+3 7A EB	MRV1	MRV2	MRAD	STPG
08	NA0	JRLE	JIG	IMMD	DMRF	IMRF	IXMA	IXMB	IND	INX	OTD	OTX	MVGP	SPFG	IRBM	NA15
09	RAXI+1 95 94		RSEX	RUNR	RACI	RAXI	SAXI	SLZJ		SZDS+1 9A	SXDS	SNZD	MRV1+1 7C	MRV2+1 7D		FETCH+2 1F
0A	RAXR+1 A4	SAXR+1 A6	SNCF	SSCF	RACR	RAXR	SAXR	SLZR	PSM+2 78 B8	ISJG+3 D9	CPG2+1 AD 7A	BYTE+3 AC AE	BYTE+2 BC AB	CPG2	BYTE+4 AB 2E	INTER+9 6F AD
0B	RXFI	IMML+1 B2	IMML	IMMS	RACF	RAXF	SAXF	SLZF	PSM+3 A8 0F	XRTN+2 B9 BA	LRTN	XRTN	BYTE+1 CC AC	CLOP2+1 0D	XRTN+1 BB B9	INTER+1 FF 5F
0C	MVXR+1 C4 C1	MVXR+2 C0	NCY	SCY	MVXR	MVRX	MOD	PGRP	PGRP+1 C7	ISJG+1 CF D9	NOSK	SKIP	BYTE	RSGP	SCJG	ISJG
0D	POPP+1 DF 30	XDA+1 D6 0F	CPG2+7 F2 0D	PPAL+1 DC D8	NDA	ODA	XDA	ADA	PPAL+2 D3 E8	ISJG+2 D9 A9	PAXE	PAXC	PPAL	PPRA	PPAX	POPP
0E	JIG+2 E2 50	MXRX	JIG+1 82 E0		LDA	LDX	PDS	ADX	PPAL+3 D8 E9	PPAL+4 E8		CPG2+4 7B FB	ILGA	ILPX	NAI1	NAI2
0F	PXA+4 70 F1	PXA+5 F0 0F	CPG2+6 FB D2			MRXX	PDS1	ADA1	ISDA+2 68 F7	SDX+2 79 F7		CPG2+5 EB F2				INTER

Chart 3

			JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	JFL, JCF, JZF COLUMN RESTRICT f,c,z=1	JLL COLUMN RESTRICT						JFL, JCF, JZF COLUMN RESTRICT f,c,z=0	JFL, JCF, JZF COLUMN RESTRICT f,c,z=1	JRL COLUMN RESTRICT			
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	INIT	JAGT+1	JTR1	JNT1	TFAL	TTRU	OTD1	JALE+1	IND1	MADD+1	IMRF1+1	XATS	INIT+12 FC	CLOP2	PPRA+4 1E	FETCH
01	SNEQ+1 12	INIT+4 21 41	SNEQ	SWEQ	LBYA	LBYR	SBYA	SBYR	SBYA+1 16 79	SBYA+2 18	STLB	STRB	STLB+1 1A 1D	SRBI	PPRA+5 0E 9E	FETCH+1 0F 9F
02	DBIA	INIT+3 91 11	LBYT	RBYT	JXGX	JXLX	JXEX	JXNX	INDI+1 08 0F	PPRA+13 2B 0F	PPRA+11 EA 2B	PPRA+12 2A 29		PPRA+1 DD 2F	BYTE+5 AE	PPRA+2 2D 4F
03	JUNC	JAGE	JALT	JXGA	JAEQ	JANE	JCEZ	JXLA	JALE	JAGT	JCNZ	JXEA	CPSS	PXA	CLOP	JXNA
04	PXA+2 4D 70	INIT+5 11 44	APE2	AN2	INIT+6 41 F4	IXMB+3 47 85	IXMA+1 86 56	IXMB+2 57 45	INIT+11 49 0C	INIT+10 4A 48	INIT+9 FA 49	XATS+1 0B 5B	SCJG+2 4E 3C	PXA+1 3D 40	SCJG+1 CE 4C	PPRA+3 2F 0E
05	JIG+3 E0 60	XATS+4 5A	TFEQ	TXNG	DMRF+1 84	IMRF1	IXMA2	IXMB+1 87 47	INTER+3 5F 59	INTER+4 58 69	XATS+3 5B 51	XATS+2 4B 5A	CPSS+1 3C 5D	CPSS+2 5C AD	IRBM+1 8E 56	INTER+2 BF 58
06	JIG+4 50	JRDR+1 71	JNT2	JTR1	LAA	LAX	PAS	ISDA	ISDA+1 67 F8	INTER+5 59 6C	MADD	MLOAD	INTER+6 69 6D	INTER+7 6C 6F	CLOP+1 3E 0D	INTER+8 6D AF
07	PXA+3 40 F0	JRDR 81 61	APRE	ANPE	SAM	SXM	PSM	SDX	PSM+1 76 A8	SDX+1 77 F9	CPG2+2 AA 7B	CPG2+3 7A EB	MRV1	MRV2	MRAD	STPG
08	NA0	JRLE	JIG	IMMD	DMRF	IMRF	IXMA	IXMB	IND	INX	OTD	OTX	MVGP	SPFG	IRBM	NA15
09	INIT+1 00 91	INIT+2 90 21	RSEX	RUNR	RACI	RAXI	SAXI	SLZJ	RAXI+1 95 94	SZDS+1 9A	SXDS	SNZD	MRV1+1 7C	MRV2+1 7D	PPRA+6 1E FE	FETCH+2 1F
0A	RAXR+1 A4	SAXR+1 A6	SNCF	SSCF	RACR	RAXR	SAXR	SLZR	PSM+2 78 B8	ISJG+3 D9	CPG2+1 AD 7A	BYTE+3 AC AE	BYTE+2 BC AB	CPG2	BYTE+4 AB 2E	INTER+9 6F AD
0B	RXFI	IMML+1 B2	IMML	IMMS	RACF	RAXF	SAXF	SLZF	PSM+3 A8 0F	XRTN+2 B9 BA	LRTN	XRTN	BYTE+1 CC AC	CLOP2+1 0D	XRTN+1 BB B9	INTER+1 FF 5F
0C	MVXR+1 C4 C1	MVXR+2 C0	NCY	SCY	MVXR	MVRX	MOD	PGRP	PGRP+1 C7	ISJG+1 CF D9	NOSK	SKIP	BYTE	RSGP	SCJG	ISJG
0D	POPP+1 DF 30	XDA+1 D6 0F	CPG2+7 F2 0D	PPAL+1 DC D8	NDA	ODA	XDA	ADA	PPAL+2 D3 E8	ISJG+2 D9 A9	PAXE	PAXC	PPAL	PPRA	PPAX	POPP
0E	JIG+2 E2 50	MXRX	JIG+1 82 E0	PPRA+9 F3 EA	LDA	LDX	PDS	ADX	PPAL+3 D8 E9	PPAL+4 E8	PPRA+10 E3 2A	CPG2+4 7B FB	ILGA	ILPX	NAI1	NAI2
0F	PXA+4 70 F1	PXA+5 F0 0F	CPG2+6 FB D2	PPRA+8 FE E3	INIT+7 44 FA	MRXX	PDS1	ADA1	ISDA+2 68 F7	SDX+2 79 F7	INIT+8 F4 4A	CPG2+5 EB F2	INIT+13 0C		PPRA+7 9E F3	INTER

APPENDIX A

THE DESIGN EXAMPLE INSTRUCTION SET

The basic machine uses a 16-bit word. All instructions are single word instructions except the long immediate forms. Macroprograms are fully relocatable without reassembly. The data segment is also independently relocatable. There are five basic instruction catagories: memory reference, immediate data, jumps (including calls and returns), register moves and manipulations, and input-output functions.

The machine has seven registers as follows:

REGISTER		ASSIGNED CPE REGISTER
(A)	Accumulator	R0
(X)	Index Register	R1
(B)	Data-Base Register	R5
(E)	Program Execution Base Register	R6
(P)	Program Counter	R3
(S)	Stack Pointer	R4
(W)	Status Word Register*	R7

*A carry flip-flop designated C is the high order bit of the status word register W.

