$10.00

TECHNICAL REFERENCE MANUAL
for the
BEST OPERATING SYSTEM

MARCH 1977 - PRELIMINARY PRINTING

QANTEL CORPORATION

TABLE OF CONTENTS

SECTION	TITLE	PAGE
	3. Keyed Only	5.1-3a
	4. Contiguous	5.1-4a
	5. Object	5.1-5a
	6. Standalone	5.1-6a
	7. BEST Directory	5.1-7a
5.2	File Operations	
	1. CREATE	5.2-1a
	A. Calculation of Record Size	5.2-1b
	2. ERASE	5.2-2a
	3. OPEN	5.2-3a
	A. Disc Files	5.2-3a
	B. Peripheral Devices	5.2-3b
	4. CLOSE	5.2-4a
	5. READ	5.2-5a
	A. Disc Files	5.2-5a
	B. Keyed Only Files	5.2-5b
	C. Key and Record Lengths	5.2-5c
	D. Peripheral Devices	5.2-5c
	6. WRITE	5.2-6a
	7. READ/WRITE IMAGE	5.2-7a
	8. DELETE	5.2-8a
	A. Key Sectors	5.2-8a
	B. Data Sectors	5.2-8a
	9. KEY/ORD	5.2-9a
	10. EXTRACT	5.2-10a
	11. UPDATE	5.2-11a
	12. LOCK/UNLOCK	5.2-12a
	13. GET/PUT	5.2-13a
5.3	Disc Layout and Handling	
	1. Software Initialization	5.3-1a
	2. Disc Layout on a BEST Pack	5.3-2a
5.4	System Operations and Control	
	1. Sector Allocation	5.4-1a
	2. File System Control	
	A. In Memory	5.4-2a
	B. On Disc	5.4-2c
	3. Physical Structure of Pointers	5.4-3a
	A. Key Pointers	5.4-3a
	B. Directory Entry Pointers	5.4-3a
	4. Key Splits	5.4-4a
	A. Other than the Top Key Sector	5.4-4a
	B. Top Key Sector	5.4-4d
	5. Directory Updates	5.4-5a
	6. End of File Detection	5.4-6a

1.0 INTRODUCTION TO THE MANUAL

The purpose of the BEST Manual is to fully document the features of the BEST Operating System and its interaction with Qantel hardware. Used in conjunction with the QIC Technical Reference Manual and the REAL Assembly Language Manual, this manual will provide more specific information about the "internal workings" of statements allowed in either language and thus, a more effective tool in optimizing user programs. Additionally, this manual provides information about many of the unique Operating System features that make BEST a highly effective software system.

The BEST Manual will be periodically updated as new features are added to the operating system. Updates should be filed in the appropriate section of the Manual to provide current documentation for software personnel.

2.0 A GENERAL INTRODUCTION TO BEST

2.1 The Overall Picture

1. Hardware

With any Qantel Computer, the following configuration is standard:

Entry Device	- Some method to enter data to be acted upon by the Central Processing Unit, usually a QCRT or Printing Terminal.
Storage Device	- Some medium used to save data from operation to operation, or day to day, such as a disc or tape.
Output Device	- Used to provide hard copy of computer data, usually a line printer or any of the auxiliary character printers.
Memory	- Temporary storage medium. That part of the computer where programs are loaded to execute, and where data manipulations are performed.
Central Processor	- That part of a computer that controls all other parts using instructions that are hardwired into a ROM (Read Only Memory).

2. Software

Qantel provides a specialized system of software, designed to work with its full line of processors. This software system provides:

1) QIC Language - An easy, english-like language used to develop business application systems. Includes associated system utilties used to enter and maintain the QIC Source Code.

2) Compiler - The program that converts the Source Code into machine understandable object code for execution.

3) BEST - The Operating System that executes the Object code and provides file management, input/output control, and multi-user operations.

2. Software (cont)

Additionally, the REAL Language and Assembler are provided as the assembly language alternative to QIC. The QIC Language and Compiler are discussed in the QIC Technical Reference Manual. The REAL Language and Assembler are discussed in the REAL Assembly Language Manual. The following pages provide information on the BEST Operating System.

3. The Elements of BEST

The BEST Operating System is comprised of the Scheduler, Runtime, File Management, and Input/Output Subsystems:

The Scheduler

The Scheduler handles the functions of starting, switching and stopping the execution of programs, as well as assigning possession of non-sharable system resources.
(See Section 3.0 for a detailed discussion of the Scheduler Subsystem).

The Runtime Subsystem

The Runtime portion of the system is primarily the user interface with the File and Input/Output Subsystems. Runtime allows the user to request system actions in a simple, concise form such as a QIC Statement or REALLINK Macro Call. Runtime minimizes memory requirements of a user program and standardizes the calling mechanism, making the user program independent of changes in the operating system. Also included in Runtime is a collection of routines which provide a variety of special purpose functions supplied in the QIC Language. Maintaining these routines in the BEST system reduces the memory requirements for the user program. (See Section 4.0 for a detailed description of the Runtime System).

The File Management Subsystem

The BEST Operating System uses the disc as a mass storage device for programs and data. All programs and data are considered to be FILEs. A FILE has a name for unique identification, and contains RECORDs. All RECORDs within a file are of the same length (same number of characters or bytes), and have further identifiers such as KEYs or ORDINALS or POSITION in the file that make them unique.

3. The Elements of BEST (cont)

The File Management subsystem handles all accesses to the disc for CREATEing and ERASEing files, OPENing and CLOSEing files, or READing, WRITEing, or DELETEing records within an existing file. The user's program deals with the file names and record keys and is not concerned with the physical location of the record on the disc. The types of files supported by the BEST Operating System are:

Sequential - This type of file contains data records written in sequential (chronological) order and retrieved in the same order. Sequential files afford higher access speed and require less disc space than keyed files, but are restricted in use by their nature, i.e., a single record in the file cannot be located directly, each record in the file must be examined in sequence. (See Section 5.1.1).

Keyed - Keyed file organization affords the greatest accessing flexibility in that any record in the file can be found with equal ease. Each record has a unique identifier, a KEY, that is used to access the data record. (See Section 5.1.2).

Contiguous - This type of file combines the access speed of a sequential file and the flexibility of a keyed file. Data records are written sequentially and assigned a record number or "ordinal" based on the position within the file. There is no overhead for key sectors, but the records may be accessed directly if the ordinal is known. Unlike other file types, the boundaries of a Contiguous file are set up when CREATEd. (See Section 5.1.4).

Object - Executable programs are stored on disc as object files, where their file names are used in a RUN statement which starts their execution. (See Section 5.1.5).

3. The Elements of BEST (cont)

The Input/Output Subsystem

The Input/Output Subsystem is an interface between a user program and the various I/O devices, thus relieving the user program of the need to deal with I/O devices directly. It receives a standard calling sequence from a QIC/REAL program, where the I/O device is OPENed as a file and is either "written to" or "read from". Customized driver programs exist in the I/O Subsystem to handle the different I/O devices. These drivers convert the generalized interface from the QIC/REAL program to the unique commands and status tests for a particular device.

Drivers exist for all Qantel supported devices. These drivers all interface to the program executive in the same manner, so the actual device being accessed is transparent to the executive. (See Section 6.0 for a detailed description of the I/O Subsystem).

2.2 How to Begin

1. IPL and Bootstrap

The IPL button located on the front of the Qantel Computer provides a very specialized function. Pressing the IPL button will:

1) Reset all devices
2) Generate a Read Hex Instruction from device 0
3) Execute the instruction entered

The machine language instruction, 002281XY, entered from device 0, is referred to as the "Disc Bootstrap". The elements of this instruction are:

0022 - Read/Branch Location. This memory address contains the first loader instruction, once the loader has been read into memory.

81 - Machine language operation code. When executed it breaks down into 3 instructions:

a) Seek to sector 0
b) Read into memory
c) Branch to the first loader instruction

X - Refers to the platter of the disc being accessed, where:

X=0 for a fixed platter or 30MB, and
X=1 for a removable platter

Y - Refers to the device number of the disc.

On execution of the "disc bootstrap", the contents of sectors 0,1, and 2 are read into memory, and execution of the program loader begins.

The Qantel Model 1300 initiates a "disc bootstrap" to disc 0D on IPL and Transmit. To bootstrap any other disc device, enter only the device number (0C,1C,1D) and transmit. The full disc bootstrap may also be used.

Pressing the IPL button during system operation completely resets the system, devices, and in essence, memory. Although memory is not cleared, all pointers to the last operations are lost. IPL does not

1. IPL and Bootstrap (cont)

provide an organized shutdown of the system. To protect all operations IPL should always be preceded by a Flag 3/Transmit.

The command word IPL executed at the monitor prompt "READY::" does essentially the same operations as pressing the IPL button. The command word does not reset all devices, only printing terminals. But the command word is executed from the monitor which closes all files and provides a safer means to IPL the system. The IPL command will only function when all controlling terminals are CLOSEd, or executing *MONITOR, and all Background partitions are free. (See *IPL, Section 2.4.2).

2. The Program Loader

The Program Loader is the initial interface to the Qantel Software System. It provides the ability to search a program directory for a specified program name, and load that program at a specified address in memory. Two types of programs can be loaded by the program loader:

Core Image Program	-	A System program that resides in a fixed location on disc, and must be loaded without modification into a fixed location in memory to be executed. Core Image Programs are not part of the BEST Directory.
Standalone Object Program	-	An assembly language program that is accessed through the BEST Directory and performs its own I/O Operations; does not run under Best.

Core Image programs have a separate and distinct directory (in sectors 11-15) that contains the names and load information for all Core Images. Examples of Core Image programs are BEST, QIC, DKIN and CFIG. Standalone Object Programs are the result of the REAL assembler, and are maintained in the BEST directory. Examples of Standalone Object Programs are *BACKUP, *LDLD, and *CIUT.

When a program name is entered at the loader prompt message, the Core Image Directory is searched for the specified program name. If the program is not found, the BEST Directory is searched. If the program name is found in the BEST Directory, the file type must be $30 in order to be loaded. If not, the message "INV TYPE" will be displayed. If a program name of all blanks (XM) is entered at the loader prompt, the loader will automatically load BEST.

The loader resets each printing terminal and parity memory device; then rewrites all of memory in place (up to 128K on the Model 1300). Finally, it enables parity error checking and error storage (fuse blowing) on each memory device. This operation occurs while the operator is typing a filename to load. (If the loader is called by a program to run an overlay, this device initialization does not occur.)

Operating instructions for all programs executable from the program loader prompt are available in the BEST Utility Manual, and the QIC Technical Reference Manual.

2. The Program Loader (cont)

A. Error Messages

Error messages provided by the program loader are:

1) NOT FOUND
Program name entered was not found in the Core Image or BEST Directory.
Press Return or Transmit to clear the screen for a new entry.

2) DISC ERROR XXXXX
While searching the Core Image or BEST directory, or while loading a Standalone Object Program, a disc error occurred.
Press START/STOP to retry the sector.

3) INV TYPE
Program name was found in the BEST directory, but the file type was not $30.
Press Return or Transmit to clear the screen for a new entry.

4) LOAD ERR
While loading a Standalone Object Program, an invalid record type was encountered. (Similar to BEST Error 91).
This error usually means the object file has been destroyed.

5) START/STOP
A hard halt with no error message means a disc error occurred while loading the Core Image. Reload the Operating System.

2. The Program Loader (cont)

B. Loader and Core Image Utilities

*LDLD

This program will create a program loader on any pack. It also provides the ability to change the loader prompt message. (See the BEST Utility Manual for Operating Instructions).

*CIUT

This program provides the means to change or list the Core Image Directory. It is normally used to build a Core Image Directory on a user pack, or to list the contents of the Core Image Directory. (See the BEST Utilities Manual for Operating Instructions).

2.3 Under User Control

1. Definition of a User

Under the BEST Operating System, all attributes of a particular hardware system are defined prior to loading the operating system. These attributes include the description of all terminal devices, partitions, and other peripheral devices.

In the process of defining a user, several conventions must be considered. The BEST Operating system works in a "fixed partition" environment, e.g., a specific segment of memory between two absolute addresses is reserved for execution of a single program at one time. This fixed partition may or may not be controlled by a specific terminal device. If a partition is not controlled by a terminal it is a "Background" partition; a "Foreground" partition is assigned to a particular terminal device. At the same time, a terminal device may or may not control a partition. If a terminal device does not control a partition it is a "non-controlling" or "passive" device; a "controlling terminal" is always associated with the same partition. Under the BEST Operating System, a USER is defined as a controlling terminal.

In this manner, to refer to a 5-User system is to define a system with 5 controlling terminals. This particular system may have 10 terminal devices (5 passive terminals), and be capable of running 10 separate tasks (5 foreground partitions and 5 background partitions).

In the Core Image Program, CFIG, partitions and terminals are defined separately. Each partition is defined by its absolute memory addresses and assigned a name "P00", "P01", etc. In turn, each terminal device is described by its terminal name, device number, terminal type, and the partition name to which it is assigned. (See Appendix B for Operating Instructions for CFIG).

2.3 Under User Control

2. The Configurator

A. Memory

The section on how the Configurator builds its tables within memory will be written at a later date.

2. The Configurator (cont)

B. Peripheral Devices

In the same manner as users are defined prior to using the BEST Operating System, so are all peripheral devices. All devices present on a particular hardware system are manually set to a unique hardware device number. Using the Core Image Program, CFIG, each peripheral device is assigned a "device name", and that device name is associated with a specific device number. The Configurator uses this information to build a device table available to all users. When any program requires a device such as a printer or magtape, that device is OPENed by its device name in the program. That device name is then unavailable to any other user until it is CLOSEd. This applies to all devices except the disc.

Peripheral devices are assigned device names during Configuration. The order of configuration determines the last digit of the device name.

DEVICE	DEVICE NAME	
Terminals	TXX	(T00,T01...)
Partitions	PXX	(P00,P01...)
Discs	DKX	(DK1,DK2...)
Card Readers	CRX	(CR1,CR2...)
Magnetic Tapes	MTX	(MT1,MT2...)
Clock	CL1	(Only one allowed)
Communication Line	CM1	(Only one allowed)
Printers	LPX	(LP1,LP2...)

where X is a numeric digit in the range 1 through 9. (0 is permitted only in the case of terminals and partitions).

For all CRT devices, two device numbers are relevant: a) The CRT controller device number, and b) the device number of the CRT on that controller. Incorrect assignment of these device numbers will cause the terminal to not respond to Flag3/Transmit. (See Appendix B for CFIG Operating Instructions).

Peripheral devices will generate an "inoperable" condition if configured with a device number that is not present on the system. "Unpredictable results" will occur if: a) Devices are configured with a device number that is present on the system, but incorrect, and b) two devices are manually set to the same device number.

3. Partition Control

A. Task Headers

Every partition has associated with it a "Task Header" which contains all the information necessary for proper execution of a job in that partition. The Task Header provides the means for a job to be suspended while another task is performed, and then restart where it left off. When a task is activated by the system, the Task Header is moved into working storage. Only one Task Header is in working storage at any one time. When interrupted, or the next task is scheduled, the elements of working storage are moved back into the permanent Task Header until that task is scheduled again.

3. Partition Control (cont)

B. File Control Blocks

Eight File Control Blocks (FCBs) are associated with every partition. An FCB directly corresponds to a Logical Unit Number (LUN) that is OPENed in a program. The FCB is a user's "pointer" table that points to the common system tables for files and devices.

3. Partition Control (cont)

C. Active File List

If the OPENed LUN references a file, the FCB contains the current position in that file and a pointer to the associated file in the Active File List (AFL). The AFL is a table of all files currently OPEN by all partitions. This table provides the means for multiple terminals to work on the same file at the same time.

It is through the AFL that all access really takes place. The AFL contains the file directory header which is updated by any partition changing that file, so all paritions have access to the same copy of the file header. The AFL also contains an OPEN/CLOSE count which is incremented when a file is OPENed, and decremented when it is CLOSEd. This OPEN/CLOSE count is used to determine when the directory on disc is updated.

The maximum number of AFL entries possible for a specific system is defined during Configuration. This should be the maximum number of unique file names that will ever be OPENed by all users at any one time. If the number of AFLs configured is exceeded during operation, Best Error 86 will be generated, indicating an "Active File List Overflow". This is a fatal error and can be corrected by decreasing the activity on the system, or by configuring more AFLs. However, the number of AFLs should not be determined arbitrarily, since that space is available for other devices and, in some configurations, for partitions.

3. Partition Control (cont)

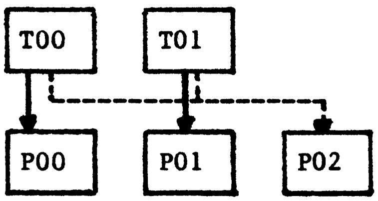
D. Device Descriptor Table

If the LUN on OPEN references a device, the FCB contains a pointer to the Device Descriptor Table (DDT). This table is established by the configurator and contains vital information (e.g., device number, default record length, etc.) for accessing any particular device configured into the system.

When a device is OPENed by a particular partition, the Task Header address of that partition is stored in the DDT for that device. No other partition may access that device unless its task header address matches that in the DDT (i.e., same partition). In this way, a device is "locked out" to any other partition until that device is CLOSEd and the DDT address cleared.

4. Background/Foreground

BACKGROUND is defined as the ability to execute a program in a partition that is not controlled by an input device. A Background partition is defined by the Configurator as a partition with no controlling terminal. A program is "started up" (ACTIVATEd) in a Background partition by another partition, either Background or Foreground. Once the Background partition is ACTIVATEd it can RUN overlays or any other program that does not attempt to WRITE to TERM$ or LUN 0 (unless it is OPENed intentionally for a file or device).



T00 and T01 are Controlling Terminals to Partitions P00 and P01. Partition P02 is a Background Partition and can be ACTIVATEd by any program in partitions P00 or P01.

——— Direct terminal to partition relationship.

------ Ability to ACTIVATE Background partition.

Two command words are associated with a Background Partition:

ACTIVATE - Starts up a program in a Background or Foreground partition, if that partition is Clear. Functions as a "Remote" RUN Statement.

TERMINATE - Ends a program running in a Background Partition. TERMINATE will function from the program running in the Background Partition, or can be initiated from another controlling terminal. Functions as a "Remote" Escape (Flag3/Transmit for QCRTs, Flag1 for Printing Terminals). A Foreground partition can only TERMINATE itself.

There are four "states" for any partition:

CLEAR - No program running. Partition is available for any program.

PROGRAM RUNNING - A program is currently running in the partition. If Background, this partition must be TERMINATEd before being ACTIVATEd; if Foreground, this partition is unavailable.

NORMAL TERMINATION - A STOP or END was issued by the program running in this partition. If Background,

4. Background/Foreground (cont)

this partition must be TERMINATEd before being ACTIVATEd; if Foreground, this partition may be ACTIVATEd.

ERROR TERMINATION - An unexpected (not-handled by EXCP branch) error was encountered in the program running in this partition. All error information from *ENDITOR can be retrieved by *CONSOLE. If Background, this partition must be TERMINATEd before being ACTIVATEd; if Foreground, this partition may be ACTIVATEd.

