IBM Instruments Inc.

Computer System
Operating System Reference Manual

Part 2.
Assembler Programmer's Guide to Logical I/O and System Services

Release 1.1

GC22-9200-1

IBM Instruments Inc.

Computer System
Operating System Reference Manual

Part 2.
Assembler Programmer's Guide to Logical I/O and System Services

Release 1.1

GC22-9200-1

Second Edition (October 1983)

The contents of this edition are subject to change. Changes will be included in subsequent Technical Newsletters or editions of this publication.

Requests for copies of IBM Instruments, Inc., publications should be made to your IBM Instruments, Inc., representative or by calling, toll-free, 800-243-3122 (in Connecticut, call collect 265-5791).

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Instruments, Inc., Department 79K, P.O. Box 332, Danbury, CT 06810. IBM Instruments, Inc. may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

01107830

PREFACE

This manual describes the logical I/O facilities and programming interfaces of the IBM Instruments Computer System. It consists of 20 chapters and five appendixes.

- Chapter 1 -- "Introduction to Logical I/O and System Services" -- is divided into two parts; one discusses the general concepts of logical I/O briefly, and one goes into more detail about the commands used to perform logical I/O or gain system services.

- Chapter 2 -- "Keyboard Driver" -- describes how the keyboard can be controlled.

- Chapter 3 -- "Keypad Driver" -- describes how to control the computer system's keypad through the keypad driver.

- Chapter 4 -- "CRT Alphanumeric Display Driver" -- describes the information required by this driver for its control of the CRT screen.

- Chapter 5 -- "CRT Graphics Driver" -- describes the type of information that must be provided to this driver to perform various graphics functions of the CRT.

- Chapter 6 -- "Printer Driver" -- describes the various settings required by the driver to control print color, etc.

- Chapter 7 -- "File Access and Structure" -- describes methods for accessing files stored on diskette and on hard disk.

- Chapter 8 -- "Diskette Driver" -- describes the use of the diskette driver.

- Chapter 9 -- "Hard Disk Driver" -- describes the use of the hard disk driver.

- Chapter 10 -- "RS-232 Asynchronous Communications Driver" -- describes how to use the RS-232 driver.

- Chapter 11 -- "IEEE-488 Interface Driver" -- describes how to use the IEEE-488 driver.

- Chapter 12 -- "Parallel Port Driver" -- describes the use of that driver.

- Chapter 13 -- "Intertask Communication Channels Driver" -- describes the Read and Write channels used to exchange data between two concurrently running tasks.

Chapters 14 through 18 describe the drivers that are associated with the computer system's optional Sensor I/O board:

- Chapter 14 -- "A/D Converter Driver" -- describes how analog to digital conversion can be accomplished through the A/D driver.

- Chapter 15 -- "Switch Input Driver" -- describes how to use this driver to set up switch input.

- Chapter 16 -- "LED Output Driver" -- describes how the LED driver can accomplish LED output.

- Chapter 17 -- "Sensor Board Parallel Ports Driver" -- shows how the parallel ports driver associated with the Sensor I/O board works.

- Chapter 18 -- "Counter Driver" -- shows how to use the driver to manipulate the counter.

- Chapter 19 -- "Semaphore Manager" -- describes how tasks can be controlled with semaphores.

- Chapter 20 -- "System Calls" -- describes how system calls can be made.

- Appendix A -- "Error Messages and Codes" -- provides a list of the most commonly used error messages and codes.

- Appendix B -- "Command Summary" -- provides a brief summary of the I/O-related command presented in this manual.

- Appendix C -- "Sample Coding" -- shows some samples of how various drivers and I/O routines could be coded.

- Appendix D -- "Include Files on Extension Diskette" -- lists the equate statements and macros that can be used with the assembler.

- Appendix E -- "Disk Structure" -- describes the internal structure disks and diskettes.

Related Publications:

Publications that discuss related aspects of the Computer System are:

Computer System Product Description, GC22-9183

Computer System BASIC Reference Manual, GC22-9184

Computer System Pascal Reference Manual, GC22-9190

Computer System FORTRAN Reference Manual, GC22-9194

Computer System Operating System Reference Manual
Part 1: Operating System, GC22-9199

Computer System Problem Isolation Manual, GC22-9192

Additional References:

A good understanding of assembler language programming is assumed in much of this manual. There are many sources of information on the subject, whether formal classroom education or through use of books and other student material.

A knowledge of the facility of the 68000 microprocessor is also important, especially for understanding the instruction types and notation conventions which apply to this microprocessor. There are several books available on the 68000, two of which are:

- Motorola MC68000
  16-Bit Microprocessor User's Manual, 3rd edition
  Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982

- 68000 Assembler Language Programming
  by Gerry Kane, Doug Hawkins, and Lance Leventhal
  Osborne/McGraw-Hill, Berkeley, CA, 1981

CONTENTS

1.0 INTRODUCTION TO LOGICAL I/O AND SYSTEM SERVICES

The IBM Instruments Computer System 9000 Operating System (CSOS) contains two major facilities:

1. Logical I/O

   This part of CSOS handles input-output. An introduction to logical I/O follows, and Chapters 2-6, 8-12, 14-18 describe the various device drivers. Chapter 7 describes how to access files, and Chapter 13 describes two pseudo devices that allow intertask communication.

2. System Services

   This part of CSOS provides facilities for program loading, message printing, command parsing, task startup and exit, high-resolution timing, and semaphores. An introduction to system services can be found in this chapter (Section 1.2).

1.1 AN OVERVIEW OF LOGICAL I/O

One of the features of the IBM Instruments Computer System is the variety of peripheral devices that can be attached to it and the flexibility of its own I/O devices such as the CRT, the keypad, and the printer/plotter. The Computer System's operating system enhances this flexibility by providing a number of programming facilities that allow users to easily configure and access the various devices and interface ports directly, at run-time. The remainder of this section outlines these facilities.

1.1.1 DEVICE DRIVERS

The Computer System handles device control through device "drivers". Each device or communications port is controlled by at least one driver. In some cases, more than one driver may be associated with a device. The drivers provided by the Computer System are separated from the rest of the operating system so that new devices can be added without difficulty and without affecting system operation. Facilities also exist that allow users to write their own drivers and attach them to the system -- all without regenerating the operating system itself.

1.1.2 LOGICAL UNIT NUMBERS (LUN) AND LOGICAL DEVICE NAMES

The Computer System's I/O structure simplifies the transfer of data to and from devices by making direct references to specific physical devices unnecessary. Instead, devices are referred to generically by logical rather than physical labels. This has a number of advantages, the first of which is that users need not deal with physical locations at all -- the system will do that. All devices can be dealt with logically by logical device names. Another concept -- that of logical unit numbers, or "LUNs" -- allows the association between a logical unit number and a logical device name to be deferred until run-time when it can be made by command line parameters or by a user response to a program menu. In this way, a logical unit number specified in a user's program can refer to the CRT for one program run and the printer for another run without rewriting the program itself.

The Computer System associates or "binds" logical unit numbers (LUNs) with logical devices by means of a system OPEN command. (This and other system commands are discussed later in this chapter.) A single task running in the system may open up to 127 LUNs (if space permits) and any number of LUNs may be assigned to a single device. Once a device is OPENed, it can be referred to by its LUN only. Logical unit numbers are "local" variables; that is they are known only to the tasks that OPENed them. OPENing a device is further explained in Section 1.1.4, "The Structure of Logical I/O".

1.1.3 USING THE HIGHER LEVEL LANGUAGES

The Computer System's I/O facilities are accessible not only through Assembler language but also through the higher level languages like BASIC, FORTRAN, and Pascal. Essentially, this access consists of a higher level language "CALL" to the Computer System's I/O function. (See the BASIC Programming Manual, GC22-9184; FORTRAN Reference, GC22-9194, or Pascal Reference, GC22-9190 for further details.) Each of these languages has its own interface to the Computer System's I/O facilities, but while these interfaces may differ from one higher level language to another, the underlying structure of I/O management remains constant.

1.1.4 THE STRUCTURE OF LOGICAL I/O

This section briefly explains the ways in which a user can provide I/O-related information to the system at the Assembler language level and how the Computer System uses that information. (The higher level languages allow users to present information to the system without necessarily using all of the data structures described below.) Figure 1-1 shows the relationship between the user and system programming and you should refer to it throughout this discussion.

How Information Can Be Passed

Information can be passed to the Computer System by I/O or system service commands and through data structures called "control blocks". These commands and control blocks should be included as part of any user-written program that deals with input or output.

The I/O commands include the OPEN command briefly mentioned in section 1.1.2 and others used to establish or release associations between logical unit numbers and devices, or perform read/write operations. The OPEN command (SYSIO OPEN) includes a logical unit number, a label that identifies an error handling routine which is branched to if the operation fails. For example:

```
SYSIO OPEN,#10,GRDIB,OPENERR
```

The control blocks that can be used are as follows:

- DIB

  The Device Initialization Block; used to provide information about a device at Open time. This includes the device name and any specific options that a user wishes the device to have.

- DTCB

  The Data Transfer Control Block; used only for operations that involve the transfer of data. This control block tells the system where a user program is storing the data that is to be transferred.

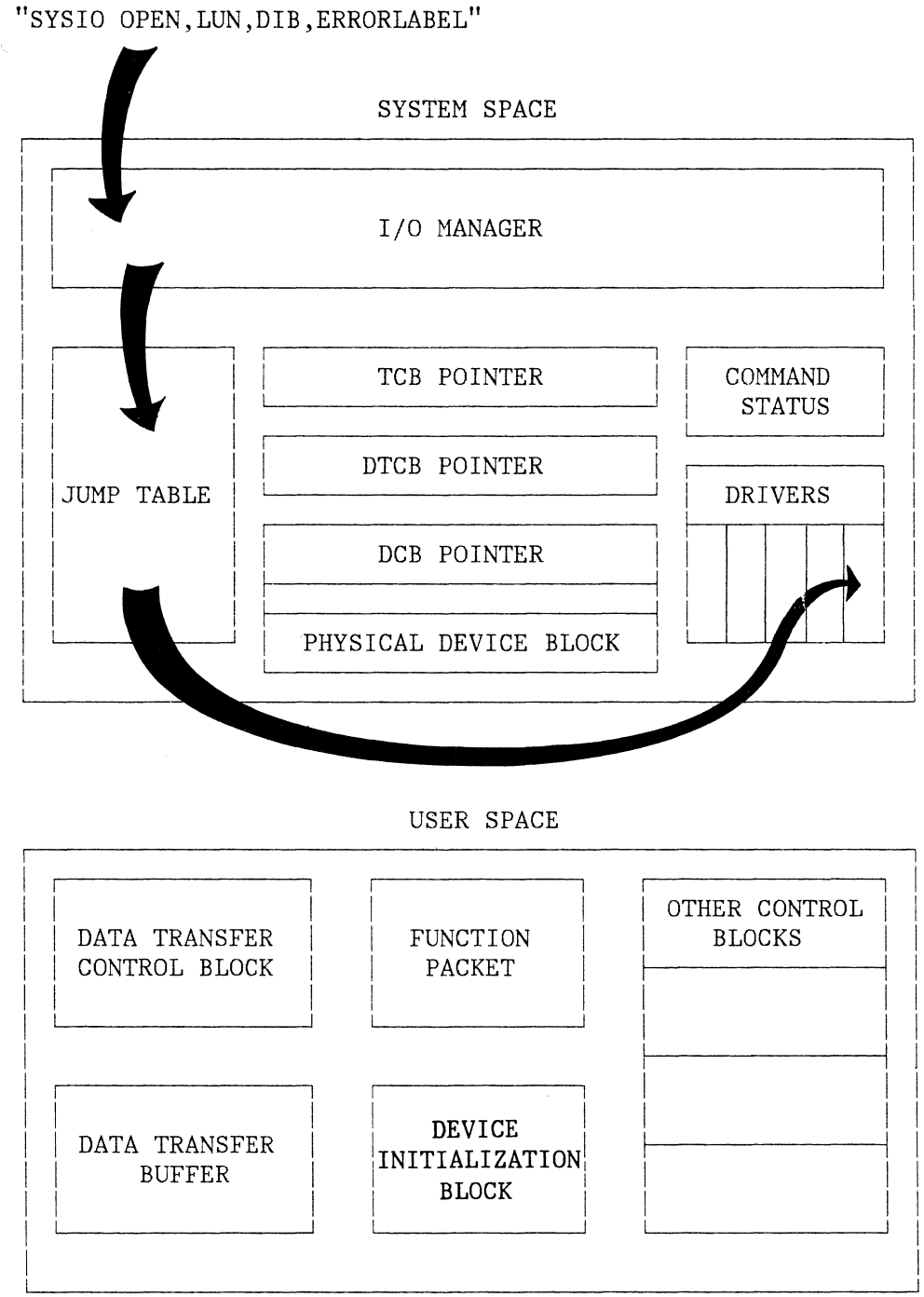


Figure 1-1. Data Structures in System and User Space

- **FPKT**

  The Function Packet; used with the FUNCTION command to change a device state to something other than the system default or the state set by the options specified in the DIB at initialization.

  The format of this control block is identical for all devices. It's use is driver specific, for example, it can be used for setting a baud rate or reading a cursor position. It is the control block that is most often used by the higher level languages.

1.1.4.1 Device Initialization Block (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. First it is checked for validity, then it is copied from user space into the appropriate control block in system space. The DIB must be aligned on a word boundary.

DIB FORMAT

The following fields describe the DIB format.

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Volume or device name. Use the appropriate names from section 1.3.5.3.
DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional.
DIBTRN	DS.B 1	Enter 0 for fixed length or 1 for variable length transfers.
DIBRS0	DC.L 0,0	User sets this field to 0.
DIBOPT	DS.W 1	Configuration Options. (Used for File I/O)
DIBFCN	DS.L 1	Insert pointer to function packet or set this field to $0000 0000 to select the default mode.

DIBBIO	DS.L 1	Byte I/O field. To open the driver for byte I/O enter -1 ($FFFF FFFF), otherwise set it to $0000 0000. After open the I/O manager fills this field with an identifier which is used for SYSIO-BREAD, SYSIO-BWRITE and SYSIO-BTEST.

DIB EXTENSION

DIBXXX	Only File I/O may use a DIB extension.

1.1.4.2 Data Transfer Control Block (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READ's and WRITE's. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made. The DTCB must be aligned on a word boundary. (See also Figure 1-2.)

DTCB Format

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable Length transfer here.
DTCTBL	DS.B 1	User puts lower limit to be used for Transfer Termination characters in Variable Length transfer here.
DTCRS0	DC.B 1	This field is reserved. User puts $00 here.
DTCBFS	DS.L 1	User puts Buffer starting address here.

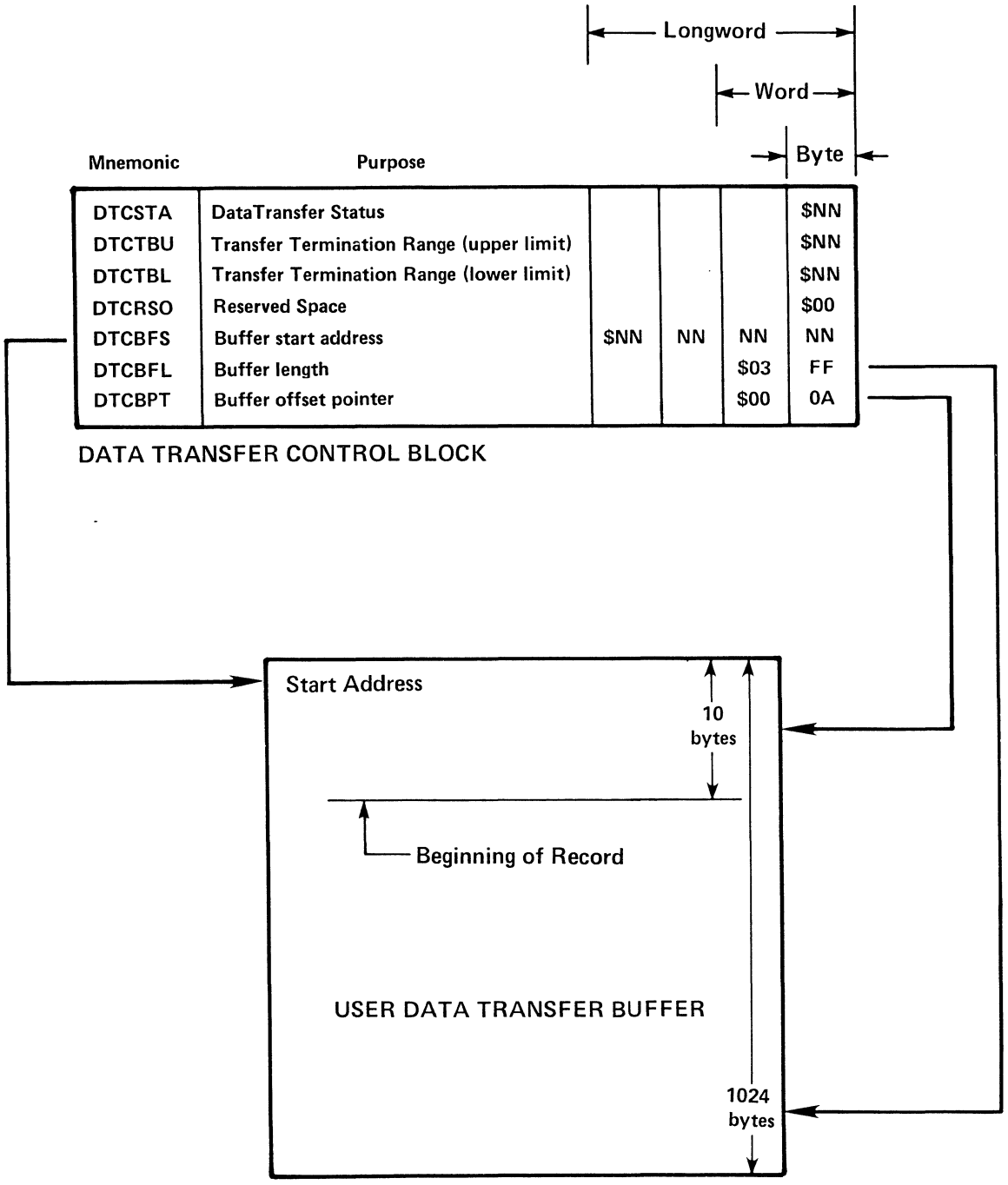


Figure 1-2. Data Transfer Buffer and Control Block

DTCBFL DS.W 1 User puts count of number of bytes in data buffer here.

DTCBPT DS.W 1 User puts byte offset into buffer (if any) to the first byte of the record to be transferred. This offset value will be incremented by the driver for every byte transmitted. It should be reset after every READ/WRITE (when the buffer is reused).

-----The next field is required for Disk Drivers or file access only----

DTCREC DS.L 1 User puts the Relative or Logical Record number of the particular record to be transferred here.

DIBBIO DS.L 1 System used Byte I/O field. To OPEN the RS-232 driver for Byte I/O enter -1($FFFF FFFF), otherwise set it to zero. After OPEN the I/O manager fills this field with an identifier which is used for SYSIO-BREAD, SYSIO-BWRITE, and SYSIO-BTEST.

1.1.4.3 Function Packet (FPKT)

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like reading the cursor position and selecting a font table. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The Function Packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero, indicating END-OF-LIST. The COMMAND word must be a positive number 1 or higher. The COMMAND word is followed by zero or more words, or longwords that send or receive the immediate DATA for the COMMAND, or a longword that points to the DATA for that COMMAND.

Each of the I/O commands and system service commands is described in greater detail later in this chapter. Chapters 3 through 18 explain what control blocks are required by each device driver and how to specify options in those control blocks to configure a device. Configuring a device or port consists of defining all of its pertinent characteristics for use by a specific program -- the color used to print output on the printer/plotter or the baud rate for a port, for example. Default characteristics exist in each case; these are listed in each driver's chapter.

How Information is Used

When an OPEN command is encountered in a user's program, the Computer System's "I/O Manager" is given control and several things happen:

- The system "binds" the logical unit number provided as part of the command, to the logical device name identified in the DIB specified with the command. (It is the I/O Manager that carries this one step further to identify the actual physical device.) In our example above, the logical unit number #10 would be associated with the logical device name specified in the DIB called GRDIB.

- The I/O Manager determines which device driver is required and uses its "jump table" of addresses to locate the appropriate portion of that driver for an OPEN operation. (If another type of I/O command were entered, the I/O Manager would "jump" to another location within the driver.)

- The device is configured to either its default characteristics or to characteristics specified in the optional function packet specified in DIBFCN.

After a logical device has been OPENed, the options specified for it can still be changed. Another SYSIO command, SYSIO FUNCTION, allows users to alter a device's characteristics after they have been set through the OPEN.

1.2 AN OVERVIEW OF SYSTEM SERVICES

Understandably, when many tasks with various priorities are requesting the use of many I/O devices, things can get complicated. The Computer System provides a number of system services that can help manage the situation. The I/O Manager's role in controlling task and device management has already been briefly mentioned. The other services provided by the system are:

- System Calls

- Timer Services

- Asynchronous I/O

- Event Posting

- Semaphores, and

- The SUSPEND Command

Each of these will be briefly discussed below. Detailed information about how these services function is found later in this chapter.

1.2.1 SYSTEM CALLS

The System Calls are very helpful to users programming in assembler. They can be used in the areas of parsing, error messages, directory searching, loading programs, task manipulation, and timing operations. For details, see Chapter 20, "System Calls".

1.2.2 TIMER SERVICES

The Computer System is equipped with a timer that can be used to control the execution of a task. For example, one task can be stopped for a specified period of time to provide an opportunity for another task to run-- perhaps using the same device(s) as the first task. The first task resumes running when the time period has elapsed.

Timers must be OPENed and CLOSEd in the same way as devices. System commands exist that allow users to start the timer, make inquiries about its status, or stop it as necessary. They are described in Section 1.3.4.

1.2.3 ASYNCHRONOUS I/O

Normally, the operations that take place in a program are synchronous. That is, if the program calls for data transfer (a request for data from diskette storage, for example) execution of further commands in that program cannot continue until the I/O operation has completed.

Asynchronous operations like AREAD and AWRITE (discussed later in this chapter) work differently, however. When a program initiates an asynchronous I/O operation, an "interrupt" structure is used to signal the completion of the data transfer. This allows the program to continue executing. The program receives word that the I/O operation has started successfully and can continue to do other things while the I/O operation takes place. The program can check at various points during its execution to see whether the data transfer has completed or it can be "interrupted"

by the news that the transfer operation was completed by use of the SUSPEND facility. See below.

1.2.4 EVENT POSTING

The Computer System's Event Posting facility is similar to the status checking capabilities provided by asynchronous I/O except that other non-I/O events can be checked. Event posting allows users to set aside a control block in their user space that is associated with the occurrence of a specific event. This control block is "kept posted" by the operating system and user programs can check-- or "poll" it to determine the status of the event. This status could be that the event occurred, occurred more than once, or did not occur at all. The event control block can also be used to pass information from a driver to a user program.

In addition to polling, a user can suspend the execution of a task until a particular event has taken place.

1.2.5 SEMAPHORES

The computer system's semaphore facility provides a means of synchronization in a multitasking environment. The semaphore is typically used as a BATON which is passed from task to task to indicate the owner of a non-shareable resource. The "BATON-PASS" requires only a binary semaphore. More complicated strategies are possible if one takes advantage of the counting semaphore facility implemented on the CS 9000. Uses of the counting semaphore include:

resource management (non-shareable)
Task prioritization control
queue management

Chapter 20 describes the counting semaphore implementation on the CS 9000.

1.2.6 THE SUSPEND COMMAND

The SUSPEND command is used in conjunction with the other system services outlined above. User application programs can use the SUSPEND command to wait for the completion of timer, asynchronous I/O, event posting, or

semaphore requests. The program can be made to wait for the completion of one or many requests. The way in which this command works is more fully described later in this chapter (see Section 1.3.3, "More Information on the SUSPEND Command").

1.3 DETAILED INFORMATION ON LOGICAL I/O

This section provides more detailed information about the commands and facilities outlined earlier in this chapter. It also describes some additional aspects of the operating system that are not driver- or device-specific, such as byte I/O and the naming conventions used by the Computer System.

This section presumes a moderate level of understanding of Assembler language and a familiarity with Computer System Operating System Reference: Part 1, GC22-9199. Computer system operating system commands are implemented as macros available to Assembly language programs. High level languages implement operating system facilities through use of these macros.

1.3.1 WHEN TO USE SYNCHRONOUS, ASYNCHRONOUS, AND BYTE I/O

Synchronous Commands: OPEN, CLOSE, SREAD, SWRITE, FUNCTION

These forms of SYSIO might be called the "normal" type of I/O facility in that your task is blocked from further execution until the desired operation has completed. For general purpose application programming it is perfectly satisfactory to have your program wait for the outcome of each I/O before continuing along.

The operating system uses any available CPU time during synchronous I/O for other READY tasks. If there are no READY tasks, then the IDLE task consumes unused CPU time.

Asynchronous Commands: AREAD, AWRITE, CANCEL

These forms of SYSIO allow your program to perform buffered I/O operations and to take advantage of any available CPU time that occurs while the I/O device is waiting. You would want to use this method in situations where

you want to provide multiple buffers or where you wish to overlap the operation of more than one I/O device from a single task.

It would be possible to construct programs which perform several AREAD's, and which perform a different action depending on which one completes. A program can intermix AREAD and AWRITE with Timer and Semaphore calls to have all kinds of event driven control. You would use CANCEL to abort any outstanding I/O requests that you no longer wanted to occur.

It would also be possible to chain together several requests that always proceed one after the other. For example, an editor program might always perform 24 writes to disk from a screen buffer. This could be performed by 24 AWRITES, allowing the editor to continue while the 24 transfers take place.

Please remember that each outstanding AREAD or AWRITE requires a separate DTCB.

Byte I/O Commands: BREAD, BWRITE, BTEST

These forms of SYSIO allow your program to perform I/O operations at the lowest possible physical level. You would use this facility when you want to exercise real-time byte-by-byte control of I/O without device driver buffering or pre-processing. By handling each byte individually you are able to practically make up your own "device driver" within your application program.

This facility can be useful for capturing keystrokes from the optional keyboard device, or for sending ASCII characters directly to the next character position of an open alpha window.

Remember that in byte I/O there is no waiting performed by the driver. Your program always gets immediate control back after SYSIO. It is up to you to check D7.W for error status on each request. Zero indicates successful transfer of the byte, non-zero usually means that the driver cannot send or receive a byte right now, but that you should try again at some later time.

1.3.2 MORE INFORMATION ON SYSIO COMMANDS

Application programs can use SYSIO to perform all logical I/O operations. SYSIO allows you to issue one of twelve commands as shown on the following pages. There are two ways of transferring data with SYSIO. One is a

block-oriented path that can be used for record I/O, OPEN, CLOSE, and FUNCTION. The second is a "fast path" available for byte operations such as SYSIO-BREAD, SYSIO-BWRITE, SYSIO-BTEST. Each is described below.

1.3.2.1 Block-Oriented I/O Commands

The commands have a common syntax. First specify the command itself, such as SYSIO OPEN. Next provide the logical unit number; then the name of a control block, and finally the address of a user exception-handling routine (ERRORLABEL, in the examples which follow.)

The logical unit number may be specified either in immediate notation using a # sign, such as #1, as the label of a byte containing the value, such as LOGUNIT or in a register such as D3. If register D5 already contains the logical unit number, simply code D5 and no load instruction will be generated by the macro. The same scheme is used to specify the control block. It can be an address label at the start of the block, such as GRDIB, or in an address register.

The SYSIO macro generates in-line code to carry out the command. It loads register A6 with the address of the control block. Then it loads register D5.B with the logical unit number. Next, it issues a TRAP #6 instruction followed by the command word. This is followed by a branch to the user's exception handling routine for error handling. If the command is unsuccessful, this branch is taken and register D7.W will contain an error code. If the call is successful, the branch is not taken and control will continue with the next sequential instruction following the macro.

The AREAD and AWRITE commands are asynchronous operations while all other commands are synchronous. Synchronous operations execute in the invoking program's instruction stream. This means that your program will be blocked from continued execution until the operation is complete. When you get control back from SYSIO, register D7.W will contain the completion status. Asynchronous operations are initiated in the invoking program's instruction stream with the interrupt used to signal completion of data transfer. The invoking program gets control back after queuing the request, with a -1 in register D7.W to indicate that the operation has started successfully. Any other word in register D7.W means that an exception has occurred and that the operation has been aborted.

With asynchronous I/O it is possible for the invoking program to perform other things while the data transfer takes place. When the program wants to see if the transfer is complete it can do either of two things: It can keep checking the contents of the DTCSTA field of the Data Transfer Control Block (-1 indicates still busy; zero indicates successful

completion, any other positive status is an exception completion code). Or the program can use the SUSPEND macro to wait for completion.

Asynchronous I/O is particularly useful for multi-buffering, since each I/O request is queued with a separate DTCB and a companion buffer.

SYSIO OPEN,LUN,DIB,ERRORLABEL

This command performs an OPEN operation for the logical unit number specified, "binding" it to the logical device name specified in the DIB.