Memory Reference Group

ADDRESS MODE	ADDRESS COMPUTATION	M-FIELD CODES
Direct	B+D	0100
Indirect	(B+D)	0101
Indirect Relative	(B+D)+B	1110
Indirect Indexed	(B+D)+X	0110
Indirect Indexed Relative	(B+D)+X+B	0111

SUMMARY OF MEMORY REFERENCE MODES

Note: Values enclosed in ( ) designate indirect addresses.

The operations supported under these five modes are as follows:

MNEMONIC	FUNCTION	O
NDA	AND data to A	0000
LDA	Load data to A	0001
LAA	Load address to A	0010
SAM	Store A in memory	0011
ODA	OR data to A	0100
LDX	Load data to X	0101
LAX	Load address to X	0110
SXM	Store X in memory	0111
XDA	Exclusive OR data to A	1000
PDS	Push data to stack	1001
PAS	Push address to stack	1010
PSM	Pop stack to memory	1011
ADA	Add data to A	1100
ADX	Add data to X	1101
SDA	Subtract data from A	1110
SDX	Subtract data from X	1111

Immediate Group

MNEMONIC	FUNCTION	M-FIELD	O-FIELD
LAI	Load to A immediate	0011	0001
AAI	Add to A immediate	0011	1100
NAI	AND to A immediate	0011	0000
OAI	OR to A immediate	0011	0100
XAI	Exclusive OR to A immediate	0011	1000
PSI	Push to stack immediate	0011	1001
LXI	Load to X immediate	0011	0101
AXI	Add to X immediate	0011	1101

If D is equal to zero, the contents of the memory location following the instruction is used as the immediate value.

Jump Group

MNEMONIC	FUNCTION	RELATIVE M	RELATIVE O	INDIRECT M	INDIRECT O
JRU,JIU	Jump unconditional	0001	0000	0010	0000
JRGE,JIGE	Jump if A.GE.O	0001	0001	0010	0001
JRLT,JILT	Jump if A.LT.O	0001	0010	0010	0010
JRXG,JIXG	Jump if X.GT.A	0001	0011	0010	0011
JREZ,JIEZ	Jump if A.EQ.O	0001	0100	0010	0100
JRNZ,JINZ	Jump if A.NE.O	0001	0101	0010	0101
JRCZ,JICZ	Jump if C.EQ.O	0001	0110	0010	0110
JRXL,JIXL	Jump if X.LE.A	0001	0111	0010	0111
JRLE,JILE	Jump if A.LE.O	0001	1000	0010	1000
JRGT,JIGT	Jump if A.GT.O	0001	1001	0010	1001
JRCN,JICN	Jump if C.NE.O	0001	1010	0010	1010
JRXE,JIXE	Jump if X.EQ.A	0001	1011	0010	1011
JRXN,JIXN	Jump if X.NE.A	0001	1111	0010	1111

Unconditional and conditional jumps:

Relative: P = P+D' where D'=D−128
Indirect: P = E+(E+D)

Subroutine Call Group

MNEMONIC	FUNCTION	ABSOLUTE M	ABSOLUTE O		
CAS	Call absolute, push P, E, W, B P ← (D)	1101	XX11		
		M	O	M	O
CLS	Call local subroutine, push P	N.A.		0010	1110
CVS	Call global subroutine, push W. B, E, P	N.A.		0010	1100
Local:	Push P to stack P = E+(E+D)				
Value:	Push W, B, E, P to stack E = E+(E+D) P = E'+(E') where E'=E+(E+D)				

Subroutine Return Group

MNEMONIC	FUNCTION	M	O
RLS	Pop P	1100	1111
RVS	Pop P, E, B, W	1100	1101
RSA	Pop A, X, P, E, B, W	1100	1100

Register Manipulation Group

MNEMONIC	FUNCTION	M	O
RAR	Rotate A right, include CFF	1101	0001
RAX	Rotate A and X right, include CFF	1101	0101
SAX	Shift A and X right, preserve sign	1101	1001
SAL	Shift A left, fill with zeros	1101	1101

The shift count is given by D if D is non-zero or by the least significant seven bits of the X register if D is zero.

Base and Status Register Move Group

MNEMONIC	FUNCTION	M	O
MSX	Move S to X, adjust	1100	0100
MBX	Move B to X, adjust	1100	0101
MEX	Move E to X, adjust	1100	0110
MWX	Move W to X, adjust	1100	0111
MXS	Move X to S, adjust	1100	0000
MXB	Move X to B, adjust	1100	0001
MXE	Move X to E, adjust	1100	0010
MXW	Move X to W, adjust	1100	0011
NO.OP	Nothing implemented	1100	10XX

The destination register is adjusted by D–128.

Byte Load and Store Group

MNEMONIC	FUNCTION	M	O
LBA	Load byte absolute	1101	0000
LBR	Load byte relative	1101	0100
SBA	Store byte absolute	1101	1000
SBR	Store byte relative	1101	1100

Absolute mode: Byte address = (B+D)+X/2

Relative mode: Byte address = (B+D)+B+X/2

Input/Output Group

MNEMONIC	FUNCTION	M	O
IND	Input one word A ← (D)	1000	XXXX
OTD	Output one word (D) ← A	1001	XXXX
INX	Input one word A ← (X)	1010	XXXX
OTX	Output one word (X) ← A	1011	XXXX

Special Memory Reference Instruction

MNEMONIC	FUNCTION	M	O
ISZ	Increment and skip if zero	1101	XX10

Stack Push and Pop Group

MNEMONIC	FUNCTION	M	O	M	O
PHAX	Push A, X onto stack	0001	1101	0010	1101
PPAX	Pop A, X	1100	1110		