A Background partition can only be placed in the CLEAR state through a TERMINATE command from itself or another partition. If a Background partition only TERMINATEs itself on successful end of job, then the partition will be CLEAR for the next Background job. In the case of an unexpected error, information about the error will be available through the utility *CONSOLE, until the partition is TERMINATEd. A Foreground Partition can TERMINATE itself, which causes *MONITOR to be loaded into the partition. *MONITOR running in a Foreground Partition makes that partition busy. A Background partition can only be ACTIVATEd if it is CLEAR while a Foreground partition can be activated if it is CLEAR, at NORMAL TERMINATION, or ERROR TERMINATION.

All information for any partition is available through the Utility, *CONSOLE.

The System Variable, PARTITION$, contains the name of the partition in which the program is running. PARTITION$ may be used to create unique filenames or keys for the same program running in different partitions. The System Variable, ITERM$, contains the value of the initiating terminal for the partition. Whenever an ACTIVATE is performed, the value of TERM$ for the partition initiating the ACTIVATE is placed in ITERM$ for the partition being ACTIVATEd. TERM$ for a Background Partition will always be " ". The value of TERM$ can be tested within a program to determine whether the partition in which the program is executing is Background, e.g.,

```
IF TERM$ NE '  ' PRINT (0,100)
```

See Appendix C for the QIC Syntax of ACTIVATE and TERMINATE commands; see Appendix D for the REAL Syntax of ACTIVATE and TERMINATE commands.

1. *MONITOR

*MONITOR

*MONITOR is loaded into a user partition on Flag3/Transmit (or Flag 1). This program is the initial interface to a terminal for loading programs under BEST. When *MONITOR is loaded in a partition, the prompt message "READY::" is displayed on the bottom line of the CRT, or the current line of a Printing Terminal.

When BEST is requested at the Core Image Loader prompt, the Core Image Directory is read to determine the starting sector for the BEST Core Image. The loader issues a seek to that sector and begins reading the Core Image into memory for the appropriate number of sectors. The System sets up a table of configured terminals and partitions (from CFIG) in memory, and loads *MONITOR into the first configured partition ("P00"). The "READY::" prompt will appear on the terminal associated with partition "P00".

Once the "READY::" prompt is issued, control is passed to the scheduler, and the terminals set up in the table are scanned, waiting for either a Flag3/Transmit from any terminal, or a read request from the terminal attached to "P00". On Flag3/Transmit from any terminal, *MONITOR is loaded into the associated partition.

*MONITOR closes all LUNs and OPENs TERM$ on LUN 0. Any user supplied programs substituted for *MONITOR should follow the same procedure. *MONITOR uses CRT and Printing Terminal devices in typewriter mode (ET). Exiting from *MONITOR leaves QCRT devices in normal mode (EN), and Printing Terminals in typewriter mode (ET).

From the "READY::" prompt issued by *MONITOR, several actions from the terminal are possible:

1) RUN PROGRAM, (XXX), ("MESSAGE") , where:

PROGRAM = The program name to be executed under BEST

XXX = (Optional) The disc label from which the program should be loaded. Default is the first configured disc where the program name is found.

"MESSAGE" = (Optional) Any message to be passed to the program and loaded into the first String Common variable declared in the program. The message must be contained between two

1. *MONITOR (cont)

successive occurrences of the same non-blank character, (e.g., ").

2) IPL COREIMAGE, (NX), ("MESSAGE") , where:

COREIMAGE = The name of the Core Image or Standalone program to be loaded by the Core Image loader, if no other partitions are active (i.e., running any program besides *MONITOR).

NX = (Optional) The disc device number from which the Core Image or Standalone program should be loaded. Default is the IPL device from which BEST was originally loaded.

"MESSAGE" = (Optional) Any message to be passed to the Core Image or Standalone program, e.g.,

```
IPL QIC,0D,"COMMFILE,CDISC,CKEY"
IPL BEST,0D,"*EDIT    DSK'
```

The message must be contained between two successive occurrences of the same non-blank character, (e.g., ").

*MONITOR, on recognition of the keyword, IPL, passes control to *IPL to perform the IPL function.

3) CLOSE

Executes the "END" statement. This statement causes the system to run *ENDITOR which produces the message "PROGRAM END. USE ESCAPE TO START". CLOSE deactivates the terminal and the associated partition, and closes all LUNs (0-7). After CLOSE, the terminal may be OPENed by any other partition.

4) ACTIVATE PARTITION, PROGRAM, (XXX), ("MESSAGE") where:

PARTITION = The name of the partition to be ACTIVATEd, (e.g., "P01", "P02", etc.)

PROGRAM = The program to be loaded into the partition being ACTIVATEd.

XXX = (Optional) The label of the disc from which the program should be loaded.

1. *MONITOR (cont)

"MESSAGE" = (Optional) Any message to be passed to the program being ACTIVATEd.

The message must be contained between two successive occurrences of the same non-blank character, (e.g., ").

The "MESSAGE" passed at the RUN, IPL or ACTIVATE command can be as long as the number of available characters on the transmit line. All characters within the delimiters are passed, including trailing blanks. If the command is IPL BEST the first 11 characters of the message are assumed to be the program name (8 bytes) and the disc label (3 bytes). The remaining characters are a message to that program. If a disc label is specified, the program name must be 8 characters, so trailing blanks must be appended to any name less than 8 characters. *MONITOR sets the first String Common variable to the empty String if no message is specified.

"MESSAGE" passing between Core Images is accomplished by writing the message to sector 10 of the destination disc. The disc loader is loaded into memory, the length of the message is moved into bytes $28 and $29 of memory, and the Core Image Loader is executed. The new program checks locations $28-29 of memory and, if it is non-zero, sector 10 is read by the new program to retrieve the message.

2. *IPL

*IPL is executed by *MONITOR when the command word IPL is recognized at the "READY::" prompt. *IPL performs a disc bootstrap and load of the Core Image program specified in the IPL command.

This program also closes all files for the user, and tests for any other partitions active on the system. If any partitions are running a program besides *MONITOR, an Error 60 is generated. *IPL will then process any message passed to it in the IPL command. The syntax of the IPL command is:

IPL (COREIMAGE),(DEVICE),("MESSAGE"), where:

COREIMAGE, DEVICE, and MESSAGE are optional parameters.
If no parameters are specified the program will only perform a disc bootstrap to the disc which was originally bootstrapped. (See *MONITOR, Section 2.4.1).

If a Core Image name is supplied, *IPL will attempt to load that specified name. Error messages are as explained in the section on the Core Image Loader.

If a device number is specified, *IPL will perform the disc bootstrap to the requested device. If the device number is incorrect (invalid hex digits, non-disc, unavailable), the program will execute a disc bootstrap to the disc currently executing BEST.

If all information is so far correct, and a message is included in the IPL command, the MESSAGE is written to sector 10 of the requested disc device with all delimiters omitted.

Finally, all device numbers in the hardware system are checked. If any are parity memory devices, the parity interrupt mechanism is temporarily disabled, to be subsequently enabled by any program capable of handling the interrupts. Additionally, a Reset I/O is issued to all Printing Terminal devices, clearing pending READs.

3. *ENDITOR

*ENDITOR provides the error message description displayed when an error occurs in a program that is not handled by an exception branch. The Error Message format is:

ERROR #;LUN # 'FILE', 'DISC'; INDEX; SYSTEM FUNCTION #; NEXT USER ADDRESS; PROGRAM NAME; ERROR DESCRIPTION; SYSTEM FUNCTION DESCRIPTION

where:

ERROR #	- The BEST Error Code encountered
LUN # 'FILE', 'DISC'	- The last LUN referenced when the error occurred, and the file or device assigned to that LUN.
INDEX	- (Supplied if relevant.) The next index (Key or Ord) in the file if the access was sequential, or the specified index if the access was indexed.
SYSTEM FUNCTION #	- A system defined number indicating the type of the last system function attempted before the error occurred.
NEXT USER ADDRESS	- The next user address from the QIC compiler listing (executable code) that would have been executed, if the error had not occurred.
PROGRAM NAME	- The name of the program being executed when the error occurred.
ERROR DESCRIPTION	- The literal description (from *ERRFILE) of the error that occurred.

3. *ENDITOR (cont)

SYSTEM FUNCTION DESCRIPTION	- The literal description (from *ERRFILE) of the last system function executed when the error occurred.

When an error occurs that is not handled by an exception branch in the user program, the system calls the ERROR routine to CLOSE all files and build the error message. This message is then passed to *ENDITOR, which OPENs TERM$. For hard disc errors, the standard disc error message is built and displayed on TERM$. This message provides:

Device XX
Sector XXXXX
Status XX

On non-disc errors, all information except the description is displayed. If the directory of any configured disc contains *ERRFILE, *ENDITOR reads the file and displays the description of the error and the System function. No description is displayed if there is no *ERRFILE, but the Error and System Function provided can be checked against a previous listing of *ERRFILE.

*ERRFILE has a record size of 40 and a Keysize of 3. It can be printed using *QDUMP to obtain a current list of the BEST error codes. *ERRFILE may also be used to provide a file of error messages for application programs by displaying the first 37 bytes of the record obtained by an indexed READ of *ERRFILE, where:

```
IND='E' + SUB(STR(EXCP+100),15,2)
```

4. *CONSOLE

*CONSOLE allows any terminal to examine the current status of any partition or device, and to ACTIVATE or TERMINATE programs in any partition.

On execution of the program from the "READY::" prompt, the following message is displayed:

CONSOLE UTILITY XX.X (MM-DD-YY)
(A,T,L,I,S)::

The program is then ready to accept any of the following command words:

A ACTIVATE PARTITION,PROGRAM,(DISC),("MESSAGE") , where

PARTITION = The partition name to be ACTIVATEd, e.g., "POO"

PROGRAM = The program name to be ACTIVATEd in the partition

DISC = (Optional) The disc name from which the program should be loaded

"MESSAGE" = (Optional) Any message to be passed as the first String Common variable to the program in the partition. The message must be contained between two successive occurrences of the same non-blank character, (e.g., ").

The ACTIVATE command starts up a program in any partition that is not currently busy. If the partition is in any state besides CLEAR, and is a background partition, it must be TERMINATEd before being ACTIVATEd.

T TERMINATE PARTITION , where:

PARTITION = The partition name to be TERMINATEd, e.g., "POO"

The TERMINATE command terminates any background partition (by performing a "remote escape"). If the partition is Foreground, it can only TERMINATE itself. (Program returns to *MONITOR.)

4. *CONSOLE (cont)

L LOG

The LOG command displays the current status of all partitions configured for the system. There are four possible states of a partition:

CLEAR
Partition is available for a new program.

PROGRAM RUNNING: "PROGRAM NAME"
Provides the name of the program currently running in the partition.

NORMAL TERMINATION: "PROGRAM NAME"
Indicates the name of the program that was ended by a STOP or END statement in the program. This is a normal end and if the partition is background, it must be TERMINATEd before being activated again.

ERROR TERMINATION: "PROGRAM NAME" EXCP=XX
Indicates the name of the program that encountered a fatal or unexpected error during processing, and the error encountered.

I INFORMATION PARTITION

The INFORMATION request displays more detailed information about the current activities of any partition.

PARTITION CLEAR
No activity. Partition available for program.

NORMAL TERMINATION: "PROGRAM NAME"
STOP or END encountered in the program name specified. A Background partition must be TERMINATEd, then ACTIVATEd; a Foreground partition may be ACTIVATEd.

ERROR TERMINATION
All information provided by *ENDITOR is available:

Error # XX
Last LUN: "File", "Disc"
Index:_________ (if relevant)
Last system function: XX
Next user address: $XXXX
Current Program: Programname
Error: (Description)
System Function: (Description)

4. *CONSOLE (cont)

and, in the case of a disc error,

```
Device #XX
Sector #XXXXX
Status XX
```

(See Section 2.4.3, *ENDITOR).

(PROGRAM RUNNING)

The following information is available when a partition is executing a program:

```
Current Program        "Programname" at $XXXX
Last File            : (LUN) "File", "Disc"
Last System Function : XX  Description
Last Exception       : XX  Description
Controlling Terminal : "Txx" or "   "
Initiating Terminal  : "Txx" or "   "
```

S STATUS

The STATUS request displays the status of all devices currently connected to the computer, in the form:

```
DEVICE # X -- STATUS : XX
```

Any initial part of the command words for *CONSOLE is an acceptable entry. Any unrecognizable command word will produce a list of the available command words and their required parameters. A null entry exits the program to *MONITOR.

The Scheduler handles the functions of starting, switching, and stopping the execution of programs, as well as assigning possession of non-sharable system resources. This involves:

- Processing RUN requests by loading programs for execution in a user partition.
- Processing system resource requests such as printers, buffers, etc., if the resources are available. If they are not, the Scheduler is responsible for suspending the user until the resource becomes available.
- Recognizing completion of I/O operations requested by users, notifying the user of completion, and re-establishing their "ready" status.
- Executing tasks on a "Round Robin" basis whenever any user is suspended.
- Processing "Escape" requests which normally terminate a program immediately.
- Processing "Errors" or exception conditions. The Scheduler executes user error-handling routines, if specified for non-fatal errors; or terminates the program with a display of information describing the error and the action being performed at the time.
- Processing IPL requests by verifying that no other user is actively executing a program (besides *MONITOR), before calling the Core Image Loader.

The Runtime Section of the BEST Operating System enhances the flexibility of the Language and system operations by:

- Providing the user interface with the File and I/O Systems. This allows the user to request complicated system functions in a simple, concise form such as QIC statements or REALLINK Macro Calls. Runtime fetches, passes, and converts all parameters between the user program and the system.

- Providing functions that are extensions to user programs but, if compiled into a program, would make it prohibitive in size. Instead, the compiler generates in-line a system call to the specific Runtime function.

- Performing certain routines for QIC programs that require parameter resolution not available at compile time. For example, routines such as String Assignment are done by Runtime to insure correct length attributes.

- Allowing the operating system to change without requiring a user program to change. This makes the language independent of the operating system and provides upward compatibility.

In general; addition, subtraction, multiplication, numeric compares, GOTO statements (branches), and FOR/NEXT statements are compiled into user code. All other operations are compiled as Runtime calls.

4.1 QIC/BEST

The section on RUNTIME for QIC/BEST will be added to at a later date.

1. String Variable Handling

A STRING is a series of ASCII characters treated not as a numeric value, but as a literal entity. A STRING VARIABLE is used typically to store alphabetic data such as descriptions, file names, messages, or indexes to Keyed Files. Arithmetic operators (except +) may not be used on string variables and constants. String variables may be shortened (SUB function) or appended together (concatenation, +).

STRING VARIABLES are represented in memory as they appear, with the addition of one byte in which the LENGTH of the string variable is stored. This length is the value returned by the LEN function. The length byte determines the number of characters and blanks that will appear in that string in any subsequent operation.

Although this length byte is the ruling factor for the appearance of the string, there are two methods of definition for this length, based on how the string is filled.

When a string is filled by an unformatted I/O operation such as INPUT, the length of the string is always set to the actual number of characters entered. When a string is filled by a formatted I/O operation such as READ, the length of the string is set to the maximum possible length. This maximum length is determined in the declarative section of the program and is sometimes referred to as the "declared length". The one exception to formatted I/O is in the case of an IMAGE file where every string variable is written to disc with a length byte that contains the actual number of non-blank characters in the variable. When an IMAGE file is read, all string variable lengths in the format will be set to this "actual length".

4.2 REAL/BEST

The section on RUNTIME for REAL/BEST will be included at a later date.

5.0 THE FILE MANAGEMENT SUBSYSTEM

5.1 File Types

1. Sequential Files

Records in Sequential Files are stored in an "as entered" order and can only be retrieved in that same order. Sequential Files provide a fast access, transaction-type file, but do not provide for any type of direct access. The sequential method of storing data is the basic element to all file types under BEST.

When a Sequential File is CREATEd, it is allocated 5 sectors (one Allocation Unit, AU). Each sector is linked to the next assigned sector by the forward link. The File Directory Header contains the first record sector, which is the beginning of data for a Sequential File, as well as the next available sector and offset which is used for a WRITE. Each time a record is written to the Sequential File, the offset is incremented to the next available record position. When the initial five sectors are depleted, the File System gets the next available AU from the AU Map (not necessarily adjacent) and links the last sector of the original AU to the first sector of the new AU. This procedure continues throughout the file, always posting the sector number of the new AU in the forward link of the last sector in the last AU.

When a Sequential File is OPENed, the internal record pointer (in the FCB) is always positioned to the first record in the file. Each successive READ bumps the internal pointer to the next consecutive record. The system generates an End of File (EXCP=2) when the current position in the sector is greater than or equal to the sector displacement (offset) <u>and</u> there is still room in the sector to WRITE a record. End of File is not generated by a forward link of all zeroes. Only the last sector of the last AU will have a zero forward link.

Since the File Directory Header contains the next available sector and position for a WRITE, the system can position itself to that position (End of File) to begin writing. This is accomplished through the use of the (EOF) mnemonic in a FORMAT statement. WRITEing to a Sequential File with a format that contains an EOF mnemonic, immediately positions to the End of File. In the same manner, the (BOF) mnemonic can be used to position the record pointer to the beginning of the file. (A Sequential File Directory Entry is shown on the following page).

1. Sequential Files (cont)

DIRECTORY ENTRY FOR SEQUENTIAL FILES

Byte(s)	Directory Information
1 - 3	First Record Sector
4	(Unused)
5 - 12	Sequential File Name
13	File Type (Sequential File = 01, $01)
14 - 16	Last Record Sector
17 - 18	Record Size
19 - 20	AU Count
21 - 29	(Unused)
30 - 31	Next Record Offset
32 - 48	(Unused)

2. Keyed Files

Any record in a Keyed File can be accessed with equal ease through the use of a record identifier. This record identifier is some value (customer number, invoice number, etc.) assigned to that record at the time it was written.

When a Keyed File is CREATEd, it is allocated 10 sectors (2 AUs: One for keys; the other for data). The Data Section of Keyed File contains the actual records written to the file through a FORMAT statement. Data is maintained in this section in an "as entered" order, exactly as for a Sequential File. The Key Section, or "Key Tree", is a separate part of the Keyed File which contains the record identifiers provided in the IND= expression of a WRITE statement. These identifiers, or keys, are sorted in ascending ASCII order within the Key Tree. Each key is associated with 6 bytes of system overhead, which indicates a) the sector in which the corresponding data record is located, and b) the relative position of that data record within the sector.

Data records in a Keyed File are always accessed through the Key Tree of the file.

In order to provide equal access of any record through the Key Tree, the key sectors are organized in a pyramid fashion. This pyramid maintains progressively higher level pointers that point to ranges of keys. In this way, keys and their associated pointers to data are sorted into unique level 0 key sectors, each level 0 containing a range of key values not duplicated in any other key sector. The File System then builds a higher level sector which contains an entry pointing to each of these level 0 key sectors. This pointer in the higher level sector is actually made up of the lowest key entry in the level 0 sector and the sector number where that level 0 is located. When this higher level sector is full, the File System creates an even higher level, which contains an entry pointing to each of these next lower levels. In this manner, any direct access to the file can check the highest level and determine a "path" down through the key sector levels to finally retrieve the data. The tree is always balanced so that access time for all keys in the file is the same.