Register Usage:

A6.L points to the DIB
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8400 Inadequate system space
$8500 Duplicate logical unit number
$8600 Device not found
$8700 Not device owner
$8800 Nonsharable device is already open
$8900 Device does not support byte I/O
$8A00 Non-null request queue for byte I/O
$8C00 Invalid DIB field (DIBTRN, DIBDTD, DIBRSO)

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

DIB (on a word boundary)

Optional Control Block:

FPKT (on a word boundary) specified within the DIB

Coding Example:

```
470
471 0 00000000                    SYSIO   OPEN,#21,DIBBLK,ERROR
    0 00000000 4DF900000200       LEA     DIBBLK,A6           + LOAD ADDRESS OF CONTROL BLOCK
    0 00000006 006E00400010       ORI.W   #$0040,DIBOPT(A6)   + 1.1 OR LATER SYSIO OPEN
    0 0000000C 1A3C0015           MOVE.B  #21,D5              + LOAD LOGICAL UNIT NUMBER
    0 00000010 4E46               TRAP    #6                  + TRAP TO I/O MANAGER
    0 00000012 0001               DC.W    SIOOPEN             + COMMAND WORD
    0 00000014 600001F2           BRA.L   ERROR               + BRANCH TO USER ERROR HANDLER
```

SYSIO CLOSE,LUN,0,ERRORLABEL
- or -
SYSIO CLOSE, LUN, FPKT, ERRORLABEL

This command performs a CLOSE operation on the device specified by the logical unit number. Note that no control block need be specified, however 0 is required by the macro as a placeholder.

Register Usage:

A6.L points to the Function Packet
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Optional Control Blocks:

Function Packet(on a word boundary)

Comments:

You may optionally specify that a Function Packet be part of CLOSE for files. Other device drivers do not accept a function packet at close time.

Coding Example:

```
478
479 0 00000018                    SYSIO   CLOSE,#21,0,ERROR
    0 00000018 4DF80000           LEA     0,A6            + LOAD ADDRESS OF FPKT
    0 0000001C 1A3C0015           MOVE.B  #21,D5          + LOAD LOGICAL UNIT NUMBER
    0 00000020 4E46               TRAP    #6              + TRAP TO I/O MANAGER
    0 00000022 0002               DC.W    SIOCLOSE        + COMMAND WORD
    0 00000024 600001E2           BRA.L   ERROR           + BRANCH TO USER ERROR HANDLER
```

SYSIO SREAD,LUN,DTCB,ERRORLABEL

This command performs a synchronous read operation from the device associated with the logical unit number.

Register Usage:

A6.L points to the DTCB
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner
$8F00 Invalid buffer address for read

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

DTCB (on a word boundary)

Coding Example:

```
486
487 0 00000028                    SYSIO   SREAD,#21,DTCBLK,ERROR
    0 00000028 4DF900000210       LEA     DTCBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 0000002E 1A3C0015           MOVE.B  #21,D5             + LOAD LOGICAL UNIT NUMBER
    0 00000032 4E46               TRAP    #6                 + TRAP TO I/O MANAGER
    0 00000034 0003               DC.W    SIOSREAD           + COMMAND WORD
    0 00000036 600001D0           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

SYSIO SWRITE,LUN,DTCB,ERRORLABEL

This command performs a SYNCHRONOUS WRITE operation to the device associated with the logical unit number.

Register Usage:

A6.L points to the DTCB
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

DTCB (on a word boundary)

Coding Example:

```
494
495 0 0000003A                    SYSIO   SWRITE,#21,DTCBLK,ERROR
    0 0000003A 4DF900000210       LEA     DTCBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000040 1A3C0015           MOVE.B  #21,D5             + LOAD LOGICAL UNIT NUMBER
    0 00000044 4E46               TRAP    #6                 + TRAP TO I/O MANAGER
    0 00000046 0004               DC.W    SIOSWRITE          + COMMAND WORD
    0 00000048 600001BE           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

SYSIO AREAD,LUN,DTCB,ERRORLABEL

This command performs an ASYNCHRONOUS READ operation from the device associated with the logical unit number.

Register Usage:

A6.L points to the DTCB
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner
$8F00 Invalid buffer address for read

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

DTCB (on a word boundary).

Each outstanding AREAD must have its own DTCB.

Coding Example:

```
502
503 0 0000004C                    SYSIO    AREAD,#21,DTCBLK,ERROR
    0 0000004C 4DF900000210       LEA      DTCBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000052 1A3C0015           MOVE.B   #21,D5             + LOAD LOGICAL UNIT NUMBER
    0 00000056 4E46               TRAP     #6                 + TRAP TO I/O MANAGER
    0 00000058 8003               DC.W     SIOAREAD           + COMMAND WORD
    0 0000005A 600001AC           BRA.L    ERROR              + BRANCH TO USER ERROR HANDLER
```

SYSIO AWRITE,LUN,DTCB,ERRORLABEL

This command performs an ASYNCHRONOUS WRITE operation to the device associated with the logical unit number.

Register Usage:

A6.L points to the DTCB
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

DTCB (on a word boundary).

Each outstanding AWRITE must have its own DTCB.

Coding Example:

```
509
510 0 0000005E                    SYSIO   AWRITE,#21,DTCBLK,ERROR
    0 0000005E 4DF900000210       LEA     DTCBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000064 1A3C0015           MOVE.B  #21,D5             + LOAD LOGICAL UNIT NUMBER
    0 00000068 4E46               TRAP    #6                 + TRAP TO I/O MANAGER
    0 0000006A 8004               DC.W    SIOAWRITE          + COMMAND WORD
    0 0000006C 6000019A           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

SYSIO CANCEL,LUN,0,ERRORLABEL

This command performs a CANCEL on the device associated with the logical unit number. Normally you would use CANCEL to abort one or more outstanding asynchronous I/O operations which had not yet completed. You might be waiting for keyboard or RS-232 input using AREAD and then decide to cancel the input request. 0 is required by the macro as a placeholder.

Register Usage:

A6.L contains 0
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8600 Device not found
$8700 Not device owner

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

None.

Coding Example:

```
517
518 0 00000070                  SYSIO   CANCEL,#21,0,ERROR
    0 00000070 4DF80000         LEA     0,A6            + LOAD ADDRESS OF CONTROL BLOCK
    0 00000074 1A3C0015         MOVE.B  #21,D5          + LOAD LOGICAL UNIT NUMBER
    0 00000078 4E46             TRAP    #6              + TRAP TO I/O MANAGER
    0 0000007A 0007             DC.W    SIOCANCEL       + COMMAND WORD
    0 0000007C 6000018A         BRA.L   ERROR           + BRANCH TO USER ERROR HANDLER
```

SYSIO FUNCTION,LUN,FPKT,ERRORLABEL

This command performs a FUNCTION on the device associated with the logical unit number. The specified function packet (FPKT) contains one or more device specific functions.

Register Usage:

A6.L points to the Function Packet
D5.B contains the LUN
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8400 Inadequate system space
$8600 Device not found
$8700 Not device owner

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Required Control Blocks:

FPKT (on a word boundary)

Coding Example:

```
525
526 0 00000080                  SYSIO    FUNCTION,#21,FPKTBLK,ERROR
    0 00000080 4DF900000214     LEA      FPKTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000086 1A3C0015         MOVE.B   #21,D5              + LOAD LOGICAL UNIT NUMBER
    0 0000008A 4E46             TRAP     #6                  + TRAP TO I/O MANAGER
    0 0000008C 0006             DC.W     SIOFUNCTION         + COMMAND WORD
    0 0000008E 60000178         BRA.L    ERROR               + BRANCH TO USER ERROR HANDLER
```

1.3.2.2 Byte I/O Facilities

The Computer System has provided an efficient mechanism for transferring data on a byte basis. Upon opening a device for byte I/O and a specific direction, subsequent transfers may be initiated using SYSIO-BREAD, SYSIO-BWRITE or SYSIO-BTEST trap calls. These calls require the following register usage: D0.B contains the byte on write (or D0.W for the CRT driver -- See Chapter 4) and is returned with a byte on test or reads. Register D7.W is returned with a status code. No scheduling is done for these requests and if the device is busy with another request, the code returned in D7.W will indicates that condition. This facility is best used for application programs using non-shareable devices (the first task to open a non-shareable device owns the device until it is closed) or devices supported by reentrant drivers such as the CRT. In addition, register A6.L must contain the system identifier returned in the DIBBIO field of the device initialization block. If a device will not support byte I/O, an I/O Manager error code will be returned in D7.W at open time.

Note that single-byte transfers (either asynchronous or synchronous) may be initiated using the other I/O facilities, but these transfers must be scheduled and may therefore require some system overhead.

1.3.2.3 Byte I/O Commands

The Byte I/O mechanism is meant to be a "fast path" between application programs and device drivers. There is no scheduling and no use of dynamic control blocks. On the average, less than 25 instructions are executed by the I/O manager between the SYSIO call and entry to the driver. This section describes the SYSIO commands used to perform byte I/O operations.

Unlike the block-oriented requests, byte I/O requests do not have an error return label. It is up to the application program to monitor D7.W for completion status. Register D0.B is used to pass or receive the byte. Register A6 is loaded with the system identifier stored in the DIBBIO field after successful completion of the OPEN command. Register D7.W contains the completion status (see ERROR CODES of appropriate driver and Appendix A).

To OPEN a device for BYTE I/O you must place a -1 in the DIBBIO field and a specific direction (either 0 for write or 1 for read) in the DIBDTD field. After OPEN, the DIBBIO field will contain an identifier to be used for SYSIO-BREAD, SYSIO-BWRITE, and SYSIO-BTEST.

BTEST operates as a BREAD, but does not increment the buffer pointer. It will return a $0C error code in D7.W if the buffer is empty otherwise the byte value is returned in D0.B.

The SYSIO macro will load the system identifier, obtained from DIBBIO, into register A6.L. If the register already contains the proper information place A6 in the BYTEID field of the call. This eliminates redundant register loading.

SYSIO BREAD,BYTEID -or- SYSIO BREAD,A6

This command reads a byte from the device associated with the system identifier BYTEID.

Register Usage:

A6.L points to the driver Byte Read Entry point
D0.B receives the byte
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8A00 Non-null request queue for byte I/O
$8B00 Not open for byte I/O

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Coding Example:

```
533
534 0 00000092                    SYSIO    BREAD,BYTEID
    0 00000092 2C7900000218       MOVEA.L  BYTEID,A6          + LOAD SYSTEM IDENTIFIER
    0 00000098 4E46               TRAP     #6                 + TRAP TO I/O MANAGER
    0 0000009A 0008               DC.W     SIOBREAD           + COMMAND WORD
```

Remember that there is no error branch on Byte I/O. Your program must always check the contents of D7.W to determine whether the operation was successful or whether the byte was not transmitting for some reason.

SYSIO BWRITE,ID -or- SYSIO BWRITE,A6

This command writes a byte to the device associated with the system identifier BYTEID.

Register Usage:

A6.L points to the driver Byte Write Entry Point
D0.B or D0.W (CRT Driver) contains the byte to be written
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8A00 Non-null request queue for byte I/O
$8B00 Not open for byte I/O

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Coding Example:

```
540
541 0 0000009C                    SYSIO    BWRITE,BYTEID
    0 0000009C 2C7900000218       MOVEA.L  BYTEID,A6          + LOAD SYSTEM IDENTIFIER
    0 000000A2 4E46               TRAP     #6                 + TRAP TO I/O MANAGER
    0 000000A4 0009               DC.W     SIOBWRITE          + COMMAND WORD
```

SYSIO BTEST,IDFIELD -or- SYSIO BTEST,A6

This command reads a byte from the device associated with the system identifier BYTEID without incrementing the internal buffer pointer.

Register Usage:

A6.L points to driver Byte Test Entry Point
D0.B passes the byte
D7.W returns status

Completion Codes:

$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8A00 Non-null request queue for byte I/O
$8B00 Not open for byte I/O

Note: This is only a partial listing of exception codes which may be returned when this macro is used. Other codes will be specific to the device driver named in the DIB. The chapter describing that particular device driver should be consulted for additional error codes.

Coding Example:

```
547
548 0 000000A6                  SYSIO    BTEST,BYTEID
    0 000000A6 2C7900000218     MOVEA.L  BYTEID,A6          + LOAD SYSTEM IDENTIFIER
    0 000000AC 4E46             TRAP     #6                 + TRAP TO I/O MANAGER
    0 000000AE 000A             DC.W     SIOBTEST           + COMMAND WORD
```

1.3.2.4 Event Posting With EVENTMGR Commands

Asynchronous event posting allows users to detect the occurrence of specific device driver events. By associating an Event Block (EVB) in the user's data area with a specific event known to the operating system the user can detect when that event occurs. Currently the events available to the user are:

1. IEEE-488 Bus Service request
2. RS-232 Break Key
3. Keyboard. CTRL-BREAK, CTRL-NUMLK, Functions Keys and State Change
4. RS-232 Character Input

For more information on these events refer to the section for those specific device drivers. The EVENTMGR MACRO generates in-line code to carry out the command. It loads register A6 with the address of the current control block. A '0' is required by the Macro as a place holder. It is loaded into D5 but not used. Register D7.W returns with a completion code.

1.3.2.5 Opening Events

The association of event with EVB is done by the user issuing an EVENTMGR OPNEVBLK command. The format of the EVB is as follows:

```
EVBSEM   DS.B 1 (S)    Event indicator
EVBRSO   DS.B 1 (S)    Reserved.  Set this byte to $00.
EVBSTA   DS.B 4 (S)    User sets these bytes to $00.
EVBNAM   DS.B 4 (U)    Event mnemonic

(S) means set by system.
(U) means set by user.
```

The EVBSTA field is used to pass optional information to the user.

1.3.2.6 Closing Events

Once a user is finished processing events, he should issue an EVENTMGR CLSEVBLK to remove any further association of the EVB with the event.

1.3.2.7 Arming Events

If a user would prefer to wait for an event rather than poll for it, he can 'arm' the event. This command followed by the SUSPEND macro will suspend the user until the event occurs. The user must rearm the event each time it occurs in order to be resumed when the event occurs again.
Once the event has been opened the user can poll the EVB to determine if the event has occurred, or he can arm the event and suspend himself until the event occurs. These two methods are described in the following sections.

1.3.2.8 Polling Events

Polling an EVB consists of a Test-and-Set (TAS) instruction on the EVBSEM field. TAS sets one of the following three condition codes in the 68000 Status Register:

MI (minus) - No event has occurred
EQ (equal) - One event has occurred since the last TAS
Pl (plus) - More than one event has occurred since the last TAS

When the condition code is PL the seven low-order bits contain the number of times the event has occurred since the last TAS.

Coding Example:

```
            INCLUDE   I0MCLB50.INC       event block offset equator
POLLEVENT   EQU       *
            LEA       EVBLK,A0           point A0 to event block
            TAS       EVSEM(A0)          test indicator byte
            BMI       EXIT               nothing happened
            BEQ       EVTOCCUR           process event's occurrence
            BPL       OVERRUN            more than one happened
EXIT        RTS                          return
OVERRUN     MOVE.B    EVBSEM(A0),D0      pick up event count
            AND.B     #$7F,D0            turn off upper bit
            MOVE.B    D0,OVERCNT         save overrun count
            BSR       EVTOCCUR           event handler subroutine
            BRA       EXIT               return
EVTOCCUR    EQU       *                  process event here
            :
            :
            RTS
EVBLK       DS.B      10                 10 bytes of storage for the EVB
OVERCNT     DS.B      1                  overrun count storage
```

EVENTMGR OPNEVBLK,#0,EVENTBLK,ERRORLABEL

Register Usage:

A6.L points to the Event Block (EVB)
D5.B contains the LUN
D7.W returns status

Completion Codes:

$0000 Success
$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8400 Inadequate system space
$8600 Device not found
$8700 Not device owner
$8D00 Event Descriptor Block not found
$8E00 Event not opened

Data Structures:

Event Block (EVB)

Coding Example:

```
555
556 0 000000B0                    EVENTMGR OPNEVBKL,#0,EVENTBLK,ERROR
    0 000000B0                    SYSIO    OPNEVBKL,#0,EVENTBLK,ERROR + INTERNALLY GENERATED COMMAND
    0 000000B0 4DF900000021C      LEA      EVENTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 000000B6 1A3C0000           MOVE.B   #0,D5                + LOAD LOGICAL UNIT NUMBER
    0 000000BA 4E46               TRAP     #6                   + TRAP TO I/O MANAGER
    0 000000BC 000B               DC.W     SIOOPNEVBKL          + COMMAND WORD
    0 000000BE 60000148           BRA.L    ERROR                + BRANCH TO USER ERROR HANDLER
```

EVENTMGR CLSEVBLK,0,EVENTBLK,ERRORLABEL

Register Usage:

A6.L points to the Event Block (EVB)
D5.B contains the LUN
D7.W returns status

Completion Codes:

$0000 - success
$8E00 - event not opened

Data Structures:

EVB

Coding Example:

```
563
564 0 000000C2                    EVENTMGR CLSEVBLK,#0,EVENTBLK,ERROR
    0 000000C2                    SYSIO    CLSEVBLK,#0,EVENTBLK,ERROR + INTERNALLY GENERATED COMMAND
    0 000000C2 4DF90000021C       LEA      EVENTBLK,A6           + LOAD ADDRESS OF CONTROL BLOCK
    0 000000C8 1A3C0000           MOVE.B   #0,D5                 + LOAD LOGICAL UNIT NUMBER
    0 000000CC 4E46               TRAP     #6                    + TRAP TO I/O MANAGER
    0 000000CE 000C               DC.W     SIOCLSEVBLK           + COMMAND WORD
    0 000000D0 60000136           BRA.L    ERROR                 + BRANCH TO USER ERROR HANDLER
```

EVENTMGR ARMEVENT,#0,EVENTBLK,ERRORLABEL

Register Usage:

A6.L points to the event block
D5.B contains the LUN
D7.W returns status

Completion Codes:

$0000 Success
$8200 Invalid SYSIO call
$8300 Logical unit number not opened
$8D00 Event Descriptor Block not found
$8E00 Event not opened

Data Structures:

Event Block (EVB)

Coding Example:

```
570
571 0 000000D4                  EVENTMGR ARMEVENT,#0,EVENTBLK,ERROR
    0 000000D4                  SYSIO    ARMEVENT,#0,EVENTBLK,ERROR + INTERNALLY GENERATED COMMAND
    0 000000D4 4DF90000021C     LEA      EVENTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 000000DA 1A3C0000         MOVE.B   #0,D5                + LOAD LOGICAL UNIT NUMBER
    0 000000DE 4E46             TRAP     #6                   + TRAP TO I/O MANAGER
    0 000000E0 000D             DC.W     SIOARMEVENT          + COMMAND WORD
    0 000000E2 60000124         BRA.L    ERROR                + BRANCH TO USER ERROR HANDLER
```

1.3.3 MORE INFORMATION ON THE SUSPEND COMMAND

User programs can use the SUSPEND command to wait for the completion of outstanding asynchronous requests. The program can be made to wait for the completion of one request or all requests.

The functions of the SUSPEND command include:

SYNCH	Retrieve the number of outstanding I/O requests.
SUSPEND TILLANY	Wait until any outstanding request is completed.
SUSPEND TILLALL	Wait until all outstanding requests are complete.

The SUSPEND function will not wait for a specific request but only for a change in the count of outstanding requests that is kept for each task. User programs that are interested in the completion of one specific request out of several outstanding requests must check the status field of the Data Transfer Control Block, timer block, semaphore block, or event block in question.

If the program uses decision logic based on the status fields, there is a possibility that the status will change between the time that the program tests the field and the time that a SUSPEND is issued. To allow synchronization, there is a SUSPEND-SYNCH command to capture the number of outstanding requests before any decision logic takes place. This number will subsequently be used as an operand of a SUSPEND-TILLANY command so the I/O manager can work with the same set of initial conditions.

The macro generates in-line code to carry out the command. It issues a TRAP #9 to the I/O manager and either passes or receives a value contained in register D0.

SUSPEND TILLALL,,ERRORLABEL

The current task is suspended until all outstanding requests are complete.

Register Usage:

D0.B returns the number of outstanding I/O requests
D7.W returns status

Completion Codes:

On error branches D7.W will be loaded with $8200 (invalid call); otherwise D7.W is set to zero.

Required Data Structures:

None

Coding Example:

```
578
579 0 000000E6                  SUSPEND  TILLALL,,ERROR
    0 000000E6 4E49             TRAP     #9                  + TRAP TO I/O MANAGER
    0 000000E8 0002             DC.W     $0002               + TILLALL COMMAND WORD
    0 000000EA 60000124         BRA.L    ERROR               + BRANCH TO USER ERROR HANDLER
```

SUSPEND TILLANY,D0,ERRORLABEL

The current task is suspended until any outstanding request is completed. The command is non-specific as to which request will satisfy the wait. Register D0.B must contain the number of requests that were outstanding before it was decided to perform the SUSPEND. This value is obtained via SUSPEND-SYNCH. The I/O manager will resume the task as soon as the number of outstanding requests is less than the number passed in register D0.B.

Register Usage:

D0.B must contain the number of outstanding requests
D7.W returns status

Completion Codes:

On error branch. D7.W=$8200 Invalid call
D7.W=$8300 Invalid Synch value.

Required Control Blocks:

None

Coding Example:

```
586
587 0 000000EE                    SUSPEND  TILLANY,D0,ERROR
    0 000000EE 00000000           DS.W     0                 + D0 ASSUMED TO CONTAIN SYNCH COUNT
    0 000000EE 4E49               TRAP     #9                + TRAP TO I/O MANAGER
    0 000000F0 0003               DC.W     $0003             + TILLANY COMMAND WORD
    0 000000F2 60000114           BRA.L    ERROR             + BRANCH TO USER ERROR HANDLER
```

SUSPEND TILLANY,NOSYNCH,ERRORLABEL

The current task is suspended until any outstanding asynchronous request is completed. The operand NOSYNCH instructs the system to use the current request count as a starting number. The command is non-specific as to which request (I/O, timer, event) will satisfy the wait. No wait will take place if the number of outstanding requests is zero.

Register Usage:

D0.B
D7.W returns status

Completion Codes:

On error branches D7.W will be loaded with $8200 (invalid call)

Required Control Blocks:

None

Coding Example:

```
594
595 0 000000F6                    SUSPEND  TILLANY,NOSYNCH,ERROR
    0 000000F6 103CFFFF           MOVE.B   #-1,D0              + INDICATE TILLANY-NOSYNCH CALL
    0 000000FA 4E49               TRAP     #9                  + TRAP TO I/O MANAGER
    0 000000FC 0003               DC.W     $0003               + TILLANY COMMAND WORD
    0 000000FE 60000108           ERA.L    ERROR               + BRANCH TO USER ERROR HANDLER
```

SUSPEND SYNCH,DO,ERRORLABEL

The number of outstanding requests for the current task is returned in register D0.B.

Register Usage:

D0.B returns the number of outstanding requests
D7.W returns status

Completion Codes:

On error branches D7.W is loaded with $8200 (invalid call); otherwise D7.W contains $0000.

Required Control Blocks:

None

Comments:

As an example of the SUSPEND SYNCH command, let us assume that there are three outstanding requests which use the Data Transfer Control Blocks: AAADTCB, BBBDTCB, and CCCDTCB. A user program wishes to wait until AAADTCB is complete; the other requests do not matter. Furthermore, the fields AAASTA, BBBSTA, and CCCSTA are all set to -1 indicating that the request is not yet complete. The program statements might look as follows:

```
WAITAAA  SUSPEND  SYNCH,D0,ERR       Get current number of requests.
         IF.B AAASTA <EQ>#-1 THEN   If the one you want is not done
           SUSPEND TILLANY,D0,ERR1  Then wait for any request to
           BRA WAITAAA              complete and see if it is the one
         ENDI                       you want.
```

Coding Example:

```
602
603 0 00000102                     SUSPEND  SYNCH,D0,ERROR
    0 00000102 4E49                TRAP     #9                  + TRAP TO I/O MANAGER
    0 00000104 0001                DC.W     $0001               + SYNCH COMMAND WORD
    0 00000106 60000100            BRA.L    ERROR               + BRANCH TO USER ERROR HANDLER
```

1.3.4 TIMER SERVICES AND THE "REAL-TIME" MANAGER

The Computer System's timer services, controlled by a "real-time" manager, allow programs to access a real-time clock with a resolution of 1/32768 seconds. User programs can query the time of day or specify a time interval that will elapse before an "alarm" is set in either polled or "wake-up" mode. This interval has a 1/1024 second resolution and may be specified as relative (to current date and time) or absolute.

The "Real-Time" Manager is based on the CS 9000 time-of-day clock and therefore provides a much more accurate timing facility than the DELAY system call which is based on the time-slice clock.

Please keep in mind when using these timing facilities that user programs are run at hardware interrupt level zero and that any interrupt activity at all, whether disk, keyboard I/O, time slicing, or whatever, will necessarily preempt your application program and cause delays.

It is not realistic to presume that you can program time periods of only a few milliseconds and maintain accuracy while at the same time you have initiated I/O activity which will preempt your program during the data transfer process. In such cases the Real Time Manager will make your program READY at the proper time, but your program will not run at that time because other activity of a higher priority level is taking place.

The Real Time Manager is most suitable for providing periodic interrupts at an accurate repetition rate, and for providing the ability to schedule events in the future at a precise time. These characteristics are useful for direct instrument control and for taking periodic samples at a precise rate.

1.3.4.1 Polled and "Wake-up" Mode

When an alarm timeout is started with either ALRM or ALRMR, the RTT mode field is checked to determine whether polled or wake-up mode was selected. In the polled mode, the alarm will simply provide an indication in the RTT status area. In wake-up mode, an asynchronous operation will be started and when the alarm occurs, the RTT status will be updated and the user program is awakened if in a suspended state according to its priority.

1.3.4.2 Time and Date Format

The time format for the RTT data output and the RTT data input areas is identical. Byte fields of hexadecimal values are used for the time and date, while the "fractional seconds field" is a 15-bit counter value. The format is as follows:

```
RTTDA   | 00  | year  | month  | day  |    (4 byte field)
RTTTM   | 00  | hour  | minute | second|   (4 byte field)
RTTMS   |     count     |                  (2 byte field)
```

The date "January 2, 1983" and time "13:30:59" would be:

```
RTTDA0  DC.L  $00530102        83/01/02
RTTTM0  DC.L  $000D1E3B        13:30:59
RTTMS0  DC.W  $0000            0
```

1.3.4.3 RTT Block Format

The information below represents the data in the system include file for the RTT block. The notation "S" means that the system sets this field while "U" means that the user program sets this field. This control block must be aligned on a word boundary.

```
RTTSTA     DS.B     1      (S) status (Idle = 0,
                                Ticking = $FF,
                                Alarm occurred = 1)
RTTMOD     DS.B     1      (U) mode (Polled = 0, Wakeup = 1)
RTTIDX     DS.W     1      (S) index
RTTDA0     DS.L     1      (U) date -- data output area
RTTTM0     DS.L     1      (U) time
RTTMS0     DS.W     1      (U) count
RTTDAI     DS.L     1      (S,U) date -- data input area
RTTTMI     DS.L     1      (S,U) time
RTTMSI     DS.W     1      (S,U) count
```

The real-time manager commands are listed on the following pages.

For all of the commands, a branch to an error routine identified as ERRORLABEL in these pages is executed. Status is also returned in D7.W; zero indicates success but a positive value indicates an error.

1.3.4.4 Printing the Time

One way of printing the time is to convert it to GETTIM output, use the FMTTOD system call (to get ASCII format), and then display that. The conversion program is as follows:

```
LEA     STRING,A6          18-character string
CLR.L   D1
CLR.L   D2
MOVE.L  RTTDA(Ax),D3       year/month/day
MOVE.B  D3,D2
LSR.L   #8,D2
LSL.L   #8,D3
MOVE.B  D3,D2
LSR.L   #8,D2
LSL.L   #8,D3
MOVE.B  D3,D2              day/month/year in D2
MOVE.L  RTTTM(Ax),D3       hour/minute/second
MOVE.B  D3,D1
LSR.L   #8,D1
LSL.L   #8,D3
MOVE.B  D3,D1
LSR.L   #8,D1
LSL.L   #8,D3
MOVE.B  D3,D1              second/minute/hour in D1
FMTTOD                     input is D1,D2,A6

. . .                      code for printing desired
. . .                      string or portion of string
```

RTMGR OPEN,RTT,ERRORLABEL

This command opens the timer facility and points to the task data block that will be used for timer control and date/time transfer.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful open, D7.W=5 unsuccessful (cannot allocate RTT block).

Probable cause for an error during OPEN is that all RTMGR system timer blocks are currently in use.

Data Control Blocks:

RTT

Note: You must issue the RTMGR OPEN Command before using any of the other RTMGR commands.

Coding Example:

```
610
611 0 0000010A                    RTMGR   OPEN,RTTBLK,ERROR
    0 0000010A 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000110 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 00000112 0001               DC.W    RTOPEN             + COMMAND WORD
    0 00000114 600000F2           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR CLOSE,RTT,ERRORLABEL

This command releases the timer facility associated with the specified RTT block.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful completion. D7.W=5, timer completed, RTMGR stop must be issued before CLOSE.

It is illegal to try and close an RTT block that still has an active timeout running on it.