APPENDIX B

MICROPROGRAM LISTING

```
RECORD
NUMBER

    1  /* BIPOLAR MICROCOMPUTER MACRO-MACHINE
    2     REGISTER MACHINE--12/13/74
    3     UPDATED 3/18/75
    4
    5  MACHINE HAS 7 REGISTERS AS FOLLOWS:
    6   A        ACCUMULATOR      R0
    7   X        INDEX REGISTER   R1
    8   P        PROGRAM COUNTER  R3
    9   S        STACK POINTER    R4
   10   B        DATA BASE REG    R5
   11   E        PROG. BASE REG.  R6
   12   W        STATUS WORD      R7
   13
   14   C=CARRY,LINK FLIP-FLOP=HOB OF W
   15
   16   DEFINITION OF KBUS FIELD                  */
   17
   18   KB       FIELD   LENGTH=4         DEFAULT=0
   19            MICROPS(K0000=0   K007F=1   K00FF=3   K7FFF=7
   20                    K8000=8   KFF00=12   KFF80=14   KFFFF=15);
   21
   22   KB       KBUS;
   23
   24
   25  /* DEFINITION OF BUS CONTROL FIELD        */
   26
   27   MCF      FIELD   LENGTH=3         DEFAULT=0
   28            MICROPS(NMO=000B   INH=001B   RMW=010B   CNB=011B
   29                    RIN=100B   ROT=101B   RRM=110B   RWM=111B);
   30
   31  /*        NBO     NO BUS OPERATION
   32   INH      INHIBIT CPE ARRAY
   33   RMW      READ-MODIFY-WRITE
   34   CNB      CPU NEEDS BUS
   35   RIN      REQUEST INPUT
   36   ROT      REQUEST OUTPUT
   37   RRM      REQUEST READ MEM.
   38   RWM      REQUEST WRITE MEM.
   39
   40   SET UP FOR SYMBOLIC REPRESENTATION OF REGISTER DESIGNATIONS */
   41
   42   A        STRING  'R0';
   43   X        STRING  'R1';
   44   P        STRING  'R3';
   45   S        STRING  'R4';
   46   B        STRING  'R5';
   47   E        STRING  'R6';
   48   W        STRING  'R7';
   49
   50  /* SET UP A SPECIAL NO.OP STRING */
   51
   52   NO.OP    STRING  'NOP(R2)';
   53
   54  /* NEXT WE SPECIFY A DEFAULT TO FF1 IN THE FO FIELD FOR THE SDR
   55     MICROP IN THE CPE FIELD. SDR IS NORMALLY USED AS A STORE
   56     OPERATION. WHEN A DECREMENT OPERATION IS ALSO DESIRED, FF0
   57     WILL HAVE TO BE EXPLICITLY SPECIFIED */
   58
   59   SDR      IMPLY   FO=11B;
   60
```

```
RECORD
NUMBER

   61
   62
   63
   64  /* INITIALIZATION SEQUENCE
   65     ZERO A, X, AND W */
   66
   67   000H:   INIT:   CLR(A);
   68   090H:           CLR(X);
   69   091H:           CLR(W);
   70
   71  /* ZERO T AS TEMPORARY POINTER, WRITE W TO INTERRUPT STRUCTURE   */
   72
   73   021H:           CLR(T);
   74   011H:           LMI(T);
   75   041H:           ILR(W) ROT;
   76
   77  /* SET S = (0), T = 1 FOR NEXT OPERATION  */
   78
   79   044H:           LMI(T) FF1 RRM;
   80   0F4H:           ACM(AC)  ;
   81   0FAH:           SDR(S);
   82
   83  /* SET B = (1), T = 2 FOR NEXT OPERATION */
   84
   85   04AH:           LMI(T) FF1 RRM;
   86   049H:           ACM(AC);
   87   048H:           SDR(B) STC;      /* THIS SETS THE C FLAG TO INSURE
   88                                       A CORRECT JUMP TO XRTN */
   89
   90  /* GET (2), JUMP TO XRTN TO SET E = (2), P = (E) */
   91
   92   00CH:           LMI(T) RRM;
   93   0FCH:           ACM(AC)                  JCF (*,XRTN);
   94
   95  /* FETCH SEQUENCE & START OF MACRO-INSTRUCTION PROCESSING
   96     P IS ISSUED TO MAR AND INCREMENTED, MACRO-INSTRUCTION
   97     IS FETCHED AND TESTED BY JPX MICRO-OPERATION.  NOTE
   98     FETCH IS IN LOCATION 15 TO STROBE INTERRUPT ON ENTRY. */
   99
  100   00FH:   FETCH:  LMI(P) FF1 RRM;
  101
  102  /* LOAD DISPLACEMENT AND TEST FOR ZERO USING Z FLAG */
  103
  104   01FH:           LTM(AC) STZ K00FF;
  105
  106  /* SAVE DISPLACEMENT, TEST 4 BITS OF MACRO-OP.  TEST IS
  107     DELAYED TO ALLOW PIPELINE PROPAGATION.  ALSO C FLAG IS
  108     SET FOR LATER USE IN PSEUDO-SUBROUTINES. */
  109
  110   09FH:           SDR(R9) STC    JPX(NA0,JREL,JIG,IMMD,DMRF,IMRF,IXMA,IXMB,IND,
  111                                      INX,OTD,OTX,MVGP,SPFG,IRBM,NA15);
  112
  113  /* UNASSIGNED OP-CODE GROUPS--NOPS FOR THIS VERSION */
  114
  115   080H:   NA0:    NO.OP                    JZR(FETCH);
  116   08FH:   NA15:   NO.OP                    JZR(FETCH);
  117
  118  /* IMMEDIATE GROUP OF MACRO-INSTRUCTIONS--TEST FOR LONG OR SHORT
  119     FORM--D IS IN AC AND R9--ADJUST AC BY -128 */
  120
  121   083H:   IMMD:   LMI(AC) KFF80            JZF(IMML,IMMS);
  122
  123  /* LONG FORM: FETCH NEXT WORD TO AC */
  124
  125   082H:   IMML:   LMI(P) FF1 RRM;
```

```
RECORD
NUMBER

  126   0B1H:            ACM(AC)                    JRL(ILGA,ILPX,NAI1,NAI2);
  127
  128   /* SHORT FORM: NO PROCESSING NEEDED */
  129
  130   0B3H:   IMMS:    NO.OP                      JRL(ILGA,ILPX,NAI1,NAI2);
  131
  132   /* PREPROCESSING FOR ARITHMETIC AND LOGIC ROUTINES?  NONE NEEDED */
  133
  134   0ECH:   ILGA:    NO.OP                      JLL(NDA,ODA,XDA,ADA);
  135   0EDH:   ILPX:    NO.OP                      JLL(LDA,LDX,PDS,ADX);
  136
  137   /* NOTE:  NAI1 AND NAI2 ARE NON-VALID INSTRUCTIONS!!  THEY ARE
  138      MADE INTO NO-OPS IN THIS VERSION OF THE MACRO-MACHINE */
  139
  140   0EEH:   NAI1:    NO.OP                      JZR(FETCH);
  141   0EFH:   NAI2:    NO.OP                      JZR(FETCH);
  142
  143   /* BASIC ARITHMETIC AND LOGIC PROCESSING--UPDATE C FF OF MACRO-
  144      MACHINE FOR ADA--TOGGLE IT ON CARRY FROM ADA */
  145
  146   0D7H:   ADA:     ADR(A);
  147   0F7H:   ADA1:    NO.OP                      JFL(NCY,SCY);
  148   0C2H:   NCY:     NO.OP                      JZR(FETCH);
  149   0C3H:   SCY:     LMI(W) K8000               JZR(FETCH);
  150
  151   /* LOGICALS */
  152
  153   0D4H:   NDA:     ANR(A)                     JZR(FETCH);
  154   0D5H:   ODA:     ORR(A)                     JZR(FETCH);
  155   0D6H:   XDA:     CMR(AC);
  156   0D1H:            XNR(A)                     JZR(FETCH);