There must always be one sector that is the highest level, and which references all next lower levels. This sector is referred to as the "Top Key Sector". The procedure by which the File System maintains the key pyramid is referred to as a "Key Split" (explained in Section 5.4.4). The Top Key Sector is always the first key sector of the Keyed File. Since disc space for Keyed Files is always allocated on an "as needed" basis, the key sectors are scattered throughout the file and do not exist in a contiguous area. Key sectors are linked together primarily by the key pointers. They are also linked in

2. Keyed Files (cont)

allocation order by the "forward link", but this link is used only to return sectors during an ERASE.

The Directory Entry for Keyed Files is shown on the following page.

2. Keyed Files (cont)

DIRECTORY ENTRY FOR KEYED FILES

Byte(s)	Directory Information
1 - 3	Top Key Sector
4	(Unused)
5 - 12	Keyed File Name
13	File Type (Keyed File = 04, $04)
14 - 16	Last Record Sector
17 - 18	Record Size
19 - 20	AU Count
21 - 23	Last Key Sector
24	Key Size
25 - 26	Delete Chain Record Offset
27 - 29	Delete Chain Record Sector
30 - 31	Next Record Offset
32 - 34	First Record Sector
35 - 37	First Sector of First Key AU
38 - 40	Deleted Chain for Key Sectors
41 - 48	(Unused)

3. Keyed Only Files

Keyed Only Files are CREATEd with a record length of 0 and are not allocated a data AU.

The primary purpose of a Keyed Only File is as a pointer file to other keys in other files. Since there is no data look up in this type file, it is at least one disc access faster than a similar Keyed File.

If it is necessary to access one file in two separate orders, a Keyed Only File can preclude the necessity for sorting that file to get the secondary access. For Example, if the primary access to a file is by "customer number", and a secondary access is needed by "order number", the following approach can be taken: Set up the master file with a key of Customer Number + Order Number. Every time a WRITE occurs to the master file, WRITE a second index to a Keyed Only File with a key of Order Number + Customer Number. When it is necessary to find an Order Number in the primary file, READ the Keyed Only File with a "dummy read", specifying the Order Number to position the record pointer just before the Order Number. Use the KEY function to get the value of the next Key, Substring it and reverse the order, then READ the master file with the resulting key.

To READ a Keyed Only File sequentially, first take the KEY function to get the value of the key, then READ with a dummy format to advance the record pointer to the next key. If the READ does not occur, the KEY function will continually return the same Key value.

The Directory Entry for a Keyed Only File is the same as for a Keyed File.

4. Contiguous Files

Contiguous Files carry no overhead for pointers, but can be accessed in a direct manner. These files are contained in a specific area of the disc, and are allocated a maximum amount of space when CREATEd. A Contiguous File can only be CREATEd using the utility *CREATE.

Based on the amount of space required for the maximum number of records (specified during *CREATE), the File System searches the AU Bit Map for that much contiguous (immediately adjacent) space on the disc. If the required amount of space is not available, *CREATE will report an unsuccessful CREATE. If the space is available, it will be allocated to the file, and the extents of the file placed in the File Directory Header.

When a Contiguous File is CREATEd, a "fill" character is requested. This may be any character, and all sectors allocated to the Contiguous File are written with this character. This fill character can be useful in determining whether the requested record number has been written before, or is a new record. Once a Contiguous File is CREATEd, it is considered full and all utilities will report the number of records as the maximum possible.

Unlike a Keyed or Sequential File, unused space in a Contiguous File is carried as overhead; when the maximum number of records has been entered, no more space can be allocated to the file. A new file must be CREATEd with a larger area specified, and the old records copied into the new file.

Because the BEST File System allocates space on an as needed, as available basis, when an attempt to CREATE a Contiguous File fails, it is not straightforward to determine what files can be ERASED to gain the needed space in the proper area. Instead, a complete, logical copy should be made to compact all existing files, and thus free up blocks of space for the Contiguous File.

Once a Contiguous File is CREATEd, it can be accessed in a sequential or direct manner. That is, it can be READ/WRITTEN sequentially; READ/WRITTEN with an index; EXTRACTed with or without an index; and UPDATEd. A Contiguous File can be an IMAGE file. Sequential access of a Contiguous File is in the same manner as for a Sequential File; direct access is accomplished through the use of a numeric index, where the numeric index refers to the record number in the contiguous file.

4. Contiguous Files (cont)

The QIC Program interface for a direct access to a Contiguous File is:

```
READ(1,10) IND=4  or,
READ(1,10) IND=NUMBER
```

where the IND= parameter must be a numeric constant or variable.

The numeric index is exactly the record position for the record in the file. The actual sector address of the data record can be determined by the file system once the extents of the file are known. The first sector of the Contiguous File, at offset 0, contains record 1 of the file. Since the system knows the size of the record, it also knows how many records will fit in one sector. When a numeric index is supplied, that number is divided by the number of records that fit in each sector. The result of this calculation added to the first record sector of the file determines the sector number where the requested record is located. The remainder (if any), of this calculation determines the offset, or where the record begins in that sector.

Implemented with Contiguous Files is the ORD function:

```
NUMBER=ORD(1,EXCP=9000)
```

which reports the next record number of the file, based on the current position of the record pointer. This value is the number of the record that will be obtained if the next access is sequential.

When an attempt is made to WRITE or READ to a Contiguous File with an ordinal greater than the extents of the file will allow, an Error 2 is issued. If an attempt is made to WRITE or READ with an ordinal that is less than 1, an Error 32 is issued. Ordinals are always integer numbers. A fractional IND= parameter will be truncated to the integer value of the parameter specified. The maximum Ordinal size is 16 digits (the size of the accumulator).

The (BOF) mnemonic can be used on Contiguous Files. The (EOF) mnemonic is ignored since all records in a Contiguous File are considered written at the time the file is CREATEd.

A Contiguous File Directory Entry is shown on the following page.

4. Contiguous Files (cont)

DIRECTORY ENTRY FOR CONTIGUOUS FILES

Byte(s)	Directory Information
1 - 3	First Record Sector
4	(Unused)
5 - 12	Contiguous File Name
13	File Type (Contiguous File = 02, $02)
14 - 16	Last Record Sector
17 - 18	Record Size
19 - 20	AU Count
21 - 23	(Unused)
24	Value of Fill Character
30 - 31	Next Record Offset
32 - 48	(Unused)

5. Object Files

Object Files are loaded into memory by a Sequential Read through the Object data on the disc and executed under the control of the BEST Operating System. The Directory Header (see next page) contains all the information necessary for the monitor to load the program into memory.

An Object File is the product of a Compile or Link, and is written to disc in the same manner as a Sequential File.

5. Object Files (cont)

DIRECTORY ENTRY FOR OBJECT FILES

Byte(s)	Directory Information
1 - 3	First Record Sector
4	(Unused)
5 - 12	Object File Name
13	File Type (Object File = 16, $10)
14 - 16	Last Record Sector
17 - 18	Program Code Length
19 - 20	Execution Start Address
21 - 28	LUN's OPENed if QIC Program
29 - 30	I/O Buffer Length
31 - 32	(Unused)
33 - 34	(Unused)
35 - 36	Common Length
37 - 38	(Unused)
39 - 40	Local Length
41 - 48	(Unused)

6. Standalone Object Files

Standalone Object Files are object programs that can only be executed from the Program Loader. These programs handle their own I/O and do not execute under BEST. They require the full capabilities of the machine, and must be executed from terminal "T00". Standalone Object Programs are part of the BEST Directory but are loaded into memory by the Core Image Loader. The Directory Header (see next page) has the same elements as does a BEST Object File, but the file type is $30.

6. Standalone Object Files (cont)

DIRECTORY ENTRY FOR STANDALONE OBJECT FILES

Byte(s)	Directory Information
1 - 3	First Record Sector
4	(Unused)
5 - 12	Standalone File Name
13	File Type (Standalone File = 48, $30)
14 - 16	Last Record Sector
17 - 18	Program Code Length
19 - 20	Execution Start Address
21 - 28	LUN's OPENed if QIC Program
29 - 30	I/O Buffer Length
31 - 32	(Unused)
33 - 34	(Unused)
35 - 36	Common Length
37 - 38	(Unused)
39 - 40	Local Length
41 - 48	(Unused)

7. BEST Directory

The BEST Directory is the part of any disc that contains the names and file header information for any file added to the pack under the BEST Operating System.

The BEST Directory is maintained in much the same manner as a Keyed File. Sector 5 is established as the Directory Entry for the BEST File Directory, and contains the pointer to the top directory sector. At Software Initialization, sectors 6 through 9 are allocated to the BEST File Directory and sector 5 is set to point to sector 6. As files are added to the directory sectors are split in the same manner as for a Keyed File. When sectors 6 through 9 are depleted, the File System gets a new AU (in the user file area), and sets the forward link in sector 9 to point to this new AU. As more files are added, new AUs are obtained in the same manner, scattering the directory throughout the disc. In this manner, there is no pre-defined limit to the number or type of files that can be added to a BEST pack.

Every directory entry is 48 bytes. Within these 48 bytes, all information necessary to access the file or load the program is available. (This is the data loaded into the system AFL when the file is OPENed by any user). When the file is CLOSEd, the information in the AFL is written back to the directory entry, making sure that the entry is completely up to date.

As in key sectors of Keyed Files, the higher level directory entries are only pointers to the lower levels. Although the pointers are also 48 bytes, their information is not current for the file. Only the level 0 directory entry is updated by the File System.

A Directory Entry for the BEST Directory is shown on the following page.

7. BEST Directory (cont)

DIRECTORY ENTRY FOR THE BEST DIRECTORY - SECTOR 5

Byte(s)	Directory Information
1 - 3	Pointer to the top level of the BEST Directory
4	(Unused)
5 - 12	Dummy File Name ($FFFFFFFFFFFFFFFF)
13	File Type (Directory = 04, $04)
14 - 16	(Unused)
17 - 18	Record Size (Directory Entry = 48 bytes, $30)
19 - 20	AU Count
21 - 23	Last Directory Sector
24	Key Size (Directory Entry Key = File Name 08, $08)
25 - 37	(Unused)
38 - 40	Deleted Directory Sector Chain
41 - 42	Number of Directory sectors "thrown away" because of disc errors during allocation
43 - 48	(Unused)
756 -758	Disc Label
767 -768	Sector Displacement

5.2 File Operations

1. CREATE

CREATE provides the ability to add a new file name to a BEST Directory and allocate a minimum amount of space based on the file type.

When the system executes a CREATE, it first searches through the configured set of discs to determine if the specified disc label is available. If it is not, (not there, inop), an Error 40 is issued: Attempted Create on an Unavailable Disc.

If the correct label is found, the system gets the following number of AUs from the AU Bit Map, based on the file type specified:

Keyed File	= 2 Allocation Units (10 sectors)
Sequential File	= 1 Allocation Unit (5 sectors)
Object File	= 1 Allocation Unit (5 sectors)
Contiguous File	= (Handled by a REAL subroutine to the QIC Program, *CREATE).

The system verifies the AUs by writing a copy of the AU Bit Map into each of the new sectors, and setting the forward links in each sector to point to the next sector. If an error occurs on this "initialization", the AU is thrown away, a new AU is obtained, and the system records in sector 5 that a "bad" AU was found, but not allocated to any file. The sector numbers obtained from the AU Map are placed in the directory entry being built in working storage.

The BEST Operating System then checks the directory for a duplicate file name. If a duplicate name is found, the AUs are returned to the AU Map, and an Error 12 is issued: Attempted Create on Existing File.

If there is no duplicate name in the directory, the system inserts the entry built in working storage into the appropriate directory sector and rewrites the updated sector to the disc.

1. CREATE (cont)

A. Calculation of Record Sizes for Data Files

The actual size of a data record is not necessarily the size specified in a CREATE Statement. BEST determines the number of records of the specified size that will fit into a sector by dividing 756 bytes (maximum available) by the specified size. If there is no remainder, or the remainder is less than the number of records that fit, the actual size is the specified size. If the remainder is greater than the number of records that fit, BEST evenly distributes the remaining bytes to each record. Those bytes that cannot be distributed evenly are unused bytes.

Example 1:

Specified record size = 94
756/94 = 8 with a remainder of 4.
Since the 4 remaining bytes cannot be distributed to the 8 records in each sector, they become unused bytes at the end of the sector.
In this case: Specified Record Size = Actual Record Size.

Example 2:

Specified record size = 88
756/88 = 8 with a remainder of 52.
The remainder of 52 bytes can be evenly distributed over the 8 records by adding 6 "extra" bytes to each record.
52/8 = 6 with a remainder of 4.
In this case: BEST will create the file with an actual record size of 94 (88 + 6) with 4 remaining, unused bytes.

A WRITE to this file with 88 bytes of true data will appear in the file as the data with 6 appended blanks. This provides the ability to increase record sizes without having to reformat the file.

2. ERASE

ERASE provides the ability to remove a file name from the BEST directory and return all space assigned to that file to the system for reallocation.

On execution of the ERASE statement, the system searches the directory for the file name specified. If a disc label is provided, only that label is checked. If no label is provided, the system looks for the first occurrence of that file name on the configured set of discs.

The system then checks to see if that file is OPEN by any other user (has an active AFL). If the file is OPEN, an Error 36 is generated: Attempted ERASE of an OPEN File.

If the file is not OPEN, the file name is removed from the directory. The file name is not flagged as ERASEd, but is physically removed by "shuffling up" all directory entries in the directory sector. In other words, every entry below the specified file name is moved up one entry in the directory, and the offset of the sector is changed to reflect one less entry. When the "shuffling up" is complete the directory sector is rewritten to disc.

Once the directory is rewritten, the system starts with the first key AU specified in the Directory Header and follows the forward links through the file, returning each AU to the AU Bit Map as it is encountered. When a forward link of all zeroes is encountered, the ERASE is terminated. In reality, only the forward link in the last sector of each AU is used to obtain the sector number of the next AU. If an ERASE prematurely aborts due to a disc error or Escape, only those AUs already encountered will be returned to the AU Bit Map.

Since an ERASE can be a lengthy process if the file is large, the system takes a Task Break after returning each AU. This allows other users to be activated.

3. OPEN

The OPEN statement is used to gain access to a disc file or a peripheral device. In the case of a disc file, the OPEN statement moves the directory information from the disc into an AFL which provides the common entry for all users to access and update the file. In the case of a device, the device name is an entry in the DDT, and the device is reserved for the exclusive use of the program (partition) that OPENed it.

A. Disc Files

The syntax of the OPEN statement provides an LUN as an internal program reference for that file. When an OPEN is executed, the supplied LUN is converted to a system reference and checked against the corresponding FCB to determine if the LUN is already in use. If it is, an Error 34 is issued: Logical Unit Number Not Available for OPEN.

The System then checks the AFLs in memory to see if any other user has that file OPEN. If so, the user count in the AFL is incremented by one, and the directory information is not read from disc. The user's FCB is made to point to the AFL that corresponds to the file. In this manner, all users acting on a particular file have their own separate FCB entry to determine their position, each pointing to a single AFL to record all updates.

If an AFL does not already exist, and a disc label is specified on the OPEN, that disc is searched for the specified name. If no label is specified, the discs are searched for the first occurrence of the name in disc configuration order. If the file is not found, an Error 11 is issued. Once the file name is found, the directory information is moved into the AFL.

In either case, the record pointers in the user's FCB are set to the first key in the file (if Keyed), and the position of the first record (if Sequential or Contiguous).

The system starts with the first Key AU and "flinks" through the forward links of the file to be sure the extents of the file match the Directory Header. The system also checks to see if the last sector of the last Key AU points to the first data sector. Any discrepancies are corrected during the OPEN process.

Once the OPEN process is complete, any record of the file may be accessed. The FCB will remain active in memory until that user CLOSEs the file. The AFL will remain available in memory until the last user to have the file OPEN, CLOSEs the file.

3. OPEN (cont)

B. Peripheral Devices

When a device is OPENed, the system again converts the LUN to the corresponding FCB reference to see if the LUN is already in use. The entry for the device name being OPENed is located in DDT. If the Task Header address of the DDT is 0, the device is available for OPEN. If the Task Header address of the DDT is the same as that for the user attempting the OPEN, the device can be OPENed again by that user. If the Task Header address does not match the address of the user attempting the OPEN, an Error 31 will be issued Device Unavailable.

Once the FCB is updated with the address of the DDT, the system calls the appropriate driver for that device to perform any functions necessary on OPEN. For example, the printer driver would check to see if a VFU was needed on the Model 5041 printer, and load it if necessary.

Operations supplied by the drivers on OPEN are explained in Section 6.1, Devices and their Software Drivers.

4. CLOSE

CLOSE is used in two contexts under the BEST Operating System:

- o The QIC keyword CLOSE in reference to an LUN, and
- o CLOSE all files (Abort), due to a Flag 3/Transmit, or system error.

The QIC keyword, CLOSE, calls a system routine that checks to see if the LUN actually references an OPEN file. If it does not, no error is reported and the system continues. If the open LUN refers to a device, the appropriate driver is called to perform the CLOSE routine. If the LUN references an OPEN file, the system clears any Extract flag for that user, decrements the user count in the AFL entry for that file, and, if no one else has the file open (User Count = 0), and the file has changed, updates the directory on the disc.

On Flag 3/Transmit the system RUNs the utility *MONITOR, which performs a CLOSE on all possible LUNs (0-7). In the case of an error, the system sets an "Abort Flag" and performs all of the CLOSEs itself. This makes sure that all directory entries are properly updated on fatal errors.

5. READ

The READ statement causes an input of a record from a disc file or peripheral device, into variables within the user program.

A. Disc Files

There are two types of READs in the BEST Operating System, Sequential and Indexed (Keyed). A Sequential READ can be performed on any file type. An Indexed READ can only be performed on a Keyed or Contiguous File.

1) Sequential READ

Sequential access to a file is based on the fact that the user's FCB for the file contains the next record position. If the file is Sequential, the next record position consists of a sector number and offset for the next record. Execution of a READ causes a SEEK to that sector, and the entire sector is moved into the System Buffer. The offset indicates the beginning position of the record in the System Buffer. Control is then passed to the Runtime Format Handler to move the record from the System Buffer to the User Buffer. When the READ is accomplished, the next record position in the FCB is updated with the sector number and offset of the next record.

For a Keyed File, the record pointer consists of the value of the next sequential key in the file, and the sector and offset of that key's position in the Key Tree. Any type of READ to a Keyed File (indexed or sequential) causes the system to get the next key value in the key tree, and the offset and sector where that key is located. If the next access to the file is sequential, the sector and offset of the next record position are used to immediately get the key sector; check that the key is still in the file; and if so, read the data associated with that key.

If the next key value has been deleted prior to the next Sequential Read, the file system forces a search of the key tree to get the next key value. Once the READ is accomplished, the next key value, sector and offset are updated with the next values from the file in the FCB.

A Sequential READ of a Contiguous File is handled in much the same manner as a Sequential File, except there is a next index value that is obtained by incrementing the current index value by one. The ORD function will move this value from the FCB to the user program.