Data Control Blocks:

RTT

Coding Example:

```
618
619 0 00000118                    RTMGR   CLOSE,RTTBLK,ERROR
    0 00000118 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 0000011E 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 00000120 0002               DC.W    RTCLOSE            + COMMAND WORD
    0 00000122 600000E4           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR QTIME,RTT,ERRORLABEL

This command returns the current date/time to the RTT block input area. The time is at a resolution of 1/32768 seconds.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful completion. No error code is defined.

Data Control Blocks:

RTT

Coding Example:

```
626
627 0 00000126                    RTMGR   QTIME,RTTBLK,ERROR
    0 00000126 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 0000012C 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 0000012E 0003               DC.W    RTQTIME            + COMMAND WORD
    0 00000130 600000D6           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR QALRM,RTT,ERRORLABEL

This command returns the current setting of the task alarm to the RTT input area. The alarm may be active or inactive as indicated by the RTT status.

Register Usage:

A6.L point to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful completion. D7.W=5 unsuccessful (RTT invalid or not opened).

Error if RTT not opened, or if user has modified SYSTEM fields in the RTT.

Data Control Blocks:

RTT

Coding Example:

```
634
635 0 00000134                    RTMGR   QALRM,RTTBLK,ERROR
    0 00000134 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 0000013A 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 0000013C 0004               DC.W    RTQALRM            + COMMAND WORD
    0 0000013E 600000C8           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR STOP,RTT,ERRORLABEL

This command can be used to terminate a currently active alarm.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful completion
D7.W=5 unsuccessful (RTT invalid or not opened)

Data Control Blocks:

RTT

Coding Example:

```
642
643 0 00000142                    RTMGR   STOP,RTTBLK,ERROR
    0 00000142 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000148 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 0000014A 0006               DC.W    RTSTOP             + COMMAND WORD
    0 0000014C 600000BA           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR ALRM,RTT,ERRORLABEL

Set the absolute date/time for an alarm from the RTT data output area. The time will be truncated to a 1/1024 second resolution. You must set RTTMOD to either polled or wakeup mode before issuing this macro.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 successful completion
D7.W=5 unsuccessful completion (RTT invalid or not opened)

An error is returned if the alarm setting is for a time in the "past", or is less than 1 millisecond into the future.

Data Control Blocks:

RTT

Coding Example:

```
650
651 0 00000150                    RTMGR   ALRM,RTTBLK,ERROR
    0 00000150 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000156 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 00000158 0007               DC.W    RTALRM             + COMMAND WORD
    0 0000015A 600000AC           BRA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR ALRMR,RTT,ERRORLABEL

Set the alarm from the relative date/time in the RTT data output area. The resulting absolute alarm time may then be queried with the QALRM command. You must set RTTMOD to either polled or wakeup mode before issuing this macro.

Register Usage:

A6.L points to RTT block
D7.W returns status

Completion Codes:

D7.W=0 Successful completion
D7.W=5 Unsuccessful completion (RTT invalid or not opened).

An error is returned if the alarm setting is for a time in the "past", or is less than 1 millisecond into the future

Data Control Blocks:

RTT

Coding Example:

```
658
659 0 0000015E                    RTMGR   ALRMR,RTTBLK,ERROR
    0 0000015E 4DF900000220       LEA     RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000164 4E4A               TRAP    #10                + TRAP TO REAL-TIME MANAGER
    0 00000166 0008               DC.W    RTALRMR            + COMMAND WORD
    0 00000168 6000009E           ERA.L   ERROR              + BRANCH TO USER ERROR HANDLER
```

RTMGR SUM,RTT,ERRORLABEL

This command returns the sum of the date/time in the RTT data output area and the RTT input area to the data input area in the correct format.

Register Usage:

A6.L points to RTT block
D7.W returns the status

Completion Codes:

D7.W=0 successful completion
D7.W=5 unsuccessful completion, (RTT invalid or not opened)

Data Control Blocks:

RTT

Coding Example:

```
666
667 0 0000016C                    RTMGR    SUM,RTTBLK,ERROR
    0 0000016C 4DF900000220       LEA      RTTBLK,A6          + LOAD ADDRESS OF CONTROL BLOCK
    0 00000172 4E4A               TRAP     #10                + TRAP TO REAL-TIME MANAGER
    0 00000174 0009               DC.W     RTSUM              + COMMAND WORD
    0 00000176 60000090           BRA.L    ERROR              + BRANCH TO USER ERROR HANDLER
```

1.3.4.5 Sample Program Using the Real-Time Manager

The program on the opposite page illustrates one use of the real-time manager. It opens the RTT block, reads the current time, sets the mode to wakeup, issues an alarm for 15 seconds into the future, issues a suspend for the timeout, and then closes the RTT.

1.3.5 NAMING AND OTHER CONVENTIONS

1.3.5.1 File Name Format

A fully qualified filename consists of three fields: A volume identifier, a filename, and an extension. When file names are specified within system commands, you must use specific delimiters to separate the fields. The standard format is shown below:

<volume:>filename.ext

volume — is one to six alphanumeric characters always terminated by a colon. This field can be omitted in which case the default volume is used. Specifying a logical drive number in the volume field (0-3 for diskette drives #FD00-#FD03) OPENs the volume name that is mounted in the specified drive.

filename — is one-to-eight alphanumeric characters with the leading character alphabetic. This is always followed by a period and the filename extension.

ext — is one to three alphanumeric characters with the leading character alphabetic.

EXAMPLES:

```
123456:BLUE.SRC                SAMP:TESTCASE.BIN
VOL7:CALCPGM.REL               SPECT1.REL
```

The "Wildcard Feature" is the same as described in Computer System Operating System Reference, Part 1, GC22-9199, except that it does not apply to volume names.

```
  1                       RTMTEST  IDNT    1,1

 54                       * RTMMAC50.INC REV 22-SEP-83 HAS BEEN INCLUDED
357                       * SYSMAC50.INC REV 22-SEP-83 HAS BEEN INCLUDED
597                       * IOMCLB50.INC REV 22-SEP-83 HAS BEEN INCLUDED
598
599 0 00000000                     TYPE    (OPENING RTT BLOCK)
600 0 00000022                     RTMGR   OPEN,RTTBLOCK,ERREXIT
601 0 00000030                     TYPE    (READING CURRENT TIME)
602 0 00000056                     RTMGR   QTIME,RTTBLOCK,ERREXIT

603                       * NOTE THAT WE ARE NOT USING THIS READING FOR ANYTHING IN THE SAMPLE
604                       * PROGRAM BUT IT IS AVAILABLE IN THE RTT INPUT AREA IF WE WANT IT.
605                       * FOR EXAMPLE WE COULD USE IT TO COMPUTE AN "ABSOLUTE" WAKEUP TIME.

606 0 00000069                     TYPE    (SET ALARM 15 SECONDS FROM NOW, WAKEUP MODE)

607                       * WE'RE USING THE RELATIVE ALARM... 15 SECONDS FROM NOW

608 0 000000A0 42B900000164        CLR.L   RTTBLOCK+RTTDAO       NO CHANGE TO DATE
609 0 000000A6 23FC0000000F        MOVE.L  #$0000000F,RTTBLOCK+RTTTMO RELATIVE 15 SECONDS TO TIME
               00000168
610 0 000000B0 427900000160        CLR.W   RTTBLOCK+RTTMSO       NO CHANGE TO FRACTIONAL SECONDS
611 0 000000B6 13FC00010000        MOVE.B  #1,RTTBLOCK+RTTMOD    SET MODE TO WAKEUP
               0161
612 0 000000BE                     RTMGR   ALRMR,RTTBLOCK,ERREXIT
613 0 000000CC                     TYPE    (SUSPEND ISSUED)
614 0 000000EC                     SUSPEND TILLALL,#0,ERREXIT
    0 000000F0 60000066            BRA.L   ERREXIT               + BRANCH TO USER ERROR HANDLER
615 0 000000F4                     TYPE    (PROGRAM AWAKENED, ISSUING RTT CLOSE)
616 0 00000128                     RTMGR   CLOSE,RTTBLOCK,ERREXIT
    0 00000132 60000024            BRA.L   ERREXIT               + BRANCH TO USER ERROR HANDLER
617 0 00000136                     TYPE    (PROGRAM ENDED)
618 0 00000154                     EXIT
619 0 00000158             ERREXIT PRTERR
620 0 0000015C                     EXIT

621 0 00000160 00000000    RTTBLOCK DS.W    0                   ALIGN TO WORD BOUNDARY
622 0 00000160 00000001             DS.B    1                   RTTSTA  STATUS
623 0 00000161 00000001             DS.B    1                   RTTMOD  MODE
624 0 00000162 00000002             DS.W    1                   RTTIDX  INDEX
625 0 00000164 00000004             DS.L    1                   RTTDAO  ;
626 0 00000168 00000004             DS.L    1                   RTTTMO  ; DATA OUTPUT
627 0 0000016C 00000002             DS.W    1                   RTTMSO  ;
628 0 0000016E 00000004             DS.L    1                   RTTDAI  ;;
629 0 00000172 00000004             DS.L    1                   RTTTMI  ;; DATA INPUT
630 0 00000176 00000002             DS.W    1                   RTTMSI  ;;
631                                 END