  157
  158   /* LDA AND LDX OPERATIONS */
  159
  160   0E4H:   LDA:     SDR(A)                     JZR(FETCH);
  161   0E5H:   LDX:     SDR(X)                     JZR(FETCH);
  162
  163   /* STACK PUSH--ADVANCE STACK POINTER TO NEXT LOCATION (FOR THE
  164      REVERSE DIRECTION STACK--A DECREMENT OF S), THEN WRITE */
  165
  166   0E6H:   PDS:     DSM(S);
  167   0F6H:   PDS1:    LMI(S) RWM                 JZR(FETCH);
  168
  169   /* ADX - SHARES CODE FOR ADA - ALSO TOGGLES C FF OF MACRO MACHINE */
  170
  171   0E7H:   ADX:     ADR(X)                     JMP(ADA1);
  172
  173   /* MEMORY REFERENCE INSTRUCTION GROUPS
  174      DIRECT--GET B+D INTO AC--ALSO R9 */
  175
  176   084H:   DMRF:    ILR(B);
  177   054H:            ALR(R9)                    JRL(MRV1,MRV2,MRAD,STPG);
  178
  179   /* INDIRECT-ABSOLUTE--GET (B+D) INTO AC--C FLAG USED FOR PSEUDO-SUBROUTINE */
  180
  181   085H:   IMRF:    ILR(B);
  182   055H:   IMRF1:   ALR(R9);
  183   00AH:            LMI(R9) RRM                JCF(MADD,MLOAD);
  184   06BH:   MLOAD:   ACM(AC)                    JRL(MRV1,MRV2,MRAD,STPG);
  185
  186   /* NOTE:  MADD WILL BE USED FOR OTHER INDIRECT OPERATIONS WHERE
  187      B, X, ETC. HAS BEEN LOADED TO R8 */
  188
  189   06AH:   MADD:    ACM(AC);
  190   009H:            ALR(R8)                    JRL(MRV1,MRV2,MRAD,STPG);
  191
  192   /* INDIRECT INDEXED ABSOLUTE - CLEAR C FLAG, MOVE X TO R8 */
  193
  194   086H:   IXMA:    ILR(X) STC;
  195   046H:            SDR(R8);
  196
```

```
RECORD
NUMBER

  197  /* NOTING THAT ASSIGNMENT RULES WOULD NOT ALLOW THE DESIRED
  198     JUMP TO IMRF UNLESS IXMA+1 WERE IN ROW ZERO--AN EXTRA STATE
  199     IS ADDED HERE */
  200
  201   056H:   IXMA2:  ILR(B)                    JMP(IMRF1);
  202
  203  /* INDIRECT INDEXED RELATIVE - CLEAR C FLAG, PUT B+X IN R8 */
  204
  205   087H:   IXMB:   ILR(X) STC;
  206   057H:           SDR(R8);
  207   047H:           ILR(B);
  208   045H:           ADR(R8)                   JMP(IMRF);
  209
  210  /* INDIRECT RELATIVE (TO B) - CLEAR C FLAG, PUT B IN R8 */
  211
  212   08EH:   IRBM:   ILR(B);
  213
  214  /* AGAIN ASSIGNMENT RULES PREVENT JUMPING TO IXMA+1 UNLESS IT IS
  215     LOCATED IN ROW ZERO--PLACEMENT THERE COULD FREE TWO WORDS */
  216
  217   05EH:           SDR(R8)                   JMP(IXMA2);
  218
  219  /* THE FOLLOWING PROCEDURES IMPLEMENT THE BASIC PREPROCESSING FOR
  220     VALUE AND ADDRESS LOADING.
  221
  222     VALUE-GROUP 1:  GET (AC) IN AC */
  223
  224   07CH:   MRV1:   LMI(AC) RRM;
  225   09CH:           ACM(AC)                   JLL(NDA,ODA,XDA,ADA);
  226
  227  /* VALUE GROUP 2 */
  228
  229   07DH:   MRV2:   LMI(AC) RRM;
  230   09DH:           ACM(AC)                   JLL(LDA,LDX,PDS,ADX);
  231
  232  /* MRAD GROUP INCLUDES ADDRESS LOADS AND SUBTRACT FROM A */
  233
  234   07EH:   MRAD:   NO.OP                     JLL(LAA,LAX,PAS,ISDA);
  235
  236   064H:   LAA:    SDR(A)                    JZR(FETCH);
  237   065H:   LAX:    SDR(X)                    JZR(FETCH);
  238   066H:   PAS:    DSM(S)                    JMP(PDS1);
  239
  240  /* FOR SUBTRACT, ADD 1'S COMPLEMENT PLUS 1 */
  241
  242   067H:   ISDA:   LMI(AC) RRM;
  243   068H:           LCM(AC);
  244   0F8H:           ADR(A) FF1                JMP(ADA1);
  245
  246  /* STPG GROUP INCLUDES STORES AND SUBTRACT FROM X */
  247
  248   07FH:   STPG:   LMI(AC)                   JLL(SAM,SXM,PSM,SDX);
  249
  250   074H:   SAM:    ILR(A) RWM                JZR(FETCH);
  251   075H:   SXM:    ILR(X) RWM                JZR(FETCH);
  252
  253  /* POP STACK TO MEMORY - SAVE ADDRESS, POP STACK */
  254
  255   076H:   PSM:    SDR(T);
  256   078H:           LMI(S) FF1 RRM;
  257   0A8H:           ACM(AC);
  258   0B8H:           LMI(T) RWM                JZR(FETCH);
  259
  260  /* SUBTRACT FROM X */
  261
  262   077H:   SDX:    LMI(AC) RRM;
  263   079H:           LCM(AC);
  264   0F9H:           ADR(X) FF1                JMP(ADA1);
  265
  266  /* JUMP GROUPS--USE JPR MICRO-OPERATION TO RESOLVE CONDITION SELECTION
  267     DESTINATION ADDRESS IS COMPUTED FIRST--PLACED IN AC AND R9
```

```
RECORD
NUMBER

   268      JUMP RELATIVE TO P--ADDRESS=P+D-128 */
   269
   270    081H:   JREL:   ILR(P);
   271    071H:   JRDR:   LMI(AC) KFF80;
   272    061H:           ALR(R9)             JPR(JUNC,JAGE,JALT,JXGA,JAEQ,JANE,JCEZ,JXLA,
   273                                            JALE,JAGT,JCNZ,JXEA,CPSS,PXA,CLOP,JXNA);
   274
   275   /* JUMP INDIRECT - GET E+(E+D) IN AC AND R9 */
   276
   277    082H:   JIG:    ILR(E);
   278    0E2H:           ADR(R9);
   279    0E0H:           LMI(R9) RRM;
   280    050H:           AMA(AC);
   281    060H:           SDR(R9)             JPR(JUNC,JAGE,JALT,JXGA,JAEQ,JANE,JCEZ,JXLA,
   282                                            JALE,JAGT,JCNZ,JXEA,CPSS,PXA,CLOP,JXNA);
   283
   284   /* UNCONDITIONAL JUMP */
   285
   286    030H:   JUNC:   SDR(P)              JZR(FETCH);
   287
   288   /* TESTS FOR A.GE.0, ETC. */
   289
   290    031H:   JAGE:   TZR(A) K8000 INH    JMP(TTRU);
   291    032H:   JALT:   TZR(A) K8000 INH    JMP(TFAL);
   292    034H:   JAEQ:   TZR(A)              JMP(TTRU);
   293    035H:   JANE:   TZR(A)              JMP(TFAL);
   294
   295    039H:   JAGT:   TZR(A) K8000 INH;
   296    001H:           TZR(A)              JFL(APRE,ANPE);
   297
   298    072H:   APRE:   NO.OP               JFL(JNT2,JTR2);
   299    073H:   ANPE:   NO.OP               JZR(FETCH);
   300
   301    038H:   JALE:   TZR(A) K8000 INH;
   302    007H:           TZR(A)              JFL(APE2,AN2);
   303
   304    042H:   APE2:   NO.OP               JFL(JTR1,JNT1);