5. READ (cont)

2) Indexed (Keyed) READ

When a READ which specifies an IND= parameter is executed, the system forces a search of the Key Tree. To find a specific key value, the Top Key Sector is read into memory, and examined. If the Top Key Sector is not a level 0, the system checks for the key closest in value to the specified index, but not greater than the index. The pointer for this key value is used, and the next lowest level sector is read. The search through the key sector starts at the beginning of the sector (highest value) and ends at the sector offset. If the sector is a level 0 and no entry is found that matches the specified value, an error 32 is returned. If a match is found, the key pointer provides the sector and offset of the corresponding data record. Whether the key is found or not found, the next key value and position in the FCB are updated with the next value in the key tree.

An Indexed READ to a Contiguous File forces the system to calculate the sector and offset of the record, based on the index supplied. Any type of access to a Contiguous File always updates the next index value in the FCB by adding one to the current index value.

A Sequential READ of a Keyed File is faster than a KEY function and Indexed READ. For large files, a key search may have to examine four levels of key sectors before retrieving the data. With a Sequential READ, the access is immediately to the appropriate level 0, based on the last access. Unless a file is being deleted from by another user, most accesses are relatively fast.

The KEY function moves the next key value from the FCB into a variable in the user program. If the KEY Function followed by an indexed READ is used to pass an entire file, and that file is being deleted from by another user, it is possible to get an Error 32 on the READ. When an entire file is to be read, it is more advantageous to use a Sequential READ. If the actual key value is needed, the KEY function followed by a Sequential READ should be used since the KEY function does not advance the record pointer.

B. Keyed Only Files

Keyed Only Files have a 0 record length and are used as pointer files to other keys. Performing a READ on a Keyed Only File is only to advance the record pointer. There is no data AU for a Keyed Only File, and if the structure of the key can contain

5. READ (cont)

the value of the key in a secondary file, that value can be retrieved by one access to the Keyed Only File (e.g., KEY Function, Sequential READ to advance the pointer).

C. Key and Record Lengths

When the value supplied in the IND= parameter is shorter than the actual Key size for the file, the key value supplied to the File System will be padded with blanks. If the key value is longer, it will be truncated.

Since a short key will be padded with blanks, and since the File System always gets the next key value on an indexed access, supplying a short key for a READ is a method of positioning to a starting value in a file. This procedure is called a "dummy read". If a file is organized by Customer Number + Invoice Number, a READ with a short key of only the Customer Number will return an Error 32, but the record pointer will be positioned to the first key with that customer number. A sequential READ following the dummy read will return the record that matches the first invoice for that customer.

The FORMAT specified for a READ does not necessarily have to have all of the variables for that record. If only certain variables are needed, the FORMAT may contain only those variables necessary with a position parameter. If the total length of the variables specified in a FORMAT statement is longer than the actual record size, part of the next physical record will be read, on all but the last record. Using a larger format than necessary does not alter the beginning position for the next READ.

D. Peripheral Devices

A READ from a peripheral device is handled by that device's particular software driver once the device has been OPENed. See Section 6.1 for a description of the software drivers for all BEST supported peripheral devices.

6. WRITE

The Section on WRITE will be added at a later date.

7. READ/WRITE IMAGE

READ/WRITE IMAGE is designed primarily to reduce the amount of disc storage space required for records which contain a large number of numeric variables.

The standard format for numeric variables requires one byte for the sign and one byte for the decimal for any precision variable. By contrast, numeric data under READ/WRITE IMAGE is packed. The formula to calculate the required space for a packed numeric field is

(LENGTH/2) + 1

where length is the total number of digits declared in the precision (e.g., the length of a field with precision 8.2 is 8). In the case where the calculation has a fractional result, rounding is always downward.

String fields under READ/WRITE IMAGE are exactly the declared length of the field, plus one length byte. This length byte carries the "actual" or current length of the string variable. Thus, READIMAGE is able to recover the true length of the string variable. This feature differs from the normal READ, which forces the declared length for string variables.

To calculate the effective record size for an IMAGE file, calculate all numeric fields as explained above, and add the declared length plus one for all string variables.

READ/WRITE IMAGE is available for all data file types.

Example:

```
    LENGTH 8.2 & LOCAL A,B        !Image Length = (8/2) + 1 = 5
    LENGTH 5.0 & LOCAL C,D        !Image Length = (5/2) + 1 = 3
    LENGTH 10  & LOCAL A$,B$      !Image Length = 10 + 1 = 11
100 FORMAT A$;A;B;C;D;B$          !Record Length Image =
                                   ! 11 + 5 + 5 + 3 + 3 + 11 = 38
                                  !Record Length Normal =
                                   ! 10 + 10 + 10 + 7 + 7 + 10 = 54
```

8. DELETE

The DELETE statement is used to remove an existing record from a Keyed File. This statement applies only to Keyed Files.

The DELETE statement requires that an index or key be supplied. On execution, the key is found in the key tree and physically removed. The corresponding data record is "flagged" as DELETEd.

DELETEd space within a file can be reused by that file only. The File System will always attempt to use available deleted space before allocating new space. This is accomplished by maintaining a Delete Chain in the File Directory Header. There is a Delete Chain entry for key sectors and an entry for data sectors. These entries contain the last DELETEd record or key sector addresses.

A. Key Sectors

When the last key in a sector is DELETEd, the File System places the sector number of that sector in the Delete Key Chain entry in the File Header. When a new Key Sector is needed, this sector will be used. The key sector is flagged as DELETEd by placing three hex "FE"s in bytes 757 - 759 of the key sector. If another key sector has its last entry DELETEd the header is updated to point to this new DELETEd key sector, and the backward link (bytes 761 - 763) is set to point to the first DELETEd key sector. This type of chain continues throughout the key sectors with the header always pointing to the most recently DELETEd key sector.

B. Data Sectors

When a key is DELETEd, it is physically removed from the Key Tree and the corresponding data record is flagged by placing one hex "FE" in its sixth byte. The first five bytes are then set to point to the record DELETEd just prior to this record. As in key sectors, the file directory header is always set to point to the most recently DELETEd record.

Before attempting to re-use DELETEd space, the File System always checks to see if the DELETE flag is present. If there is no DELETE flag, the system assumes that something is wrong and zeroes out the Delete Chain Entry in the File Header. Because space is re-used on a last in, first out basis, this effectively clears the Delete Chain. The utilities *DELREC and *DELKEY can be used to restore the Delete Chain for records and keys, respectively, for

8. DELETE (cont)

any Keyed File.

The record pointer is advanced to the next record on DELETE.

9. KEY/ORD

The KEY/ORD function is used to determine the next (in logical order) available index value in a specified Keyed or Contiguous File.

Typically, every access to a Keyed or Contiguous File obtains the record specified, and in addition, gets the Key or Ordinal value of the next record to be accessed if that access is sequential. The KEY/ORD function merely moves the value obtained from the system work area into the user partition.

In the case of the KEY function, the system does not check that the Key supplied during the KEY function is still the next Key. If another user were active, that Key could have been DELETEd so an attempt to read with an index supplied by the KEY Function could produce an Error 32, Key Not Found. The only error produced by the KEY function would be End of File.

The ORD function returns the record number of the last access, plus one. Since a Contiguous File cannot be DELETEd from, the ORD function will always return the next record number.

The KEY or ORD function does not advance or change the record pointer.

10. EXTRACT

The EXTRACT statement provides a means to prevent multiple users from accessing the same record at the same time. EXTRACT insures that the record obtained is the most current copy of the data record, and thus protects volatile files such as Inventory Files.

EXTRACT may be used on all data file types: Keyed, Sequential and Contiguous. The syntax of the EXTRACT statement is the same as that for a READ.

Whenever a record is EXTRACTed, an EXTRACT count is incremented in the file AFL. For all operations such as READ, WRITE and DELETE, the File System checks the AFL to see if the EXTRACT count is not zero, before proceeding with the operation. If the EXTRACT count is not zero, the system must check the EXTRACT pool to determine if the record being accessed is already EXTRACTed by some other user.

The size of the EXTRACT pool is determined at Configuration Time. If an inadequate number of EXTRACT entries is configured, attempts to EXTRACT when the pool is full will generate a BEST Error 33 until space is made available by releasing an entry.

A record is released from the EXTRACT when the user that EXTRACTed the record accesses the file again through a READ, WRITE, DELETE, UPDATE or CLOSE. KEY or ORD functions do not release the EXTRACTed record.

EXTRACT does not advance the record pointer to the next key or record. This feature allows many programming techniques to be employed where a user program may want to re-read a record based on some condition in the record. Additionally, when an EXTRACT is pending, a WRITE may be executed without supplying an index. Since the record pointer is already positioned to the record, the speed of the WRITE is improved by eliminating a search through the Key Tree. By specifying an index on a WRITE after the EXTRACT, the speed improvement is lost. NOTE: An attempt to write without an index when no EXTRACT is pending will generate an Error 04.

Since EXTRACT causes the system to do some additional work to determine if a given record is EXTRACTed, indiscriminate use of this statement can slow down processing. This slowdown would be a function of the number of records EXTRACTed at any one time and the number of files involved.

11. UPDATE

The UPDATE operation provides the ability to READ, change some specified value, and rewrite a record without declaring the entire file format.

Internally, the File System first EXTRACTs the record called for and then calls the Runtime routine for unindexed WRITE branching around the routine that would normally clear the record buffer. The field or fields being UPDATEd are moved into the buffer using the positions specified in the file format. The record is then rewritten to disc. The record pointer is only advanced after the WRITE. UPDATE may be used with or without an index specified.

In order to use this feature to UPDATE a field based on some condition in the record, a separate READ or EXTRACT must occur in order to test the condition. For an unindexed UPDATE, the test read should be done on a separate LUN, since READ advances the record pointer. A test EXTRACT can be used on the same LUN since EXTRACT does not advance the pointer.

For Example:

If a field, X$, is equal to 'N', set the variable A to 10.00

```
 10 FORMAT X$,@(10)                       !FORMAT TEST FIELD
 20 FORMAT A @(30)                        !FORMAT UPDATE FIELD
    OPEN (1) 'FILE'                       !LUN FOR READ
    OPEN (2) 'FILE'                       !LUN FOR UPDATE
100 READ (1,10) EXCP=9000
    IF X$='N' A=10.00 & GO 200            !BRANCH OUT IF TRUE
    READ (2,20)                           !KEEP 2ND LUN IN STEP
    GO 100
200 UPDATE (2,20)                         !UPDATE VARIABLE A
    GO 100
```

If a String field is being UPDATED, the length of that field is governed by the <u>current</u> (actual) length, and <u>not</u> by the <u>declared</u> length. If the current length is less than the declared length, the string field should be padded to the declared length to UPDATE the entire field. If the field is not padded, the actual characters are moved into the buffer and the remainder of the field is not changed.

11. UPDATE (cont)

For Example:

Given: Record 1 = $\underbrace{11111}_{X\$}\underbrace{22222222222222222222}_{A\$}\underbrace{333}_{Y\$}$

If the following program is executed,

```
     LENGTH 20
     LOCAL A$
  10 FORMAT A$,@(5)
     OPEN (1) 'FILE'
 100 LET A$='444'                  !Set current length to 3
     UPDATE (1,10)                 !UPDATE every record
                                   !with new value of A$
     GOTO 100
```

the resulting record would be:

Record 1 = $\underbrace{11111}_{X\$}\underbrace{44422222222222222222}_{A\$}\underbrace{333}_{Y\$}$

If statement 100 is replaced with:

```
 100 LET A$='444                 '  !set current length to 20
```

the result would be:

Record 1 = $\underbrace{11111}_{X\$}\underbrace{444\qquad\qquad\qquad\qquad\qquad}_{A\$\ A\$}\underbrace{333}_{Y\$}$

12. LOCK/UNLOCK

The Section on LOCK/UNLOCK to be written at a later date.

13. GET/PUT

The Section on GET/PUT to be written at a later date.

5.3 Disc Layout and Handling

1. Software Initialization

Software Initialization (accomplished by the Core Image program, DKIN), provides three specific functions for proper execution of all file system operations:

1) Clears all BEST files from the BEST Directory by resetting the pointer to the top directory sector. The pointer is set to point to sector 6 (the first available sector for the directory) and sector 6 is rewritten with a copy of the AU Bit Map.

2) Resets the AU Bit Map in sectors 3 and 4. The AU Bit Map determines the effective capacity of any disc pack Based on the answer to "Disc Type", DKIN sets certain bits "on" for available ($FF), and "off" for unavailable ($00).

   3 + 3 - The first 100 bytes of sector 3 are set to $FF, and all remaining bytes in sectors 3 and 4 are set to $00. Effective capacity is 4000 sectors.

   6 + 6 - The first 200 bytes of sector 3 are set to $FF, and all remaining bytes in sectors 3 and 4 are set to $00. Effective capacity is 8000 sectors.

   30MB - The first 756 bytes of sector 3 and 244 bytes of sector 4 are set to $FF, and all remaining bytes in sector 4 are set to $00. Effective capacity is 40,000 sectors.

   DKIN makes no check as to what kind of drive is being used when the pack is software initialized.

3) Reserves special sectors for system use. There are two types of initialized packs

   System Pack - Defined as a pack which has reserved the first 1000 sectors of the disc to store the Operating System, Compiler, and other Core Image utilities. Effectively, the first 25 bytes of sector 3 are set to $00, and cannot be allocated to BEST Files.

   Data Pack - Defined as a pack which will reserve the first 40 sectors for "selected" Core Images. Effectively, the first byte of sector 3 is set to $00, and cannot be allocated to BEST Files.

1. Software Initialization (cont)

Once a pack is initialized, the Best Operating System makes no check for the type of drive being used. The AU Map is used to automatically allocate sectors to any file (in increments of 5 sectors). If the pack has been initialized incorrectly, the system will try to allocate sectors beyond the physical range of the disc drive, and the hardware will generate an "Invalid Seek". Conversely, if the pack is initialized with less space available than the actual range of the disc, that space will be wasted.

2. Disc Layout on a BEST Pack

The BEST Operating System defines its reserved space on a disc as follows:

SECTOR NUMBER	SYSTEM PACK	SECTOR NUMBER	DATA PACK
0- 2	Disc Program Loader	0- 2	Disc Program Loader
3- 4	AU Bit Map	3- 4	AU Bit Map
5	BEST Directory File header (pointer to top level of BEST Directory)	5	BEST Directory File header (pointer to top level of BEST Directory)
6- 9	BEST Directory Area	6- 9	BEST Directory Area
10	Message Passing Area	10	Message Passing Area
11- 14	Coreimage Directory	11- 14	Coreimage Directory
15-999	System Coreimages	15- 39	Selected Coreimages
1000+	User Files	40+	User Files

DISC PROGRAM LOADER - Provides the ability to access Core Image and Standalone programs.

AU BIT MAP - Controls sector allocation for all files on that pack.

BEST DIRECTORY FILE HEADER - Is the Directory entry for the BEST Directory, and provides the pointer to the top level of the BEST Directory.

BEST DIRECTORY AREA - Reserved for the beginning of the BEST Directory. When this space is used, additional AUs in the user file area are allocated.

MESSAGE PASSING AREA - Used by Core Images to pass messages to each other.

CORE IMAGE DIRECTORY - Contains directory information for Core Image programs on that pack. (Separate from the BEST Directory).

CORE IMAGES - Contains programs that are loaded into memory from the disc loader prompt.

2. Disc Layout on a BEST Pack (cont)

USER FILES - Contains programs that are added to the pack under control of the BEST Operating System.

5.4 System Operations and Control

1. Sector Allocation

The BEST File System allocates space on an "as needed" basis to all file types, except Contiguous Files. Each BEST disc maintains two sectors as a map (the Allocation Bit Map) to the sectors used and available for that disc pack. In this manner files do not carry overhead space for unused sectors, and can expand whenever volume demands

All space is allocated by the File System in increments of five sectors, called an Allocation Unit (AU). At software initialization (DKIN), the AU Bit Map is set up for the particular type of disc drive being used. (See Software Initialization, Section 5.3.1). In essence, the program "turns on" the bytes that represent the sectors available for that disc, and the remainder of the bytes are "turned off", to prevent attempts to access beyond the range of the disc. In this case, each byte in sectors 3 and 4 represents 8 allocation units, or 40 sectors.

On CREATE, Keyed Files are allocated 2 AUs, one for keys and one for data. Sequential Files are allocated 1 AU, and Object Files are allocated 1 AU. When this space is depleted, BEST reads in sector 3, determines the first AU available by finding the first byte that is not $00, and converts that byte to the corresponding sector numbers of the available AU. Sector 3 (or 4) is then updated to show that the AU is no longer available and the sector is rewritten. On allocation of the new AU the system initializes the AU by writing a copy of the AU Bit Map into each sector of the new AU, and setting the forward links in each sector to point to the next sector ("preflinking"). The last data sector of the last data AU will have a forward link of all zeroes.

If a disc error occurs when the AU is being initialized, the operator is not notified, but the AU is "thrown away" and a new AU is obtained from the AU Map. The system updates bytes 41 and 42 in sector 5 to indicate that an AU was allocated but not used. The number of sectors "thrown away" in this manner is available through the utility, *SCOUNT.

Every File Directory entry contains the number of AUs allocated to that file. This number will increase in size, but never decrease. If records are DELETEd in a file, the Delete Chain pointers are updated so that the space can be reused <u>by that file</u>, but no space is returned to the AU Map. Once an AU is allocated to a particular file, it remains allocated to that file until the file is ERASEd. ERASE and DKIN are the <u>only</u> operations (programs) that return AUs to the AU Map. When a file is ERASEd, the system links through all of the sectors in the file, by forward links, and turns

1. Sector Allocation (cont)

on the corresponding bit in the AU Bit Map to indicate the space is available. If an ERASE prematurely aborts, or if the forward links are not correct, all space may not be returned to the AU Map. If space is lost in this manner, it can be regained by a logical copy of all files to another pack. (See the BEST Utilities Manual for Operating Instructions for copy programs.)

2. File System Control

A. In Memory

A File System in a multi-user environment requires file control at two levels:

1) At the user level to distinguish the user's own position in any file, and
2) At the system level where any action by any user is recorded for all users attempting to access that file.

Under BEST, this control is accomplished by an FCB (File Control Block) that records the files each user has OPEN and its own position in that file; and an AFL (Active File List) which is a list of all files OPEN by all users. An AFL contains all pertinent information for accessing that file.

Each file OPENed by a user is recorded in an FCB. There are 8 FCBs for each user, corresponding to LUNs 0 through 7. For every unique file name OPENed, one AFL entry is created in the Active File List. The AFL entry contains the File Directory information as read from the disc on OPEN. All users accessing that file use the same AFL entry. As the file is added to or deleted from, the current information is maintained in the AFL entry to be rewritten to the disc on CLOSE, Flag3/Transmit, or on an error, when no other users have the file OPEN.

The FCB entry maintains the user's current position in the file, his next key value, and the address of the AFL being accessed.

On OPEN, the LUN specified in the OPEN statement is converted to an FCB reference. If that FCB is currently in use an Error 34 is generated. If the FCB is free, BEST checks the AFL list to see if that file is currently OPEN. If there is already an AFL entry for the file, BEST places the address of that AFL into the FCB and increments the user count in the AFL by one. If no AFL exists, BEST reads the directory, creates an AFL entry, and places the address in the FCB.