****** TOTAL ERRORS      0-- 616
```

1.3.5.2 File Extensions

BIN	Binary programs and files
ASM	Assembler source statement files
INC	Assembler include files
OBJ	Relocatable object files - individual
LNK	Relocatable object files - linked together
SUB	Submit files
DRV	Binary device drivers
SYS	Operating system programs and files
LIB	Library files

Files extensions to avoid:

TMP	These extensions are used by the editor program.
BAK	

1.3.5.3 Logical Device Naming

Logical device names are established by the device drivers. The names of devices attached to the system can be displayed by the operating system LISTDEV command. These names are "bound" to user program logical unit numbers with the SYSIO-OPEN command and a Device Initialization Block (DIB) supplied by the user's program. Task logical unit assignments can be displayed with the SHOW command. The pre-assigned logical device names are shown below.

Hard Disk	#HD00 through #HD03
Floppy Diskette	#FD00 through #FD03
IEEE-488 Bus	#BUS00 (additional devices would be #BUS01, etc.)
Parallel Port	#PPU
Keyboard/Keypad	#CON
Keypad/control	#KPD
CRT Graphics	#GR
Printer	#PR
RS-232 Asynch	#SER01, #SER02
RS-232 Bisynch/Asynch	#SER00
CRT (see note)	#SCRN0 for page 0 #SCRN1 for page 1 #CNSLO

Note: The CRT has ability to display one of the two memory pages at any given time. The page to be displayed can be set by using a function call or the PAGE command.

1.3.5.4 Numbers

- Unless otherwise noted, numbers are decimal.
- Hexadecimal numbers are preceded by a dollar sign, e.g. $4FFF0.

2.0 KEYBOARD DRIVER

2.1 DRIVER DESCRIPTION

The Keyboard Driver supports the keyboard option. Note: The optional keyboard must be installed to use this driver. The IBM Instruments Computer System can be configured with just keypad and softkeys. See Chapter 3 for information related to the keypad and softkeys and the keypad driver. The Keyboard Driver is sharable and supports Byte I/O, asynchronous event posting and asynchronous requests but does not support attach or detach devices. The keyboard and keypad driver share the same resources. If either is "detached" the other will also be detached.

EVENTS SUPPORTED BY #CON DRIVER

Name	When Signalled
"CBUF"	Any key depression or shift state change. The EVBSTA field returns the current BYTE I/O longword.
"CTLB"	When CTRL-BREAK is pressed.
"CTLN"	On CTRL-NUMLOCK EVBSTA=-1, on RESUME EVBSTA=0.
"F1 "	When Function Key 1 is pressed.
"F2 "	When Function Key 2 is pressed.
"F3 "	When Function Key 3 is pressed.
"F4 "	When Function Key 4 is pressed.
"F5 "	When Function Key 5 is pressed.
"F6 "	When Function Key 6 is pressed.
"F7 "	When Function Key 7 is pressed.
"F8 "	When Function Key 8 is pressed.
"F9 "	When Function Key 9 is pressed.
"F10 "	When Function Key 10 is pressed.

KEYBOARD ENCODING AND USAGE

ENCODING

The keyboard routine provided by IBM in ROM is responsible for converting the keyboard scan codes into what will be termed "Extended ASCII".

Extended ASCII encompasses one byte character codes with possible values of 0-255, an extended code for certain extended keyboard functions and functions that are handled within the keyboard routine or through interrupts.

CHARACTER CODES

The following character codes are passed through the keyboard routine to the system or application program. A "-1" means the combination is suppressed in the keyboard routine.

KEY	BASE CASE	UPPER CASE	CTRL	ALT
1	ESC	ESC	ESC	-1
2	1	½	-1	Note 1
3	2	@	NUL (000) Note 1	Note 1
4	3	#	-1	Note 1
5	4	$	-1	Note 1
6	5	%	-1	Note 1
7	6		RS (030)	Note 1
8	7	&	-1	Note 1
9	8	*	-1	Note 1
10	9	(	-1	Note 1
11	0	)	-1	Note 1
12	-	_	US (031)	Note 1
13	=	+	-1	Note 1
14	Backspace (008)	Backspace (008)	DEL (127)	-1
15	→ (009>	← (Note 1)	-1	-1
16	q	Q	DC1 (017)	Note 1
17	w	W	ETB (023)	Note 1
18	e	E	ENQ (005)	Note 1
19	r	R	DC2 (018)	Note 1
20	t	T	DC4 (020)	Note 1
21	y	Y	EM (025)	Note 1
22	u	U	NAK (021)	Note 1
23	i	I	HT (009)	Note 1
24	o	O	SI (015)	Note 1
25	p	P	DLE (016)	Note 1
26	[	{	ESC (027)	-1
27	]	}	GS (029)	-1
28	CR	CR	LF (010)	-1
29	-1	-1	-1	-1
30	a	A	SOH (001)	Note 1
31	s	S	DC3 (019)	Note 1
32	d	D	EOT (004)	Note 1
33	f	F	ACK (006)	Note 1
34	g	G	BEL (007)	Note 1

CHARACTER CODES (Cont.)

KEY	BASE CASE	UPPER CASE	CTRL	ALT
35	h	H	BS (008)	Note 1
36	J	J	LF (010)	Note 1
37	k	K	VT (011)	Note 1
38	l	L	FF (012)	Note 1
39	;	:	-1 -1	
40	'	"	-1 -1	
41		~	-1 -1	
42SHIFT	-1	-1	-1 -1	
43	\		FS (028)	-1
44	z	Z	SUB (026)	Note 1
45	x	X	CAN (024)	Note 1
46	c	C	ETX (003)	Note 1
47	v	V	SYN(022)	Note 1
48	b	B	STX (002)	Note 1
49	n	N	SO (014)	Note 1
50	m	M	CR (013)	Note 1
51	'	<	-1 -1	
52	.	>	-1 -1	
53	/	?	-1 -1	
54SHIFT	-1	-1	-1 -1	
55	*	(Note 2)	(Note 1)	-1
56ALT	-1	-1	-1 -1	
57	SP	SP	SP SP	
58CAPS LOCK	-1	-1	-1 -1	
59	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
60	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
61	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
62	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
63	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
64	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
65	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
66	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
67	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
68	NUL (Note 10	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
69NUM LOCK	-1	-1	Pause (Note 2)	-1
70SCROL LOCK	-1	-1	Break (Note 2)	-1

Note 1: Refer to Extended Codes (see below).

Note 2: Refer to Special Handling (see below).

Keys 71-83 have meaning only in base case, in NUMLOCK (or shifted) states, or in CTRL state. It should be noted that the shift key temporarily reverses the current NUMLOCK state.

KEY #	NUM LOCK	BASE CASE (Note 3)	ALT	CTRL (Note 3)
71	7	Home (Note 1)	Note 1	Clear Screen
72	8	↑ (Note 1)	Note 1	-1
73	9	Page Up (Note 1)	Note 1	Top of Text & Home
74	-	-	-1	-1
75	4	← (Note 1)	Note 1	Reverse Word
76	5	-1	Note 1	-1
77	6	→ (Note 1)	Note 1 (Note 1	Adv Word
78	+	+	-1	-1
79	1	End (Note 1)	Note 1	Erase to EOL
80	2	↓ (Note 1)	Note 1	-1
81	3	Page Down (Note 1) (Note 1)	Note 1	Erase to EOS
82	0	INS	Note 1	-1
83	.	DEL (Notes 1,2)	Note 2	Note 2

Note 1: Refer to Extended Codes (see below).
Note 2: Refer to Special Handling (see below).
Note 3: The meanings of keystrokes is a recommendation for applications, and is not implemented in the operating system command interpreter.

A. EXTENDED CODES

For certain functions that cannot be represented in the standard ASCII code, an extended code is used. This indicates that the system or application program should examine a second code that will indicate the actual function. Usually, but not always, this second code is the scan code of the primary key that was pressed. Extended ASCII cannot be read with a normal SYSIO SREAD command. The extended code can be accessed through BYTE I/O, through Function Key Mode, or through a "table lookup facility." These capabilities are described later on.

Keyboard Extended Codes

SECOND CODE	FUNCTION (Note 3)
3	NUL Character
15	← 16-25 ALT Q,W,E,R,T,Y,U,I,O,P
30-38	ALTYA,S,D,F,G,H,J,K,L
44-50	ALTYZ,X,C,V,B,N,M
59-68	F1-F10 Function Keys Base Case
71	Home
72	↑
73	Page Up & Home Cursor
75	←
77	→
79	End
80	
81	Page Down & Home Cursor
82	INS
83	DEL
84-93	F11-F20 (Upper Case F1-F10)
94-103	F21-F30 (CTRL F1-F10)
104-113	F31-F40 (ALT F1-F10)
114	CTRL PRTSC Key 55
115	CTRL← Reverse Word
116	CTRL→ Advance Word
117	CTRL END Erase EOL
118	CTRL PG DN Erase EOS
119	CTRL HOME Clear Screen and home
120-131	ALT 1,2,3,4,5,6,7,8,9,0,-,=(Keys 2-13)

B. SHIFT STATES

Most shift states are handled within the keyboard routine transparently to the system or application program. In any case, the current set of active shift states are available by using BYTE I/O or by monitoring the event "CBUF". The following keys result in altered shift states.

Shift - Temporarily shifts keys 2-13, 15-27, 30-41, 43-53, 55, 59-68 to upper case (lower case if in CAPSLOCK state). Temporarily reverses NUMLOCK/NONUMLOCK state of keys 71-73, 75, 77, 79-83.

CTRL - Temporarily shifts keys 3, 7, 12, 14, 16-28, 30-38, 43-50, 55, 59-71, 73, 75, 77, 79, 81 to CTRL state. Used with ALT and DEL to cause "system reset" function described in Section 2.4. Used with

SCROLL LOCK to cause 'break' function described in Section 2.4. Used with NUMLOCK to cause "pause" function described in Section 2.4.

ALT - Temporarily shifts keys 2-13, 16-25, 30-38, 44-50, and 59-68 to ALT state. Used with CTRL and DEL to cause system reset function described in Section 2.4.

CAPS LOCK - Shifts keys 16-25, 30-38, 44-50 to upper case. A second depression of CAPS LOCK reverses the action. Handled internal to keyboard routine.

NUM LOCK - Shifts keys 71-73, 75-77, 79-83 to numeric state. A second depression of NUM LOCK reverses the action. Handled internal to keyboard routine.

SCROLL LOCK - Interpreted by appropriate application programs as indicating that the use of the cursor control keys should cause windowing over the text rather than cursor movement. A second depression of SCROLL LOCK reverses the action. The keyboard routine simply records the current shift state of SCROLL LOCK. It is up to the system or application program to perform the function.

C. SHIFT KEY PRIORITIES AND COMBINATIONS

If combinations of ALT, CTRL and SHIFT are pressed and only one is valid, the precedence is as follows: Highest is ALT, then CTRL, then SHIFT. The only valid combination is ALT CTRL, which is used in system reset.

SPECIAL HANDLING

A. SYSTEM RESET

The combination of ALT CTRL DEL (Key 83) will result in the keyboard routine initiating the equivalent of a system restart. Handled internal to keyboard routine.

B. BREAK

The combination CTRL BREAK will result in the keyboard routine signaling event "CTLB". See also Functions 7 and 8.

C. PAUSE

The combination CTRL NUM-LOCK will cause the keyboard interrupt routine to loop, waiting for any key except PRTSC to be pressed. This provides a system/application transparent method of suspending list/print/etc. temporarily, and then resuming. The event "CTLN" is

signalled whenever the system enters or leaves "Pause" state. A value of -1 is returned if PAUSE is on. A zero is returned if PAUSE is off. The "Unpause" key is thrown away. Handled internal to keyboard routine. See also Function 9.

D. TYPAMATIC ACTION

The following keys will have their typamatic action suppressed by the keyboard routine: CTRL, SHIFT, ALT, NUM-LOCK, SCROLL-LOCK, CAPS LOCK, INS.

E. PRINT SCREEN

The combination SHIFT-PRINT SCREEN (Key 55) will result in an interrupt invoking the print screen routine. This routine works in graphics mode.

DRIVER PRINCIPLE OF OPERATION

The Keyboard driver processes key strokes from the keyboard. Each keystroke generates two hardware scan-codes; one when the key is pressed and one when the key is released. See Figure 2-1. An interrupt is generated, the associated character (if any) is echoed to an Alpha Window on the CRT and the scan-codes are stored in a 32 keystroke buffer where they are held until they can be individually processed by the keyboard driver. The keyboard driver stores them, along with their extended ASCII translation and two other bytes which represent the state of the keyboard at the time the key was pressed. For example Control Keys, Alternate Keys, Numbers Lock etc. From this 32 entry circular buffer (#CON) the information is available to transfer to a user buffer or for use by an applications program. Extended ASCII encompasses one byte character codes with possible values of 0 to 255, an extended code for certain extended keyboard functions and functions that are handled by the keyboard driver or through interrupts. Entries may also be made into the #CON buffer from the keypad driver. See Chapter 3 for information on how this is accomplished.

If the circular buffer becomes full and an additional keystroke comes in, then the last keystroke in the buffer will be overlaid. This procedure allows user programs to use EVENTS without ever reading from the buffer. (If the additional keystrokes were ignored, then the user program would need to keep reading from #CON in order to make the buffer not become full.)

Each SYSIO-BREAD will obtain a longword from the #CON buffer. Register D0.L will always contain information on the last key that was struck, regardless of whether there is a byte available in the buffer or not.

BYTE I/O LONGWORD DESCRIPTION

BYTE 0: Contains "KBFLAG" defined as follows:
- Bit 7: Insert state
- Bit 6: Caps lock has been toggled
- Bit 5: Numeric lock has been toggled
- Bit 4: Scroll lock has been toggled
- Bit 3: Alt shift key depressed
- Bit 2: Control shift key depressed
- Bit 1: Left shift key depressed
- Bit 0: Right shift key depressed

BYTE 1: Contains "KBFLAG1" defined as follows:
- Bit 7: Insert key depressed
- Bit 6: Caps lock key depressed
- Bit 5: Numeric lock key depressed
- Bit 4: Scroll lock key depressed

BYTE 2: Contains the scancode of the key that was pressed.

BYTE 3: Contains the translated ASCII code of the key that was pressed. Contains zero if there is no ASCII representation.

Each SYSIO-AREAD or SYSIO-SREAD will transfer a line of console input to the buffer specified in the Data Transfer Control Block (DTCB). There are several line-editing options that may be specified with function packets using SYSIO-OPEN or SYSIO-FUNCTION. See Paragraph 2.4.2.

There is a "table lookup facility" in the keypad driver that may be used to translate non-ASCII keystrokes such as function keys and cursor keys into program control functions or into ASCII strings. See functions 12, 13 and 19 of the Keypad Driver for an explanation on how to use this facility.

Function and cursor keys are normally ignored if they do not appear in the lookup table. However, you can use "Function Key Mode" to gain access to them. In this mode you will receive an exception code of $10 whenever a control or function key is pressed that could not be translated. Three bytes will be placed in the buffer specified in your data transfer control block starting at the offset contained in the DTCB buffer pointer. The first byte is KBFLAG, the second is KBFLAG1, and the third is the SCANCODE. Text entered before the exception key was pressed remains in the buffer. You may also use SYSIO-BREAD to gain access to function and cursor keys.

2.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at open time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the keyboard the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. When this is done all standard SYSIO operations described in Chapter 1 are allowed except writes.

2.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION OF USE
DIBVOL	DS.B	6	Device name. Use #CON for keyboard driver.
DIBDTD	DS.B	1	Data transfer direction. Use 1.This driver is read only.
DIBTRN	DS.B	1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRS0	DC.L	0,0	Not used. User sets this field to 0.
DIBOPT	DC.W	0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L	1	Insert pointer to function packet or set this field to $0000 0000 to select the default mode.
DIBBIO	DS.L	1	System used Byte I/O Field. To open the keyboard for Byte I/O enter -1 ($FFFF FFFF), otherwise set it to zero. After OPEN the I/O manager fills this field with an identifier which is used for SYSIO-BREAD and SYSIO-BTEST

2.3 KEYBOARD DATA TRANSFER CONTROL BLOCK (DTCB).

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs. It is required operand of the SYSIO macro for READs or WRITEs. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

2.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts lower limit to be used for Transfer Termination characters in Variable length transfer here.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ.

2.4 KEYBOARD FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like setting the amount of space between tab columns or causing a prompt string

such as ENTER: to appear on the CRT. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more words or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

2.4.1 SUMMARY OF KEYBOARD FUNCTIONS

The function listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	$0021	WORD
SET EDIT OPTIONS	2 ($0002)	NONE	WORD
PROMPT ON	3 ($0003)	NONE	NONE
PROMPT OFF	4 ($0004)	NONE	NONE
ECHO ON	5 ($0005)	NONE	NONE
ECHO OFF	6 ($0006)	NONE	NONE
RESET CTRL-BREAK	7 ($0007)	NONE	NONE
GET CTRL-BREAK ADDRESS	8 ($0008)	NONE	LONGWORD ADR RETURNED
GET CTRL-NUMLOCK ADDRESS	9 ($0009)	NONE	LONGWORD ADR RETURNED
SET ECHO	10 ($000A)	NONE	LONGWORD
SET TAB	11 ($000B)	$002B	WORD
SET PROMPT	12 ($000C)	NONE	LONGWORD POINTER
PARSE NEXT TOKEN	13 ($000D)	$002D	NONE
FUNKY MODE ON	14 ($000E)	NONE	NONE
FUNKY MODE OFF	15 ($000F)	NONE	NONE
PARSING ON	16 ($0010)	NONE	NONE
PARSING OFF	17 ($0011)	NONE	NONE
SET SCROLL LOCK ON	18 ($0012)	None	NONE
SET SCROLL LOCK OFF	19 ($0013)	NONE	NONE
GET SCROLL LOCK STATE	20 ($0014)	NONE	WORD RETURNED
SET NUM LOCK ON	21 ($0015)	NONE	NONE
SET NUM LOCK OFF	22 ($0016)	NONE	NONE
GET NUM LOCK STATE	23 ($0017)	NONE	WORD RETURNED
SET CAPS LOCK ON	24 ($0018)	NONE	NONE
SET CAPS LOCK OFF	25 ($0019)	NONE	NONE
GET CAPS LOCK STATE	26 ($001A)	NONE	WORD RETURNED

SET INSERT MODE ON	27 ($001B)	NONE	NONE
SET INSERT MODE OFF	28 ($001C)	NONE	NONE
GET INSERT MODE STATE	29 ($001D)	NONE	WORD RETURNED
GET EDIT OPTIONS	30 ($001E)	NONE	WORD RETURNED
GET PROMPT STATE	31 ($001F)	NONE	WORD (0/1) RETURNED
GET ECHO STATE	32 ($0020)	NONE	WORD (0/1) RETURNED
GET TAB AMOUNT	33 ($0021)	NONE	WORD RETURNED
GET FUNKY MODE	34 ($0022)	NONE	WORD (0/1) RETURNED
GET PARSING MODE	35 ($0023)	NONE	WORD (0/1) RETURNED
GET TRANSFER MODE	36 ($0024)	NONE	WORD (0/1) RETURNED
SET CTRL-BREAK	37 ($0025)	NONE	NONE

2.4.2 KEYBOARD FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminate processing of the function packet.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	Allows user to specify either fixed length or variable length transfers. When fixed length transfers are specified the driver refers to the buffer length specified in the Data Transfer Control Block (DTCB) to determine the amount of the data to transfer. When variable length transfers are specified the driver will check each byte to see if it lies within the range specified in the Data Transfer Control Block for termination characters and will terminate transfer of data it is does.
	Function Data	Word, integer.
	Data = $0000	Selects fixed length transfers.
	$0001	Selects variable length transfers.

	Error Code	$0021 Data out of limits.
2	SET EDIT OPTIONS	Allow the user to set the bits in the Edit Control Word. You may select whether backspaces and forward tabs are to participate in the line editing process. Keep in mind, however, that the control characters ($08 and $09) will be echoed to the window (if echoing is enabled) and also placed in your input buffer. If you specify that nulls are to be transferred rather than ignored, this means that you will receive an ASCII character of $00 in your buffer if keys are pressed which do not result in an ASCII representation.
	Function Data	One Word, integer
	Data Bit 8 = 0 9 = 0 11 = 0	Enables backspace. Enable forward tab. Disregard nulls not in look up table.
	Error Code	None
	Default Setting:	Driver initializes to zero at open time.
3	PROMPT ON	Causes the character string established by SET PROMPT to be sent to the echo device before a READ.
	Function Data	None
	Error Code	None
4	PROMPT OFF	Disables the prompt character string from being sent to the echo device. This is the default setting at open time.

	Function Data	None
	Error Code	None
5	ECHO ON	Cause the characters to be echoed to the CRT window specified by SETECHO or to the default window. This is the default setting at open time.
	Function Data	None
	Error Code	None
6	ECHO OFF	Disables the echo function. Characters will not appear on the CRT.
	Function Data	None
	Error Code	None
7	RESET CTRL-BREAK	When the CTRL-BREAK Keys are depressed a CTRL-BREAK byte is set to $FF . This function resets this global byte to $00 so that the occurrence of the CTRL-BREAK can be detected and used within an application program. It is often used to abort a function or leave a loop.
	Function Data	None
	Error Code	None
8	GET CTRL-BREAK ADDRESS	Returns the address of the CTRL-BREAK byte. See Function 7 above and Function 37.

	Function Data	Leave space for one longword, integer.
	Error Code	None
9	GET CTRL-NUMLOCK ADDRESS	Returns the address of the CTRL-NUMLOCK byte. This global byte is set to $FF whenever CTRL-NUMLOCK is pressed and reset to $00 whenever any other key is pressed. Many programs need a way to temporarily halt their operations and then later resume via operator command. One example is screen scrolling. The CTRL-NUMLOCK global byte may be tested within an application program to accomplish this end.
	Function Data	Leave space for one longword, integer.
	Error Code	None
10	SETECHO	Allows the user who has opened a window for Byte I/O to use that window for keyboard echoing.
	Function Data	Longword, integer.
	Data = $FFFF FFFF	Specifies the default screen.
	Data = Longword returned in DIBBIO	Identifies the window opened for Byte I/O. (A longword window identifier is returned in DIBBIO by OPEN.)
	Error Code	None
11	SET TAB AMOUNT	Allows the user to specify the number of spaces between tab columns. When the tab key is pressed the cursor is moved to the next tab column. The default tab amount of 10 is set at open

		time. Each LUN has its own tab control.
	Function Data	One word, integer.
	Data = $000A	Sets 10 spaces between tab columns.
	Error Code	$002B Error if tab amount specified is zero or is greater than 79.

12	SET PROMPT	Allows the user to create a prompt string of up to 8 characters which will be sent to the echo device before reads. If less than 8 characters are required fill the bytes on the right with blanks. The long word specified in the data points to the character string. Note that you must issue Function 3 to turn prompting on.
	Function Data	One longword, integer Note: This address is not checked by the driver. A DTAK TRAP error will occur if you supply an invalid address.