   305    043H:   AN2:    SDR(P)              JZR(FETCH);
   306
   307   /* TESTS OF C FLIP-FLOP (HIGH ORDER BIT OF W) */
   308
   309    036H:   JCEZ:   TZR(W) K8000 INH    JMP(TTRU);
   310    03AH:   JCNZ:   TZR(W) K8000 INH    JMP(TFAL);
   311
   312   /* TEST EXECUTION FOR ABOVE TESTS - ROW ZERO USED */
   313
   314    005H:   TTRU:   NO.OP               JFL(JTR1,JNT1);
   315
   316    002H:   JTR1:   SDR(P)              JZR(FETCH);
   317    003H:   JNT1:   NO.OP               JZR(FETCH);
   318
   319    004H:   TFAL:   NO.OP               JFL(JNT2,JTR2);
   320
   321    062H:   JNT2:   NO.OP               JZR(FETCH);
   322    063H:   JTR2:   SDR(P)              JZR(FETCH);
   323
   324   /* TESTS FOR X.GT.A, X.LE.A, X.EQ.A, X.NE.A--SHARED PSEUDO-
   325      SUBROUTINE USES JLL FOR AN EXIT TEST--ROUTINE ENTRY IN ROW 0
   326      C FLAG IS SET FOR X.GT.A, FL TEST FOR X.EQ.A */
   327
   328    033H:   JXGA:   ILR(X)              JMP(XATS);
   329    037H:   JXLA:   ILR(X)              JMP(XATS);
   330    03BH:   JXEA:   ILR(X)              JMP(XATS);
   331    03FH:   JXNA:   ILR(X)              JMP(XATS);
   332
   333   /* SAVE X AT T, FETCH AND COMPLEMENT A */
   334
   335    00BH:   XATS:   SDR(T);
   336    04BH:           ILR(A) STC;      /* CLEAR C FLAG */
   337    05BH:           CMA(AC);
   338
```

```
RECORD
NUMBER

339  /* ADD HOB'S OF A' AND X - CARRY MEANS X NEG., A.GE.0 */
340
341   05AH:            ADR(T) K8000;
342
343  /* EXECUTE PREVIOUS TEST, SET UP TO TEST HOB OF RESULT--IF 1,
344     THE SIGNS OF A AND X WERE THE SAME */
345
346   051H:            TZR(T) K8000 INH          JFL(TFEQ,TXNG);
347
348  /* TXNG IMPLIES X NEG AND A.GE.0--I.E. X.NE.A AND X.LT.A--DO A
349     DUMMY OPERATION TO FORCE THE PROPER F FLAG */
350
351   053H:   TXNG:   ILR(A)                    JLL(JXGX,JXLX,JXEX,JXNX);
352
353  /* PERFORM A TEST ADDITION AND EXECUTE SIGN-EQUAL TEST
354     C WILL BE SET IF SIGNS WERE THE SAME AND X.GT.A */
355
356   052H:   TFEQ:   ADR(T) STC K7FFF          JFL(SNEQ,SWEQ);
357
358  /* SNEQ IMPLIES SIGNS NOT EQUAL--I.E. X.GE.0, A NEG--X.GT.A */
359
360   012H:   SNEQ:   SDR(AC) STC;    /* DUMMY OP TO SET C FLAG */
361   010H:           NO.OP                     JLL(JXGX,JXLX,JXEX,JXNX);
362
363  /* FOR SIGNS EQUAL, IF X=A RESULT WOULD BE 1111...1.  INCREMENT
364     WILL GENERATE A CARRY IF SO */
365
366   013H:   SWEQ:   ILR(AC) FF1              JLL(JXGX,JXLX,JXEX,JXNX);
367
368  /* EXECUTION OF JUMP TESTS */
369
370   024H:   JXGX:   ILR(R9)                   JCF(JNT2,JTR2);
371   025H:   JXLX:   ILR(R9)                   JCF(JTR1,JNT1);
372   026H:   JXEX:   ILR(R9)                   JFL(JNT2,JTR2);
373   027H:   JXNX:   ILR(R9)                   JFL(JTR1,JNT1);
374
375  /* SUBROUTINE CALLS
376     CALL LOCAL AND PUSH W, B, E, P =CPSS
377     CALL LOCAL AND PUSH P ONLY=CLOP
378     C FLAG IS USED FOR EXIT TEST AFTER PUSHING P */
379
380   03CH:   CPSS:   DSM(S);
381   05CH:           ILR(W);
382   05DH:           LMI(S) RWM;
383
384   0ADH:   CPG2:   DSM(S);
385   0AAH:           ILR(B);
386   07AH:           LMI(S) RWM;
387
388   07BH:           DSM(S);
389   0EBH:           ILR(E);
390   0FBH:           LMI(S) RWM;
391
392   0F2H:           DSM(S);
393   0D2H:           ILR(P);
394   00DH:   CLOP2:  LMI(S) RWM;
395
396  /* E+(E+D) INTO AC */
397
398   0BDH:           ILR(R9)                   JCF(LRTN,XRTN);
399
400   0BBH:   XRTN:   SDR(E);
401   0BEH:           LMI(E) RRM;
402   0B9H:           AMA(AC);
403
404   0BAH:   LRTN:   SDR(P)                    JZR(FETCH);
405
406   03EH:   CLOP:   DSM(S);
407   06EH:           ILR(P) STC                JMP(CLOP2);
408
409  /* PUSH INSTRUCTION */
```

```
RECORD
NUMBER

  410
  411   03DH:   PXA:    DSM(S);
  412   04DH:           ILR(X);
  413   040H:           LMI(S) RWM;
  414
  415   070H:           DSM(S);
  416   0F0H:           ILR(A);
  417   0F1H:           LMI(S) RWM              JZR(FETCH);
  418
  419  /* MOVE GROUP OF INSTRUCTIONS--USES JCE TO SELECT REGISTER--NOTE
  420     THAT REGISTER ASSIGNMENT BECOMES IMPORTANT
  421     FIRST MODIFY D TO GET D-128 */
  422
  423   08CH:   MVGP:   LMI(R9) KFF80           JLL(MVXR,MVRX,MOD,PGRP);
  424
  425  /* MOVE X TO REG. - GET X, MODIFY BY D'=D-128 */
  426
  427   0C4H:   MVXR:   ILR(X);
  428   0C0H:           ALR(R9);
  429   0C1H:           SDR(R7)                 JCE(MXRX);      /* REGISTER OVERRIDE */
  430   0E1H:   MXRX:   NO.OP                   JZR(FETCH);
  431
  432  /* MOVE REG TO X - FETCH REG USING JCE OVERRIDE */
  433
  434   0C5H:   MVRX:   ILR(R7)                 JCE(MRXX);
  435   0F5H:   MRXX:   ALR(R9)                 JMP(LDX);
  436
  437  /* MOD NOT IMPLEMENTED IN THIS VERSION */
  438
  439   0C6H:   MOD:    NO.OP                   JZR(FETCH);
  440
  441  /* ADJUST STACK AND RETURN GROUP
  442     PPAL--POPS A, X, P, E, B, AND W
  443     PPRA--POPS P, E, B, AND W
  444     PPAX--POPS ONLY A AND X
  445     POPP--POPS ONLY P */
  446
  447   0C7H:   PGRP:   ILR(R9);
  448   0C8H:           ADR(S)                  JRL(PPAL,PPRA,PPAX,POPP);
  449
  450   0DCH:   PPAL:   LMI(S) FF1 RRM;
  451   0D3H:           ACM(AC);
  452   0D8H:           SDR(A);
  453
  454   0E8H:           LMI(S) FF1 RRM;
  455   0E9H:           ACM(AC)                 JCF(PAXE,PAXC);
  456   0D8H:   PAXC:   SDR(X);
  457
  458   0DDH:   PPRA:   LMI(S) FF1 RRM;
  459   02DH:           ACM(AC);
  460   02FH:           SDR(P);
  461
  462   04FH:           LMI(S) FF1 RRM;
  463   00EH:           ACM(AC);
  464   01EH:           SDR(E);
  465
  466   09EH:           LMI(S) FF1 RRM;
  467   0FEH:           ACM(AC);
  468   0F3H:           SDR(B);
  469
  470   0E3H:           LMI(S) FF1 RRM;