The OPEN process records in the FCB that the user's position is at the beginning of the file, and the next key is the first key in the file. Subsequent accesses to the file adjust the FCB to always point to the next position in the file, based on the last access.

The User Count in the AFL keeps track of how many users have a file OPEN. A CLOSE always decrements the User count, but until the count reaches 0, no updates are made to the directory. Additionally, no updates are made to the directory if no changes have been made

2. File System Control (cont)

to the file since it was OPENed.

When the User Count reaches 0, the CLOSE operation updates the directory and clears the AFL from memory.

2. File System Control (cont)

B. On Disc

The BEST File System provides the ability to store and retrieve two types of data from disc:

1) Record data for input/output/calculation, and
2) Object program data for execution.

All data is stored on disc in the same manner, but methods of accessing vary between file types. Each sector of 768 bytes ($300), has the last 12 bytes reserved for system control information, leaving 756 bytes available for data:

BYTES	HEX	DESCRIPTION
1-756	$ 00-$2F3	Record Data
757-759	$2F4-$2F6	Disc Label
-760	-$2F7	Key Sector Level
761-763	$2F8-$2FA	Backward Link
764-766	$2FB-$2FD	Forward Link
767-768	$2FE-$2FF	Sector Offset

The System Control bytes are used in the following manner:

DISC LABEL - The three bytes reserved for the disc label should always be $000000, unless

a) The sector is a top key sector of a file. If it is, the three bytes will contain some disc label, but not necessarily the current label of the pack.

b) The sector is the top directory sector of the pack. If it is, again the three bytes will contain some disc label.

c) The key sector is a deleted key sector. When all keys are deleted from a sector, the sector is added to the Delete Key Chain and the three bytes are filled with $FEFEFE.

KEY SECTOR LEVEL - This byte always contains the level of the key sector. If the sector is not a key sector it will contain $00. If the sector is the top key sector of a file and the key sector level contains $00, the keys contained in that sector are the only keys

2. File System Control (cont)

to the file.

BACKWARD LINK - The backward link contains the pointer to the next deleted key sector, if that sector is indeed a valid deleted key sector, i.e., the disc label contains $FEFEFE. Otherwise, it contains the number of the sector itself or the number of the sector that was split to create this sector. These three bytes are only important if the sector is a deleted key sector.

FORWARD LINK - The three bytes reserved for the forward link always contain a pointer to the next logical sector, or to the next AU allocated to that file. All sectors allocated by BEST are pre-flinked when allocated. The last key sector of the last Key AU always contains a forward link to the first data sector. If the File System determines that this link is not valid it will force the link to be correct, based on information found in the directory. Forward links are primarily used for ERASE. During ERASE, the File System flinks through all of the forward links in the file, returning AUs as they are encountered. Should an ERASE be aborted, all sectors may not be returned to the AU Map. The last sector of the last data AU has a forward link of all zeroes ($000000). When that sector is filled, the system gets a new AU, and sets the forward link to point to the first sector in this new AU.

SECTOR OFFSET - The two bytes reserved for the sector offset contain the starting position where the next key or record will be written in that sector. All data appearing before that offset is valid for that file. All data after that offset is not current for that file. The maximum value for a sector offset is $02F4.

3. Physical Structure of Pointers

A. Key Pointers

The key pointers within a Key Tree are made up of the key, a sector number, and a sector offset. A key pointer appears as:

```
XX XX    YY YY YY    ZZ     KK KK KK KK ... KK
-----    --------    --     -----------------
Sector    Sector    Unused   The Key (1-32 characters)
Offset    Number
```

This structure is the same throughout the levels of the Key Tree. The Sector Offset is only relevant in the level 0 key sectors, where it indicates the relative starting position of the data within the sector. The Sector Number always points to the next lower level key sector, whose first valid key is the same as the value of the key pointer. If the key sector is a level 0, the sector number is the sector where the data is located.

The offset and key are carried as hexadecimal numbers. The sector number is represented as a decimal number within the 3 bytes to reduce the overhead. (It would take 5 bytes to represent the sector number hexadecimally).

A key of "1234" that points to a data record in the beginning of sector 2001 would have a level 0 key pointer of:

```
00 00    00 20 01    00     31 32 33 34
-----    --------    --     -----------
Sector    Sector    Unused   The Key '1234'
Offset    Number
```

indicating the record for key "1234" is found at offset 0 of sector 2001.

B. Directory Entry Pointers

A directory entry pointer is much the same as a key pointer. Every directory entry is exactly 48 bytes long and carries all information, in the level 0, vital to accessing the file. In levels other than level 0, only the first 3 bytes and the file name are relevant as pointers to lower levels. The directory information for each file type is specialized. See Section 5.1 for a complete description of the directory entry for each file type.

4. Key Splits

Adding keys to a file and maintaining access to those keys are accomplished through the Key Split operation. As explained in Section 5.1.2, Keyed Files, the File Directory Header maintains a pointer to the "top level" key sector. This is the trunk of the tree that points to all other keys, or to lower level key sectors that in turn point to keys. Adding new records to a file must always preserve the pointer to the top level, or access to the file is lost.

As keys are written to a new file, they are added to the first allocated key sector of the file. When that sector is full, the File System gets the next two available key sectors, divides the original (full) sector in half, and builds a pointer sector to point to the two "new" key sectors created. This pointer sector is the top level key sector, and the operation is called Key Split. There are essentially two types of Key Split operations in the File System: One for splitting most key sectors; and the second for the special case of splitting the top key sector.

A. Splitting Key Sectors other than the Top Key Sector

Given a file with a level 1 pointing to a full level 0, inserting a new key in the full level 0 will cause that sector to split. The File System obtains two new sectors in the same AU, or if none are available, gets a new AU. The full level 0 is split in half and the two new sectors are written to disc. At this point the full level 0 is still intact, pointed to by the original entry in the level 1. This entry in the level 1 is also the lowest key value of one of the new sectors just created (since it was the lowest in the original full sector).

The level 1 sector is read into memory and the pointer to the the original full sector is changed to point to the new, half full sector. A second entry is created for the other level 0, using the lowest valid key value in that sector.

A Delete pointer is put into the system stack pointing to the original full sector and, if the addition to the level 1 sector does not cause it to split, the level 1 is written back out to disc, and the original full level 0 is Deleted (added to the Delete Key Chain for re-use.)

If the addition to the level 1 causes that sector to to split, the modified level 1 is not written to disc, but is in turn split into two level 1's and a new level 2 is created (see Splitting Top Key Sector), or an existing level 2 is read in and

4. Key Splits (cont)

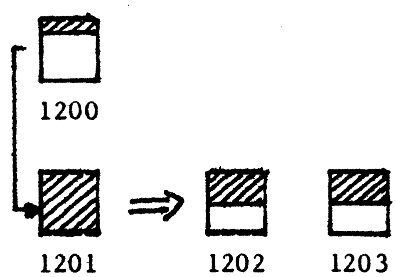
modified to point to the two new level 1's just created.

When finally, the last split is made, the highest level affected is rewritten to disc, which contains all of the new key sectors, and the sectors in the Delete stack are added to the Delete Chain. In this manner, the system actually monitors two separate trees, the old tree before the split, and the new tree reflecting the insert. Should anything happen during the split operation, the old tree remains intact, missing only the key that caused the original split.

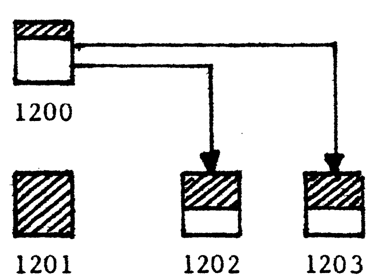
Example A-1:

To split a full level 0:

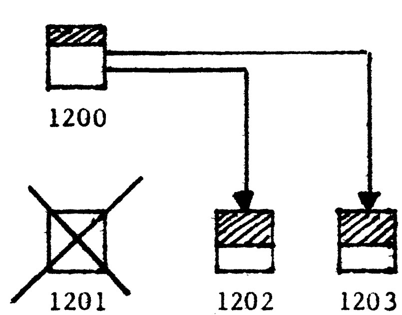
Sector 1201 is divided into 2 new key sectors. Sector 1200 points to 1201. Sectors 1202 and 1203 are written to disc.



Sector 1200 is read in and adjusted to point to 1202 and 1203. If no new splits are required, sector 1200 is written to disc.



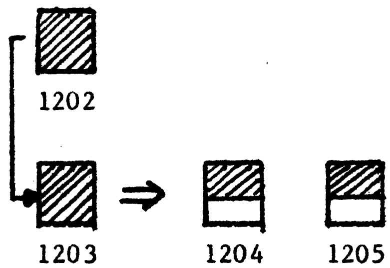
Sector 1201 is added to Key Sector Delete Chain.



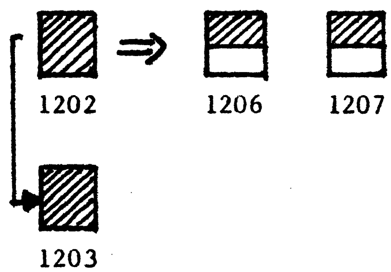
4. Key Splits (cont)

Example A-2:

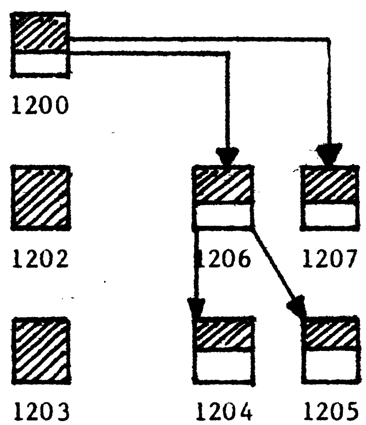
Sector 1203 is divided into 2 new key sectors. Sector 1202 points to 1203 and sectors 1204 and 1205 are written to disc. Sector 1203 is added to system stack for Delete.

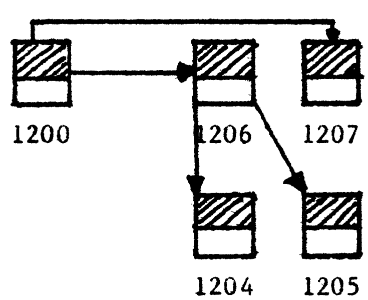


Adding new entry to 1202 for split above causes 1202, level 1, to split. 1202 is divided in half and two new sectors, 1206 and 1207 are written to disc. Sector 1202 is added to system stack for Delete.



Sector 1200 is read in, the pointer pointing to 1202 is changed to point to one of the new level 1's, and an entry is made for the additional level 1. Sector 1200, which reflects all of the splits made, is rewritten to disc and the sectors in the system stack are Deleted. (See "Splitting Top Key Sector" if addition to the level 2 causes it to split).





4. Key Splits (cont)

B. Splitting the Top Key Sector

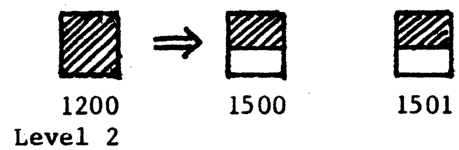
The top key sector of a file is maintained as the first key sector of the file, and never moves as the file expands. (For files built prior to version 13 and 14, the top key sector is not necessarily the first key sector).

The BEST File System splits the top key sector in a different manner than other key sectors. Since the top key sector must always stay in the same place, it cannot be added to the Delete Chain as can other sectors. Therefore, when the top key sector must be split, the File System gets two available key sectors, divides the top key sector in half, and writes out the two new sectors to the disc. In memory, a new top key sector is built, pointing to these two new sectors; the level is increased one level higher than before; and the new top sector is written to disc in the same place as the old full sector. In this manner, should any problem occur while the new sector is being built in memory the original top key sector still exists, and the integrity of the file is maintained.

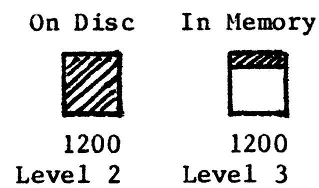
4. Key Splits (cont)

Example B-1:

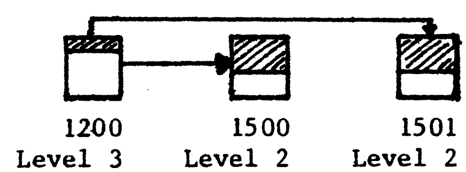
Sector 1200 is the top sector and needs to split. Sector 1200 is read into memory, two available key sectors are obtained, sector 1200 is divided in half, and the two new sectors are written to disc.



A new level 3 sector, containing pointers to the two, new level 2 sectors, is built in memory.



The new sector 1200 is written to disc, in the same place as before, pointing to the two new level 2 sectors.



5. Directory Updates

When a file is OPENed, the File Directory Header is read into the AFL in memory. This AFL entry is used by all users acting on the file and all changes to the file are recorded in the AFL in memory.

If the structure of a file is altered, the updated information from memory will be rewritten to the Directory File Header. This update occurs when the last user to have the file OPEN does one of three things:

1) CLOSEs the file in a QIC Program

2) Processes an error through *ENDITOR

3) Initiates a Flag3/Transmit or Flag1 (Escape).

The key structure for files should insure that all keys and levels of keys will be accessible if the directory is not updated. This feature, combined with the fact that OPEN attempts to link entirely through a file and correct a directory entry if it is wrong, provides a reasonable amount of protection for a file. At the same time, if a directory entry is not updated, neither are the Delete Chain entries. This means that re-use of deleted space will not be possible until the Delete Chains are restored.

The Directory Entry for a file will not be updated if any of the following conditions occur:

- o The Operating System "hangs" due to hardware or software forcing an IPL and reload of BEST.
- o The IPL button is pressed before all files are CLOSEd.
- o A pack that contains OPEN files is removed from the system. If this pack is replaced with another pack with the same label and the same files that are OPEN, the system will update the second pack, perhaps incorrectly.

To insure continuing integrity of all files, IPL should always be preceded by an ESCAPE on all terminals. Turning a terminal "OFF" does not close files.

6. End of File Detection

End of File for a Keyed File is determined when there is no next key in the file. Each time a key is accessed in a Keyed File, the File System automatically gets the next key and saves it in the system key buffer. When there is no next key in the file, this key buffer is filled with $FF. If the next operation performed is a Sequential READ or a KEY function, an Error 2 will be generated. If the next operation is a direct access (Indexed READ), the system again gets the next logical key after the specified index, and places that value in the key buffer.

End of File for a Sequential File is determined when reading a record in the data sector and the current position in the sector is greater than the offset of the sector, and there is still room in the sector for another record. When a sector is initially read into the system buffer, the current position is 0. This current position is always checked against the sector offset (displacement); when the current position is smaller, there are more records in the sector. The record is read into the record buffer starting from the current position for the length of the record. The current position is then increased to the start of the next record (based on the File Header record length). When the current position is greater than the sector offset, the File System adds one record size to the current position and if the result is less than $02F4 (the maximum offset possible), End of File is issued. If it is greater than $02F4, the flink is used to get the next data sector and the current position is reset to 0.

6.0 INPUT/OUTPUT CONTROL SUBSYSTEM

6.1 Devices and Their Software Drivers

1. CRTs

The section on CRTs will be written at a later date.

2. Printing Terminals

The section on Printing Terminals will be written at a later date.

3. Discs

Several types of disc drives are supported by the BEST Operating System:

Split Platter Disc	-	Two distinct platters, fixed and removable. The fixed disc is stationary; the removable disc is a portable cartridge that can be removed entirely from the disc drive unit.
Removable Only Disc	-	With more capacity than the Split Platter disc, the Removable Only disc consists of one cartridge unit that is completely removable from the disc drive unit.
Fixed Platter Disc	-	Effectively one cartridge unit which cannot be removed from the disc drive unit.

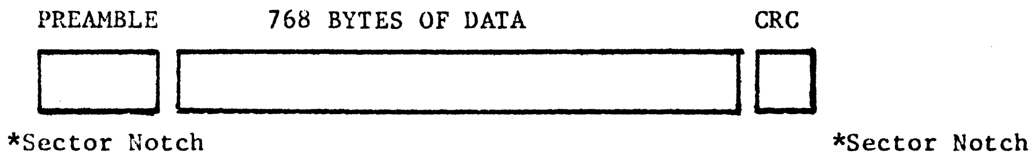
The Configurator for the BEST Operating System considers each platter of a split platter disc drive as a separate entity. Once a disc pack is software initialized, BEST takes no notice of what type of drive is on the system. All drives are handled by the same software disc driver, in the same manner. This provides software compatability through the entire Qantel product line.

A. Physical Structure of Disc Packs

Every Qantel disc pack has at least two SURFACEs (top/bottom). Each surface is accessed by a Read/Write Head. The SURFACE of a platter is divided into TRACKs, where a TRACK is a strip equidistant from the center spindle. Within this TRACK are 10 contiguous SECTORs. A CYLINDER is the same TRACK on all surfaces of the disc pack. (See Diagram 6.1.3-A).

The BEST Operating System requires 768 byte sectors. On a TRACK, each sector is preceded by a PREAMBLE and followed by a CRC check character. The PREAMBLE and the CRC are not included in the 768 bytes of data. The PREAMBLE, data, and CRC characters are contained between Sector Notches on the pack. The Sector Notch is the mechanism by which the drive determines the beginning of any sector on that TRACK.

3. Discs (cont)



B. Reading and Writing from Disc

When a certain sector on the disc is needed for either a READ or a WRITE, the operation is always preceded by a SEEK instruction. The sector provided by the Operating System becomes the Seek Parameter and, when the SEEK is performed, the head is positioned to the correct CYLINDER on all SURFACEs and the appropriate head is selected. When a Sector Notch is detected, the Read Gate is opened and any PREAMBLE is read. The PREAMBLE is made up of 31 bytes of $00 and a Track Check Character. The controller waits for 16 consecutive "0" bits to clock by and then looks for the first "1" bit which precedes the Track Check Character. If that PREAMBLE does not have 16 consecutive "0" bits, the controller waits for the next PREAMBLE and checks it. This process repeats until all PREAMBLEs have been checked. If they all fail, a Marked Sector Status is posted. Once a PREAMBLE with the 16 consecutive "0" bits is found, the Track Check Character is compared with the Seek Parameter, modulo 16. If it compares correctly, a successful SEEK is reported; if not, an Invalid Seek is returned.