	Data = $NNNN NNNN	Points to user created character string.
	Error Code	None

13	PARSE NEXT TOKEN	Causes the next token in the input line to be parsed when the PARSING ON function has been activated. Same as NXTOK system call.
	Function Data	None
	Error Code = $002D	Parsing mode is not on.

14	FUNKY MODE ON	Activate the " Function Key Mode ". In this mode any keystroke which results in an ASCII byte of $00 will cause an exception code of $10 to be returned to the caller. The last three bytes in the input buffer will contain KBFLAG, KBFLAG1, and the scancode of the key that was pressed which had a zero ASCII code. The buffer will contain ASCII codes for any key that was pressed before the ASCII code of zero was pressed.
	Function Data	None
	Error Code	None

15	FUNKY MODE OFF	This function turns off the " Function Key Mode " described above. This is the default setting at open time.
	Function Data	None
	Error Code	None

16	PARSING ON	Activates the parsing mode in which the input line on each READ is sent to the parsing routine. This routine parses the first token in the line and makes the rest of the line available for parsing through use of the " PARSE NEXT TOKEN " function 13. When parsing mode is on, each SREAD behaves something like the System Call GTCMD in that the input line is passed to the NXTOK routine and the first token is parsed. You use the same method of obtaining parsing information as you do with GTCMD/NXTOK, i.e. issue GSTAT to

		to get address of parsing fields in register A5. You may not concurrently parse from both the command line and the SREAD or AREAD buffer since the same parsing area is being used for holding variable data. Parsing is only available for the SYSTEM task.
	Function Data	None
	Error Code	None
17	PARSING OFF	Turns off the parsing mode described above. This is the default setting at open time.
	Function Data	None
	Error Code	None
18	SET SCROLL LOCK ON	This function turns scroll lock on.
	Function Data	None
	Error Code	None
19	SET SCROLL LOCK OFF	This function turns scroll lock off. This is the default setting at cold start.
	Function Data	None
	Error Code	None

20	GET SCROLL LOCK START	This function returns a 0 if scroll lock is off and 1 if scroll lock is on.
	Function Data	One word is returned.
	Error Code	None
21	SET NUM LOCK ON	This function turns numeric lock on. This is the default setting at cold start.
	Function Data	None
	Error Code	None
22	SET NUM LOCK OFF	This function turns numeric lock off.
	Function Data	None
	Error Code	None
23	GET NUM LOCK STATE	This function returns a 0 if numeric lock is off and 1 if numeric lock is on.
	Function Data	One word is returned.
	Error Code	None
24	SET CAPS LOCK ON	This function turns caps lock on. This is the default setting at cold start.
	Function Data	None
	Error Code	None

25	SET CAPS LOCK OFF	This function turns caps lock off.
	Function Data	None
	Error Code	None
26	GET CAPS LOCK STATE	This function returns a 0 if caps lock is off and 1 if caps lock is on.
	Function Data	One word is returned.
	Error Code	None
27	SET INSERT MODE ON	This function turns insert mode on.
	Function Data	None
	Error Code	None
28	SET INSERT MODE OFF	This function turns insert mode off. This is the default setting at cold start.
	Function Data	None.
	Error Code	None
29	GET INSERT MODE STATE	This function returns a 0 if insert mode is off and 1 if insert mode is on.
	Function Data	One word is returned.
	Error Code	None
30	GET EDIT OPTIONS	This function returns a word with bits set according to the current edit options. See function 2.

	Function Data	Word returned
	Error Code	None
31	GET PROMPT STATE	This function returns a zero if prompting is off and a one if prompting is on.
	Function Data	Word returned
	Error Code	None.
32	GET ECHO STATE	This function returns a zero if echo is off and a one if prompting is on.
	Function Data	Word returned
	Error Code	None
33	GET TAB AMOUNT	This function returns a word containing the current tab amount
	Function Data	Word returned
	Error Code	None
34	GET FUNKY MODE	This function returns a zero if funky mode if off and a one if funky mode is on.
	Function Data	Word returned
	Error Code	NONE
35	GET PARSING MODE	This function returns a zero if parsing is off and a one if parsing mode is on.
	Function Data	One word is returned
	Error Code	None

36	GET TRANSFER MODE	This function returns the transfer mode to the user 0=fixed, 1=variable
	Function Data	One word is returned
	Error Code	None

37	SET CTRL-BREAK	This function sets the global byte (accessed by Function 8) to $FF. It also causes the 'CTLB' Event to be signalled. This may be useful in terminating the execution of programs and SUBMIT files.
	Function Data	None
	Error Code	None

2.5 KEYBOARD USAGE

This section summarizes a set of guidelines for key usage when performing commonly used keyboard functions.

FUNCTION	KEY(S)	COMMENT
Home Cursor	HOME	Editors; word processors
Return to outermost menu	HOME	Menu-driven applications
Move cursor up	↑	Full screen editor, word processor
Page up, scroll backwards 25 lines & home	PG UP	Editors; word processors
Move cursor left	←Key 75	Text, command entry
Move cursor right	→	Text, comand entry

FUNCTION	KEY(S)	COMMENT
Scroll to end of text Place cursor at end of line	END	Editors; word processors
Move cursor down	↓	Full screen editor, word processor
Page down, scroll forwards 25 lines & home	PG DN	Editors; word processors
Start/Stop insert text at cursor, shift text right in buffer	INS	Text, command entry
Delete character at cursor	DEL	Text, command entry
Destructive backspace	←Key 14	Text, command entry
Tab forward	|→	Text entry
Tab reverse	←|	Text entry
Clear screen and home	CTRL HOME	Command entry
Scroll up	↑	In scroll lock mode
Scroll down	↓	In scroll lock mode
Scroll left	←	In scroll lock mode
Scroll right	→	In scroll lock mode
Delete from cursor to EOL	CTRL END	Text, command entry
Exit/Escape	ESC	Editor, 1 level of menu, etc
Start/Stop Echo screen to printer	PRTSC CTRL K55	Any time
Delete from cursor to EOS	CTRL PG DN	Text, command entry
Advance word	CTRL →	Text entry
Reverse word	CTRL←	Text entry

FUNCTION	KEY(S)	COMMENT
Window Right	CTRL →	When text is too wide to fit screen
Window Left	CTRL←	When text is too wide to fit screen
Enter insert mode	INS	Line editor
Exit insert mode	INS	Line editor
Cancel current line	ESC	Command entry, text entry
Suspend system (pause)	CTRL NUMLOCK	Stop list, stop program, etc. Resumes on any key
Break interrupt	CTRL BREAK	Interrupt current process
System reset	ALT CTRL DEL	Reboot
Top of document and home cursor	CTRL PG UP	Editors, word processors
Standard Function Keys	F1-F10	Primary function keys
Secondary function keys	SHIFT F1-F10 CTRL F1-F10 ALT F1-F10	Extra function keys if 10 are not sufficient
Extra function keys	ALT Keys 2-13 (1-9,0,-,=	Used when stickers are put along top of keyboard
Extra function keys	ALT A-Z	Used when function starts with same letter as one of the alpha keys

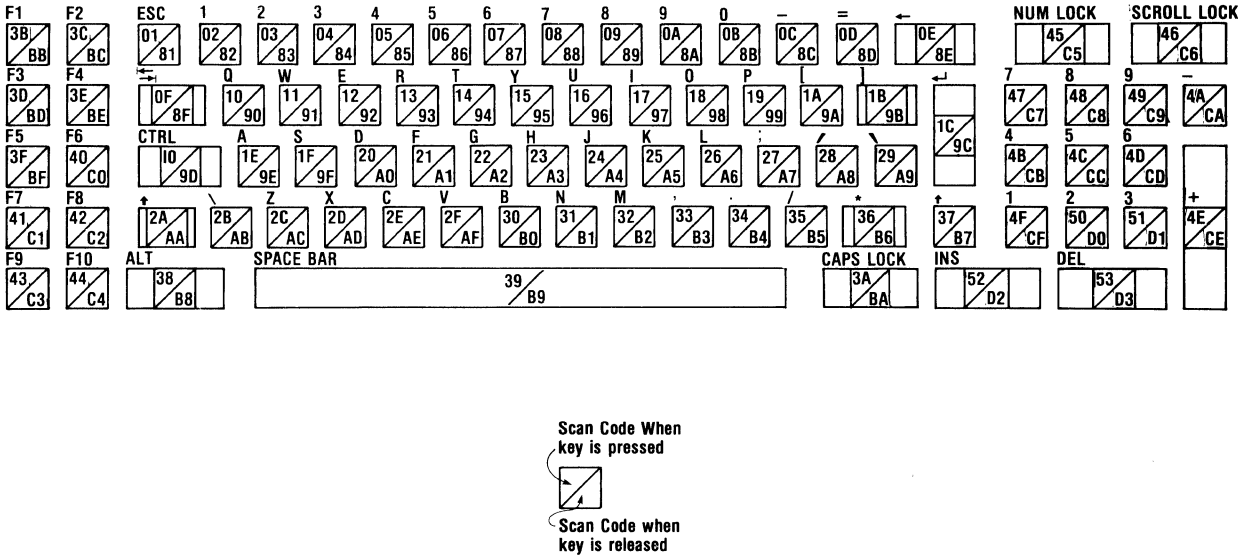


Figure 2-1. Keyboard Scan Codes

3.0 KEYPAD DRIVER

3.1 DRIVER DESCRIPTION

This driver processes keystrokes from the 3 row x 19 column tactile switch matrix located on the front of the printer and the 1 x 10 softkey matrix located directly beneath the CRT screen. It also processes non-ASCII keystrokes from the optional keyboard attachment. The six LED's and the system audible alarm is also controlled by this driver through FUNCTION calls. This driver is sharable and supports Byte I/O and asynchronous requests but does not support asynchronous events or Attach or Detach Device. Detach driver is supported, however this will also detach the #CON driver since they share resources and cannot exist separately.

The keypad driver is essentially a "TABLE LOOKUP FACILITY" where the user can associate any desired ASCII string to a keystroke. Each key actuation produces a one byte scancode which is read and decoded by the keypad driver. The scancode is used as an argument for one of four scancode lookup tables. Each lookup table associates scancodes with ASCII text strings. The keypad driver searches the appropriate table for the scancode and places the associated ASCII string in either the #KPD or #CON internal device buffer. It is from this internal buffer that you will receive data when you issue a SYSIO-SREAD, AREAD, or BREAD. This scheme allows the keypad, softkeys, and keyboard function/cursor keys to be totally programmable by the user.

The user must define a table entry for each key to be used. If a key does not appear in a scancode lookup table then that key is essentially "dead". At system startup time the tables are empty, so all keys are "dead".

The MENU program, which is shipped with the CS 9000 in both source and binary form, is an example of assigning ASCII strings to the ten keyboard function keys. If MENU is run (usually with an AUTOEXEC at system start-up) then the function keys are not dead, but instead have been made useful for command line input. You have the option of redefining these keys or any other keys in any way you desire.

A table entry is as follows:

SCANCODE	1 byte
TABLE NUMBER	1 byte
BUFFER CODE	1 byte
ASCII STRING	from 1 to 20 bytes
DELIMITER	1 BYTE

You use FUNCTION 19 or 12 to establish lookup table entries.

The SCANCODE can be $85 - $BD for the SHIFTED and NON-SHIFTED tables.
$BE - $C7 for the SOFTKEY table. (see Figure 3-1)
$01 - $84 for the FUNCTION KEY (keyboard) table.

The TABLE NUMBER is 0 for the NON-SHIFTED table
1 for the SHIFTED table
2 for the FUNCTION KEY table
3 for the SOFT KEY table.

The BUFFER CODE IS 0 for Send to last buffer
1 for Send to #KPD internal buffer
2 for Send to #CON internal buffer

The DELIMITER is either $04 or $0D. You use $0D if you want the text to go into the buffer with a carriage return ($0D) as the last character. You use $04 if text is to go into the internal buffer without an ending carriage return. The $04 is not copied into the internal buffer. This allows you to combine a series of keystrokes together, with only certain of the keystrokes causing termination of the entry.

The flowchart in Figure 3-2 shows the processing which takes place for the keypad shifted and non-shifted tables. One of the keypad keys can be designated a "shift" key, allowing almost a doubling of the number of strings that can be generated from user keystrokes. You are responsible for making a table entry for each possibility. You can specify the scancode of the key which is to be the "shift" key. The default scancode for "shift" is $AB. When the shift key is pressed the shifted LED (under key $AB... not programmable) is lit. The next keystroke is taken as "shifted", and then the LED goes out.

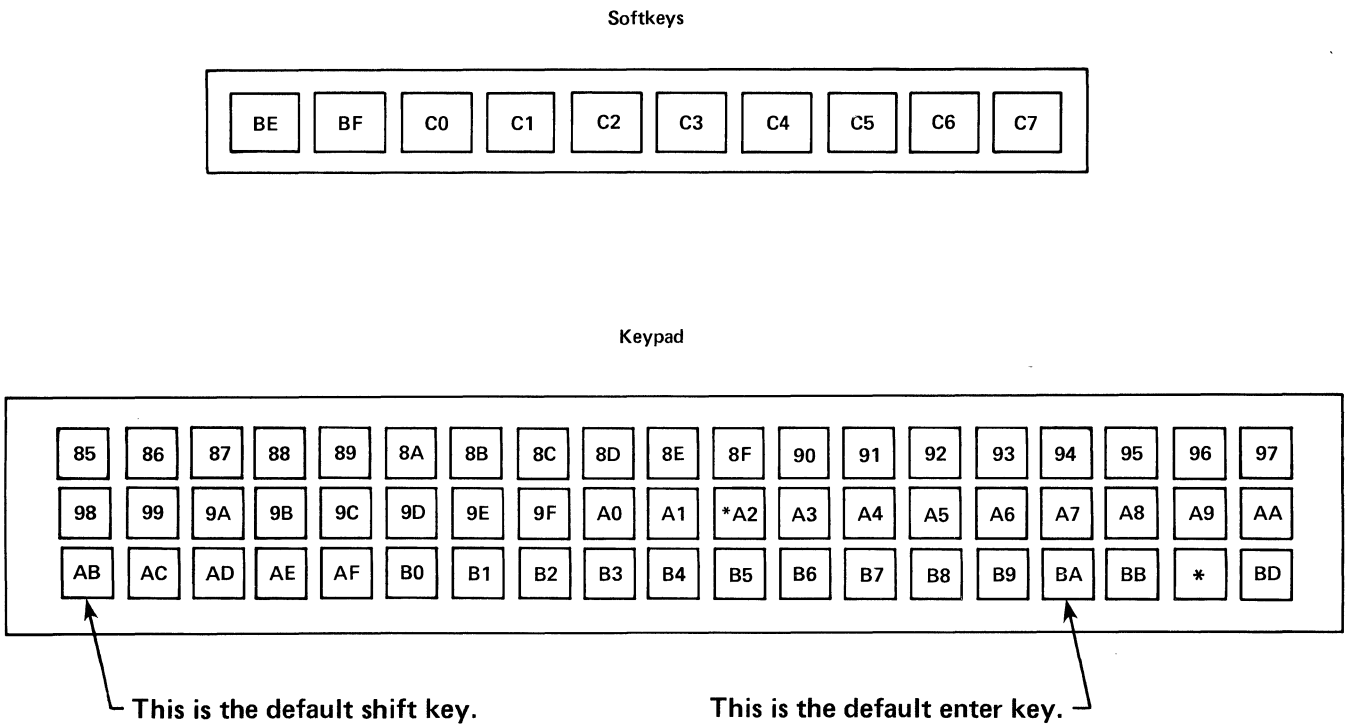
The flowchart in Figure 3-3 shows the processing involved in determining which buffer (#KPD or #CON) the ASCII string should be sent to. The BUFFER CODE is used with the BUFFER POINTER to make this selection.

BUFFER CODE	BUFFER POINTER	ASCII STRING	NEW BUFFER POINTER
0	0	to #CON buffer	unchanged 0
0	1	to #KPD buffer	unchanged 1
0	2	to #CON buffer	unchanged 2
1	Don't Care	to #KPD buffer	1
2	Don't Care	to #CON buffer	2

The keypad keys may be programmed to send ASCII characters to either the #CON buffer, the #KPD buffer, or else whichever buffer was last selected. This is done by the BUFFER CODE table entry. The keypad driver remembers the buffer which you last sent characters to in a variable called BUFFER POINTER.

A recommended use of BUFFER CODE 0 is in the definition of the numeral keys since any numeric input will most likely be intended for the buffer that was last used.

When the "enter" key is pressed, a carriage return ($0D) is sent to the last buffer (default #CON). You may specify which key is to be the "enter" key with FUNCTION 21, or use the default, which is $BA.



*** This key generates scancode A2, the same as the key located in row 2, column 11. Software cannot distinguish these keys except possibly in the context of the application.**

Figure 3-1. Keypad and softkey hex scan-codes.

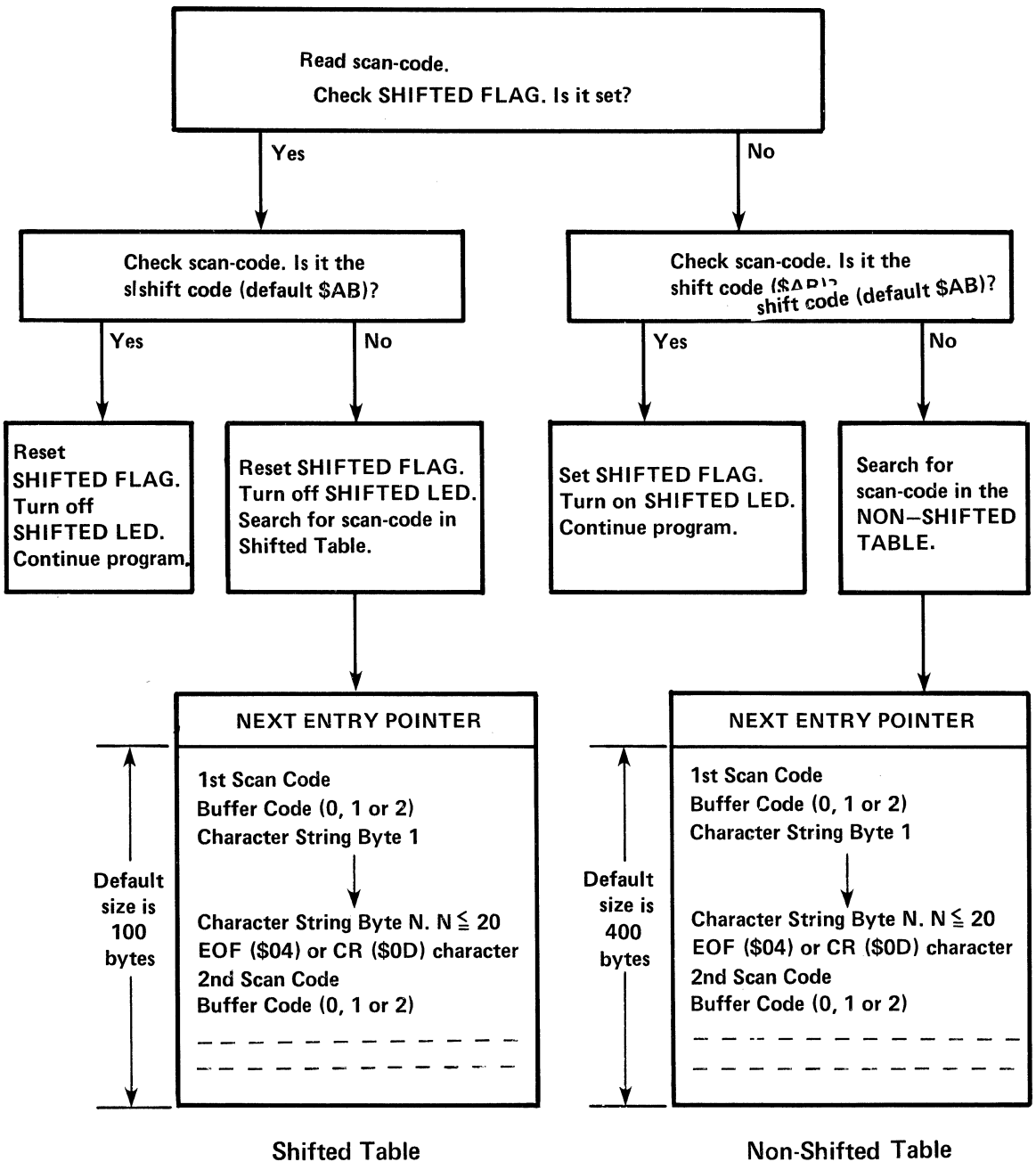


Figure 3-2. Keystroke processing flowchart and table format.

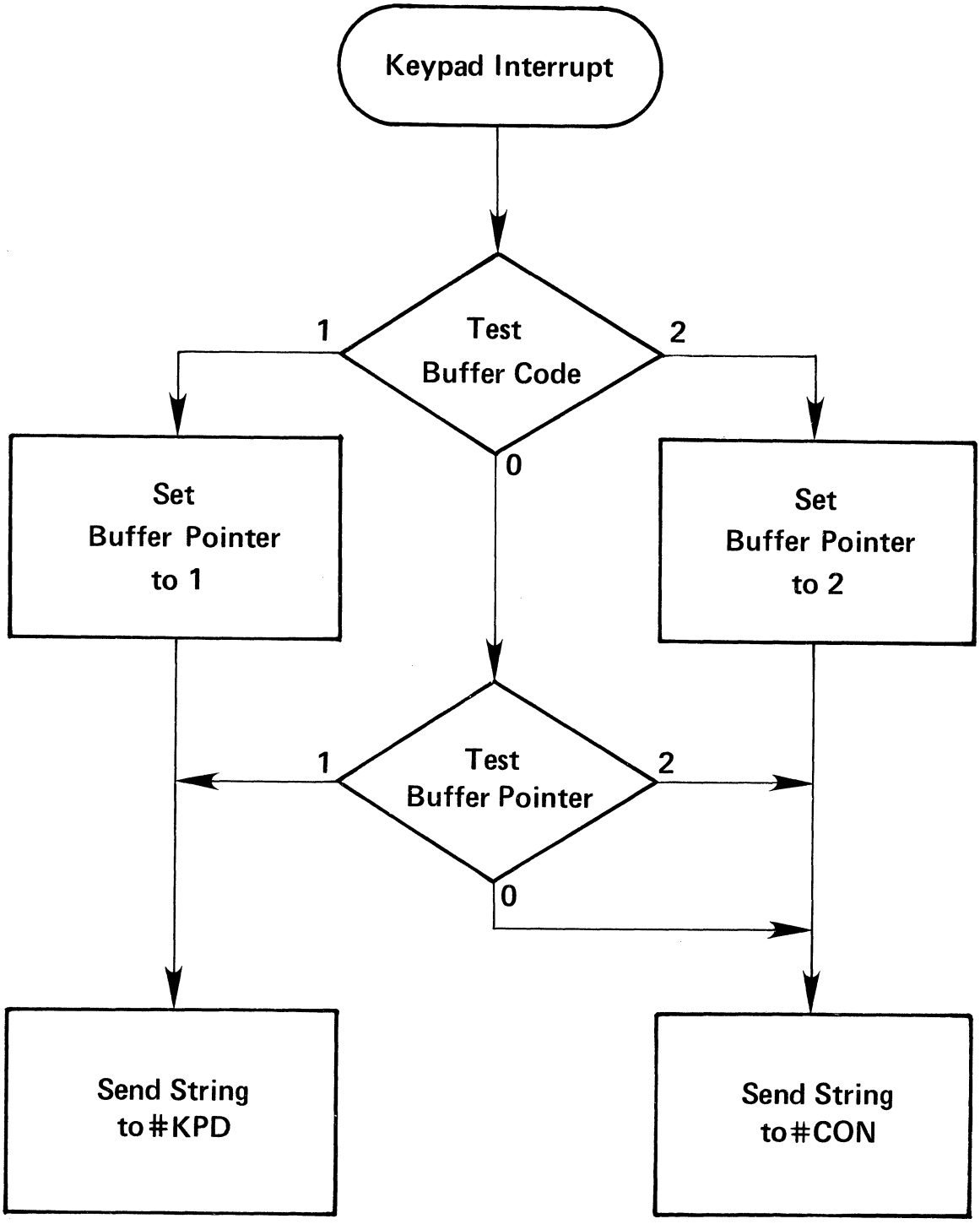


Figure 3-3. Buffer code processing flowchart.

3.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at open time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the keypad the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device.

3.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION OF USE
DIBVOL	DS.B	6	Device name. Use #KPD for keypad driver.
DIBDTD	DS.B	1	Data transfer direction. Use 1. This driver is read only.
DIBTRN	DS.B	1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRSO	DC.L	0,0	User sets this field to 0.
DIBOPT	DC.W	0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L	1	Insert pointer to function packet or null for default.
DIBBIO	DS.L	1	System used Byte I/O Field. To open the keypad for Byte I/O enter -1 ($FFFF FFFF), otherwise set it to zero. After OPEN the I/O manager fills this field with an identifier which is used for SYSIO-BREAD and SYSIO-BTEST.

3.3 KEYPAD DATA TRANSFER CONTROL BLOCK (DTCB)

The data transfer control block (DTCB) holds I/O status and buffer information during READS. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

3.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts lower limit to be used for Transfer Termination characters in Variable length transfers here.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ.

3.4 KEYPAD FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like turning a LED or the system audible alarm ON or OFF. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is following by zero or more words or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

3.4.1 SUMMARY OF THE KEYPAD FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	$0021	WORD (0 or 1)
BEEPER ON	2 ($0002)	NONE	NONE
BEEPER OFF	3 ($0003)	NONE	NONE
TIMED BEEP	4 ($0004)	$0024	WORD (1 thru 255)
SINGLE BEEP	5 ($0005)	NONE	NONE
TURN ON LED	6 ($0006)	NONE	WORD (1 thru 6)
TURN OFF LED	7 ($0007)	NONE	WORD (1 thru 6)
TURN ALL LEDS OFF	8 ($0008)	NONE	NONE
TURN ALL LEDS ON	9 ($0009)	NONE	NONE
TURN KEYPAD ON	10 ($000A)	NONE	NONE
TURN KEYPAD OFF	11 ($000B)	NONE	NONE
ADD TO TABLE	12 ($000C)	SEE 4.5	SEVERAL WORDS
DELETE FROM TABLE	13 ($000D)	NONE	TWO WORDS
CLEAR SELECTED TABLE	14 ($000E)	$002E	WORD
SET BUFFER CODE	15 ($000F)	$002F	WORD (0,1,or2)
GET BUFFER CODE	16 ($0010)	NONE	WORD RETURNED
ALLOCATE TABLES	17 ($0011)	SEE BELOW	THREE WORDS
GET TRANSFER MODE	18 ($0012)	NONE	WORD RETURNED
ADD TO TABLE	19 ($0013)	SEE BELOW	THREE WORDS PLUS

			LONGWORD
DELETE TABLE ENTRY	20 ($0014)	SEE BELOW	TWO WORDS
SET ENTER/SHIFT KEYS	21 ($0015)	$0035	WORD
GET ENTER/SHIFT KEYS	22 ($0016)	NONE	WORD RETURNED
DELETE TABLE ENTRY	23 ($0017)	SEE BELOW	TWO WORDS

3.4.2 KEYPAD FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0		
	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error code	None
1	SET TRANSFER MODE	Allow the user to specify either fixed length or variable length transfers. When fixed length transfers are specified the driver refers to the buffer length specified in the Data Transfer Control Block (DTCB) to determine the amount of the data to transfer. When variable length transfers are specified, the driver will check each byte to see if it lies within the range specified in the Data Transfer Control Block for termination characters and will terminate transfer of data if it does.
	Function Data	Word, integer
	Data = $0000	Selects fixed length transfer.
	$0001	Selects variable length transfer.
	Error Code	$0021, mode not 0 or 1

2	BEEPER ON	Turns the system audible alarm on continuously.
	Function Data	None
	Error Code	None
3	BEEPER OFF	Turn the system audible alarm off.
	Function Data	None
	Error Code	None
4	TIMED BEEP	Turn beeper on for specified period, then turn off. User specifies period in units of 50 milliseconds. Maximum beep is 255 periods or about 12.75 seconds, minimum is 1 period.
	Function Data	Word
	Error code	$0024 Data out of limits.
5	SINGLE BEEP	Turns the system audible alarm on for a period of about 20 milliseconds.
	Function Data	None
	Error Code	None
6	TURN ON LED __	Turn on the LED specified by the data.
	Function Data	One word, Integer

	Data = $0001	Turns on the top left LED.
	$0002	Turns on the middle left LED.
	$0003	Turns on the bottom left LED.
	$0004	Turns on the bottom right LED.
	$0005	Turns on the middle right LED.
	$0006	Turns on the top right LED.
	Error Code	$0026 Data out of limits.

7	TURN OFF LED __	Turns off the LED specified by the data.
	Function Data	Word, integer.
	Data = $0001	Turns the top left LED off.
	$0002	Turns the middle left LED off.
	$0003	Turns the bottom left LED off.
	$0004	Turns the bottom right LED off.
	$0005	Turns the middle right LED off.
	$0006	Turns the top right LED off.
	Error Code	$0027 Data out of limits.

8	TURN ALL LEDs OFF	Turns all LEDs off.
	Function Data	None
	Error Code	None

9	TURN ALL LEDs ON	Turns on all LEDs.
	Function Data	None
	Error Code	None

10	TURN KEYPAD ON	Allow the keypad to interrupt and be READ by the system.
	Function Data	None
	Error Code	$0027

11	TURN KEYPAD OFF	Disable keypad interrupt to the system.
	Function Data	None
	Error Code	None

NOTE: This function can also be performed by Function 19. You should use Function 19 if you are coding in PASCAL or ASSEMBLER, and you should use Function 12 if you are coding in BASIC or FORTRAN.

12	ADD TO TABLE	Allow the user to define the use and interpretation of the keypad, softkeys and function keys. See paragraph 3.1 and figure 3-1 for a better understanding of these tables.
	Function Data	Consists of from 6 to 25 words structured as follows.

Argument	Length	Definition
1	WORD	Scancode of key to be added to a table.
2	WORD	Table which this scancode should be added to 0 = keypad non-shifted table 1 = keypad shifted table 2 = console function key table 3 = soft key table

3	WORD	Buffer Code 0 = put data in last used buffer
		1 = put data in #KPD buffer
		2 = put data in #CON buffer

4	WORD	String length in words

5-25	WORD(S)	String to be inserted. Each word contains one character of the string in the low order Byte. Up to 20 words (therefore 20 characters can be specified).
Last	WORD	String Terminator. The low order Byte of this word contains the terminating Byte, either $0D or $04.
Error Codes		$0011 Scancode already in table $0012 Not enough room to add this entry $0015 Invalid table number $0016 Illogical scancode for table type $0017 String length too long, greater than 20 $0018 Bad terminator, not $04 or $0D

NOTE: This function has been superseded by Functions 20 and 23. It is included here for compatibility with Release 1.0 programs. You should not use it in new programs.

13	DELETE ENTRY FROM TABLE	Allows the user to delete an entry from one of the key tables. The data specifies which table and which entry (Scancode) is to be deleted.
	Function Data	Two Words, integer
	Word 1 = $00NN	NN = Scancode - see Figure 3-2.
	Word 2 = $000N	Specifies which table to delete from. N=0 Keypad non-shifted table. N=1 Keypad shifted table. N=2 Function key table N=3 Soft-key table
	Word 3 = $00NN	Ignored.
	Error Codes	$0015 Table number invalid $0014 Scancode not in table $0013 Table organization error

14	CLEAR SELECTED TABLE	Clears the table indicated by data.
	Function Data	One word, integer
	Data = $0001	Clears the keypad shifted table.
	$0002	Clears the keypad non-shifted table.
	$0003	Clears the keyboard function key table
	$0004	Clears all tables.
	Error Code	$002E Table type invalid

15	SET BUFFER POINTER	This allows the application program to set the status of the buffer pointer (see buffer code processing flowchart). The buffer pointer controls the logic which determines where Keystrokes are to be placed.
	Function Data	Word, integer (0, 1, or 2)
	Data = $0000	Make buffer pointer zero = unspecified
	$0001	Place data in #KPD buffer
	$0002	Place data in #CON buffer
	Error Code	$002F Invalid buffer pointer
		The buffer pointer is set to zero at cold start.

16	GET BUFFER POINTER	This allows the application program to get the current buffer pointer value.
	Function Data	Word returned
	Error code	None

---

17 ALLOCATE KEYPAD TABLES

This function allows the user to redefine the amount of space for Keypad lookup tables. The tables must be cleared before issuing this function.

Function Data: Three Words (maximum of 32767 bytes)

Word 1 = Shift Table Size — default is 100 bytes at cold start

Word 2 = Unshifted Table Size — default is 400 bytes at cold start

Word 3 = Function Key Table Size — default is 200 bytes at cold start

Error Code

$001D	Error returning memory
$001E	Tables not empty, can't reallocate
$001F	Not enough memory for tables
$0031	Negative table size

---

18 GET TRANSFER MODE

Return transfer mode to the user. 0 = fixed, 1 = variable

Function Data: Word returned

Error Code: None

---

19 ADD TABLE ENTRY

Allows the user to add strings to the Keypad lookup tables. Supersedes Function 12 which was used in Release 1.0.

Function Data: As follows:

Argument	Length	Definition
1	WORD	Scancode of key to be added to table
2	WORD	Table which the scancode should be added to: 0 = non-shifted keypad table 1 = shifted keypad table

		2 = keyboard function key table 3 = soft key table

3	WORD	Buffer code 0 = put data in last used buffe
		1 = put data in #KPD buffer
		2 = put data in #CON buffer

4	LONGWORD	Address of ASCII string to be added. Strin may be up to 21 characters long and must include a terminating character of $0D or $04. Note: This address is not checked by the driver. A DTAK take error will occur if the address supplied is invalid.
	Error Codes	$0011 Scancode already in table $0012 Not enough memory to add this entry $0015 Invalid table number $0016 Illogical scancode for table type $0019 Illogical buffer code for table type $0033 String too long or bad terminator