  471   0EAH:           ACM(AC);
  472   02AH:           SDR(W);
  473
  474  /* RESTORE INTERRUPT STRUCTURE */
  475
  476   02BH:           CLR(T);
  477   029H:           LMI(T) ROT              JZR(FETCH);
  478
  479   0DAH:   PAXE:   SDR(X)                  JZR(FETCH);
  480
```

```
RECORD
NUMBER

  481   0DEH:   PPAX:   ILR(AC) STC              JMP(PPAL);
  482
  483   0DFH:   POPP:   LMI(S) FF1 RRM;
  484   0D0H:           ACM(AC)                   JMP(JUNC);
  485
  486  /* SPECIAL FUNCTION GROUP
  487     BYTE OPERATORS--ADDR=(B+D)+B+X/2 OR (B+D)+X/2
  488     CALL TO (D) AND PUSH ALL
  489     SHIFT AND ROTATE GROUP
  490     INCREMENT AND SKIP
  491     FETCH B JUST IN CASE */
  492
  493   08DH:   SPFG:   ILR(B)                    JRL(BYTE,RSGP,SCJG,ISJG);
  494
  495  /* BYTE GROUP--COMPUTE ADDR, STORE B IN CASE NEEDED */
  496
  497   0CCH:   BYTE:   SDR(R8);
  498   0BCH:           ADR(R9);
  499   0ACH:           ILR(X);
  500   0ABH:           SRA(AC) STC;
  501   0AEH:           LMI(R9) RRM;
  502   02EH:           AMA(AC)                   JLL(LBYA,LBYR,SBYA,SBYR);
  503
  504   015H:   LBYR:   ALR(R8);
  505   014H:   LBYA:   LMI(AC) RRM               JCF(LBYT,RBYT);
  506   022H:   LBYT:   LDI(AC) FF1 K00FF         JMP(DBIA);
  507   023H:   RBYT:   LTM(AC) K00FF;
  508   020H:   DBIA:   SDR(A)                    JZR(FETCH);
  509
  510   017H:   SBYR:   ALR(R8);
  511   016H:   SBYA:   LMI(AC);                  /* LOAD MAR FOR LATER USE */
  512   018H:           ILR(A);
  513   019H:           TZR(AC) K00FF RRM         JCF(STLB,STRB);
  514   01BH:   STRB:   LTM(T) KFF00;
  515   01DH:   SRB1:   ALR(T) RWM                JZR(FETCH);
  516
  517   01AH:   STLB:   LTM(T) K00FF;
  518   01CH:           LDI(AC) FF1 CNB           JMP(SRB1);
  519
  520  /* ROTATE GROUP
  521     ROTATE A WITH C--ROTATE A AND X WITH C--SHIFT A, X RIGHT, FILL
  522     WITH SIGN--SHIFT A LEFT, FILL WITH ZEROES
  523
  524     AT ENTRY, Z FLAG IS ZERO IF D=0. DUE TO PIPELINED OPERATION, IT IS
  525     THIS CONDITION THAT IS TESTED BY THE FIRST JZF */
  526
  527   0CDH:   RSGP:   TZR(W) STZ K8000 INH      JZF(SZDS,SNZD);
  528   09AH:   SZDS:   ILR(X);
  529   099H:           SDR(R9) FF0 K007F         JLL(RACI,RAXI,SAXI,SLZI);
  530   09BH:   SNZD:   DSM(R9)                   JLL(RACI,RAXI,SAXI,SLZI);
  531
  532   094H:   RACI:   ILR(A)                    JMP(RUNR);
  533   095H:   RAXI:   ILR(X);
  534   098H:           SDR(T)                    JMP(RACI);
  535   096H:   SAXI:   TZR(A) STZ K8000 INH      JMP(RAXI);
  536   097H:   SLZI:   ILR(A)                    JMP(RUNR);
  537
  538  /* MAIN ROTATION LOOP */
  539
  540   093H:   RUNR:   DSM(R9) STC               JLL(RACR,RAXR,SAXR,SLZR);
  541
  542   0A4H:   RACR:   SRA(AC) FFZ STZ           JFL(RSEX,RUNR);
  543   0A5H:   RAXR:   SRA(AC) FFZ STZ;
  544   0A0H:           SRA(T) FFZ STZ            JCF(RSEX,RUNR);
  545   0A6H:   SAXR:   SRA(AC) FFZ STC;
  546   0A1H:           SRA(T) FFC                JCF(RSEX,RUNR);
  547   0A7H:   SLZR:   ADR(AC) STZ               JFL(RSEX,RUNR);
  548
  549   092H:   RSEX:   SDR(A)                    JLL(RACF,RAXF,SAXF,SLZF);
  550
  551   0B4H:   RACF:   TZR(W) K7FFF              JZF(SNCF,SSCF);
```

```
RECORD
NUMBER

  552   0A2H:   SNCF:   NO.OP                    JZR(FETCH);
  553   0A3H:   SSCF:   LMI(W) K8000             JZR(FETCH);
  554   0B5H:   RAXF:   ILR(T);
  555   0B0H:   RXF1:   SDR(X)                   JMP(RACF);
  556   0B6H:   SAXF:   ILR(T)                   JMP(RXF1);
  557   0B7H:   SLZF:   TZR(W) K7FFF             JZF(SNCF,SSCF);
  558
  559  /* SPECIAL CALL AND JUMP GROUP--CURRENTLY CONTAINS ONLY THE
  560     CALL TO (D) AND PUSH W,B,E,P--ALL 4 OPCODES DO THE SAME THING */
  561
  562   0CEH:   SCJG:   LMI(R9) RRM;
  563   04EH:           ACM(AC);
  564   04CH:           SDR(R9)                  JMP(CPSS);
  565
  566  /* INCREMENT AND SKIP GROUP--AGAIN 4 OPCODES ARE USED FOR ONE
  567     INSTRUCTION--LOCATION AT B+D IS INCREMENTED */
  568
  569   0CFH:   ISJG:   ALR(R9);
  570   0C9H:           LMI(R9) RMW;
  571   0D9H:           ACM(AC) FF1;
  572   0A9H:           NO.OP RWM                JFL(NOSK,SKIP);
  573   0CAH:   NOSK:   NO.OP                    JZR(FETCH);
  574   0CBH:   SKIP:   LMI(P) FF1               JZR(FETCH);
  575
  576  /* INPUT AND OUTPUT--CURRENT VERSION DOES NOT DECODE INTO
  577     SUBGROUPS--ALSO ROW ZERO IS USED TO SAVE CODE */
  578
  579   088H:   IND:    LMI(R9) RIN;
  580   008H:   IND1:   ACM(AC);
  581   028H:           SDR(A)                   JZR(FETCH);
  582   089H:   INX:    LMI(X) RIN               JMP(IND1);
  583   08AH:   OTD:    LMI(R9);
  584   006H:   OTD1:   ILR(A) ROT               JZR(FETCH);
  585   08BH:   OTX:    LMI(X)                   JMP(OTD1);
  586
  587  /* INTERRUPT--UTILIZES CALL ROUTINES FOR REGISTER SAVING
  588     I/O DEVICE #0 REPRESENTS EXTERNAL INTERRUPT STRUCTURE
  589     START BY PUSHING OLD VALUE OF STATUS */
  590
  591   0FFH:   INTER:  DSM(S);
  592   0BFH:           ILR(W);
  593   05FH:           LMI(S) RWM;
  594
  595  /* READ INTERRUPTING LEVEL FROM EXTERNAL STRUCTURE */
  596
  597   058H:           CLR(T);
  598   059H:           LMI(T) RIN;
  599   069H:           LTM(AC) K00FF ROT; /* NOTE LEVEL REWRITTEN */
  600
  601  /* STORE PRIORITY IN W - SET C FLAG FOR PROPER LOADING OF REGISTERS */
  602
  603   06CH:           SDR(W) STC;
  604
  605  /* INTERRUPT ROUTINE STARTING ADDRESS IS COMPUTED IN R9 */
  606
  607   06DH:           LMI(W) RRM;
  608   06FH:           ACM(AC);
  609   0AFH:           SDR(R9)                   JMP(CPG2);
  610
  611
  612  EOF

NO PROGRAM ERRORS
END OF PROGRAM
```