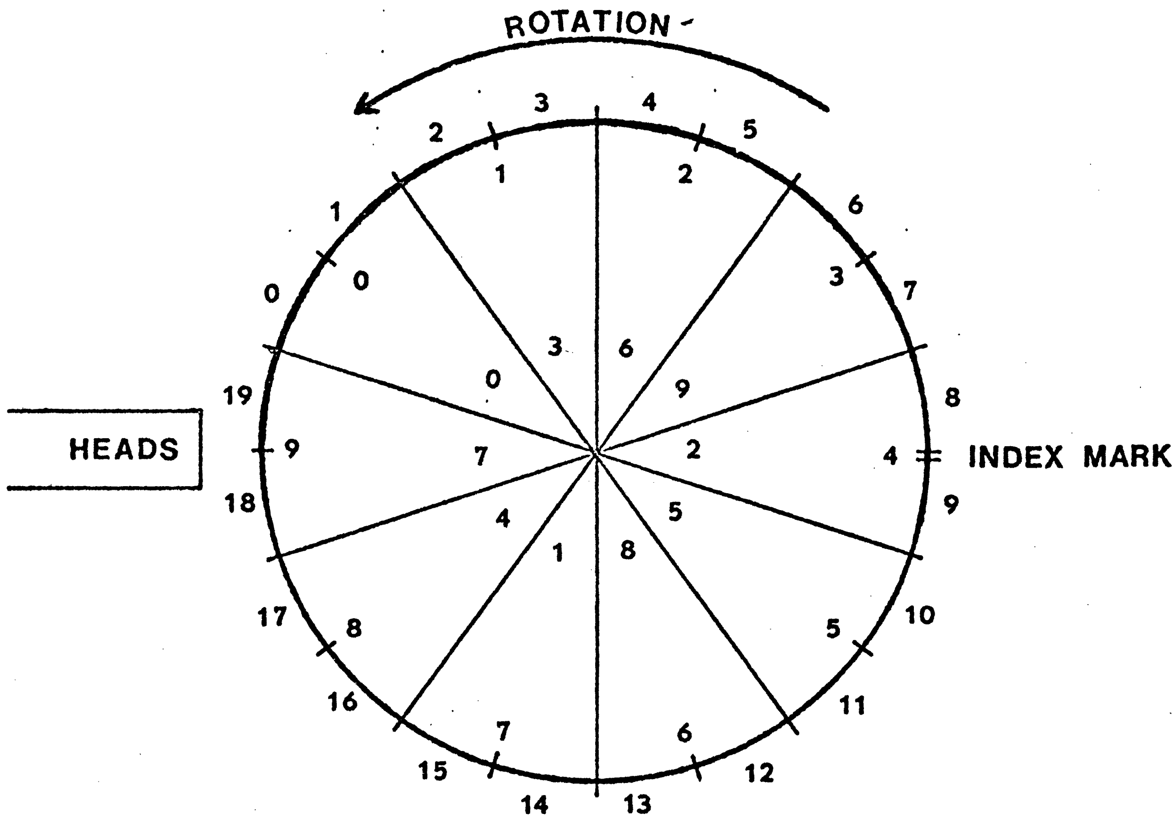
Two Sector Notches very close together provide an INDEX PULSE to the controller. This INDEX PULSE resets the internal sector counter to 0, so that any sector in the track can be chosen by counting the number of Sector Notches since the last INDEX PULSE.

To perform a READ, the Read Gate is opened when the sector counter indicates the desired sector is under the Read Head. Again, the PREAMBLE is checked for the 16 consecutive "0" bits and reports a Marked Sector if they are not found. If they are found, it waits for the first "1" bit, compares the Track Check Character against the Seek Parameter and, if the compare fails, reports an Invalid Seek. If the Track Check Character is correct, the data is transferred to the disc controller buffer, and a CRC character is calculated from the data being transferred. This CRC is compared against the CRC at the end of the sector and, if it is not the same, a Read Error Status is reported. If the CRC matches, the READ is good. The controller tries three times before reporting a Marked Sector or Read Error Status.

3. Discs (cont)

When the SEEK is performed for a WRITE, the controller waits for the processor to finish filling the disc buffer, and then performs the track check as outlined above. Once the track is verified, the controller waits for the correct sector to come around and writes the PREAMBLE, Data, and a CRC character calculated while the data is being written. After the sector is written, the controller waits one revolution and rereads the sector. If an error status is returned, the controller rewrites the sector and again attempts to read it back. If an error status appears again, the controller attempts to mark the sector by writing 1's in the PREAMBLE and posting Marked Sector Status. If it cannot mark the sector, the original error status is reported.

DIAGRAM 6.1.3-A



DISK SECTOR INTERLACE

CARTRIDGE DISK & IOU-24A/C

3. Discs (cont)

C. Disc Error Status

The BEST Operating System is capable of reporting four distinct disc errors:

Status 34 (24) - Read Error

A Read Error occurs because the CRC character calculated on the data after the READ, does not match the CRC character at the end of the sector, which was calcuated when the data was written. This is the most common of the four disc errors. It can be caused by a dirty disc pack, dirty heads, or be indicative of more serious hardware problems. If the Read Error is caused by a dirty environment, cleaning the heads and packs should solve the problem. In the event the Read Error is a true calculation error, Read Error status can only be cleared by rewriting the sector. The utility DFULL32 can be used to READ and WRITE the sector. A Read Error after a WRITE to disc indicates that the controller could not mark the sector it was attempting to WRITE (see Status 54 (44) - Marked Sector).

Status 54 (44) - Marked Sector

A Marked Sector Status can occur on a READ or a WRITE. On a READ it indicates that 16 consecutive bits of 0 were not found in the PREAMBLE. A Marked Sector Status on a WRITE is posted when the controller, after two unsuccessful attempts to WRITE data to the disc and READ it back, marks the sector by writing 1's in the PREAMBLE. (If the controller is unable to mark the sector, Read Error (34) is posted). In general, a Marked Sector is uncommon while running under BEST, and usually indicates a more serious hardware problem.

Status 74 (64) - Invalid Seek

An Invalid Seek occurs when the Track Check Character in the PREAMBLE does not match the Seek Parameter or when the Seek Parameter is beyond the physical range of the disc. An Invalid Seek is the only disc error status that can, in some cases, be generated by bad pointers on the disc or improper software initialization. When an Invalid Seek occurs, all software causes should be checked (*KEYCHEK to check pointers, etc.) before attributing the failure to hardware.

3. Discs (cont)

Status 60 - Non-numeric Sector Number

A Status 60 is the only disc error that is not generated by a hardware status from the disc. This status comes from checking the SEEK parameter before sending it to the disc and generating a disc error if the parameter is not numeric. Status 60 is generally caused by a bad pointer in a file.

4. Printers

The section on Printers will be added at a later date.

5. Magnetic Tape

A. Buffer Size Control

QIC programs set up a maximum buffer size for all Input/Output operations through the use of a FILE statement at the beginning of each program. Although many FILE statements may appear in the program, the largest record size declared will be the buffer size. If no FILE statement is specified, the system sets a default buffer size of 136.

For magnetic tape, the default record size is 768 bytes. If a smaller record size is desired, the mnemonic (REC=N) can be used in the tape FORMAT statement to force the actual record size.

For example:

```
10 FORMAT (REC=50);A$;B$;C$
```

The (REC=50) mnemonic will force the record size to be exactly 50 bytes. If there are more than 50 characters, the extra characters will be truncated.

All other devices (except the disc) strip trailing blanks and force the record size to the actual count of all non-blank characters. The (REC=N) mnemonic is only relevant to the magtape, although it can be used with no error on other devices (it will be ignored).

NOTE: Further information will be added to the Magnetic Tape Section at a later date.

6. Communications Controllers

A. Asynchronous Communication Driver

The Asynchronous Communication Driver permits communication between the BEST user and a wide variety of asynchronous terminals and computers. Once OPENed and given the appropriate control block, the driver will continuously monitor the line for any incoming data. The condition of the line can be monitored by use of the QIC STS function. Data can be transmitted over the line by issuing WRITE or PRINT statements. Once the STS function indicates that data has been received, a READ or INPUT statement will give the user access to the data

This driver works in conjunction with the Asynchronous Communication Controller (IOU-15A/B) with a crystal speed appropriate for the user's requirements. The communication driver is included in the BEST Operating System, and can be accessed if a communication line is configured (Device Type - CM).

As with other peripheral devices, the communication line must be OPENed using the statement,

```
OPEN (X) 'CM1'  EXCP=YYYY
```

where X is an available LUN, and YYYY is the user-provided exception address. Immediately following the OPEN statement, the user must write a seventeen character hexadecimal string to the communication driver. This string is the control block for the line and has the following configuration:

```
XXX  FORMAT "@00AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPP@"
```

where the alphabetic character pairs have the following meanings:

AA - The AA control character is bit encoded where the bits have the following meanings when they are "1":

- 2(7) - Don't wait for DATA SET READY
- 2(6) - Don't wait for CLEAR TO SEND
- 2(5) - Data has EVEN PARITY
- 2(4) - Data has ODD PARITY
- 2(3) - Hardware system is 4 WIRE (Keep Request to Send True)
- 2(2) - Set data rate 8 times base data rate
- 2(1) - Set data rate 1/8 times base data rate
- 2(0) - Data is 5 bit code (Baudot Code)

BB - The BB control character is bit encoded where the bits have the following meanings when they are "1":

6. Communications Controllers (cont)

2(7) - Look for EOT and ETX as Data End
2(6) - Look for ENQ, ACK, NAK, or EOT as Control End
2(5) - Data has Longitudinal Redundancy Check Character
2(4) - Double Buffer Receive
2(3) - Set Full Duplex (Receive while Transmitting)
2(2) - Don't time out until after receiving first data character
2(1) - Data End consists of multiple characters
2(0) - Data Follows Poll

CC - The CC control character is bit encoded where the bits have the following meanings when they are "1":

2(7) Buffer End signaled by ETB character
2(6) Data is 7 bit code (Correspondence Code)
2(5) - 2(0) Undefined

The control characters DD and EE are counts for the number of 300 millisecond time intervals to elapse before the driver will issue Time Out Status. (These counts are hexadecimal).

DD - Specifies the time interval that the driver will wait, PRIOR to receiving data, before Time Out Status is given.

EE - Specifies the time interval that the driver will wait, BETWEEN characters while receiving data, before time out status is given.

FF - Specifies the number of characters expected in a poll if poll is set. In most cases this should be $00.

GG - This is the SOH character or, $00.

HH - This is the STX character or, $00.

II - This is the ETX character or, $00.

JJ - This is a count of the number of characters expected after the ETX character or, $00.

6. Communications Controllers (cont)

KK - This is the ETB character or, $00.

LL - This is the ENQ character or, $00.

MM - This is the EOT character or, $00.

NN - This is the ACK character or, $00.

OO - This is the NAK character or, $00.

PP - This is a count of the number of characters expected after a control end or, $00.

The communication driver responds to the STS function with a five character Status String. The characters of the string have the following meanings:

Byte 1: The first character is the constant "6" which informs the user that the device being accessed is the Asynchronous Communication Driver.

Byte 2: The second character informs the user of the condition of the communication line.

0 = Communication line is not ready
1 = Communication line is ready.

Byte 3: The third character informs the user of the condition of the receive buffer:

0 = Receive in progress
1 = Buffer available
2 = Data End Detected (ETX or EOT Received)
3 = Control End Detected (ENQ, ACK, NAK or EOT received)
4 = Time Out before receiving any data

Byte 4: The fourth character informs the user of the error status for this buffer.

0 = No errors on receive

6. Communications Controllers (cont)

1 = Data or LRC error on receive
2 = Buffer Overflow (Data Lost)
3 = Time out between character receptions during receive

Byte 5: The fifth character informs the user of the condition of the reverse channel.

0 = Reverse channel is off.
1 = Reverse channel is on

The four mnemonic controls available for the communications driver are defined as follows:

CBF - The (CBF) mnemonic in a FORMAT statement, tells the communication driver that the next WRITE will be a new driver control block, as defined above.

DIL - The (DIL) mnemonic in a FORMAT statement, tells the communication driver that the next WRITE will consist of a number to be dialed by the automatic calling unit. The line condition must be 0 before a dial may be issued. If the line condition is 1, an exception code 30 (INOP) will be returned. After the dial, the line condition should be checked. If the line condition is still 0 after the dial, it means the dial failed and should be retried.

SRC - The (SRC) mnemonic in a FORMAT statement, tells the communication line to SET REVERSE CHANNEL ON.

RRC - The (RRC) mnemonic in a FORMAT statement, tells the communication line to SET REVERSE CHANNEL OFF.

A READ is performed in the same manner as for other peripheral devices, e.g ,

READ (X,Y) or INPUT (X,Y) or INPUT (X) Y$

where X is an available LUN and Y is the appropriate FORMAT statement number, or Y$ is a defined string variable. The READ statement has two purposes. First, it is used to get the receive buffer for the user if there is one available; second

6. Communications Controllers (cont)

it is used to clear the receive status to 0. If there is no data waiting, the driver will return a string of length 0, and clear both receive status bytes to 0.

The Asynchronous Communication Driver will supply records of up to 220 characters and automatically split longer records into two buffers. The QIC Programmer is responsible for allocating his receive record sizes according to the characteristics of the device he is communicating with.

The WRITE statement has three purposes depending on what preceded it:

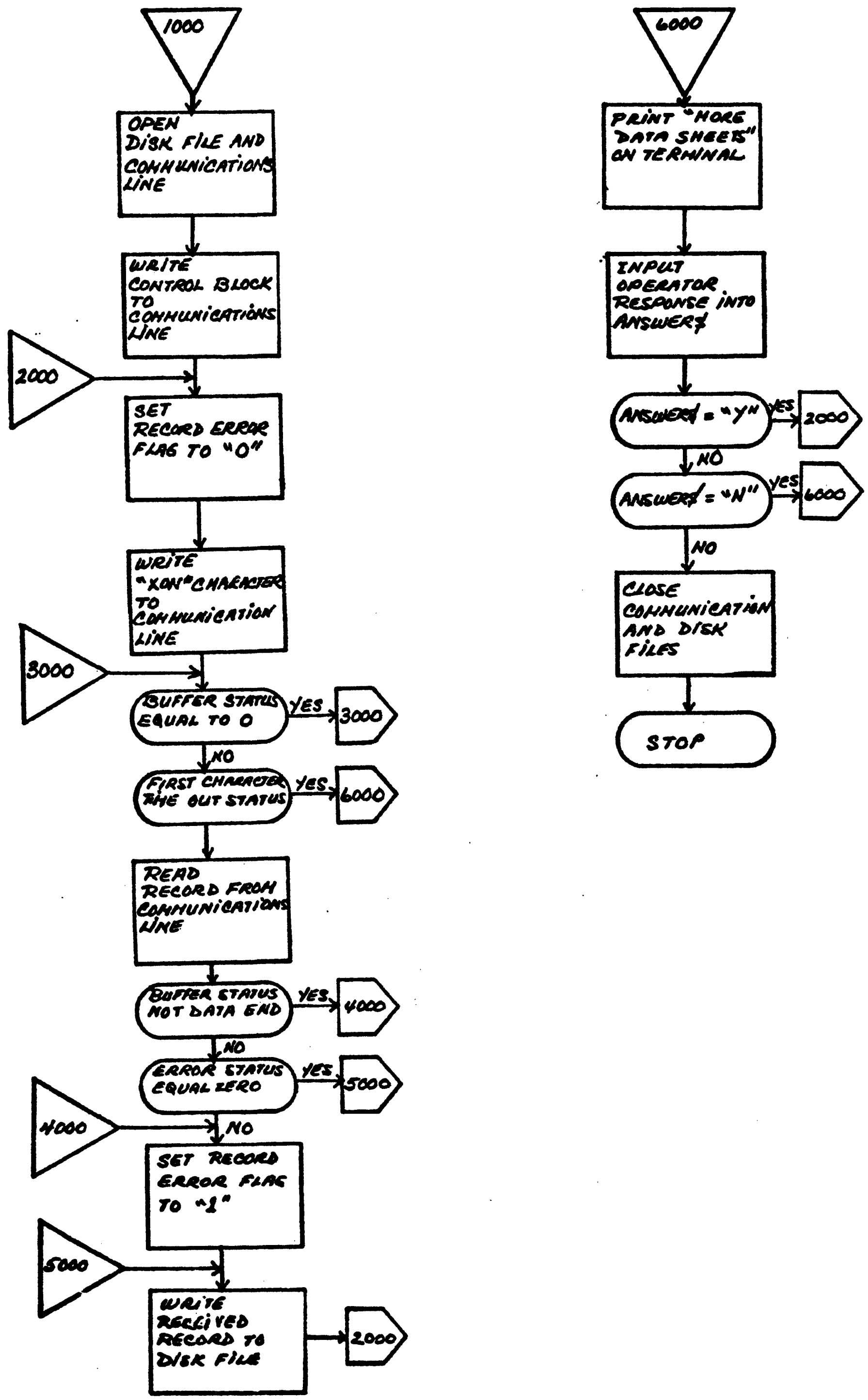
- o If the WRITE is preceded by an OPEN or CBF mnemonic, it is supplying the seventeen character control block.
- o If the WRITE is preceded by the DIL mnemonic, it is supplying a number to be dialed by the automatic dialing unit.
- o If the WRITE is preceded by neither of these, it is supplying the data to be transmitted. Any WRITE errors result in error 30.

A CLOSE statement releases the Asynchronous Communication Driver and disconnects the line.

The next three pages provide an example of a Communiations Program.