20	DELETE TABLE ENTRY	Allows the user to delete strings from the keypad lookup tables. Supersedes Function 13 which was used in Release 1.0.
	Function Data	As follows:

Argument	Length	Definition
1	WORD	Scancode of key to be deleted
2	WORD	Table from which the scancode is to be deleted. 0 = non-shifted keypad table 1 = shifted keypad table 2 = keyboard function/cursor table 3 = soft key table
	Error Codes	$0013 Table organization error $0014 Scancode not found in table

		$0015 Invalid table number

See Function 23 which is similar but which ignores scancode not in table.

21	SET ENTER/SHIFT KEY SCANCODES	Allows the user to define the keys to be used for ENTER and SHIFT. Any scancode between $85 and $BD may be defined for ENTER or SHIFT. The default is $AB for SHIFT and $BA for ENTER.
	Function Data	WORD Byte 0 (Low order) is shift key Byte 1 (High order) is enter key
	Error Code	$0035 Invalid Enter/Shift key. Must be in the range $85 to $BD.

22	GET ENTER/SHIFT KEY SCANCODES	Retrieve current definitions for these keys.
	Function Data	WORD returned.
	Error Code	None.

23	DELETE TABLE ENTRY	Same as Function 20, except no error code is generated if scancode is not found in the table.

4.0 CRT ALPHANUMERIC DISPLAY DRIVER

4.1 DRIVER DESCRIPTION

The CRT Display Driver is a tool with which custom Alphanumeric displays may be created. The Alpha Display Driver allows the user to manipulate data within a user defined display area. This display area is called an "ALPHA WINDOW". See Figures 4-1. A separate "CONSOLE BOX" of 3 Character Lines by 80 Character Blocks is provided for system messages. Although this CONSOLE BOX cannot be disabled it can be used to display other than system messages by using the PRTERMSG Macro.

The CRT Display has two independent pages of refresh memory associated with it. Only one page may be displayed at a given time. The displayed page may be switched using the SETCRTCR system call. There are three alphanumeric device names #SCRN0, #SCRN1, and #CNSL0. #SCRN0 and #SCRN1 are associated with page 0 and 1 respectively, while #CNSL0 is a reserved console display area consisting of three lines at the bottom of page 0.

Each of the 2000 (25x80) Character Blocks in the ALPHA WINDOW is addressable by specifying line and column numbers. To facilitate ease of programming, line and column numbers begin with zero. The DISPLAY BUFFER has two pointers associated with it. The cursor pointer determines the location of the cursor. This pointer may be manipulated in a variety of ways. The window pointer points to a character block within a window. It may also be manipulated in a variety of ways, and is used when the cursor is disabled. The normal user area may be divided into up to five separate display areas or WINDOWS. Each WINDOW has its own CURSOR and WINDOW POINTER and is assigned a unique Logical Unit Number when OPENed. Opening #CNSL0 counts as a window. The window's position and dimensions can be redefined without closing and reopening that logical unit. See Figure 4-2.

Each window has a one character block border around the perimeter which is not included in the dimensions specified by the user. This should be taken into account in designing screen usage. This border may be filled with any fill pattern. See Figure 4-3. Overlapping windows are allowed but the user must provide space management to avoid overwrite. If the user does not define a window the system defaults to the entire 25 x 80 character block user area.

Each window may be framed at any time at the option of the user. The top and bottom of the frame are formed by XORing pixels in the fourth scan line above and below the window, while the sides of the frame are formed

by XORing the pixels three pixel columns to the right and left of the window. These scan lines and pixels are part of the character border. Each character block is 9 pixels wide by 16 scan lines deep. The entire ALPHA WINDOW of the Display buffer is 720 pixels wide (80 character blocks each 9 pixels wide) by 400 scan lines deep (25 character lines, each of which is 16 scan lines deep). See figure 4-1. The contents of the DISPLAY BUFFER are displayed on the CRT.

A second buffer, the CHARACTER BUFFER, is used to store the ASCII data representation of the ALPHA WINDOW for later transfer to the DISPLAY BUFFER. THE CHARACTER BUFFER is organized to have a one to one correspondence with the DISPLAY BUFFER and has 2000 words arranged in 25 lines of 80 characters. Each word contains the character code in the Least Significant Byte and a 4 bit attribute code in the least significant nibble of the Most Significant Byte. See Figure 4-2. The most significant nibble of this byte is not currently used. The contents of this buffer may be "dumped" to the DISPLAY BUFFER for display on the screen. When this is done the attribute nibble determines the characteristics of the character such as inverse video or underlining. Attributes are discussed in more detail in paragraph 4.2.2 and 4.2.3.

The ASCII character code is used as a pointer to a location in a FONT TABLE during dumps to the DISPLAY BUFFER. The user may specify use of the SYSTEM FONT TABLE (Figure 4-4) or a USER created FONT TABLE. If a USER FONT TABLE is specified, the user must specify the beginning address with the FONT SELECT function. The font table entry at the location pointed to by the ASCII character code will be used by the driver, along with the attribute bits, to determine the bit makeup of the character block in the DISPLAY BUFFER. See Figure 4-5 for a pictorial relationship between the buffers and the font table. Each character is represented by 14 bytes in the font table. Each byte represents the rightmost 8 pixels on a scan line. Only the middle 14 scan lines of each character block are used. The 8 x 14 Font is displayed in a 9 x 16 field. To create a USER FONT TABLE, reserve 3584 ($0E00) bytes of memory and specify the beginning address using the FONT SELECT function. Draw the character in a 8 x 14 grid and determine the bit pattern needed to create each scan line. The bit pattern for the first scan line will become the first of the 14 bytes. Continue until all 14 bytes have been determined. These 14 bytes now become an entry in the user font table. When the 256 entries have been created they may be entered in any desired order in the font table. The first entry will be pointed to by ASCII code $00 in the character buffer -- the last entry by ASCII code $FF. See Figure 4-6.

The Character Buffer is treated as a circular buffer of 25 lines of 80 characters each or as up to 5 window buffers the size of which and location of which corresponds to the window locations specified by the user. Each window has associated with it a TOP LINE POINTER which determines which line will be displayed as the top line in the Display Buffer window when the Character Buffer is dumped to the Display Buffer. This pointer may be set or read. Each window also has a character pointer which may be set or read. This pointer determines where the next character in the window is placed. By manipulating these pointers the window may be scrolled or paged.

In addition to transfers of the Character Buffer contents to the Display Buffer the user may specify dumps from specified buffers in application memory. The manner in which these transfers are handled is determined by the attribute decoding selected. If attribute decoding is not selected each byte transferred is treated as a separate character code. If attribute decoding is specified transfers are treated as full words, the least significant byte must contain the character code and the low order four bits of the most significant byte must be set to the desired attribute code specified in paragraph 4.2.

Notes: Asynchronous write operations are not supported. The CRT driver will treat an AWRITE as an SWRITE.

CRT driver functions relating to the character buffer are subject to change or elimination (see Function Summary 4.4.1) in order to implement future improvements.

For most drivers, Byte I/O transmits a byte in register D0.B and does not use the remainder of register D0 for any purpose. The CRT driver differs from this general standard. It uses byte 0 of register D0 to transmit the ASCII character, just as other drivers do; however, it also uses byte 1 of register D0 to transmit attribute information. See Section 4.2.2 (Character Attributes) and Section 4.2.3 (Attribute Codes).

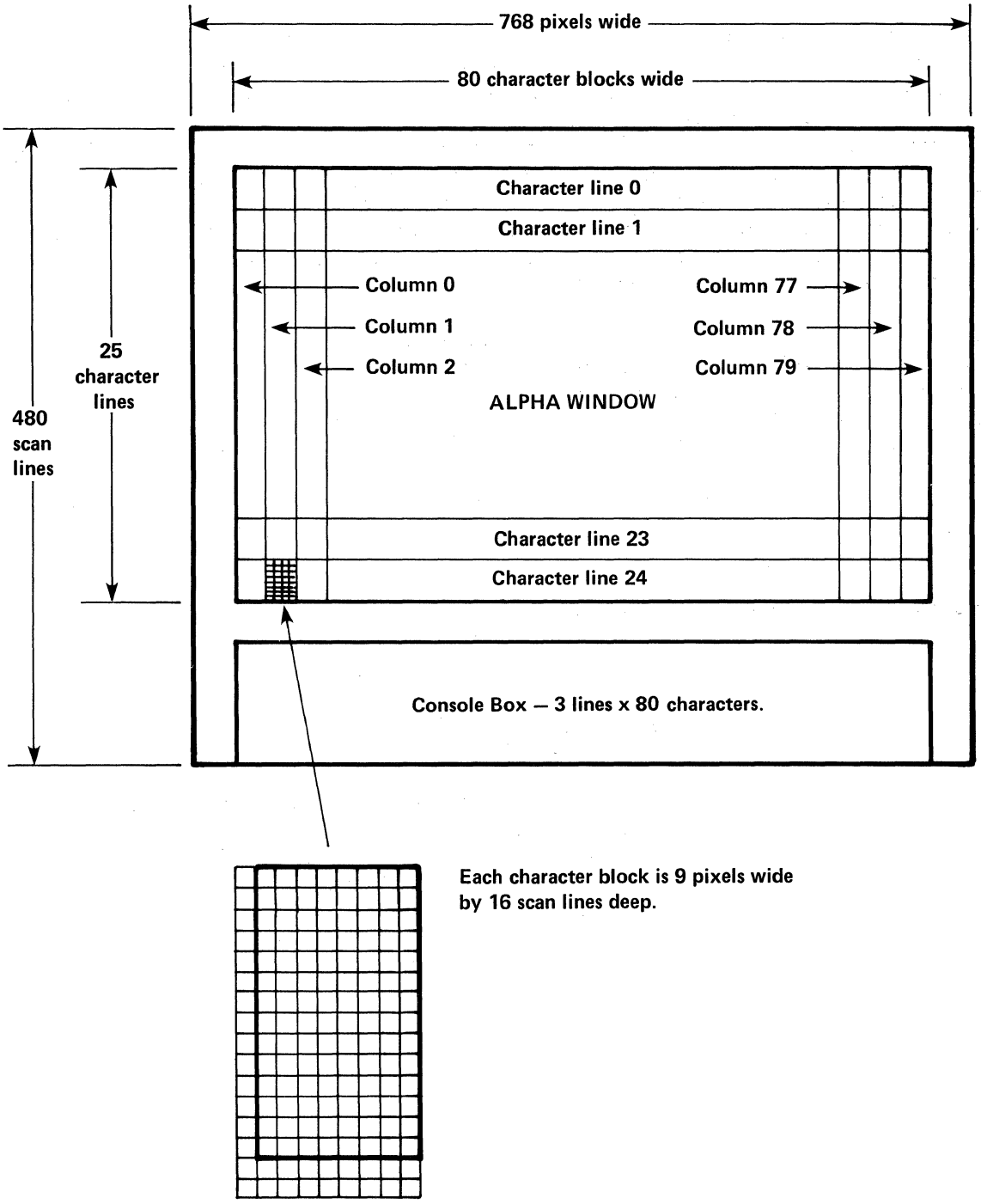


Figure 4-1. CRT Display

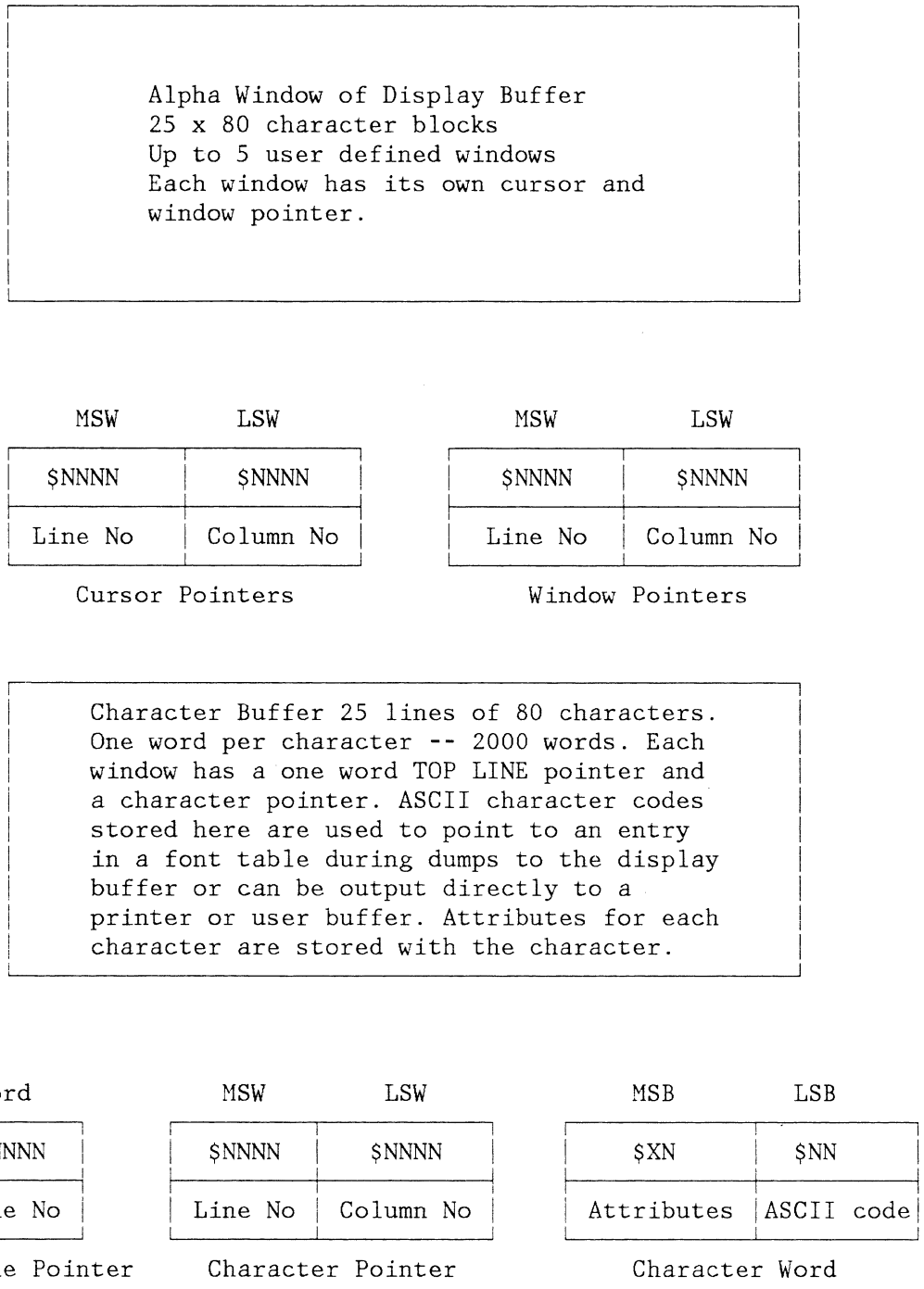


Figure 4-2. Display and Character Buffers

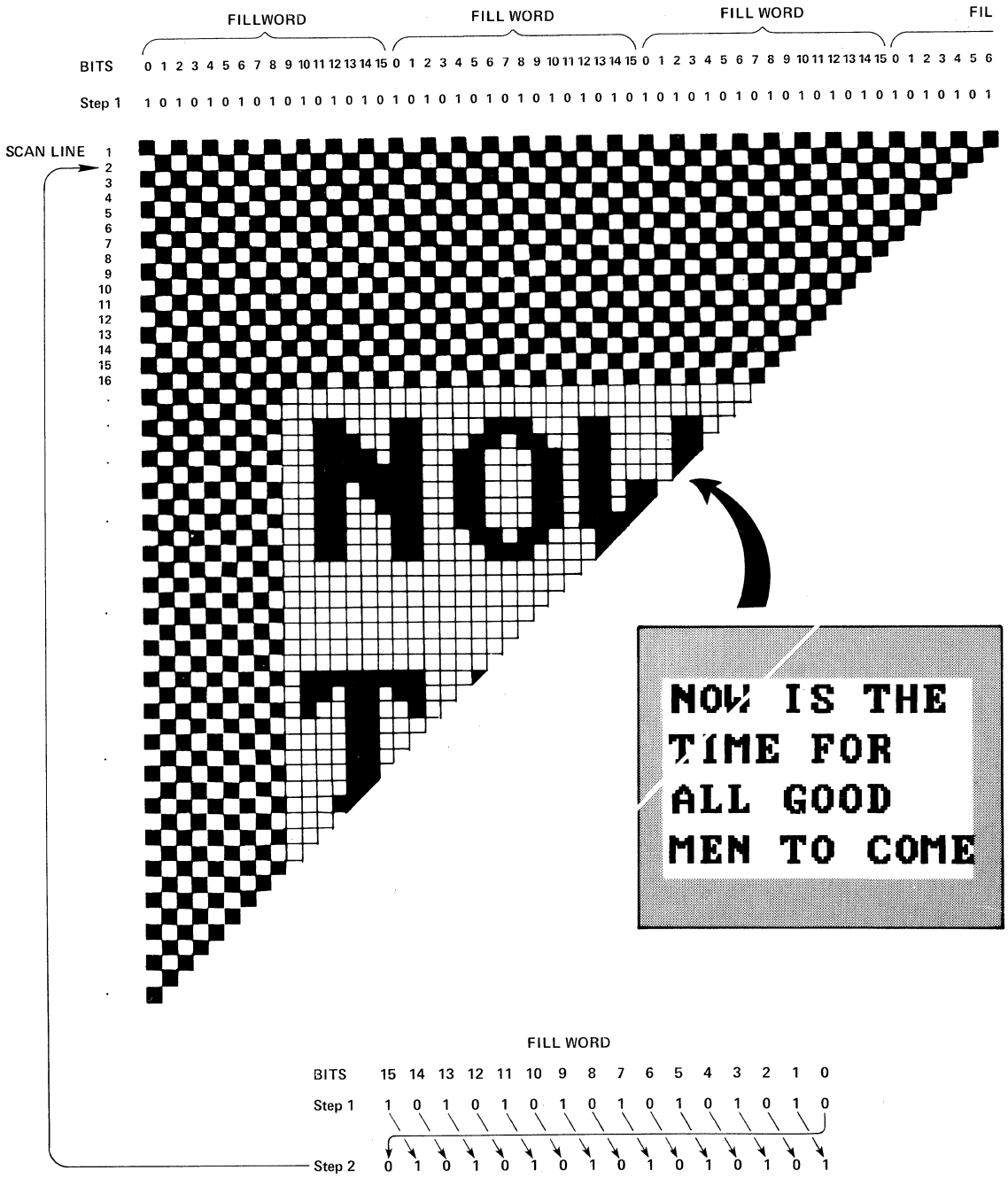


Figure 4-3. Fill Border Function

Figure 4-4. System Font Table

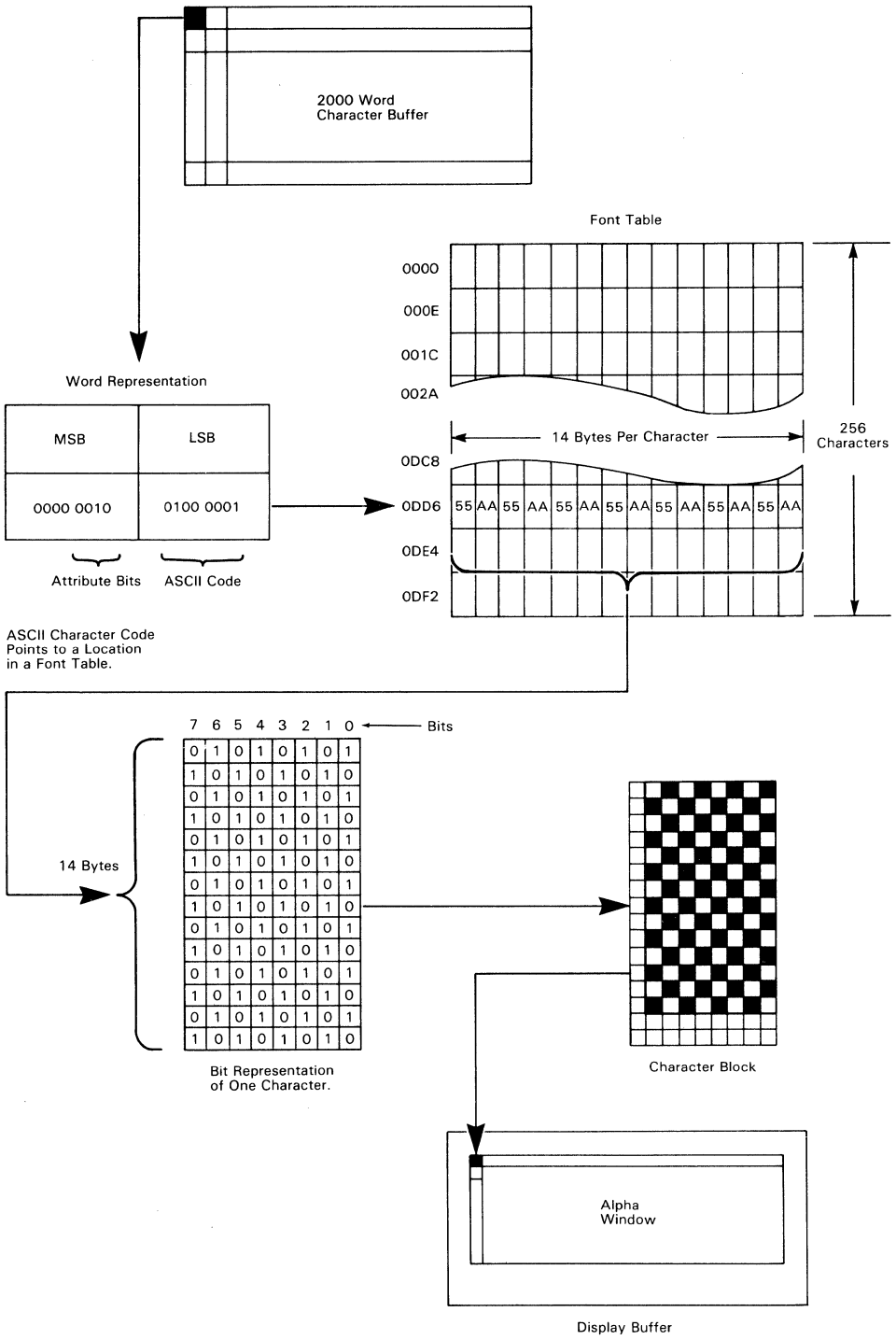


Figure 4-5. Buffer and Font Table Relationship

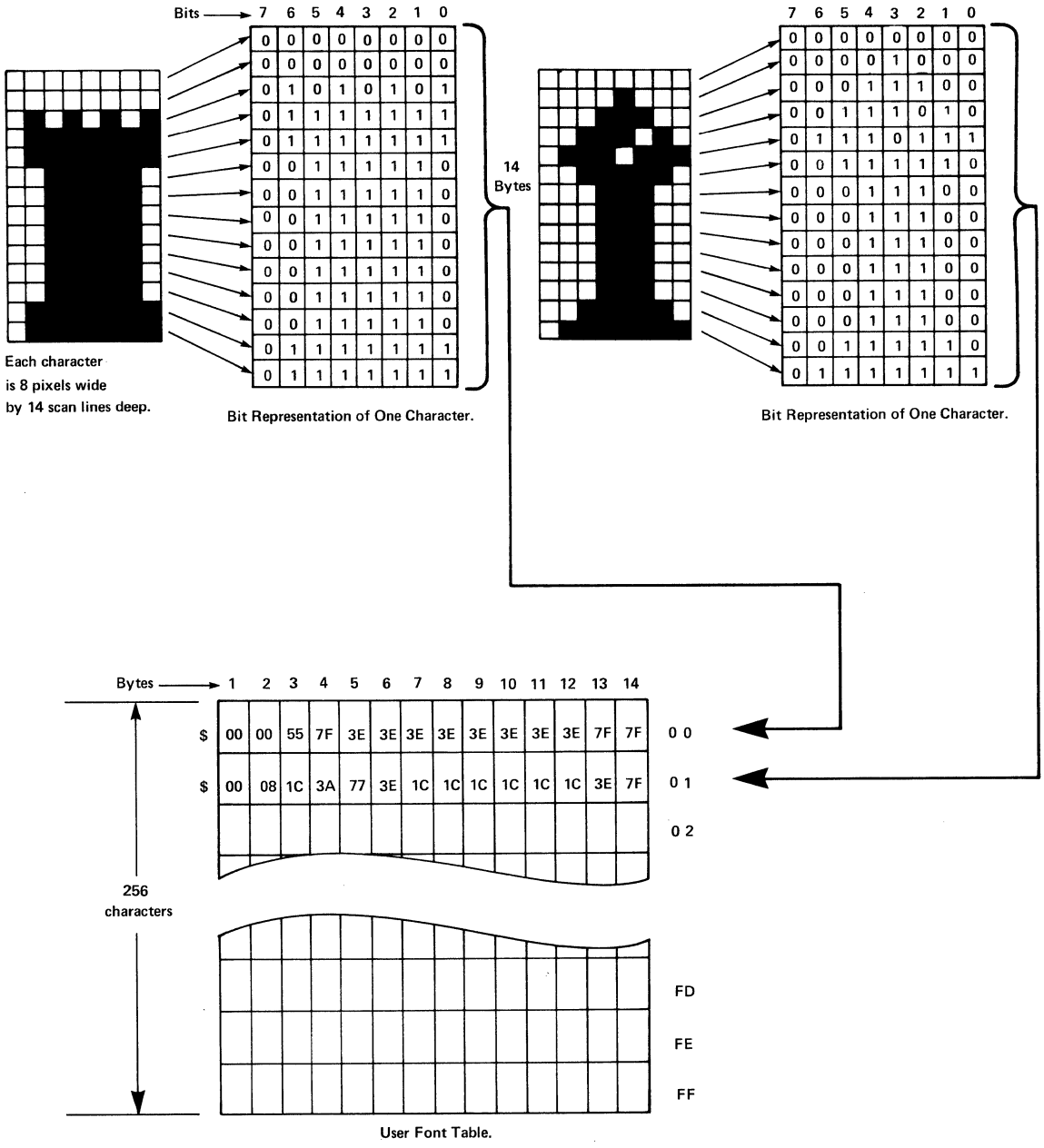


Figure 4-6. User Font Table

4.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the CRT the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to one of the devices using the Device Name specified under DIBVOL. When this is done all standard SYSIO operations described in Chapter 1 are allowed.

Characteristics of the display area or window can be specified at OPEN time.

4.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #SCRN0, #SCRN1, or #CNSL0,
DIBDTD	DS.B 1	Data Transfer Direction. Enter 0 for WRITE.
DIBTRN	DS.B 1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRS0	DC.L 0,0	Reserved space. User sets this field to 0.
DIBOPT	DC.W 0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L 1	Insert pointer to function packet; null for default.
DIBBIO	DS.L 1	System used Byte I/O Field. Byte I/O to the display buffer is really a word at a time. D0.W contains the attribute byte if attribute decoding is enabled. To open the CRT for Byte I/O enter -1 ($FFFF FFFF), otherwise set it to zero. After OPEN the I/O manager fills this field with an identifier which is used for SYSIO-BWRITE.

4.2.2 CHARACTER ATTRIBUTES

Attributes which modify the display characteristics of individual characters may be specified in an attribute byte which is stored in the most significant byte of the word in the Character Buffer. Characters with attributes are written as a word. The low byte is the ASCII value and the high byte is the attribute byte.

4.2.3 ATTRIBUTE CODE

ATTRIBUTE BYTE

Bit No	Value		Attribute
7	X)		
6	X)	-	These bits not used at present. X = 0 or 1.
5	X)		
4	X)		
3	0		Display this character.
3	1		Character blanking (Do not display this character).
2	0		Regular video.
2	1		Inverse video.
1	0		Do not underline this character.
1	1		Underline this character.
0	0		Use the system font table to display this character.
0	1		Use the user font table to display this character. (Note the FONT SELECT function must also be used in order to set the pointer to the user font table).

4.3 CRT DATA TRANSFER CONTROL BLOCK (DTCB).

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

4.3.1 DTCB FORMAT

DATA MNEMONIC	LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts lower limit to be used for Transfer Termination characters in Variable length transfers here.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ/WRITE.

4.4 CRT FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like reading the cursor position and selecting a font table. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more bytes, words or longwords that send or receive the immediate DATA for the command, or a long word that points to the DATA for that COMMAND.

4.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

Note: An asterisk (*) indicates that a function is subject to change or elimination in future releases.

	FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
	ENDLIST	0 ($0000)	NONE	NONE
	SET TRANSFER MODE	1 ($0001)	$0021	WORD
	FONT SELECT	2 ($0002)	$0022	LONG WORD POINTER
	FLOOD WINDOW	3 ($0003)	$0023	WORD
	FLOOD LINE	5 ($0005)	$0025	WORD
	CLEAR PAGE	6 ($0006)	$0026	WORD
	GET CURSOR	13 ($000D)	NONE	LONG WORD RETURNED
	WRITE CURSOR	14)$000E)	$002E	LONG WORD POINTER
*	GET POINTER	15 ($000F)	NONE	LONG WORD RETURNED
*	SET POINTER	16 ($0010)	$0030	LONG WORD POINTER
	CURSOR FONT SELECT	17 ($0011)	$0031	WORD
*	DUMP CHARACTER BUFFER	18 ($0012)	NONE	NONE
	SCROLL/PAGE SELECT	19 ($0013)	$0033	WORD
*	ATTRIBUTE DECODING	21 ($0015)	$0035	WORD

*	FRAME ENABLE/DISABLE	23	($0017)	NONE	NONE
*	TARGET BUFFER SELECT	24	($0018)	$0038	WORD
*	SET TOP LINE	25	($0019)	$0039	WORD
*	GET TOP LINE	26	($001A)	NONE	ONE WORD RETURNED
	SCROLL UP N LINES	27	($001B)	$003B	WORD
	SCROLL DOWN N LINES	28	($001C)	$003C	WORD
	TRUNCATE LINES SELECT	29	($001D)	$003D	WORD
	CONTROL CHARACTER FILTER	31	($001F)	$003F	WORD
*	CURSOR UPDATE MODE	32	($0020)	$0040	WORD
*	CHARACTER OVERWRITE MODE	34	($0022)	$0042	WORD
	ERASE CURSOR	35	($0023)	NONE	NONE
*	SPECIFY FILL WORD	36	($0024)	$0044	WORD
*	FILL BORDER	38	($0026)	$0046	WORD
	AUTO LINE FEED SELECT	39	($0027)	$0047	WORD
	MOVE CURSOR	40	($0028)	$0048	LONG WORD POINTER
	SET WINDOW	41	($0029)	$0049	FOUR WORDS
*	SET CHARACTER BUFFER ADDRESS	42	($002A)	$004A	LONG WORD ADDRESS
*	SET CHARACTER BUFFER POINTER	43	($002B)	$004B	LONG WORD POINTER
*	GET CHARACTER BUFFER POINTER	44	($002C)	NONE	LONG WORD RETURNED
*	CLEAR LINE IN CHARACTER BUFFER	45	($002D)	NONE	NONE
*	CLEAR CHARACTER BUFFER WINDOW	46	($002E)	NONE	NONE

4.4.2 ALPHA WINDOW MANAGER FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None.
	Error Code	None.
1	SET TRANSFER MODE	Activates terminal character checking to delimit a record that is being transferred.
	Function Data	One word, integer
	Data = $0000	Selects fixed length transfer mode
	$0001	Selects variable length transfer mode
	Error Code $0021	Data out of limits
2	FONT SELECT	Permits selection of the system font table or a user defined table. The user defined font table could contain APL or foreign language character sets.
	Function Data	One long word, integer.
	Data = $0000 0000	System font table is selected
	$NNNN NNNN	User font table is selected and data points to location of font table.
	Error Code $0022	Data out of limits.
	NOTE: The FONT SELECTION has no effect on character	

attributes, but if character attributes are for USER FONT, one must be selected.

3	FLOOD WINDOW	Modifies the interior of the window in accordance with the data word defined below. The fill word is defined by function 36 ($0024). Neither the frame around the window, nor the area outside the window is affected. See Figure 4-3.
	Function Data	One word, integer.
	Data = $0000	Clear window.
	$0001	Fill window with current "fill" word.
	$0002	Exclusive OR "fill" word with window contents.
	$0003	OR "fill" word with window contents
	Error Code = $0023	Data out of limits.

5	FLOOD LINE	Modifies the line from the cursor position to the end of the line in accordance with the data word defined below. The fill word is defined by function 36 ($0024). See Figure 4-3.
	Function Data	One word, integer
	Data = $0000	Clear line
	$0001	Fill line with current "fill" word.
	$0002	EXCLUSIVE OR "fill" word with contents of line.
	$0003	OR "fill" word with contents of line.

	Error Code = $0025	Data out of limits.
6	CLEAR PAGE	This is a global function which clears the entire page of memory of the graphics refresh buffer. Since it affects all windows and character buffers, including those of other tasks it should be used with caution!
	Function Data	One word, integer
	Data = $0000	Clear page 0
	$0001	Clear page 1
	Error Code = $0026	Data out of limits.
13	GET CURSOR	Returns the cursor position. A long word is returned in which the most significant word indicates the line count and the least significant word indicates the column count. Note that line and column numbering begins with zero.
	Function Data	Leave space for one long word, integer.
	Data = $0000 0000	Cursor is in upper left corner of window.
	= $000N 000N	Cursor is on line N and in column N.
	Error Code	None

14	WRITE CURSOR	Places a new cursor at the location specified by the DATA. The most significant word of this long word gives the line on which the cursor will appear and the least significant word indicates the column in which it will appear. Note that line and column numbering begin with zero. The cursor at the previous location is not erased and the cursor erased flag is reset.
	Function Data	One long word, integer
	Data = $000N 000N	Writes a cursor on line N and in column N.
	Error Code = $002E	Data out of limits.
	NOTE: This function does not change the pointer or cursor position - it only draws one.	

15	GET POINTER	This function is used to determine where the next character will be placed when operating in the stationary cursor mode. A long word is returned in which the most significant word indicates the line count and the least significant word indicates the column count. Note that line and column numbering begin with zero. The window pointer is not changed by reads but is incremented by each write to the window.
	Function Data	Leave space for one long word, integer.
	Data = $000N 000N	Pointer points to line N and column N.
	Error Code	None.

16	SET POINTER	Sets the pointer to the location specified by the DATA. (Used only with cursor disabled, otherwise no effect.) The most significant word of this long word specified the line count and the least significant word specifies the column count. Note that line and column numbering begin with zero. Any subsequent write to the window will increment this pointer if stationary cursor mode is selected.
	Function Data	One long word, integer
	Data = $000N 000N	Sets the pointer to line N and column N.
	Error Code = $0030	Data out of limits.
17	CURSOR FONT SELECT	Allows any symbol from either the System Font table or the User Font table to be used as a cursor. Each window may have a unique symbol.
	Function Data	One word, integer character code.
	Data = $003D	ASCII equals character (=) is used for cursor.
	Error Code = $0031	Data out of limits.
18	DUMP CHARACTER BUFFER	Writes the contents of the character buffer into the display buffer. The write begins with the first character of the current top line of the character buffer as pointed to by the TOP LINE pointer. The function command "SET TOP LINE" (function 25, $0019) can be used to change the current top line.

	Function Data	None required.
	Error Code	None.
	Note:	Dumping the character buffer may overwrite the cursor resulting in double cursors later on. Good practice to erase cursor/dump character buffer/ redraw cursor.
19	SCROLL/PAGE SELECT	Allows the user to select the screen update mode. In the scroll mode when the window is full the contents of the window move up one line and the bottom line is cleared. In the page mode the contents of the window do not move. When the window is full the cursor will move back up to the first line.
	Function Data	One word, integer.
	Data = $0000	Scroll mode.
	$0001	Page mode.
	Error Code = $0033	Data out of limits.
21	ATTRIBUTE DECODING	Allows the user to specify attributes such as inverse video for each character in the character buffer. Character attributes are described in paragraph 4.2.3. While in this mode, characters are word length. LSB = ASCII value, MSB = attribute byte.
	Function Data	One word, integer.
	Data = $0000	Character attributes are ignored.
	$0001	Character attributes are decoded and displayed, when the character

		buffer is written to the display buffer.
	Error Code = $0035	Data out of limits.
23	FRAME ENABLE/DISABLE	Allows the user to draw or erase a frame around a window. Subsequent calls reverse the previous state.
	Function Data	None required.
	Error Code	None.
24	TARGET BUFFER SELECT	Allows the user to specify either the display buffer or the character buffer as the "target" of information transfers from the User's Data Transfer Buffer.
	Function Data	One word, integer.
	Data = $0000	Selects display buffer as the target.
	$0001	Selects character buffer as the target.
	Error Code = $0038	Data out of limits.
25	SET TOP LINE	Allows the user to specify which line of the character buffer will be displayed as the top line on the screen when the character buffer is written to the display buffer. In the scroll mode this counter is incremented causing the contents of the screen to move up.
	Function Data	One word, integer
	Data	Sets the top line pointer to line N of the character buffer.
	Error Code = $0039	Data out of limits.

26	GET TOP LINE	Allows the user to determine which line within the character buffer the top line pointer is pointing to.
	Function Data	Leave space for one word, integer.
	Data = $000N	The top line pointer points to line N and line N will be displayed at the top of the window.
	Error Code	None.
27	SCROLL UP N LINES	The screen display is scrolled up the number of lines specified by the data. The top N lines are lost and the bottom N lines are cleared with the current fill word. The fill word is defined by function 36 ($0024). See Figure 4-3.
	Function Data	One word, integer.
	Data = $0005	Moves everything on the screen up 5 lines and fills the bottom 5 lines with the current fill word.
	Error Code = $003B	Data out of limits.
28	SCROLL DOWN N LINES	The screen display is scrolled down the number of lines specified by the data. The bottom N lines are lost and the top N lines are filled with the current fill word. The fill word is defined by function 36, $0024. See Figure 4-3.
	Function Data	One word, integer.
	Data = $000A	Moves everything on the screen down 10 lines and fills the top 10 lines

		with the current fill word.
	Error Code = $003C	Data out of limits.

29	TRUNCATE LINES SELECT	Allows the user to disable or enable line truncation for the display window. When line truncation is enabled any line which is longer than the width of the window will be truncated. If this function is not enabled the line will "overrun" onto the next line until the window is full. See example in Figure 4-5.
	Function Data	One word, integer.
	Data = $0000	Disable line truncation.
	$0001	Enable line truncation.
	Error Code $003D	Data out of limits.

TRUNCATE LINES ENABLED