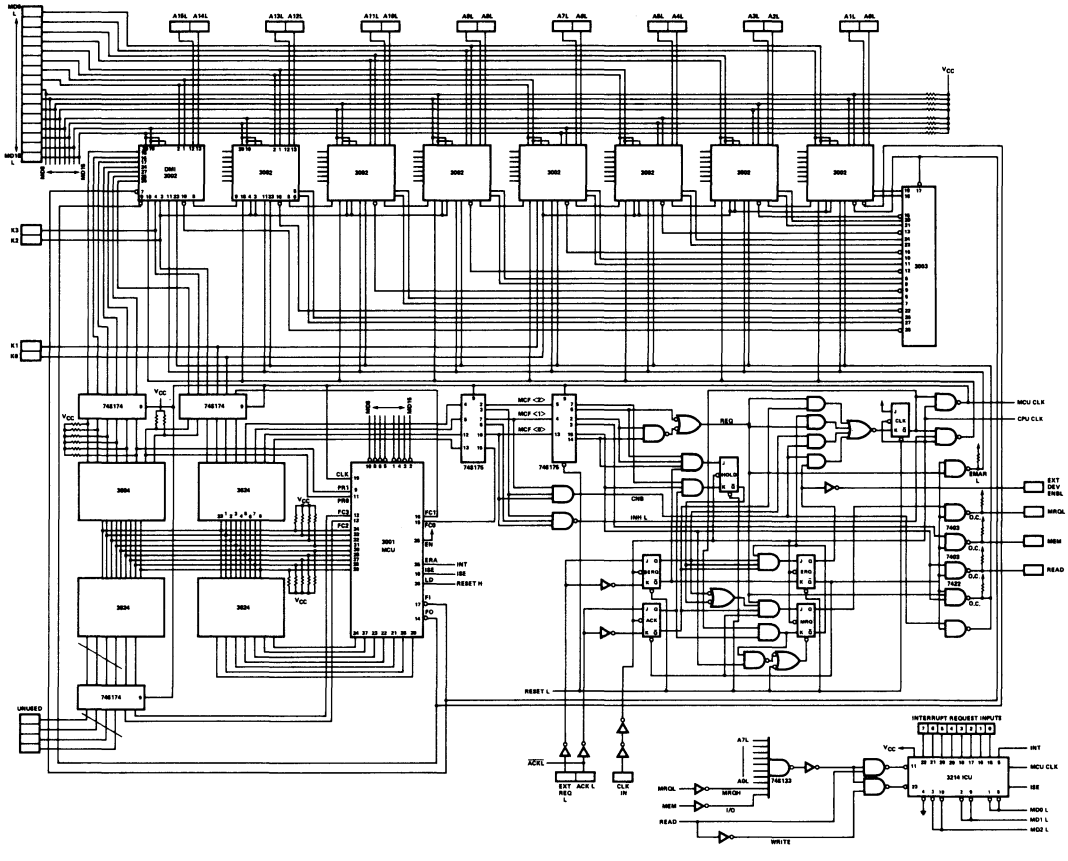
MICROPROGRAM MEMORY IMAGE

	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H	AH	BH	CH	DH	EH	FH
	JCC	JFL	JZR	JZR	JFL	JFL	JZR	JFL	JCC	JRL	JCF	JCC	JCC	JCC	JCC	JCC
	0090H	0072H	000FH	000FH	0062H	0002H	000FH	0042H	0028H	007CH	006AH	004BH	00FCH	00BDH	001EH	001FH
000H																
	67	296	316	317	319	314	584	302	580	190	183	335	92	394	463	100
	0	1	4	4	3	3	2	1	2	1	1	4	1	2	1	38
	JLL	JCC	JCR	JLL	JCF	JCR	JCR	JCR	JCR	JCF	JCR	JCR	JCR	JZR	JCC	JCC
	0024H	0041H	0010H	0024H	0022H	0014H	0018H	0016H	0019H	001AH	001CH	001DH	001DH	000FH	009EH	009FH
001H																
	361	74	360	366	505	504	511	510	512	513	517	514	518	515	464	104
	1	1	1	1	2	1	2	1	1	1	1	1	1	2	1	1
	JZR	JCC	JCR	JCR	JCF	JCF	JFL	JFL	JZR	JZR	JCR	JCR		JCR	JLL	JCC
	000FH	0011H	0020H	0020H	0062H	0002H	0062H	0002H	000FH	000FH	002BH	0029H		002FH	0014H	004FH
002H																
	508	73	506	507	370	371	372	373	581	477	472	476		459	502	460
	2	1	1	1	3	3	3	3	1	1	1	1		1	1	1
	JZR	JZR	JZR	JZR	JZR	JZR	JZR	JZR	JZR	JZR	JZR	JCC	JCC	JCC	JCC	JZR
	000FH	0005H	0004H	000BH	0005H	0004H	0005H	000BH	0007H	0001H	0004H	000BH	005CH	004DH	006EH	000BH
003H																
	286	290	291	328	292	293	309	329	301	295	310	330	380	411	406	331
	3	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2
	JCC	JCR	JFL	JZR	JCC	JCC	JCC	JCR	JZR	JCR	JCR	JCC	JCC	JCR	JCR	JZR
	0070H	0044H	0002H	000FH	00F4H	0085H	0056H	0045H	000CH	0048H	0049H	005BH	003CH	0040H	004CH	000EH
004H																
	413	75	304	305	79	208	195	207	87	86	85	336	564	412	563	462
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	JCC	JFL	JFL	JLL	JRL	JZR	JCR	JCC	JCR	JCC	JCR	JCR	JCR	JCC	JCR	JCR
	0060H	0052H	0012H	0024H	007CH	000AH	0055H	0047H	0059H	0069H	0051H	005AH	005DH	00ADH	0056H	0058H
005H																
	280	346	356	351	177	182	201	206	597	598	341	337	381	382	217	593
	1	1	1	1	1	2	2	1	1	1	1	1	1	1	1	1
	JPR	JPR	JZR	JZR	JZR	JZR	JCC	JCR	JCC	JCR	JZR	JRL	JCR	JCR	JZR	JCC
	0030H	0030H	000FH	000FH	000FH	000FH	00F6H	0066H	00F8H	006CH	0009H	007CH	006DH	006FH	000DH	00AFH
006H																
	281	272	321	322	236	237	238	242	243	599	189	184	603	607	407	608
	1	1	4	4	1	1	1	1	1	1	1	1	1	1	1	1
	JCC	JFL	JZR	JZR	JZR	JZR	JCR	JCR	JCC	JCC	JCR	JCC	JCC	JCC	JLL	JLL
	00F0H	0061H	0062H	000FH	000FH	000FH	0078H	0079H	00A8H	00F9H	007BH	00EBH	009CH	009DH	0064H	0074H
007H																
	415	271	298	299	250	251	255	262	256	263	386	388	224	229	234	248
	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3

MICROPROGRAM MEMORY IMAGE

	0H	1H	2H	3H	4H	5H	6H	7H	8H	9H	AH	BH	CH	DH	EH	FH
	JZR	JCC	JCC	JZF	JCC	JCC	JCC	JCC	JCC	JZR	JZR	JZR	JLL	JRL	JCC	JZR
	000FH	0071H	00E2H	00B2H	0054H	0055H	0046H	0057H	0008H	0008H	0006H	0006H	00C4H	00CCH	005EH	000FH
008H																
	115	270	277	121	176	181	194	205	579	582	583	585	423	493	212	116
	1	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1
	JCR	JCC	JLL	JLL	JCR	JCR	JCR	JCR	JCR	JLL	JCR	JLL	JLL	JLL	JCC	JPX
	0091H	0021H	00B4H	00A4H	0093H	0098H	0095H	0093H	0094H	0094H	0099H	0094H	00D4H	00E4H	00FEH	0080H
009H																
	68	69	549	540	532	533	535	536	534	529	528	530	225	230	466	110
	1	1	4	6	3	3	2	2	1	1	1	1	1	1	1	1
	JCF	JCF	JZR	JZR	JFL	JCR	JCR	JFL	JCC	JFL	JCC	JCR	JCR	JCR	JCC	JCR
	0092H	0092H	000FH	000FH	0092H	00A0H	00A1H	0092H	00B8H	00CAH	007AH	00AEH	00ABH	00AAH	002EH	00ADH
00AH																
	544	546	552	553	542	543	545	547	257	572	385	500	499	384	501	609
	1	1	2	2	1	1	1	1	1	1	1	1	1	2	1	1
	JCR	JRL	JCR	JRL	JZF	JCR	JCR	JZF	JZR	JCR	JZR	JCR	JCC	JCF	JCR	JCC
	00B4H	00ECH	00B1H	00ECH	00A2H	00B0H	00B0H	00A2H	000FH	00BAH	000FH	00BEH	00ACH	00BAH	00B9H	005FH
00BH																
	555	126	125	130	551	554	556	557	258	402	404	400	498	398	401	592
	2	1	1	1	2	1	1	1	1	1	3	2	1	1	1	1
	JCR	JCE	JZR	JZR	JCR	JCE	JZR	JCR	JRL	JCC	JZR	JZR	JCC	JZF	JCC	JCR
	00C1H	00E1H	000FH	000FH	00C0H	00F5H	000FH	00C8H	00DCH	00D9H	000FH	000FH	00BCH	009AH	004EH	00C9H
00CH																
	428	429	148	149	427	434	439	447	448	570	573	574	497	527	562	569
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	JCC	JZR	JZR	JCR	JZR	JZR	JCR	JCC	JCC	JCC	JZR	JCR	JCR	JCC	JCR	JCR
	0030H	000FH	000DH	00D8H	000FH	000FH	00D1H	00F7H	00E8H	00A9H	000FH	00DDH	00D3H	002DH	00DCH	00D0H
00DH																
	484	156	393	451	153	154	155	146	452	571	479	456	450	458	481	483
	1	1	1	1	2	2	2	2	1	1	1	1	2	2	1	1
	JCC	JZR	JCR	JCR	JZR	JZR	JCC	JCC	JCR	JCF	JCC	JCC	JLL	JLL	JZR	JZR
	0050H	000FH	00E0H	00EAH	000FH	000FH	00F6H	00F7H	00E9H	00DAH	002AH	00FBH	00D4H	00E4H	000FH	000FH
00EH																
	279	430	278	470	160	161	166	171	454	455	471	389	134	135	140	141
	1	1	1	1	2	3	2	2	1	1	1	1	2	2	2	2
	JCR	JZR	JCC	JCC	JCR	JCC	JZR	JFL	JCR	JCR	JCC	JCR	JCF		JCR	JCC
	00F1H	000FH	00D2H	00E3H	00FAH	00E5H	000FH	00C2H	00F7H	00F7H	004AH	00F2H	00BAH		00F3H	00BFH
00FH																
	416	417	392	468	80	435	167	147	244	264	81	390	93		467	591
	1	1	1	1	1	1	2	4	1	1	1	1	1		1	0