6.1 Communications Controllers (cont)

SAMPLE QIC PROGRAM FOR RECEIVING DATA FROM AN OPSCAN-17 OPTICAL CHARACTER READER USING OPSCAN LINE DISCIPLINE #275 AND WRITING THE RAW DATA TO A SEQUENTIAL DISC FILE



```
!
!       FILE STATEMENT
!
        FILE "RECDATA",150
!
!       DECLARATIVE SECTION
!
        LENGTH 1
        LOCAL ANSWERS,RECFLGS
        LENGTH 5
        LOCAL STATUSS
        LENGTH 140
        LOCAL RECBUFS
!
!       FORMAT SECTION
!
100     FORMAT RECFLGS/RECBUFS
!
!       COMMUNICATIONS CONTROL BLOCK
!
200     FORMAT "@00@← !CONTROL BLOCK HEADER CHARACTER
@20@← !EVEN PARITY DATA AND NORMAL DATA RATE (1200 BAUD WITH 4.92MHZ CRYSTAL)
@90@← !EOT OR ETX SPECIFY DATA END AND DOUBLE BUFFER RECEIVE
@00@← !NO RELEVANT CONDITIONS IN THIS BYTE FOR OPSCAN DISCIPLINE #275
@64@← !30 SECOND INTERVAL BEFORE NO FIRST CHARACTER TIME OUT IS ISSUED
@0A@← !3 SECOND INTERVAL AFTER START OF DATA BEFORE TIME OUT BETWEEN CHARACTERS
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@93@← !"XOFF" CHARACTER FOR ETX
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@93@← !"XOFF" CHARACTER FOR EOT
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@← !NOT RELEVANT TO OPSCAN DISCIPLINE #275
@00@"  !NOT RELEVANT OT OPSCAN DISCIPLINE #275
!
!       "XON" CHARACTER TO START EACH TRANSMISSION
!
300     FORMAT "@11@"
!
!       RECEIVE BUFFER FORMAT STATEMENT
!
400     FORMAT RECBUFS
500     FORMAT (ET);"OPSCAN-17 SAMPLE COMMUNICATIONS PROGRAM USING DISCIPLINE ←
#275"
!
!       CODE SECTION
!
!       THIS SECTION OPENS THE COMMUNICATIONS AND DISK FILES
!       IT ALSO INITTIALIZES THE COMMUNICATIONS LINE WITH THE
!       CONTROL BLOCK AND WAITS FOR THE COMMUNICATION LINE
!       TO COME READY.
!
10001
        PRINT (0,500)
        OPEN (1) "RECDATA"
```

```
PSCN17S' ON '!;      PAGE   2

                 OPEN (2) "CM1"
                 WRITE (2,200)
         1100!
                 STATUS$ = STS(2)
                 IF SUB(STATUS$,2,1) EQ "0" GOTO 1100
         !
         !       THIS SECTION IS THE COMMUNICATIONS LOOP
         !       IT WRITES AN "XON" CHARACTER TO THE OPSCAN TO
         !       START IT TRANSMITTING TO THE COMPUTER
         !       THE CODE THEN WAITS FOR A RECEIVE BUFFER
         !
         2000!
                 WRITE (2,300)     ! SEND "XON"
                 RECFLG$ = "0"     ! CLEAR RECORD ERROR FLAG
         3000!
                 STATUS$ = STS(2) ! GET COMMUNICATION LINE STATUS
                 IF SUB(STATUS$,3,1) EQ "0" GOTO 3000 ! WAIT FOR BUFFER
         !
         !       THIS SECTION HANDLES THE RECEIVED DATA
         !       IF NO DATA WAS RECEIVED THE PROGRAM BRANCHES TO CHECK FOR TERMINATION
         !       ELSE IT READS THE RECEIVED DATA AND CHECKS FOR DATA END
         !       IF NOT DATA END IT BRANCHES TO SET THE ERROR FLAG AND WRITE THE DATA
         !       TO DISK
         !       ELSE IF DATA ERROR IS TRUE IT SETS THE ERROR FLAG AND WRITES TO DISK
         !       ELSE IT WRITES THE DATA TO THE DISK
         !
                 IF SUB(STATUS$,3,1) EQ "4" GOTO 6000 !IF NO DATA RECEIVE BRANCH TO END
                 READ (2,400) ! ELSE READ THE DATA
                 IF SUB(STATUS$,3,1) NE "2" GOTO 4000 !IF NOT DATA END THEN ERROR
                 IF SUB(STATUS$,4,1) EQ "0" GOTO 5000 !IF DATA OK  THEN DON'T SET ERROR
         4000!
                 RECFLG$ = "1" !SET DATA ERROR FLAG ON
         5000!
                 WRITE (1,100)  !WRITE DATA TO DISK FILE
                 GOTO 2000      !RECEIVE NEXT RECORD
         !
         !       THIS SECTION CHECKS FOR END OF COMMUNICATION
         !       IF OPERATOR INDICATES THE THERE ARE MORE DATA
         !       SHEETS TO BE READ THE PROGRAM RETURNS TO THE
         !       COMMUNICATION LOOP
         !       ELSE IT CLOSES THE FILES AND TERMINATES
         !
         6000!
                 PRINT (0) "ANY MORE DATA SHEETS TO BE READ? (Y/N) "
                 INPUT (0) ANSWER$
                 IF ANSWER$ EQ "Y" GOTO 2000
                 IF ANSWER$ NE "N" GOTO 6000
                 CLOSE (1)  ! CLOSE DISK FILE
                 CLOSE (2)  ! CLOSE COMMUNICATION LINE
                 STOP
                 END
```

7. Clocks

Any of the Qantel Systems may optionally contain a System Clock. Once started, it will maintain the system variable, TIME$ within one second. The clock may also be OPENed and read by an INPUT or READ instruction if one second time resolution is inadequate.

As with any other peripheral device, the clock is accessed by using the statement:

```
OPEN (X) "CL1" EXCP=YYYY
```

where X is an available LUN, and YYYY is the user-provided exception address if the clock is unavailable. The clock must be OPENed before any of the clock functions are performed. Once OPENed, the clock may be read from or written to in the standard format.

The clock is initially started up by writing to the clock in the following format:

```
LENGTH 8
LOCAL TIMER$
10 FORMAT TIMER$
   OPEN (1) 'CL1' EXCP=9999
   (Assign TIMER$ the starting time value)
   WRITE (1,10)
   CLOSE (1)
```

where TIMER$ is an 8 byte unedited string value that has the appearance of HHMMSS00 (H=hour, M=minute, S=second) Once the clock is started, it may be read for the correct time or the value of TIME$ may be used. To obtain the correct time through either a READ or INPUT statement the format is

```
READ (1,10) or  INPUT (1,10) or, INPUT (1) TIMER$
```

where the lengths and formats are the same as listed in the above example. Again, the variable TIMER$ must be an 8 character, unedited string variable.

The system clock will respond to the STS function with a two character string. The first character, a '5', informs the user that the device accessed is a clock. The second byte will be a '0' if the clock has not been started, or a '1' if the clock has been started and TIME$ is being maintained within 1 second.

The system variable TIME$ is an eleven character edited string

7. Clocks (cont)

of the form HH:MM:SS:00. This variable is available to any user at any time, once the clock is started. The clock maintains 24 hour time, i.e., 1:00 PM is 1300 hours. If an AM/PM convention is desired, it must be converted by the user program.

As with other peripheral devices, the statement CLOSE (1) releases the clock and makes it available to any other user.

8. Card Readers

The Card Reader Section will be written at a later date.

6.2 Buffers

1. System Buffers

The section on System Buffers will be written at a later date.

2. Record Buffers (User)

The User Record Buffer is the area reserved within a user partition for I/O Operations. This buffer declares the maximum number of characters accessed during one operation, such as a READ or a WRITE. The buffer size is established through a FILE statement in the user program e.g ,

FILE 'CUSTMAST',252

Any number of FILE statements may appear in the program, but the user buffer area is established by the largest FILE statement. A default buffer size of 134 bytes is used if no FILE statement appears, or if the FILE statement is smaller than the default size.

On an attempt to READ a record, the data specified in the FORMAT statement is moved into the user record buffer from the system buffer. If the number of bytes specified in the FORMAT statement is greater than the defined user record buffer, the system will generate an Error 44: I/O Buffer Overflow.

On an attempt to WRITE a record, the system uses the record size from the file directory entry to determine how much data to pull from the user record buffer. If this buffer is smaller than the actual record size of the file, whatever is in memory directly behind it will be picked up and written at the end of the data record. To eliminate "garbage characters" at the end of a record, the user record buffer should always be declared the full size of the largest record being written.

CRT formats do not use the user record buffer. Instead, the CRT driver uses the system record buffer (768 bytes) for its I/O to avoid requiring large buffer areas in the user program for CRT I/O. Other drivers, such as the one for the Printing Terminal, use the user record buffer. In cases where a format is written to work on a CRT <u>and</u> on a Printing Terminal, the Printing Terminal would get an Error 44 attempting to write the same format as a CRT, unless the size of the user buffer was increased to accommodate the format.

3. Buffer Pooling

The section on Buffer Pooling will be written at a later date.

6.3 Memory

1. Memory Addressing

The QANTEL Hardware System can only "look at" a total of 32K at any given time. By definition, the first 16K of memory must always be active, and the second 16K is swapped in and out from the remaining memory based on the operation being performed. The method used for addressing memory depends on the type of processor in the system.

Standard Processor (Q7)

The maximum amount of memory available with these processors is 64K. This 64K is assigned BANK names in increments of 16K.

Increment	Addressing	Bank Name
1st 16K	\$0000-\$3FFF	Bank A
2nd 16K	\$4000-\$7FFF	Bank C
3rd 16K	\$4000-\$7FFF	Bank D
4th 16K	\$4000-\$7FFF	Bank E

Banks A and C contain the BEST Operating System and, in some configurations, 6K of user space. Banks D and E are available for user partitions. To access the user partitions in Banks D and E, the hardware responds to a SET BANK Instruction This instruction "activates" a particular 16K segment of memory, whose addresses are absolute. For this reason user partitions <u>cannot</u> overlap these 16K banks. Any user partition must "wholly" reside in a bank of memory that is activated by a SET BANK instruction. The Standard Processor can use MEM 3B (4K Modules) or MEM 5B (8K Modules) memory boards. (See Figure 6.3.1)

High Speed Processor (Q7.5)

These processors use four 17-bit base registers to access up to 128K of main memory. As with other processors, only 32K is accessible with a single instruction.

Each location in memory is specified by a base register number (0,2,4 or 6) and an offset (between \$0000 and \$1FFF). The effective memory address is computed by adding the contents of the base register to the offset. Each base register, in effect, contains the first address of an 8K block of memory.

1. Memory Addressing (cont)

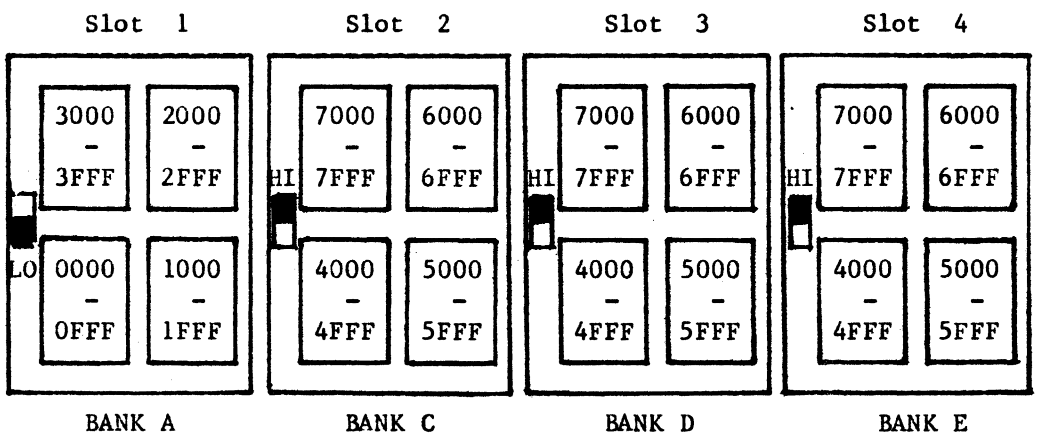
When IPL is pressed, the 1300 sets up the base registers to access the first 32K of memory:

BASE REGISTER	CONTENTS AT IPL
0	$0000
2	$2000
4	$4000
6	$6000

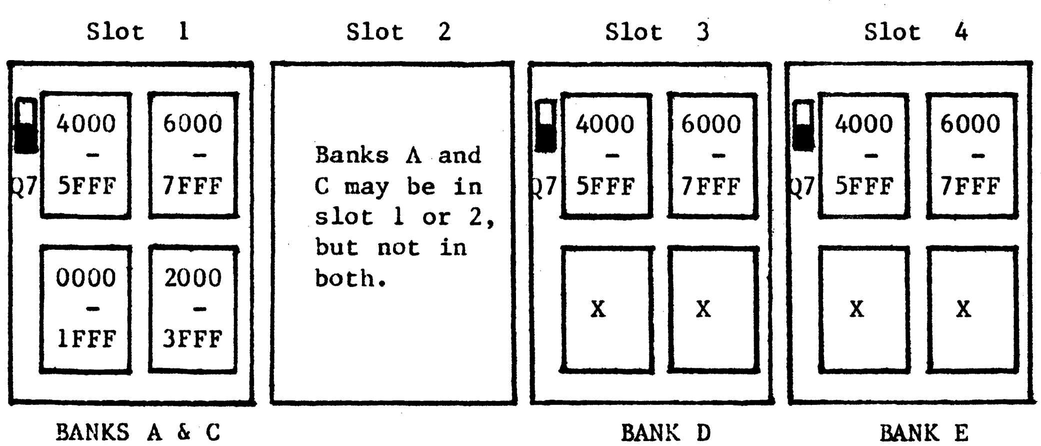
The high speed processor can simulate the banking of the standard processor When a SET BANK instruction is executed, the Q7.5 loads the base registers to access the predefined 16K bank of memory. Base Registers allow all memory to be considered "Contiguous" because any portion of memory can be described by the beginning register position and the offset. The high speed processor requires MEM 5B memory boards. (See Figure 6.3.1)

Figure 6.3.1 MEMORY LAYOUT

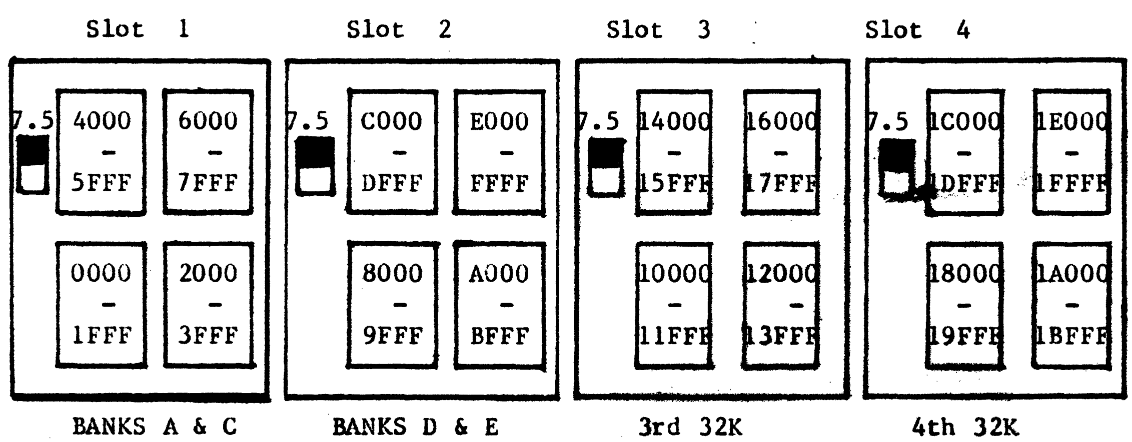
MEM 3 - Systems 800,900,950,1200



MEM 5 - Systems 800,900,1200



MEM 5 - Systems 950*,960,1300



*Switch should be in Q7 position.

2. Memory Board Assemblies

MEM 5AP

A MEM5B consists of 8K of 8-bit bytes (8192 bytes). Each MEM 5AP assembly can contain one to four MEM5B boards, allowing up to 32K per slot. The hexadecimal address range of a MEM 5AP board is $0000 to $7FFF. The following diagram shows the arrangement of a MEM 5AP board

$4000 - $5FFF	$6000 - $7FFF
$0000 - $1FFF	$2000 - $3FFF

The MEM 5AP board has one switch with a "7" (down) and "7.5" (up) position. This switch is set to the "7" position for a standard processor, and to the "7.5" position for the high speed processor. It is imperative that this switch is in the proper position for the particular processor being used.

When data is written to parity memory, a parity bit is calculated and stored as the ninth bit with each byte. Whenever a byte is accessed, its parity bit is again computed and compared with the original parity bit. If they are unequal, the memory is capable of interrupting the main processor.

BEST reports all parity errors encountered with the following message displayed on device 0:

MEM.FAIL X/YY

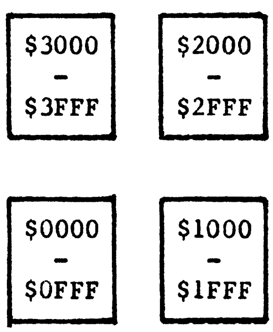
where X is the device number of the MEM 5AP, and YY is the Status. BEST will halt (Start/Stop light) on an error, and pressing Start/Stop will produce no results. When the halt occurs, the light on the MEM 5AP board corresponding to the 8K block where the error occurred will be on. If the system is IPLd, the light will stay on, but the memory failure interrupt will not occur until another parity error occurs, either in the same (or some different) location.

2. Memory Board Assemblies (cont)

In the case of a memory failure, the data in at least this location may be incorrect. Do Not Continue Processing. Leave the failure message on the screen and call a Field Service Engineer. In those systems where diagnostics are provided, write down the error information, IPL and run the Memory Test.

MEM 3AP

Some configurations may contain memory in 4K Modules (MEM 3Bs). Up to four MEM 3Bs can fit on a board assembly, allowing 16K of memory per I/O slot. This memory does not support the parity error features, and cannot be used on high speed processors. The following diagram shows the arrangement of a MEM 3AP board:



The MEM 3AP board has one switch with a "LO" (down) and a "HI" (up) position to distinguish low (first 16K) from high (next 16K) memory. This switch is set to the "LO" position if the board is the first memory board in the processor. It is set to the "HI" position on any other MEM3AP board.

7.1 BEST Error Codes

ERROR	DESCRIPTION
02	End of File
03	Disc full, no allocation unit available
04	Attempted WRITE to a Keyed File without a legal Index
05	Attempted CREATE without a Disc Label
11	Attempted OPEN on a file not found
12	Attempted CREATE of a file that already exists
24	Card Reader READ error
25	Card Reader...Hopper Empty/Stacker Full/Hold
26	End of File, Card Reader or Magnetic Tape
27	Tape READ or WRITE error
28	Printer dropped VFU on WRITE...System reloaded Standard VFU
29	Disc Read Error on Load (Communications Line)
30	Device Inoperative
31	Device Unavailable
32	Key Not Found
33	Record not available (EXTRACTed)
34	Logical Unit Number Unavailable for OPEN (already in use)
35	Attemped READ or WRITE on a file not OPENed
36	Attemped ERASE on a file that is OPEN
37	Disc Unavailable for GET or PUT
38	Device Unable to perform function
39	Disc Unavailable for LOCK or UNLOCK
40	Disc Unavailable for CREATE
41	Edit Mask length incorrect
44	I/O Buffer Overflow
46	Non-numeric input in a numeric field
47	Parameter too large
48	Keysize greater than 32 for CREATE
50	Array subscript out of range
51	Divide overflow
53	Keyed Access to a non-Keyed file
54	Cannot DELETE from a non-Keyed file
60	Cannot IPL while other users are active
61	Program not found
62	Program too large for partition
63	Cannot RUN a non-object file
64	Invalid partition name
65	Partition busy
66	Partition not BACKGROUND

System Errors...Cannot be handled by an EXCP branch

ERROR	DESCRIPTION
80	Maximum I/O Buffer size exceeded
82	Top Key Sector not in DELETE stack
83	Disc Error Reports SECTOR #, DISC DEVICE, ERROR STATUS STATUS (34) = Read Error STATUS (54) = Marked Sector STATUS (74) = Invalid Seek STATUS (60) = Invalid sector number
84	Key sector search impossible
85	Invalid parameters for REAL program
86	Active File List Overflow (Number of files OPENed exceeds maximum AFLs Configured)
90	System Return Stack Overflow/Underflow
91	Invalid Load Item in Object File
92	GOSUB Stack Underflow
93	GOSUB Stack Overflow
94	Directory Entry unavailable for CLOSE
95	Next Key Unavailable for DELETE
96	New Key found during Insert
97	New first Key found during Insert
98	EXTRACTed record not found in table
99	Re-allocation of sectors below sector 10

7.2 COMMON Structure

The ability to pass data from one program to another within the same partition is accomplished through the use of a COMMON declarative statement in a QIC program. COMMON variables establish a "reserved" data area at the start of the partition, equal to the combined length of all common variables in that program. The number and length of the declared common variables establish where the remainder of the program will be loaded in the partition. Thus, a second program to run in a partition that must access a first program's data, must have the same COMMON declared to avoid loading over and destroying data.

COMMON is stored in memory in declared order. Variables may be added in subsequent overlays without disturbing data already passed from a previous program, as long as the original variables still occur. Numeric COMMON variables are stored in memory in their formatted precision. String COMMON variables are stored in memory in their declared length, with one extra byte reserved for the length.

CLEAR CLEAR COMMON, and CLEAR LOCAL set all variables to $00.