```
NOW IS THE TIME FOR
THE QUICK BROWN FOX
```

TRUNCATE LINES DISABLED

```
NOW IS THE TIME FOR
ALL GOOD MEN TO COME
TO THE AID OF THEIR
COUNTRY.
THE QUICK BROWN FOX
```

31	CONTROL CHARACTER FILTER	This software filter screens out the ASCII control characters ($0 to $1F) from display. When disabled, it allows for the display of these characters in the font shown in Figure 4-1. Note that if the filter is disabled, carriage returns ($0D) and line feeds ($0A) will not be used for format control but will simply be displayed. When the filter is enabled these characters are used for format control.
	Function Data	One word, integer.
	Data = $0000	Control characters are not displayed.
	$0001	Control characters are displayed.
	Error Code = $003F	Data out of limits.

32	CURSOR UPDATE MODE	Allows the cursor to be enabled or disabled. Note: This function acts upon the cursor associated with a particular window. Note: If cursor is disabled (stationary mode) writes will be performed starting at the pointer (see Function 16).
	Function Data	One word, integer.
	Data = $0000	Enable cursor.
	$0001	Disable cursor.
	Error Code = $0040	Data out of limits.

34	CHARACTER OVERWRITE MODE	Allows the user to specify treatment of existing data in accordance with the data word defined below.
	Function Data	One word, integer.
	Data = $0000	Overwrite existing data.
	$0001	Exclusive OR character with existing data.
	$0002	Logical OR character with existing data.
	$0003	Overwrite in inverse video.
	Error Code = $0042	Data out of limits.
35	ERASE CURSOR	Allows the user to erase the cursor.
	Function Data	None required.
	Error Code	None.
36	SPECIFY FILL WORD	Allows the user to select a "fill word". This word will be used in the FLOOD WINDOW, FLOOD LINE, FILL BORDER, SCROLL UP and SCROLL DOWN functions and therefore should be defined prior to calling any of these function packets. It is initialized to $0000. See Figure 4-3.
	Function data	One word, integer character code.
	Data Example = $AAAA	Define fill word to be $AAAA.
	Error Code = $0044	Data out of limits.

38	FILL BORDER	Uses the current fill word to create a one character wide/high border around the outside of the window. The fill word will be written according to the function argument.
	Function Data	One word, integer.
	Data = $0000	Clear (reset pixels) with fill word
	$0001	Fill (set pixels) with fill word
	$0002	XOR (XOR pixels) with fill word
	$0003	OR (OR pixels) with fill word
	Error Code = $0046	Data out of range.

39	AUTO LINE FEED SELECT	Allows the user to elect to have a line feed generated after every carriage return or not to.
	Function Data	One word, integer.
	Data = $0000	Line feeds are not generated automatically.
	$0001	Line feeds are generated after each carriage return.
	Error Code = $0047	Data out of limits.

40	MOVE CURSOR	XOR's the cursor at the current position, resets the cursor erased flag and writes the cursor at a location specified by the data. The most significant word of this long word indicates the line and the least significant word indicates the column in which the new cursor will be. Note that line and column numbering begin with zero.

	Function Data	One long word, integer.
	Data $000N 000M	XOR's old cursor and writes new cursor on line N, column M.
	Error Code = $0048	Data out of limits.

41	SET WINDOW	Allows the user to specify the window position and dimensions to other than the default values of 25 lines by 80 columns. Four words are used. The first word specifies the column number of the left column. The second word specifies the line number of the top line. The third word specifies the window width in number of columns and the fourth word specifies the window height in number of lines. Note that line and column numbering begin at zero.
	Function Data	Four words, integer.
	Word 1 $NNNN Word 2 $NNNN Word 3 $NNNN Word 4 $NNNN	Left column number. Top line number. Window width in columns. Window depth in lines.
	Error Code = $0049	Data out of limits.

42	SET CHARACTER BUFFER ADDRESS	Allows the user to specify the address of a character buffer. All operations will then uses this address to locate the character buffer. This must be an even address.
	Function Data	One long word, integer.
	Data = $0000 0000	Points to default character buffer.
	$NNNN NNNN	Points to user specified character buffer.

	Error Code $004A	Data out of limits or an odd address was specified.
43	SET CHARACTER BUFFER POINTER	Allows the user to set the character buffer pointer to point to any location in the buffer. The most significant word in the long word specifies the line number and the least significant word specifies the column number. Note that line and column numbering begins with zero.
	Function Data	One long word, integer.
	Data = $000N 000N	Sets the pointer to point to line N and column N.
	Error Code = $004B	Data out of limits.
44	GET CHARACTER BUFFER POINTER	Returns a long word in which the most significant word specifies the line number and the least significant word specifies the column number. Note that line and column numbering begins with zero.
	Function Data	Leave space for one long word, integer.
	Data = $000N 000N	The pointer is pointing to line N and column N in the character buffer.
	Error Code	None.
45	CLEAR LINE IN CHARACTER BUFFER	Fills the line from the current pointer location to the end of the line with the ASCII blank ($00) and

		clears the attribute bits associated with the cleared line.
	Function Data	None required.
	Error Code	None.

46	CLEAR CHARACTER BUFFER WINDOW	Fills entire window with ASCII blank ($20) and clears all attribute bits.
	Function Data	None required.
	Error Code	None.

5.0 CRT GRAPHICS DRIVER

5.1 DRIVER DESCRIPTION

The CRT Graphics Driver provides the user with a means of performing graphics operations. Access to the graphics routines is primarily through the I/O manager FUNCTION call. The user OPENs a graphics window, which is treated as an independent logical unit. A graphics window is a rectangular area of the screen in which graphics primitives can be displayed. Once a window has been opened the user may perform functions such as drawing a line, filling a rectangular area, clearing the window or drawing a character string. These operations are called "graphics primitive operations." Paragraph 5.4.1 provides a summary of the various functions which are available. When the user has completed his display the window is CLOSEd by issuing a "CLOSE" call to the I/O manager using the Logical Unit Number of the window. The window is defined in terms of Screen Coordinates at OPEN time and is a region in either of the two pages of graphics memory corresponding to a rectangular area on the CRT. Figure 5-1 shows the graphics coordinates for the CRT screen as well as the coordinates for the "ALPHA WINDOWS" and the "CONSOLE BOX" used by the CRT DISPLAY Driver. See Chapter 4.

The graphics driver maps between two coordinate systems. The first is the "Screen Coordinates" referred to above and in Figure 5-1. The second is "User Coordinates" in which the user specifies information to the Graphics Driver. The driver maps these into Screen Coordinates by one of three mapping modes.

In mode 0 User Coordinates correspond to Screen Coordinates on a 1-to-1 basis. In mode 1 User Coordinates become Device Coordinates which have a window offset vector added. In mode 2 User Coordinates are scaled to fit in the window and become Screen Coordinates with a window offset vector added. See Figures 5-2 and 5-3.

When a graphics primitive is drawn it typically starts at the Current Operating Point or COP and ends at a point specified in user coordinates. Graphics primitives cause pixels to be filled according to the logic specified by Function 5 and the current fill word (See Function 19). The fill word acts as a bit mask. As each pixel is encountered the fill word is rotated one bit to the right and the least significant bit in the fill word determines what action will be performed on the pixel. If the current least significant bit of the fill word is a 1 the pixel will be SET, RESET, or EXCLUSIVE-ORed as specified by the logic selected by Function 5. If the current least significant bit of the fill word is 0 no action takes place. In this manner fill patterns, dotted lines, dashes etc. can be created. See Figure 5-4. Function 5 was used to specify "SET"

logic. This will cause the pixels in the graphics window to be SET when a one is encountered in the fill word. This figure illustrates what the driver does when Function 27, "DRAW A VECTOR" is used. The Current Operating Point is where the vector begins. The driver checks the least significant bit of the fill word. Since it is zero the driver does nothing with the first pixel. The COP is now moved to the next pixel in the direction of the End Point which was specified in Function 27 and then the driver rotates the bits in the fill word one bit to the right as is shown in Step 2. The driver checks the least significant bit of the fill word and since it is now a one the driver takes action on the next pixel. The driver SETs (turns on) the pixel at the COP. The COP is again moved to the next pixel in the direction of the End Point and the bits in the fill word are again rotated one bit to the right. The least significant bit is now zero so no action is taken on the pixel at the COP. The action proceeds in this fashion until the end point or the window boundary is reached. Fill words of $5555 as shown will produce a dotted line as will $AAAA - ($AAAA causes the first pixel to be SET rather than the second). $FFFF will produce a solid line and $3333 will produce a string of narrow dashes whereas $F0F0 will make the dashes twice as wide.

The graphics driver also allows the user to create and use a SPECIAL FONT TABLE. This could be a table of special symbols (APL or chemical symbols) or all the different playing cards in a deck of cards or chess pieces or whatever the user had a need to display on the CRT. Each character may be up to 32 columns (pixels) by 32 lines (scan lines) deep. Figure 5-5 shows how to create the SPECIAL FONT TABLE. Begin by deciding the font size and use Function 15 to specify this size as well as the beginning address of the table. Next draw the character (i.e. chess figure, playing card, chemical symbol, etc.) in a grid of the same size you specified for font size. Next convert each pixel into a binary value using 0 for the light areas and 1 for the dark areas. Begin with the top line. Each line will be represented by the smallest data size that will hold it (byte, word or longword). An 11 column wide font will require a word for each line. Use the 11 low bits and leave the 5 high order bits at 0. Each character will require a byte, word or longword for each line as specified in the font height. Enter the characters in the font table in the order you desire. Although all 256 entries must be made, unused entries can be all blanks. The entries are addressed by their relative position in the table and are pointed to by the ASCII code of the character entered with the DRAW A CHARACTER STRING Function (Function 29). For example if Function 15 is used to specify a font table and if the entries in this table at positions 64-68 ($41-$45) are the ace, deuce, three, four and five of hearts, and a character string of 5 bytes which are $41, $42, $43, $44 and $45 is specified then these playing cards will displayed in the window. If however, Function 15 is not used, the system will default to the system font table and the letters A, B, C, D and E will be displayed in the window. ($41 = A, $42 = B etc.).

Note: At open time the window is initialized with the default parameters shown in the function packet descriptions. Several of the more significant defaults are shown below for convenience.

WINDOW BOUNDARIES	(0, 0, 768, 479)
PAGE NUMBER	0
PIXEL MODE	1 SET PIXELS
MAPPING MODE	0 USER COORDS. = SCREEN COORD.
COORDINATOR INTERPRETATION	0 ABSOLUTE
CHARACTER ORIENTATION	0 LEFT TO RIGHT
CHARACTER MAGNIFICATION	1
CHARACTER FONT	0 INTERNAL
CHARACTER FIELD	9 x 16
FONT DIMENSIONS	8 x 14
FILL WORD	$FF

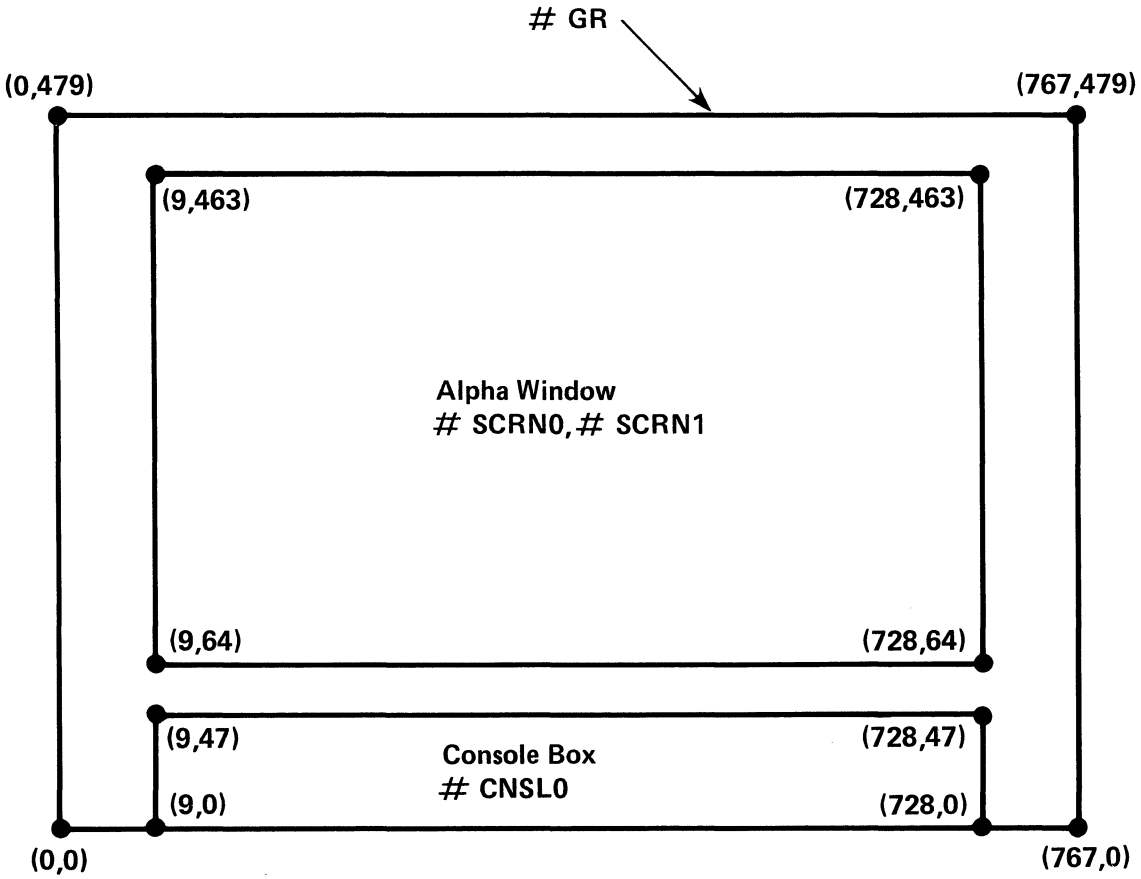
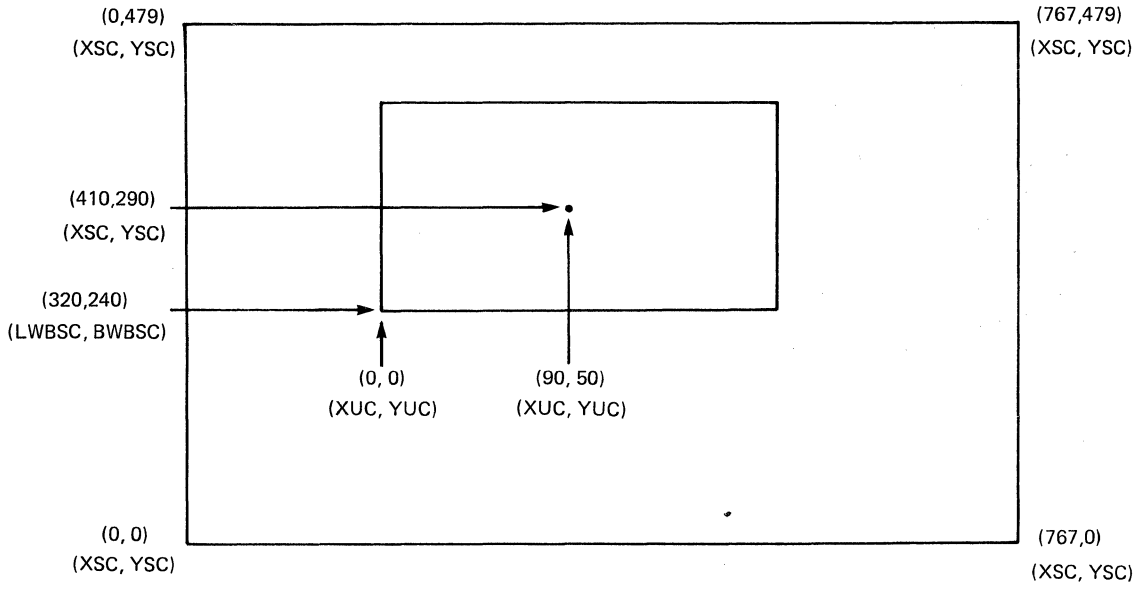


Figure 5-1. CRT Graphics Coordinates



Screen Coordinates = User coordinates plus window offset vector.

XSC = XUC + LWBSC
XSC = 90 + 320
XSC = 410

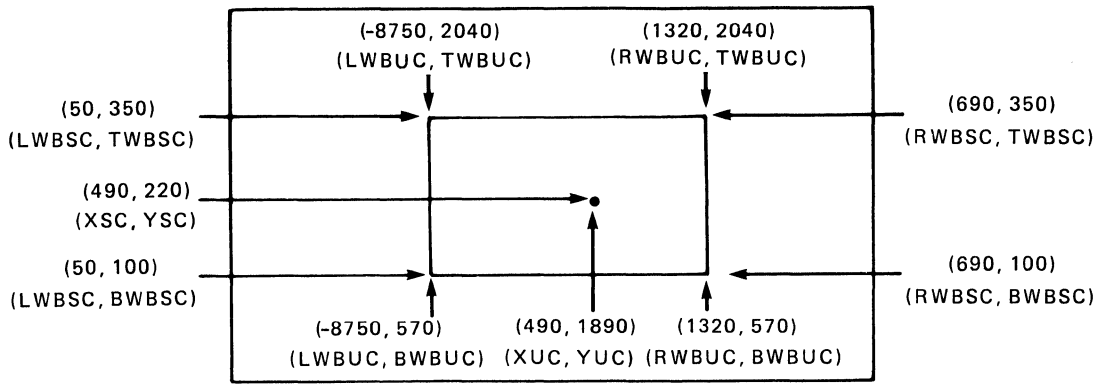
YSC = YUC + BWBSC
YSC = 50 + 240
YSC = 290

Abbreviations:

XSC	X axis in Screen Coordinates	0 to 767
XUC	X axis in User Coordinates	-16384 to 16383
YSC	Y axis in Screen Coordinates	0 to 479
YUC	Y axis in User Coordinates	-16384 to 16383
LWBSC	Left Window Boundary in Screen Coordinates.	
BWBSC	Bottom Window Boundary in Screen Coordinates.	

Note: Points outside the window are not displayed.

Figure 5-2. Mode 1 Mapping



Screen Coordinates = User coordinates plus window offset vector and scaled to fit into window.

$$XSC = (XUC - LWBUC)\left(\frac{RWBSC - LWBSC}{RWBUC - LWBUC}\right) + LWBSC$$

$$XSC = (490 + 8750)\left(\frac{690 - 50}{1320 + 8750}\right) + 50$$

$$XSC = (9240)\left(\frac{640}{10070}\right) + 50$$

$$XSC = (9240)\,(.0635) + 50$$

$$XSC = 587.249 + 50 = 637.249 = 637$$

$$YSC = (YUC - BWBUC)\left(\frac{TWBSC - BWBSC}{TWBUC - BWBUC}\right) + BWBSC$$

$$YSC = (1890 - 570)\left(\frac{350 - 100}{2040 - 570}\right) + 100$$

$$YSC = (1320)\left(\frac{250}{1470}\right) + 100$$

$$YSC = (1320)\,(.1700) + 100 = 224.489 + 100 = 324.489 = 324$$

Additional Abbreviations:

LWBUC	Left Window Boundary in User Coordinates.
RWBSC	Right Window Boundary in Screen Coordinates.
RWBUC	Right Window Boundary in User Coordinates.
TWBSC	Top Window Boundary in Screen Coordinates.
TWBUC	Top Window Boundary in User Coordinates.
BWBUC	Bottom Window Boundary in Screen Coordinates.

Screen Coordinates of a window are specified with Function 1.

User Coordinates of a window are specified with Function 21.

Figure 5-3. Mode 2 Mapping

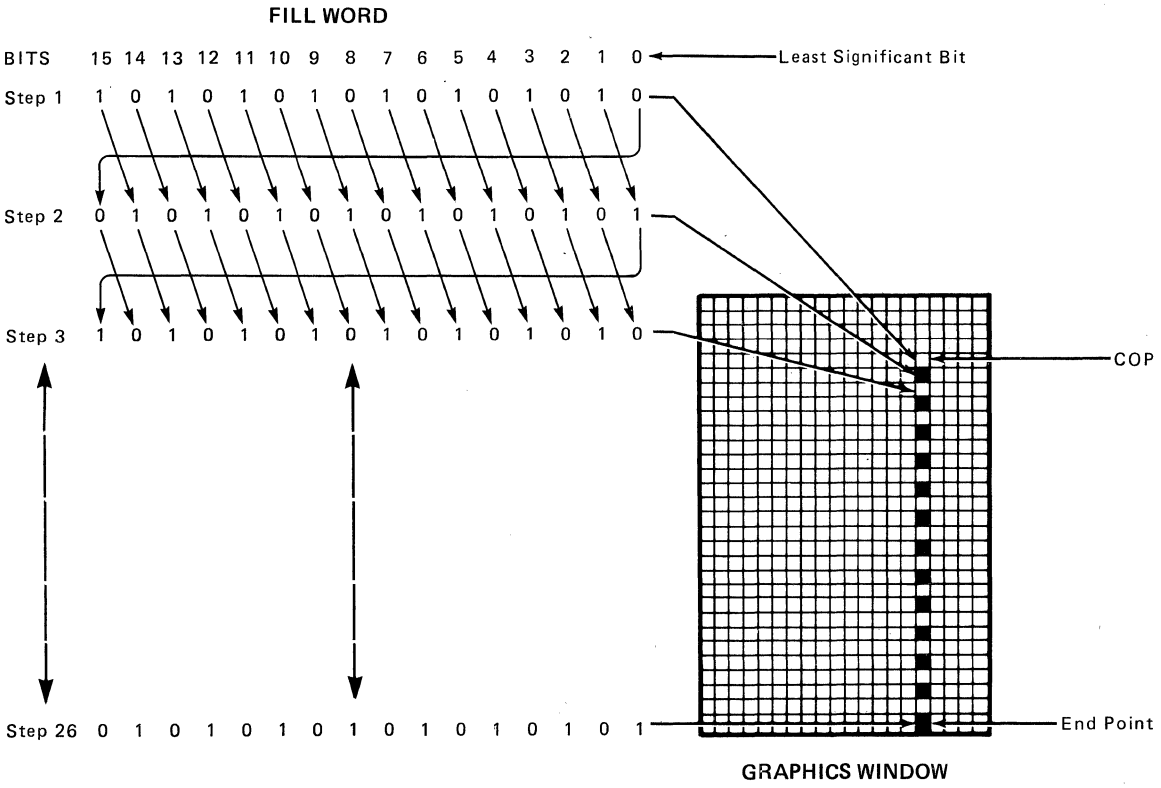


Figure 5-4. Draw a Vector Function

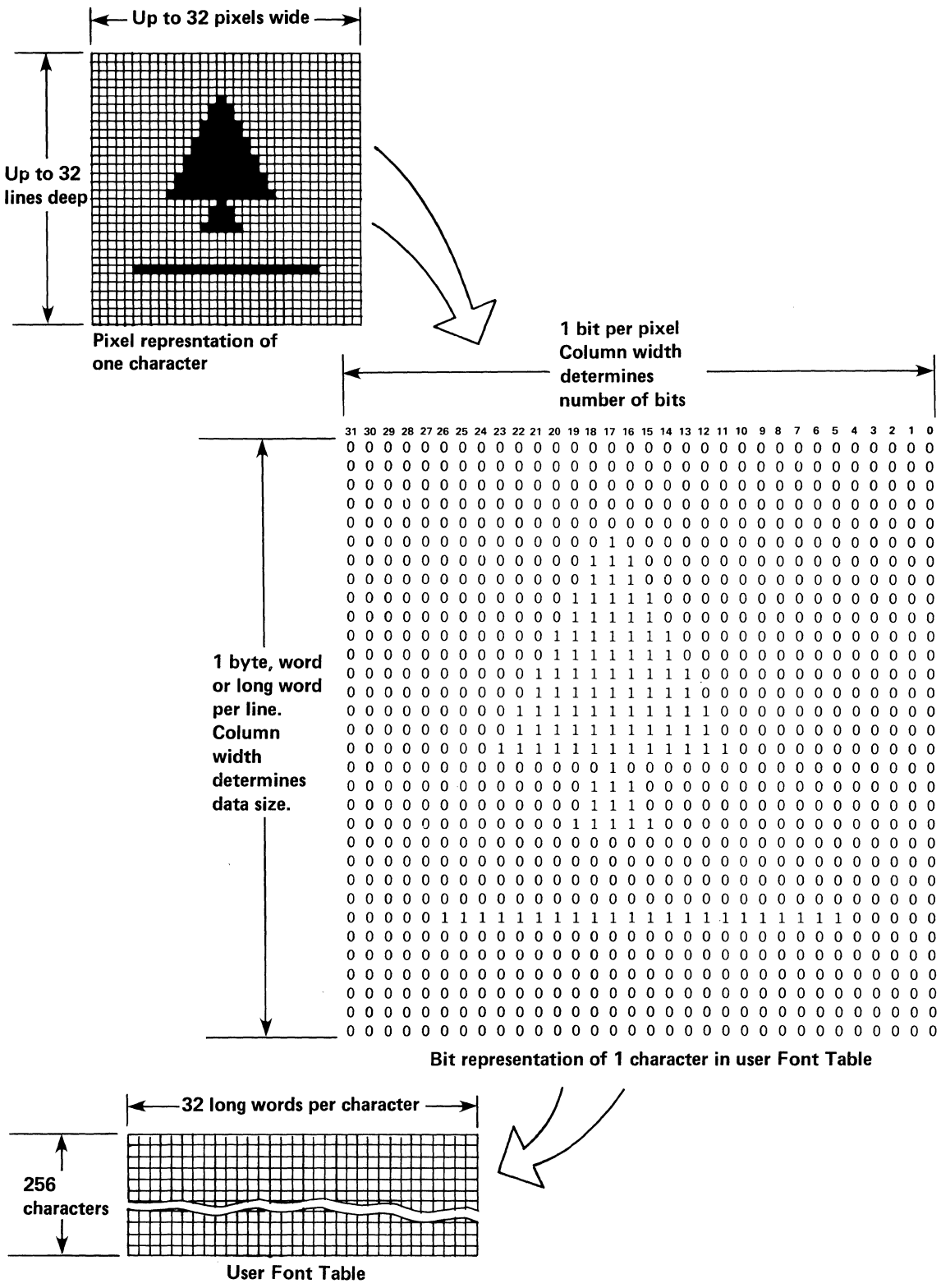


Figure 5-5. User Font Table

5.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager.

To open a graphics window, the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. Following OPEN the user may issue function calls to the graphics driver.

5.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION OF USE
DIBVOL	DS.B	6	Device name. Use #GR
DIBDTD	DS.B	1	Data Transfer Direction. Use 0. This device is WRITE only.
DIBTRN	DS.B	1	Enter 0 for fixed length or 1 for variable length transfers.
DIBRS0	DC.L	0,0	User sets this field to 0.
DIBOPT	DC.W	0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L	1	Insert pointer to function packet or set this field to $0000 0000 to select the default mode.
DIBBIO	DC.L	0	Byte I/O is not supported by this driver. User sets this field to 0.

5.3 GRAPHICS DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) is not used by this driver. The graphics function packet is the means by which requests are relayed to the driver.

5.4 GRAPHICS FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The Function Packet is a list of COMMAND/DATA structures terminated by a zero, indicating END-OF-LIST. The COMMAND word is followed by zero or more bytes, words, or longwords that send or receive the DATA for the COMMAND, or a longword that points to the DATA for that COMMAND.

5.4.1 SUMMARY OF GRAPHICS FUNCTIONS

The functions listed below can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET WINDOW BOUNDARIES	1 ($0001)	$0021	FOUR WORDS
GET WINDOW BOUNDARIES	2 ($0002)	NONE	FOUR WORDS RETURNED
SELECT PAGE NUMBER	3 ($0003)	$0023	WORD
GET PAGE NUMBER	4 ($0004)	NONE	WORD RETURNED
SELECT PIXEL MODE	5 ($0005)	$0025	WORD
GET PIXEL MODE	6 ($0006)	NONE	WORD RETURNED
SET MAPPING MODE	7 ($0007)	$0027	WORD
GET MAPPING MODE	8 ($0008)	NONE	WORD RETURNED
SET COORDINATE INTERPRETATION MODE	9 ($0009)	$0029	WORD
GET COORDINATE INTERPRETATION MODE	10 ($000A)	NONE	WORD RETURNED
SET CHARACTER ORIENTATION MODE	11 ($000B)	$002B	WORD
GET CHARACTER ORIENTATION MODE	12 ($000C)	NONE	WORD RETURNED
SET CHARACTER MAGNIFICATION	13 ($000D)	$002D	WORD

READ CHARACTER MAGNIFICATION	14 ($000E)	NONE	WORD RETURNED
SET CHARACTER FONT	15 ($000F)	$002F	LONG WORD POINTER
READ CHARACTER FONT DIMENSIONS	16 ($0010)	NONE	TWO WORDS RETURNED
SET CHARACTER FIELD	17 ($0011)	$0031	TWO WORDS
GET CHARACTER FIELD	18 ($0012)	NONE	TWO WORDS RETURNED
SET FILL WORD	19 ($0013)	NONE	WORD
GET FILL WORD	20 ($0014)	NONE	WORD RETURNED
SET MAPPING COORDINATES	21 ($0015)	$0035	FOUR WORDS
GET MAPPING COORDINATES	22 ($0016)	NONE	FOUR WORDS RETURNED
SET CURRENT OPERATING POINT	23 ($0017)	$0037	TWO WORDS
GET CURRENT OPERATING POINT	24 ($0018)	NONE	TWO WORDS RETURNED
SET A PIXEL	25 ($0019)	$0039	TWO WORDS
GET A PIXEL	26 ($001A)	NONE	THREE WORDS
DRAW A VECTOR	27 ($001B)	$003B	TWO WORDS
FILL A RECTANGLE	28 ($001C)	$003C	TWO WORDS
DRAW A CHARACTER STRING	29 ($001D)	$003D	LONG WORD POINTER
CLEAR THE WINDOW	30 ($001E)	NONE	NONE
FRAME THE WINDOW	31 ($001F)	NONE	NONE
DRAW AN ELLIPSE	32 ($0020)	$0040	TWO WORDS
SCROLL WINDOW	33 ($0021)	$0041	TWO WORDS

5.4.2 GRAPHICS FUNCTION DESCRIPTION

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET WINDOW BOUNDARIES	At open time the window is given the default boundaries of 0,0,767,479 which is the full screen. See Figure 5-1. This function allows the user to redefine the window boundaries using Screen coordinates.
	Function Data	Four words, integer

	Default	0, 0, 768, 479
	Word 1	Left window boundary 0..767 ($000..$02FF)
	Word 2	Bottom window boundary 0..479 ($000.$01DF)
	Word 3	Right window boundary 0..767 ($000.$02FF)
	Word 4	Top window boundary 0..479 ($0000.$01DF)
	Error Code=$0021	Data out of limits.
	Note:	Word 1 < Word 3 Word 2 < Word 4

2	GET WINDOW BOUNDARIES	Returns four words which specify the window boundaries in Screen Coordinates. See Function 1 above.
	Function Data	Leave space for four words, integer
	Error Code	None

3	SELECT PAGE NUMBER	Allows the user to specify which of two pages will be used. (0,1)
	Function Data	One word, integer
	Data = $0000	Selects graphics page 0
	$0001	Selects graphics page 1
	Error Code=$0023	Data out of limits
	Default =	0
	Note:	This does not change the current displayed page.

4	GET PAGE NUMBER	Returns the number of the graphics page currently in use.
	Function Data	Leave space for one word.

	Error Code	None
5	SET PIXEL MODE	Selects the logic by which the graphics primitives will cause pixels to be filled.
	FUNCTION DATA	One word, integer
	Data = $0000	RESET pixels
	$0001	SET pixels
	$0002	XOR pixels
	Error Code=$002B	Data out of limits.
	Default =	1
6	GET PIXEL MODE	Returns the current pixel control mode number in the data.
	Function Data	Leave space for one word, integer
	Data = $0000	RESET mode
	$0001	SET mode
	$0002	XOR mode
	Error Code	None
7	SET MAPPING MODE	Allows the user to select one of three coordinate mapping modes. In mode 0 there is 1-to-1 mapping between the Screen Coordinates and User Coordinates. In mode 1 User Coordinates become Screen Coordinates with a Window Offset Vector added. In mode 2 the User Coordinates are "Scaled" and become Screen Coordinates with a Window Offset Vector added.
	Function Data	One word, integer.
	Data = $0000	Mode 0. User Coordinates = Screen

		Coordinates.
	$0001	Mode 1. User Coordinates = Screen Coordinates plus Offset Vector. See Figure 5-2.
	$0002	Mode 2. User Coordinates = Screen Coordinates plus scaling plus offset vector. See Figure 5-3.
	Error Code=$0027	Data out of limits.
	Default =	0
8	GET MAPPING MODE	Returns the current coordinate mapping mode.
	Function Data	Leave space for one word, integer
	Data = $0000	Mode 0 is in effect.
	$0001	Mode 1 is in effect.
	$0002	Mode 2 is in effect.
	Error Code	None
9	SET COORDINATE INTERPRETATION MODE	Allows the user to specify either absolute or relative coordinate interpretation. Absolute coordinates are interpreted literally. Relative coordinates are interpreted as offsets to the Current Operating Point.
	Function Data	One word, integer
	Data = $0000	Selects absolute interpretation
	$0001	Selects relative interpretation
	Error Code=$0029	Data out of limits.
	Default =	0

10	GET COORDINATE INTERPRETATION MODE	Returns the current coordinate interpretation mode.
	Function Data	Leave space for one word, integer
	Data = $0000	Absolute interpretation is in effect.
	$0001	Relative interpretation is in effect.
	Error Code	None

11	SET CHARACTER ORIENTATION MODE	Allows the user to specify the orientation of text strings on the page. Four modes are available. In each mode the characters are rotated to match the text orientation.
	Function Data	One word, integer.
	Data = $0000	Characters are oriented left to right.
	$0001	Characters are oriented bottom to top.
	$0002	Characters are oriented right to left.
	$0003	Characters are oriented top to bottom.
	Error Code=$002B	Data out of limits.
	Default =	0

12	GET CHARACTER ORIENTATION MODE	Returns the current character orientation mode.
	Function Data	Leave space for one word, integer.
	Data = $0000	Character orientation is left to right.
	$0001	Character orientation is bottom to top.
	$0002	Character orientation is right to left.
	$0003	Character orientation is top to bottom.

	Error Code	None
13	SET CHARACTER MAGNIFICATION	Allows the user to specify an integer value by which the character dimensions will be magnified.
	Function Data	One word, integer.
	Data = $000N	Characters are magnified N times.
	Error Code=$002D	Data out of limits.
	NOTE: N must be greater than 0.	
	Default =	1
14	GET CHARACTER MAGNIFICATION	Returns the current character magnification value.
	Function Data	Leave space for one word, integer
	Error Code	None
15	SET CHARACTER FONT	Allows the user to select and create a user font table. This function specifies the number of columns (1-32) and rows (1-32) for each character and the start address of the font table. The font table must be organized as follows:

1-8 COLUMNS	9-16 COLUMNS	17-32 COLUMNS
1 BYTE/ROW	1 WORD = 2 BYTES/ROW	1 LONGWORD = 4 BYTES/ROW
N ROWS	N ROWS	N ROWS
256 CHARS IN TABLE	256 CHARS IN TABLES	256 CHARS IN TABLE
TOTAL BYTES = 256 x N	TOTAL BYTES = 512 x N	TOTAL BYTES = 1024 x N

		Characters are entered in the table row at a time from top to bottom starting with character 0 and ending with character 255. To invoke this function the user must pass a pointer to a font control block containing the following: FONT CONTROL BLOCK ARG 1 WORD NUMBER OF COLUMNS IN FONT ARG 2 WORD NUMBER OF ROWS IN FONT ARG 3 LONGWORD POINTER TO FONT TABLE
	Function Data	One Longword.
	Longword	Pointer to Font Control Block.
	Error Code=$002F	Data out of limits.
	Default Font	8 Columns x 14 Rows
16	GET CHARACTER FONT DIMENSIONS	Returns the dimensions currently specified for the user created font table.
	Function Data	Leave space for two words, integer.
	Word 1	Specifies the number of columns (width)
	Word 2	Specifies the number of lines (height)
	Error Code	None
17	SET CHARACTER FIELD	Allows the user to specify the field in which the font is to be set. This provides a means to clip a few columns or lines from the font or to add a few columns or lines to the font. For example the system font is 8 pixel by 14 scan lines but is displayed in a field of 9 pixels by 16 scan lines. The font is positioned in the upper right corner of the field. If the field exceeds the dimensions of the font the gap will be

		filled with zero bits.
	Function Data	Two words, integer
	Word 1	Specifies field width (1-32 pixels)
	Word 2	Specifies field height (1-32 scan lines)
	Error Code=$0031	Data out of limits.
	Default =	9 x 16
18	GET CHARACTER FIELD	Returns the dimensions of the character field.
	Function Data	Leave space for two words, integer.
	Word 1	Specifies the number of columns (width)
	Word 2	Specifies the number of lines (height)
	Error Code	None
19	SET FILL WORD	Allows the user to specify the value of the fill word which is used by DRAW A VECTOR, FILL A RECTANGLE and the DRAW AN ELLIPSE functions. The fill word acts as a mask in conjunction with the pixel control bits to SET, RESET or EXCLUSIVE OR the pixels. For example a fill word of $AAAA would create a dotted line and $F0F0 would create a string of dashes.
	Function Data	One word, integer
	Data Range	$0000 to $FFFF
	Error Code	None
	Default =	$FF
20	GET FILL WORD	Returns the value of the current fill word.

	Function Data	Leave space for one word, integer
	Error Code	None
21	SET MAPPING COORDINATES	This function establishes the user coordinate window boundaries for Mode 2 mapping described in Function 7, SET MAPPING MODE.
	Function Data	Four words, integer.
	Word 1	Left window boundary in user coordinates (-16384 to +16383).
	Word 2	Bottom window boundary in user coordinates (-16384 to +16383).
	Word 3	Right window boundary in user coordinates (-16384 to +16383).
	Word 4	Top window boundary in user coordinates (-16384 to +16383)
	Error Code=$0035	Data out of limits.
	Default =	0, 0, 768, 479
22	GET MAPPING COORDINATES	Returns the user mapping coordinates in effect.
	Function Data	Leave space for four words, integer.
	Error Code	None
23	SET CURRENT OPERATING POINT	Allows the user to specify the Current Operating Point (COP). The COP may be considered as a graphics cursor and is a point in two-dimensional space. When a primitive is drawn it starts at the COP. Some primitives cause the COP to move. These are SET CURRENT OPERATING POINT, DRAW A CHARACTER STRING, and DRAW A

		VECTOR. Other primitives simply use the COP as a reference point. The COP is initialized to 0, 0.
	Function Data	Two words, integer.
	Word 1	COP X axis in User Coordinates.
	Word 2	COP Y axis in User Coordinates.
	Error Code=$0037	Data out of limits.
24	GET CURRENT OPERATING POINT	Returns the current operating point. The COP may exceed the window boundaries.
	Function Data	Leave space for two words, integer.
	Error Code	None
25	SET A PIXEL	Allows the user to draw a pixel at the location specified in user coordinates in the Function Data. The pixel will be SET, RESET or EXCLUSIVE OR'd in accordance with the current pixel control bits.
	Function Data	Two words, integer.
	Word 1	Specifies X axis in User Coordinates.
	Word 2	Specifies Y axis in User Coordinates.
	Error Code=$0039	Data out of limits.
26	GET A PIXEL	Returns the current status of the pixel at the location specified in User Coordinates by the Function Data.
	Function Data	Three words, integer.
	Word 1	Specifies X axis in User Coordinates.
	Word 2	Specifies Y axis in User Coordinates.

	Word 3	Leave space for one word, integer.
	Data = $0000	Pixel is OFF.
	$0001	Pixel is ON.
	$FFFF	Pixel is outside of window.
	Error Code	None.

27	DRAW A VECTOR	Draws a vector to a point specified by the Function Data. The vector begins at the Current Operating Point and ends at a point specified in the DATA words. If all or a portion of the vector lies outside the window boundaries it will be clipped and only the portion within the window will be displayed. The COP is repositioned to the vector end point.
	Function Data	Two words, integer.
	Word 1	Specifies X axis in User Coordinates of the end point.
	Word 2	Specifies Y axis in User Coordinates of the end point.
	Error Code=$003B	Data out of limits.

28	FILL A RECTANGLE	This function fills the rectangular area between the Current Operating Point and a point specified in User Coordinates by the Function Data with the current fill word. If the fill area exceeds the window boundary it will be clipped and only the region within the window will be filed.