APPENDIX C CENTRAL PROCESSOR SCHEMATICS



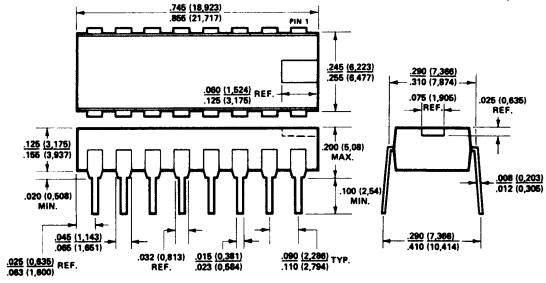
ORDERING AND PACKAGING INFORMATION

ORDERING INFORMATION

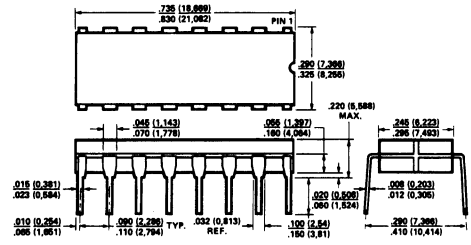
		Standard Package Type		
Component	**No. Of Pins**	**Ceramic (C)**	**CerDIP (D)**	**Plastic (P)**
3001	40	Yes	Yes	
MC3001			Yes	
3002	28	Yes	Yes	
MC3002			Yes	
3003	28	Yes	Yes	
MC3003			Yes	
3212	24		Yes	Yes
MD3212			Yes	
3214	24	Yes	Yes	Yes
MD3214			Yes	
3216/26	16		Yes	Yes
MD3216/26			Yes	

PACKAGE OUTLINES

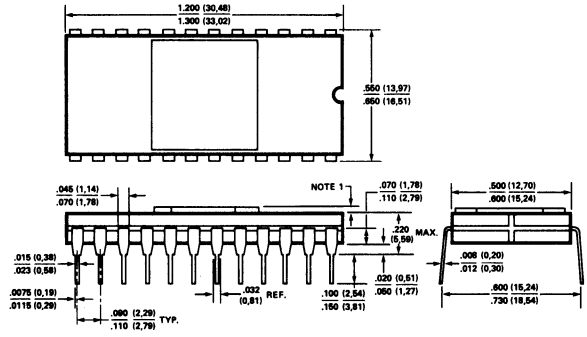
16-LEAD PLASTIC DUAL IN-LINE PACKAGE (P)



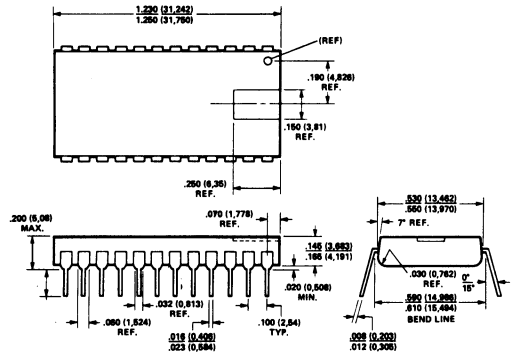
16-LEAD CerDIP DUAL IN-LINE PACKAGE (D)



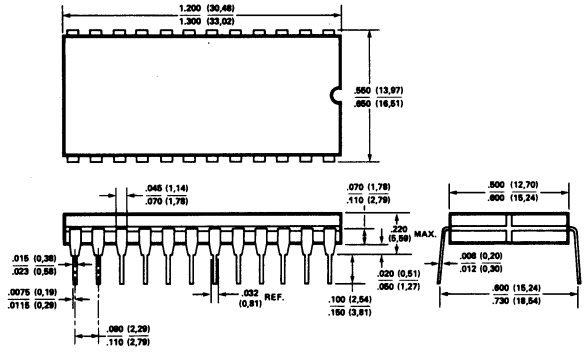
24-LEAD CERAMIC DUAL IN-LINE PACKAGE (C)



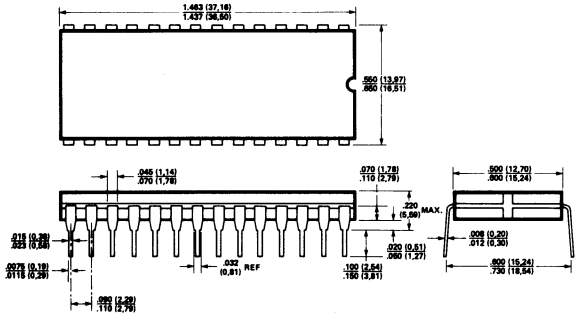
24-LEAD PLASTIC DUAL IN-LINE PACKAGE (P)



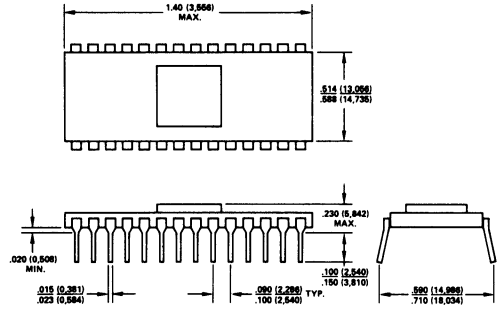
24-LEAD CerDIP DUAL IN-LINE PACKAGE (D)



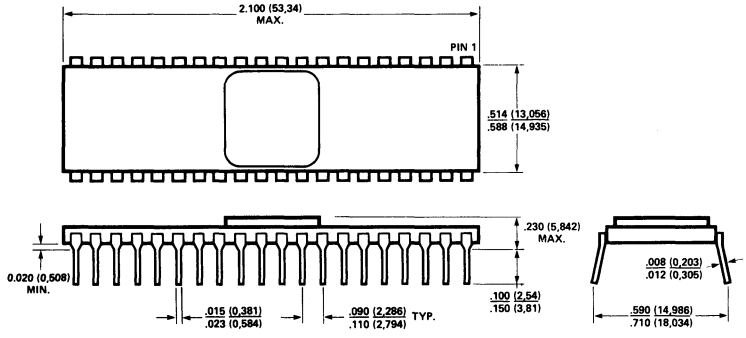
28-LEAD CerDIP DUAL IN-LINE PACKAGE (D)



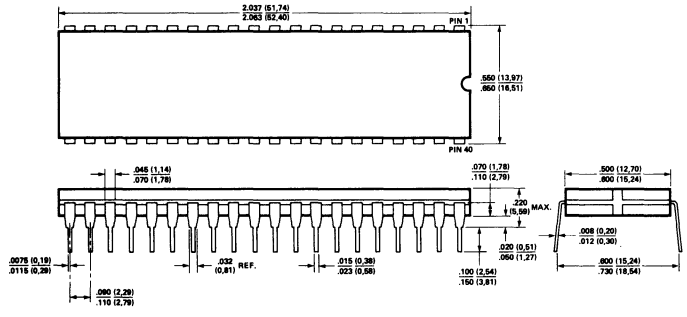
28-LEAD CERAMIC DUAL IN-LINE PACKAGE (C)



40-LEAD CERAMIC DUAL IN-LINE PACKAGE (C)



40-LEAD CerDIP DUAL IN-LINE PACKAGE (D)



INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 246-7501

Printed in U.S.A. MCS 048-0276/10K