7.3 System Variables

System Variables are data areas not declared in any program, but available to any partition on a common basis. These variables are available at any time, and are stored in the first 16K of memory. Any user may load or change the contents of the system variables, thus providing one method of passing messages between partitions. The System Variables available are:

TERM$	-	Whenever a user is activated by the system, TERM$ is updated with the device name of that terminal, i.e., "T00", "T01", etc. TERM$ will always be blank if the active partition is Background. TERM$ is only updated with a valid device name if that device is a controlling terminal. TERM$ provides a method to protect terminals from running a particular set of application programs.
TIME$	-	TIME$ is an 11 character, edited string field, in the form HH:MM:SS:00 which is maintained within one second by a system clock. If no clock is present TIME$ may be used as a message passing area.
DAY$	-	DAY$ is an 8 byte, unedited string field that can be used for a common date or any other message.
MESSAGE$	-	MESSAGE$ is a 32 byte, unedited string field that may be used as a message passing area or common data area.
PARTITION$	-	PARTITION$ is similar in function to TERM$ and may be used in the same manner. When any partition is ACTIVATEd, PARTITION$ is updated with the assigned partition name, e.g., "P00".
ITERM$	-	ITERM$ is a 3 byte variable that carries the "initiating terminal" device name. ITERM$ will be the same as TERM$ for any Foreground Partition. ITERM$ will contain a device name of the terminal that ACTIVATEd a Background Parition (TERM$ will be blank).

Any of the system variables are available to a user program directly, or through a string assignment statement, i.e., DATE$=DAY$. TIME$, DAY$, and MESSAGE$ may be set through a string assignment. Once the variables are set they are maintained until IPL or power down.

System variables provide an easy method to have a limited amount of data common to all users. However, they cannot be "locked" to prevent other users getting the same copy of the data when a task gives up control. If this effect is desirable, the access and assignment of the variable must be done in a tight QIC routine which does not perform any I/O or task breaks.

8.0 GLOSSARY

The Glossary Section will be included at a later date.

APPENDIX A - DESCRIPTION OF RESERVED MEMORY

AREA	USE
0 - 15	Accumulator positions. Used for the results of multiple add instructions, etc. When single address instructions are used, the implied second operand is the accumulator and its contents.
16 - 17	Stores the current program address while the CPU is servicing an interrupt.
18	Stores the contents of the condition switches carry, minus, and non-zero, and status of interrupt availability.
19 - 20	Contains the address which will replace the current program address when an interrupt is recognized.
21 - 22	One count more than the final address at the conclusion of an I/O operation.
23	Stores the I/O status byte when status-in or Read Status 2 is executed.
24 - 25	Stores the character match address for Search Equal, Scan Left, Scan Right, etc.
26 - 31	Micro-program utility bytes.

The Coreimage program CFIG, (Configurator), creates the system tables for the BEST Operating System, and for ALMOST (the single user operating system used by the Compiler and some REAL utilities). Based on the requirements specified, CFIG will:

1) Create tables to describe the devices present on the system, the user partitions, and the number of system buffers.

2) Allow selective updating of specific device types without requiring re-entry of all configuration information.

3) Automatically assign user partition areas, given the length of the partition in "K", decimal, or hexadecimal notation.

4) Allow printing of the configuration information to any terminal or line printer.

NOTE: "Y" or null is accepted by CFIG as "yes"; any other character is "no".

PROCESSING

IPL and Bootstrap from the disc to be configured.

PROGRAM ID:CFIG
CONFIGURATOR XX.X MM/DD/YY
E-END, C-CONFIGURE, P-PRINT CONFIGURATION
ENTER CHOICE:

Enter "E" to return to the loader prompt; "C" to configure; or "P" to print the current configuration.

If "E" is selected, control returns to the loader.

If "C" is selected:

RETAIN ALL DEVICES? (Y/N)

Enter "Y" or null to retain existing device information; enter "N" to clear the existing device information.

<u>T-TERMINAL, CR-CARD READER, DK-DISC, MT-MAG TAPE, LP-LINE PRINTER, P-PARTITIONS, CL-CLOCK, CM-COM LINE, X-SYSTEM TABLES, E-END, *-ABORT</u>

Flag 1,2, or 3 will truncate this message.

<u>DEVICE TYPE:</u>

Enter "P" or "T" or "CR" or "DK" or "MT" or "LP" or "CL" or "CM" or "X" or "E" or "*".

NOTE: If "*" is chosen, CFIG will abort before writing to the disc and will return to the "E-END, C-CONFIGURE, P-PRINT CONFIGURATION" message.

If "T" is chosen, all previously configured terminals are cleared.

<u>TYY:</u>NX

Enter the Terminal Number where X is the controller number, N is the device number of the terminal, and TYY is the assigned terminal name, (i.e., T00, T01, etc.).

<u>DEVICE TYPE (1-15 LINE, 2-27 LINE, 3-TYPEWRITER):</u>

Enter "1" for a 15-line QCRT, "2" for a 27-line QCRT, and "3" for a typewriter.

<u>PARTITION NAME:</u>

Enter any partition name (P00, P01, etc.) which will be specified in the "P" option of the configurator.

A null entry for partition name means that the device is not a controlling terminal, i.e., it is to be opened passively by another terminal.

NOTE: It is permissable to enter the device number, type and partition name on one line if they are separated by commas. For example:

<u>T00:</u> 00,2,P00	27-line QCRT assigned to partition "P00"
<u>T01:</u> 02,3	Passive typewriter

If "P" is chosen, all previously configured partitions are cleared.

PXX:

Enter the partition length, which can be expressed in three ways: K, decimal or hexadecimal. For example:
P01: 4K (=4*1024 or 4096 bytes)
P02: 4000 (=4000 decimal bytes)
P03: $1000 (=1000 hexadecimal bytes)

The minimum partition size is 2K (=$800,=2048); and the the maximum partition size is 16K (=$4000,=16384).

NOTE: Under Version 13.X, for each partition configured, 768 bytes ($300) is added by the Configurator and reserved for system user (Task Header). Therefore, if 4K is requested, 4864 ($1300) bytes is configured with 4096 ($1000) for the user and and 768 ($300) for the system.

If a partition is configured that is not assigned to a controlling terminal, then that partition is a background partition.

If "CR" is chosen, then all previously configured Card Readers are cleared.

CRX:

Enter the hexadecimal device number of the card reader (e.g , E or OE).
A null entry terminates the configuring of card readers.

Discs (DK), line printers (LP), clocks (CL), and communication lines (CM) are configured in the same manner as card readers (CR). Clocks and Comm lines are limited to a maximum of one device each. Upper and lower platters of the 3+3 and 6+6 discs are configured as separate discs, (e.g., OD and 1D). The configuration order for discs determines the order in which the platters will be accessed under BEST.

If "MT" is chosen, all previously configured magnetic tape drives are cleared.

MTX:

Enter the hexadecimal device number of the magnetic tape drive (e.g., 8 or 08).
A null entry terminates the configuring of magnetic tape drives.

FAST OR SLOW(F,S)?

Enter "F" if the tape is a "Read after Write" drive;
Enter "S" if the tape is a "Non Read after Write" drive.
NOTE: It is permissable to enter both the device number and the device type on the same line if they are separated by a comma. For example:

MT1: 8,F
MT2: 9,S
MT3: (null)

It is not necessary to configure a magnetic tape drive to use the system utilities such as *BACKUP or TAPE.
The tape drive must be configured if QIC programs access the drive, e.g., OPEN (1) 'MT1'.

If "X" is chosen, the system type, memory size and buffer information are requested. The number of buffers, AFL's and extract entries, plus the number of other peripheral devices configured determines the amount of partition space left in the first 32K. This space will be assigned a partition if the system being configured in an 800 or 900 system.

SYSTEM TYPE (800,900,950,1200):

Enter the System Model Number.
On Version 13.X this question does not appear since the hardware must be a System 960 or 1300.

TOTAL MEMORY SIZE (XK):

Enter the amount of memory available on the machine.

800,900 - 32K is assumed.
950,960,1200 - Minimum of 32K, maximum of 64K
1300 - Minimum of 32K, maximum or 128K

NUMBER OF BUFFERS:

Enter the number of buffers to be configured into the system. Under version 13.X, more space is available for buffers. Other Versions allow 3 buffers maximum for 5 users. The suggested number of buffers is:

# of Users	# of Buffers - 13.X	# of Buffers - Other
1	2	2
2	2,3	2,3
3	2 3,4	2,3,4
4	2,3,4,5	2,3,4
5	2,3,4,5	2,3
>=6	2,3,4,5	---

NOTE: Versions CFIG earlier than 13.X or 14.X allow configuration of 1 buffer. File system operations under 13.X and 14.X require a minimum of 2 buffers.

<u># OF AFL'S:</u> (Version 13.X and higher, only)
Enter the maximum number of files that will be opened by <u>all</u> users at any one time. (Each user may have a maximum of 8 files open at one time).

<u>NUMBER OF EXTRACT ENTRIES:</u> (Version 13.X and higher, only)
Enter the maximum number of records that will be EXTRACTed by all users at any one time.

<u>MEMORY DEVICE #'S:</u> (Version 13.X and higher, only)
Enter the device number(s) of the MEM 5A board(s).
For example:
<u>MEMORY DEVICE #:</u> 5
<u>MEMORY DEVICE #:</u> 6
<u>MEMORY DEVICE #:</u> (null)

If "E" is chosen, all configuration information has been entered.
<u>LOAD MODULE MESSAGE:</u>
Enter the IPL message to be displayed when BEST, DKIN or CFIG is loaded (maximum of 17 characters). A null entry means the module message is not changed from the previous configuration.
<u>END OF TABLES = $XXXX</u>
This is the address of the last location used for system table space.

(There is a pause while the coreimages are configured and written to disc).

If "P" is selected:
<u>LISTING DEVICE:</u>
Enter XY, where X is "T" for a terminal and "L" for a line printer; and Y is the hexadecimal device number for the controller (e.g. T0 for terminal 0, LF for line printer $0F, etc.)

CONFIGURATION ERROR MESSAGES

NOTE: When an error occurs on a configuration, the last valid configuration will remain intact.
If any disc errors occur (other than inop), press ST/SP to retry the I/O Operation; if errors persist the disc pack or drive may be faulty.

TABLES DO NOT FIT BEFORE $XXXX

The tables that the Configurator builds for BEST must fit between approximately $3000 and $5000. (The exact values depend on the release level and the system type.) The usual cause for exceeding this limit is configuring too many partitions or buffers.

XXXK EXCEEDED BY PARTITIONS

The partition layout designated does not fit in the memory space specified under the "X" option. The Configurator does not check how much memory is physically present on the machine. On the 800/900 systems, this message means that in the available 32K, not enough space for the requested partition was left after the system buffers and tables were configured.

TOO LARGE FOR DISC AREA

This message is displayed if the COREIMAGE being written does not fit in the disc area assigned to it (sectors 401 to the start of the BEST patch area).

DISC IS UNCONFIGURED

This message is displayed if the user attempts to configure an unconfigured disc (e.g. brand new), and answers "Y" to "RETAIN ALL DEVICES (Y/N)?"

UNAVAILABLE DEVICE TYPE

Driver associated with the device is not present in the system being configured.

AT LEAST ONE TERMINAL MUST BE ENTERED

This message is displayed if no terminals are configured for the system.

AT LEAST ONE PARTITION MUST BE ENTERED

This message is displayed if no partitions are configured for the system.

BUFFER OVERFLOW ($XXXX)
The number of buffers configured for the system does not fit in the allowable buffer area (below $4000). Try configuring one less buffer for the system.

ALMOST TABLE OVERFLOW AT $XXXX - ALMOST CONFIGURATION ABORTED
This message is displayed if, while configuring ALMOST, the tables become too large. Probable cause: Too many devices configured. The BEST configuration has already been written to disc; however, AMOST is not configured. If this message is displayed, the system must be reconfigured with fewer devices.

PRINT CONFIGURATION ERROR MESSAGES

DEVICE XX UNAVAILABLE
The listing device is invalid (status = $FF).

DISC IS UNCONFIGURED
This message is displayed if an attempt is made to print the configuration of an unconfigured disc.

ENTER 'TO' FOR TERMINAL 0, 'LF' FOR LINE PTR F, ETC
This message is displayed if an invalid response is entered to the "LISTING DEVICE" question. If the listing is to be printed on a QCRT or typewriter, enter "T" followed by the hexadecimal device number of the controller; enter "L" and the device number for display on the printer. (Use "T" for printing terminals.)

If any line printer error occurs, one of the following messages is printed:

PTR 0X INOP (SS)
PTR 0X VFU ERROR (SS)

where X is the device number and SS is the status. Reload the VFU for a VFU error. "INOP" can be caused by an invalid device number (SS="FF), out of paper, printer off-line, etc.

ACTIVATE PARTITION,PROGRAM,DISC='XXX',EXCP=NNNN where,

PARTITION = Any configured partition name, represented as a string variable or constant, e.g., "P00", "P01", etc.

PROGRAM = Any program name to be executed in the partition specified. If the partition is Background the program cannot attempt to WRITE to TERM$ or reference LUN 0, unless LUN 0 was previously OPENed for a file or device.

DISC='XXX' = (Optional) The label of the disc from which the program should be loaded. Default is the first occurrence of the program name on the configured set of disc(s).

EXCP=NNNN = The exception branch to be taken if an error occurs during the ACTIVATE.

Possible errors:

64 = Invalid Partition Name
65 = Partition Already Busy
61 = Program Not Found
62 = Program Too Large
63 = Program Not Object File
83 = Disc Error During Load
91 = Invalid Item In Object File

Errors that are normally fatal can be handled by an EXCP branch on the ACTIVATE statement only.

All errors will take the specified exception handling branch within the initiating program and the partition being ACTIVATEd will be set to the CLEAR state.

The first 256 bytes of the initiating partition are transferred to the partition being ACTIVATEd (before the program is loaded). This allows message passing via COMMON from one partition to another, if the program being ACTIVATEd has the appropriate amount of COMMON declared.

TERMINATE PARTITION, EXCP=NNNN where,

PARTITION	=	Any configured Background partition name, or PARTITION$
EXCP=NNNN	=	The exception branch to be taken if an error occurs. Possible errors are: 64 = Invalid Partition Name 66 = Partition Not Background

APPENDIX D - BACKGROUND IN REAL

The file, #ATMACRO, has been created to perform BACKGROUND related operations in REAL. This provides definitions of new macros and system variables. This file can be used by including the following statement in the READ source program:

USE #ATMACRO

The following operations will then be available:

XACTIVATE PARTITION;PROGRAM;DISC
ACTIVATE a program in another partition.

XTERMINATE PARTITION
TERMINATE a program in another background partition.

PARTITION
Is the right hand address of a 3-byte partition name, e.g.,

="P03"
PART where PART DA 3
@PRTA where PRTA DAC ="P17"

XACTIVATE and XTERMINATE both perform like other "X" system calls in that the zero condition switch is set on exit from the routine in case of error.

The general performance of these operations is similar to their QIC counterparts, ACTIVATE and TERMINATE.

The following System Variables are available to REAL programs:

Variable Name	Use	Length	Type	When Valid
ZPART$	Current Partition Name	4	Q	Always
ZITERM$	Initiating Terminal Name	4	Q	Always
ZPARTLEN	Partition Length	2	B,@	Always
ZRECSIZ	Record Size	2	B,@	After File Access
ZKEYSIZ	Key Size	1	B,@	After File Access
ZFILTYP	File Type	1	B,@	After File Access
	\$01 = Sequential			
	\$02 = Contiguous			
	\$04 = Keyed			
	\$10 = Object			
	\$30 = Stand-Alone			

where TYPE is defined as,
Q=QIC string format, left-hand address
B=Binary, right hand address
@=Value is already an indirect address

APPENDIX E - MNEMONIC TABLE

MNEMONIC	DESCRIPTION	CRT II	CRT I	VIDEO PTR	PRINTING TERM	SELECTRIC	300/600 LPM UFL PTR	200 LPM	100 LPM	240-1100 LPM 100-1800 LPM	JAPANESE PRINTER	MAGTAPE	SEA FILES	HEX VALUE
B=N	Insert N blanks			X										
BF	Blank Fill	X												0406
BS	Back Space											X		
BM	Set Blind Mode				X									
BR	Set Black Ribbon			X	X									
C=N	Skip to column N			X										
CF	Clear Foreground	X	X											0600(00)
CFLD	Clear Field	X												0407
CH	Cursor Home	X	X											010100
CR	Carriage Return	X	X		X									0D
CS	Clear Screen	X	X											0500(00)
DS	Double Space			X										
EN	Enter Normal Mode	X	X		X	X								
EM	Echo Mode				X									
ET	Typewriter Mode	X	X		X	X								
FF	Form Feed			X			X	X	X	X	X			
G=N	Gap = N Lines			X										
KN	Kana Field Follows	X	X											0F
LI	Line Insert	X												040B + 27*'00'-15 L 52*'00'-27 L
LD	Line Delete	X												040A + 27*'00'-15 L 52*'00'-27 L
LF	Line Feed						X	X	X	X	X			
LS	Line Suppress										X			
PF	Print Follows			X										040E
PR	Print Hidden Message			X										0409
PS	Print Screen			X										040F
P=N	Page = N Lines			X										
RB	Ring Bell	X	X											07
RD	Roll Down Screen	X	X											010100040B + 27*'00'-15 L 52*'00'-27 L
RJ	Right Justified Fld	X	X											0B
RR	Set Red Ribbon			X	X									
RU	Roll Up Screen	X	X											010100040A + 27*'00'-15 L 52*'00'-17 L
RW	Rewind Tape											X		
SB	Start Background	X	X											08
SF	Start Foreground	X	X											0A
SK=N	Skip to Channel N			X			0-11	0-7	0-1	0-7	0-3			
SL=N	Skip N Lines			X			0-15	0-2	1	0-15	0-3			
SSB	Suppressed Bckgrnd	X	X											040C
TF	Set Tab Rack				X									
TM	Transmit Mark	X	X											0C
TS	Triple Space			X										
UL	Unload											X		
VF	Load Vertical Format						X							
VT	Vertical Tab								X					
1F	Set Signal 1 off				X	X								
1N	Set Signal 1 on				X	X								
2F	Set Signal 2 off				X	X								
2N	Set Signal 2 on				X	X								
6L	6 Lines/Inch			X	X									
8L	8 Lines/Inch			X	X									
8C	8 Characters/Inch				X									
10C	10 Characters/Inch			X	X									
12C	12 Characters/Inch			X	X									
EOF	End of File												X	
BOF	Beginning of File												X	
0E*	Expanded Print								X					0E
08*	Backspace				X	X								08
09*	Tab				X	X								09
0A*	Index 1/2 line up				X									0A
0B*	Line Feed Up				X	X								0B
0C*	Index 1/2 line down				X									0C
0D*	Carriage Return				X	X								0D
0E*	Line Feed Down				X									0E
0408*	Force Transmit	X												0408
09*	Restore Cursor	X	X											09
	@(X,Y) Set Cursor X<=63,Y<=14,26	X	X											(01) (Y+1)(X)

*Hex Coded Mnemonic "@XX@" where XX is value given above.

QANTEL CORPORATION
Business Computer Systems

3525 Breakwater Avenue Hayward, California 94545 (415) 783-3410

102687112