	Function Data	Two words, integer.
	Word 0	Specifies the X axis in User Coordinates.
	Word 1	Specifies the Y axis in User Coordinates.
	Error Code=$003C	Data out of limits.

29	DRAW A CHARACTER STRING	Draws the character string specified by the Function Data from the Current Operating Point in a direction specified by the character orientation bits in the option word. If the string exceeds the window boundaries it will be clipped. The first word in the Function Data must specify the number of bytes (characters) in the string. The COP is repositioned to the lower left corner of the character position following the last character.
	Function Data	Pointer to a text string, where a string is defined as a one word length field followed by text data in the form of ASCII bytes.
	Long Word	
	Error Code=$003D	Data out of limits.
30	CLEAR THE WINDOW	Clears the window by complementing the action of the pixel control bits in the option word. For example if pixels are being RESET this function will clear the window by setting all the pixels.
	Function Data	None
	Error Code	None
31	FRAME THE WINDOW	Draws a frame around the window which is one scan line thick and one pixel wide.
	Function Data	None
	Error Code	None
32	DRAW AN ELLIPSE	Draws an ellipse centered on the Current Operating Point and having X and Y axis radii equal to the distance between the COP and the values specified in Function

		Data for the X and Y axes (maximum=256).
	Function Data	Two words, integer.
	Word 1	Specifies X axis radius in User Coordinates.
	Word 2	Specifies Y axis radius in User Coordinates.
	Error Code=$0040	Data out of limits.

33	SCROLL WINDOW	Scrolls the contents of the window by the distance between the Current Operating Point and a point specified by the Function Data.
	Function Data	Two words, integer.
	Word 1	Specifies X axis in User Coordinates.
	Word 2	Specifies Y axis in User Coordinates.
	Error Code=$0041	Data out of limits.

6.0 PRINTER DRIVER

6.1 DRIVER DESCRIPTION

The printer driver provides for three modes of printer operation -- one alphanumeric mode and two graphics modes. The alphanumeric mode is used to output text characters while the graphics mode is used to output graphics information. Access to the printer driver is provided through general purpose system calls to the I/O manager using the SYSIO macro. These include OPEN, CLOSE, AWRITE, SWRITE, FUNCTION, CANCEL and INIT.

In the Alphanumeric Mode text may be output either a byte at a time or in blocks of fixed or variable length. A variety of functions are supported including multiple colors, proportional spacing, text justification and character enhancement. Printing in this mode is bidirectional.

In the Graphics mode either 100 dots per inch or 200 dots per inch may be specified and only fixed length block output is permitted but successive blocks may vary in length. Bytes received by the printer in graphics mode specify the firing pattern for the upper seven wires on the print head for successive horizontal positions. See Figure 6-1. The wires are spaced 4/336's of an inch apart. Graphics output lines can be interlaced one, two or four times if desired to achieve increased vertical resolution. The amount of vertical advance can be controlled with the SET ADVANCE or the SET ABSOLUTE POSITION functions. In general, vertical Advance 1 is used for this purpose.

Note: The printer driver has the following attributes:

Non-sharable -- Only one logical unit can open the printer at a time.

Supports Asynchronous Transfers -- Asynchronous reads and writes are supported by the printer.

Non-Reentrant -- The printer driver is non-reentrant.

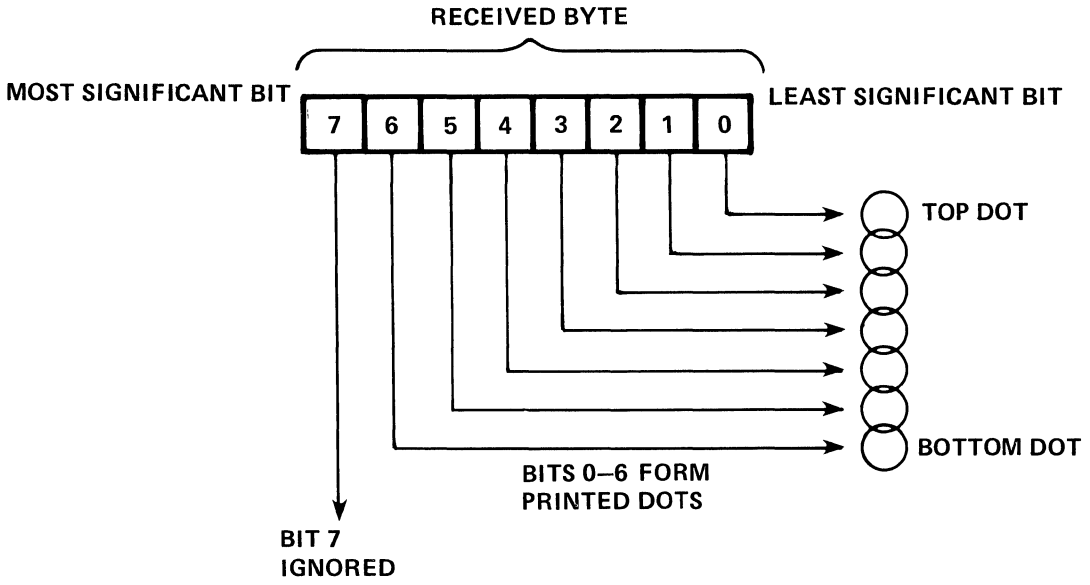
Note: During Mode 0 (normal printing) operation, control codes are filtered out of the data stream with the exception of the normal control characters $08...$0D (BS,HT,LF,VT,FF,CR). The data stream is thus limited to printable ASCII characters and normal carriage control.

ASCII Character Set and Control Codes

ASCII Control Codes

CHR	DEC	HEX	CHR	DEC	HEX	CHR	DEC	HEX
NUL	000	00	,	044	2C	X	088	58
SOH	001	01	-	045	2D	Y	089	59
STX	002	02	.	046	2E	Z	090	5A
ETX	003	03	/	047	2F	[	091	5B
EOT	004	04	0	048	30	\	092	5C
ENQ	005	05	1	049	31	]	093	5D
ACK	006	06	2	050	32	^	094	5E
BEL	007	07	3	051	33	_	095	5F
BS	008	08	4	052	34	`	096	60
HT	009	09	5	053	35	a	097	61
LF	010	0A	6	054	36	b	098	62
VT	011	0B	7	055	37	c	099	63
FF	012	0C	8	056	38	d	100	64
CR	013	0D	9	057	39	e	101	65
SO	014	0E	:	058	3A	f	102	66
SI	015	0F	;	059	3B	g	103	67
DLE	016	10	<	060	3C	h	104	68
DC1	017	11	=	061	3D	i	105	69
DC2	018	12	>	062	3E	j	106	6A
DC3	019	13	?	063	3F	k	107	6B
DC4	020	14	@	064	40	l	108	6C
NAK	021	15	A	065	41	m	109	6D
SYN	022	16	B	066	42	n	110	6E
ETB	023	17	C	067	43	o	111	6F
CAN	024	18	D	068	44	p	112	70
EM	025	19	E	069	45	q	113	71
SUB	026	1A	F	070	46	r	114	72
ESC	027	1B	G	071	47	s	115	73
FS	028	1C	H	072	48	t	116	74
GS	029	1D	I	073	49	u	117	75
RS	030	1E	J	074	4A	v	118	76
US	031	1F	K	075	4B	w	119	77
SP	032	20	L	076	4C	x	120	78
!	033	21	M	077	4D	y	121	79
"	034	22	N	078	4E	z	122	7A
#	035	23	O	079	4F	{	123	7B
$	036	24	P	080	50	|	124	7C
%	037	25	Q	081	51	}	125	7D
&	038	26	R	082	52	~	126	7E
'	039	27	S	083	53	⊞	127	7F
(	040	28	T	084	54			
)	041	29	U	085	55			
*	042	2A	V	086	56			
+	043	2B	W	087	57			

Figure 6-1. ASCII Character Set and Control Codes



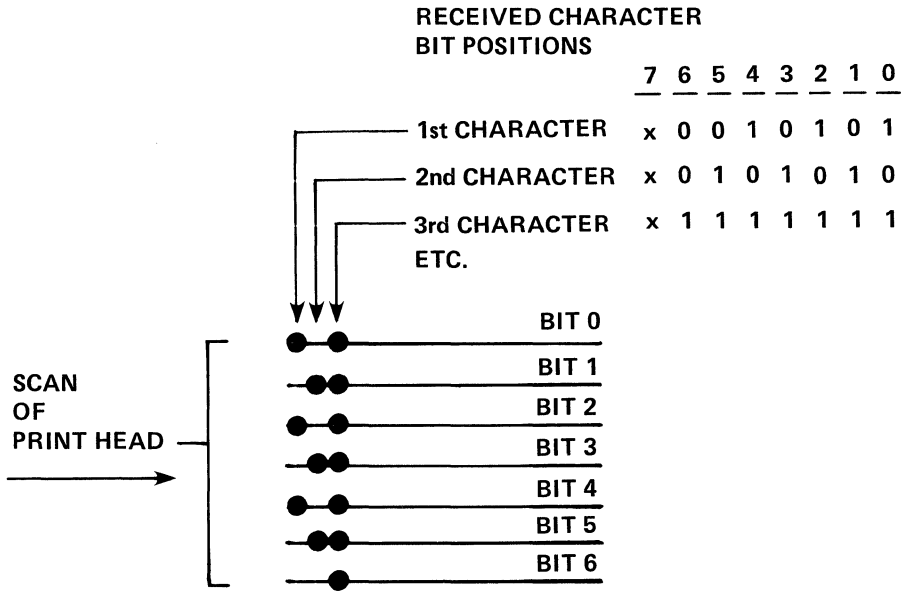


Figure 6-2. Graphics Mode Pin Firing

NOTE
DOTS ARE OVERLAPPED IN ACTUAL PRINT OPERATION

6.2 DEVICE INFORMATION BLOCK (DIB)

To OPEN the printer the user must create a Device Information Block (DIB) and within an initialization routine perform a SYSIO-OPEN to the device.

6.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION OF USE
DIBVOL	DS.B	6	Device name. Use #PR for printer driver.
DIBDTD	DS.B	1	Data transfer direction. Use 0. This driver is WRITE only.
DIBTRN	DS.B	1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRS0	DC.L	0,0	Not used. User sets this field to 0.
DIBOPT	DC.W	1	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L	1	Insert pointer to function packet or null for default.
DIBBIO	DS.L	1	System used byte I/O field. To open the printer for Byte I/O enter -1($FFFF FFFF), otherwise set it to zero. After open the I/O manager fills this field with an identifier which is used for SYSIO-BWRITE.

6.3 PRINTER DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during WRITEs. It is a required operand of the SYSIO macro for WRITEs. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

6.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper trigger byte to be used to delimit Transfer Termination character range in Variable length transfers here.
DTCTBL	DS.B 1	User puts lower trigger byte to be used to delimit Transfer Termination character range in Variable length transfers here.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every WRITE.

6.4 PRINTER FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like setting margins, tab stops and form length. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND/DATA words terminated by a zero indicating end-of-list. The COMMAND word is followed by zero or more words or longwords that send or receive the DATA for the command, or a longword that points to the DATA for that COMMAND.

6.4.1 SUMMARY OF PRINTER FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	$0021	WORD
SET PRINTER MODE	2 ($0002)	$0022	WORD
GET PRINTER MODE	3 ($0003)	NONE	WORD RETURNED
SET FONT	4 ($0004)	$0024	WORD
GET FONT	5 ($0005)	NONE	WORD RETURNED
SET COLOR	6 ($0006)	$0026	WORD
GET COLOR	7 ($0007)	NONE	WORD RETURNED
SET DENSITY	8 ($0008)	$0028	WORD
GET DENSITY	9 ($0009)	NONE	WORD RETURNED
SET FORM SIZE	10 ($000A)	$002A	TWO WORDS
GET FORM SIZE	11 ($000B)	NONE	TWO WORDS RETURNED
SET MARGINS	12 ($000C)	$002C	TWO WORDS
GET MARGINS	13 ($000D)	NONE	TWO WORDS RETURNED
SET ENHANCE	14 ($000E)	$002E	WORD
GET ENHANCE	15 ($000F)	NONE	WORD RETURNED
SET PROPORTIONAL SPACING	16 ($0010)	$0030	WORD
GET PROPORTIONAL SPACING	17 ($0011)	NONE	WORD RETURNED
SET INTERCHARACTER SPACING	18 ($0012)	$0032	WORD
GET INTERCHARACTER SPACING	19 ($0013)	NONE	WORD RETURNED
SET TABS	20 ($0014)	$0034	2-34 WORDS
GET TABS	21 ($0015)	NONE	2-34 WORDS RETURNED
SET JUSTIFICATION	22 ($0016)	$0036	WORD
GET JUSTIFICATION	23 ($0017)	NONE	ONE WORD RETURNED
SET ADVANCE	24 ($0018)	$0038	TWO WORDS
GET ADVANCE	25 ($0019)	NONE	TWO WORDS RETURNED
PERFORM ADVANCE	26 ($001A)	$003A	WORD
SET ABSOLUTE X-Y POSITION	27 ($001B)	$003B	TWO WORDS
SET ABSOLUTE ROW-COLUMN	28 ($001C)	$003C	TWO WORDS
SET RELATIVE POSITION	29 ($001D)	$003D	WORD
RESET PRINTER	30 ($001E)	NONE	NONE
SET LINE FEED MODE	31 ($0020)	$003F	WORD
GET LINE FEED MODE	32 (0020)	NONE	WORD RETURNED
SET FORM FEED MODE	33 (0021)	$0041	WORD
GET FORM FEED MODE	34 (0022)	NONE	WORD RETURNED

6.4.2 PRINTER FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	Allows the user to change the data transfer mode from that established at OPEN.
	Function Data	One word, integer.
	Data = $0000	Select Fixed length transfers.
	$0001	Select Variable length transfers.
	Error Code=$0021	Data out of limits.
	Default	Set by User in DIBTRN at open time.
2	SET PRINTER MODE	Allows the user to specify ALPHANUMERIC or GRAPHICS MODE of operation. Alphanumeric Mode 0 produces alphanumeric input from ASCII character input. Control characters $08 to $0D are accepted and the others are ignored. Graphics (Mode 1) produces 100 dots per inch graphic output from byte size integer input. See paragraph 6.1 for further details. Graphics (Mode 2) produces 200 dot per inch graphic output from byte size integer input. See paragraph 6.1 for further details. Only fixed length block data transfer are allowed in the graphics mode.
	Function Data	One word, integer.
	Data = $0000	Selects Alphanumeric Mode 0.
	$0001	Selects 100 dots per inch Graphics Mode 1.
	$0002	Selects 200 dots per inch Graphics Mode 2.

	Error Code=$0022	Data out of limits.
	Default =	0 (Alpha Mode)
3	GET PRINTER MODE	Returns the present printer mode in the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $0000	Specifies Alphanumeric Mode 0.
	$0001	Specifies 100 dots per inch Graphics Mode 1.
	$0002	Specifies 200 dots per inch Graphics Mode 2.
	Error Code	None.
4	SET FONT	Allows the user to specify the desired character font to be used for alphanumeric output.
	Function Data	One word, integer.
	Data = $0000	Selects correspondence font.
	$0001	Selects draft quality font.
	$0002	Selects -90 degree rotated graphics font.
	$0003	Selects non-rotated graphics font.
	Error Code=$0024	Data out of limits.
	Default =	0
5	GET FONT	Returns the current character font in the DATA word.
	Function Data	Leave space for one word , integer.
	Data = $0000	Specifies correspondence font.
	$0001	Specifies draft quality font.
	$0002	Specifies -90 degree rotated graphics font.
	$0003	Specifies non-rotated graphics font.
	Error Code	None.

6 SET COLOR — Allows the user to specify one of four possible ribbon positions numbered 0-3 starting at the top of the ribbon. Normally the color order from the top will be red-green-blue-black.

Function Data — One word, integer.

Data = $0000 — Selects ribbon position 0 (red)
$0001 — Selects ribbon position 1 (green)
$0002 — Selects ribbon position 2 (blue)
$0003 — Selects ribbon position 3 (black)

Error Code=$0026 — Data out of limits.

Default = — 3 (Black)

---

7 GET COLOR — Returns the current ribbon position in the DATA word.

Function Data — Leave space for one word, integer.

Data = $0000 — Specifies ribbon position 0 (red)
$0001 — Specifies ribbon position 1 (green)
$0002 — Specifies ribbon position 2 (blue)
$0003 — Specifies ribbon position 3 (black)

Error Code — None

---

8 SET DENSITY — Allows the user to select the density that is used for alphanumeric output.

Function Data — One word, integer.

Data = $0000 — Selects a 10 character per inch density.
$0001 — Selects a 12 character per inch density.
$0002 — Selects a 16.8 character per inch density.

Error Code=$0028 — Data out of limits.

Default = — 0 (10 CPI.)

Note: If auto line feed is enabled and line width is set to greater than 80, then character density will be

increased to 16.8 characters per inch.

9	GET DENSITY	Returns the current character density in the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $0000	Specifies a 10 character per inch density.
	$0001	Specifies a 12 character per inch density.
	$0002	Specifies a 16.8 character per inch density.
	Error Code	None.

10	SET FORM SIZE	Allows the user to select the form length and print field length. The form length must be larger than or equal to the print field length. Each length is expressed in 1/336 of an inch. The default value for form length is 3696. The default value for print field length is 3696.
	Function Data	Two words, integer.
	Word 1	Selects form length. This value must be the larger.
	Word 2	Selects print field length.
	Error Code=$002A	Data out of limits.

11	GET FORM SIZE	Returns the current form length and print field length in the DATA words.
	Function Data	Leave space for two words, integer.
	Word 1	Specified the current form length.
	Word 2	Specifies the current print field length.
	Error Code	None

12	SET MARGINS	Allows the user to set the right and left margins. Both are expressed in 1/120 of an inch units and the right margin must be a larger value than the left margin.
	Function Data	Two words, integer.
	Word 1 (0-960)	Left margin position in 1/120 of an inch units.
	Word 2 (1-960)	Right margin position in 1/120 of an inch units.
	Error Code=$002C	Data out of limits.
	Default =	0, 960

13	GET MARGINS	Returns the current margin positions in the DATA words.
	Function Data	Leave space for two words, integers.
	Word 1	Specifies position of the left margin.
	Word 2	Specifies position of the right margin.
	Error Code	None.

14	SET ENHANCE	Allows the user to have the characters printed with double width.
	Function Data	One word, integer.
	Data = $0000	Characters are printed in normal width.
	$0001	Characters are printed in double width.
	Error Code=$002E	Data out of limits.
	Default =	0

15	GET ENHANCE	Returns the current character enhancement in the DATA word.

	Function Data	Leave space for one word, integer.
	Data = $0000 $0001	Specifies normal width. Specifies double width.
	Error Code	None

16	SET PROPORTIONAL SPACING	Allows the user to specify proportional spacing of the output text. Inter-character spacing cannot be used with proportional spacing.
	Function Data	One word, integer.
	Data = $0000 $0001	Select regular spacing. Select proportional spacing.
	Error Code=$0030	Data out of limits.
	Default =	0

17	GET PROPORTIONAL SPACING	Returns the current selection of proportional spacing in the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $0000 $0001	Specifies regular spacing. Specifies proportional spacing.
	Error Code	None

18	SET INTERCHARACTER SPACING INCREMENT	Allows the user to adjust inter-character spacing in 1/24's of a character width. Inter-character spacing increment cannot be used with proportional spacing.
	Function Data	One word, integer.
	Data = $0007	Specifies 7/24's of a character width spacing increment.
	Error Code=$0032	Data out of limits.

	Default =	0
19	GET INTERCHARACTER SPACING	Returns the current inter-character spacing value expressed in 1/24's of a character width.
	Function Data	Leave space for one word, integer.
	Error Code	None.
20	SET TABS	Allows the user to set the horizontal and vertical tab stops for the printer. The first argument selects either horizontal or vertical tabs. The second argument is the number of tabs the user wishes to set. The following arguments are the positions of the tab stops from left to right or top to bottom. Thirty-two (32) horizontal tabs may be specified in units of 1/120's of an inch and eight (8) vertical tabs may be specified in units of 1/336's of an inch.
	Function Data	2 to 34 words, integer.
	Word 1 = $0000 $0001	Selects horizontal tabs. Selects vertical tabs.
	Word 2	Selects number of tabs to be set.
	Words 3-34	Specify positions of tab stops from left to right in 1/120's of an inch or from top to bottom in 1/336's of an inch.
	Error Code=$0034	Data out of limits.
	Default =	No tabs
21	GET TABS	Returns the current horizontal or vertical tab positions in the DATA words. In the first word the user selects horizontal or vertical tabs and in the second word specifies the number of tabs that are to be

		read. The driver will return the current tab stops in successive words in the DATA. If a tab stop has not been set the driver will return a zero in the corresponding location.
	Function Data	Two words, integer plus space for up to 32 words as required to return the information requested.
	Word 1 - $0000 $0001	Specifies that horizontal tabs be read. Specifies that vertical tabs be read.
	Word 2	Specifies up to 32 horizontal tab stops or up to 8 vertical tab stops be read.
	Words 3-34	Specifies position of tab stops from left to right in 1/120's of an inch for horizontal tabs or from top to bottom in 1/336's of an inch for vertical tabs.
	Error Code=$0035	Data out of limits.
22	SET JUSTIFICATION	Allows the printer to adjust the length of a printed line to conform to the left and right margins. Text should be sent as a single line without carriage returns or line feeds. The carriage return is used to terminate a paragraph. If indenting is desired it may be accomplished with an absolute head movement or tab. If proportional spacing is selected with this mode the printed text will approach publication quality.
	Function Data	One word, integer.
	Data = $0000 $0001	Selects regular output. Selects text justification.
	Error Code=$0036	Data out of limits.
23	GET JUSTIFICATION	Returns the current state of text justification in the DATA word.
	Function Data	Leave space for one word, integer.

	Data = $0000	Specifies that justification is not selected.
	$0001	Specifies that justification is selected.
	Error Code	None

24	SET ADVANCE	Allows the user to set one of three Vertical Advance lengths. Advance 0 is a forward advance equivalent to a line feed and used for this purpose in the alphanumeric mode. Advance 1 is also a forward advance typically used as a line feed in Graphics mode. Advance 2 is a reverse advance typically used for super-scripting in alphanumeric mode. Advance lengths are in units of 1/336's of an inch. Default values are: Advance 0 = 56/336's of an inch Advance 1 = 28/336's of an inch Advance 2 = -28/336's of an inch (reverse)
	Function Data	Two words, integer.
	Word 1 = $0000	Set Advance 0
	$0001	Set Advance 1
	$0002	Set Advance 2
	Word 2	Specifies advance length in 1/336's of an inch. (+ or - argument in case of Advance 2)
	Error Code=$0038	Data out of limits.

25	GET ADVANCE	Returns the current vertical advance length in the second DATA word for the mode selected in the first DATA word.
	Function Data	Leave space for two words, integer.
	Word 1 = $0000	Specifies Advance 0 is selected.
	$0001	Specifies Advance 1 is selected.
	$0002	Specifies Advance 2 is selected.
	Word 2	Returns the current length selected in 1/336's of an inch.
	Error Code	None

26 PERFORM ADVANCE — Performs a vertical advance in the advance mode specified by the DATA word. See Function 24 above.

Function Data — One word, integer.

Data =		
	$0000	Selects Advance 0
	$0001	Selects Advance 1
	$0002	Selects Advance 2
	$0003	Selects Advance 0 with CR
	$0004	Selects Advance 1 with CR

Error Code=$003A Data out of limits.

---

27 SET ABSOLUTE X-Y POSITION — Allows the user to move the print head to an absolute horizontal (1/120's of an inch) or vertical (1/336's of an inch) position.

Function Data — Two words, integer.

Word 1 =	$0000	Selects horizontal axis.
	$0001	Selects vertical axis.

Word 2 — Specifies position on axis. 0-960 for horizontal and any value greater than zero and less than 9999 for vertical.

Error Code=$003B Data out of limits.

Note: Positions are with respect to top of form. A form feed will reset top of form.

---

28 SET ABSOLUTE ROW-COLUMN — Allows the user to move the print head to an absolute position expressed in column or row units. The column-row unit dimensions correspond to the current character width and line height.

Function Data — Two words integer.

Word 1 =	$0000	Selects horizontal column.

		$0001 Selects vertical row.
	Word 2	Specifies number of columns or rows. 0-120 for horizontal columns and any value greater than zero for vertical rows.
	Error Code=$003C	Data out of limits.
	Note: Positions are with respect to top of form.	

29	SET RELATIVE POSITION	Allows the user to move the print head an incremental amount in the horizontal axis. Units are 1/120's of an inch.
	Function Data	One word, integer. (-9999 to +9999)
	Data = $000N	Moves the print head N/120's of an inch.
	Error Code=$003D	Data out of limits.

30	RESET PRINTER	Performs a hard and a soft printer reset.
	Function Data	None
	Error Code	None

31	SET AUTO LINE FEED MODE	Allows the user to select auto line feed. Line feeds (LF) are automatically appended to each carriage return (CR). If the line width designated by the SET LINE WIDTH command is exceeded, a CR-LF pair will be generated automatically.
	Function Data	One word, integer.
	Data = $0000	Normal mode is selected.
	$0001	Auto line feed mode is selected.
		NOTE: If auto LF is enabled the character density function will be adjusted to match the line width value.

	Error Code=$003F	Data out of limits.
	Default =	1
32	GET AUTO LINE FEED MODE	Returns the current value of the auto line feed mode in the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $0000 $0001	Specifies normal mode. Specifies auto line feed mode.
	Error Code	None
33	SET AUTO LINE FEED MODE	Allows the user to select auto form feed. Form feeds will be automatically appended whenever page depth exceeds the value specified by the SET LINE DEPTH command.
	Function Data	One word, integer.
	Data = $0000 $0001	Selects normal mode. Selects auto form feed mode.
	Error Code=$0041	Data out of limits.
	Default =	1
34	GET AUTO FORM FEED	Returns the current value of the auto form feed mode in the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $0000 $0001	Specifies normal mode is selected. Specifies auto form feed mode is selected.
	Error Code	None
35	SET LINE WIDTH	Sets the line width (characters per line) count to be used with auto line feed.

	Function Data	One word positive integer.
	Error Code=$0043	Data out of limits.
	Default =	80

36	GET LINE WIDTH	Returns the (characters per line) line width currently being used with auto line feed.
	Function Data	One word positive integer (returned).
	Error Code:	None

37	SET LINE DEPTH	Sets the number of lines per page to be used with the auto form feed function.
	Function Data	One word positive integer.
	Error Code=$0045	Data out of limits.
	Default =	60

38	GET LINE DEPTH	Returns the current lines per page value to be used with the auto form feed function.
	Function Data	One word positive integer (returned).
	Error Code:	None

7.0 FILE ACCESS AND STRUCTURE

7.1 INTRODUCTION

The file access method in release 1.0 was RSAM, Relative Sector Access Method.*

Two new file access methods are available with 1.1 level systems:

- Sequential (fixed or variable length)
- Relative Record

This chapter discusses only the new access methods. The disk format is completely changed from 1.0. File format can be requested contiguous or defaulted non-contiguous.

Note: A file can no longer be opened for write simultaneously by more than one user.

7.2 FILE ACCESS METHODS

Access methods include Sequential Access Method for both fixed length and variable length records and Relative Record Access Method for fixed length records only. Either type of record can be from 1 to 65,535 bytes in length.

* Programming written using RSAM will continue to work on 1.1 systems. If the program is ever reassembled, the same FDMCLB44.INC/IOMCLB44.INC macros must be used. Execution errors will occur if new 1.1 macros are substituted.

7.2.1 SEQUENTIAL (DTCREC = 0)

The position within the file starts at the beginning and continues through the file with the next byte to be read or written. The number of bytes read or written is controlled by the buffer length, DTCBFL, for fixed length transfers (DIBTRN = 0), or by finding a delimitering character in the buffer for variable length transfers (DIBTRN = 1). Delimiter characters are specified in DTCTBU and DTCTBL.

7.2.2 RELATIVE RECORD (DTCREC NOT 0)

The position within the file is determined by the relative record (1-n) specified in DTCREC. The record length (the number of bytes transferred) is specified in DTCBFL. DIBTRN should always be zero indicating fixed length transfer.

7.3 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time. The information in the DIB is used by the I/O manager, the disk/diskette driver and the file access methods. It is copied from user space into the appropriate control block in system space.

To OPEN for file access, a DIB and DIB EXTENSION are needed with a volume name specified in the "DIBVOL" field. The user should not have duplicate volume identifiers on the system. The error condition will be detected if it exists between two or more diskettes or between two or more hard disks. It will not be detected between a diskette and a hard disk reliably.

7.3.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Volume identifier. 1 to 6 ASCII characters padded with blanks. If all blanks, default volume is opened, or if '0'-'7', the volume mounted in the corresponding drive is opened.

DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional.
DIBTRN	DS.B 1	Enter 0 for fixed length transfers, 1 for variable length transfers.
DIBRSO	DC.L 0,0	User sets this field to 0.
DIBOPT	DC.W 0	Insert option word described in paragraph 7.2.2.
DIBFCN	DS.L 1	Insert pointer to function packet or set this field to $0000 0000.
DIBBIO	DC.L 0	Not used by file access. User sets this field to 0.
----------DIB EXTENSION----------		
DIBNMS	DS.W 1	Return area. Number of 252-byte blocks in existing file (full or partial blocks).
DIBOFS	DS.W 1	Return area. End of File offset in last 252-byte block of file (if 0, the block is the full 252 bytes)
DIBCAT	DC.L 0,0	Reserved space. Set this field to 0.
DIBNAM	DS.B 8	File name. 1 to 8 ASCII characters, padded with blanks.
DIBEXT	DS.B 3	File name extension. 1 to 3 ASCII characters, padded with blanks.
DIBTYP	DS.B 1	File type (see paragraph 7.2.2.).
DIBACS	DS.B 1	File access attributes (see paragraph 7.2.3).
DIBRS1	DC.B 0	Not used. User sets this field to 0.
DIBVER	DC.W 0	Reserved space for future use. Set this field to 0.
DIBFID	DC.L 0	Field for system use only. Set this field to 0.
DIBRLG	DS.W 1	Record length desired this open. DIBNMR and DIBOFR are calculated based on this value. If variable length sequential access is then used, record length will be ignored. User sets this field to $0000, if not desired.

DIBNMR	DS.L 1	Return area. Number of fixed length records in file, full or parital. Not set if DIBRLG = 0 or variable transfer mode specified.
DIBOFR	DS.W 1	Return area. Offset into last record. If the fixed-record-length mode has been specified, the value 0 means a full sector. Not set if DIBRLG = 0 or variable transfer mode specified. (If 0, the block is the full record length.)
DIBIET	DS.L 1	Number of clusters desired in the first extent. User sets this field to $0000 0000 to use the volume default. A nonzero number is required for contiguous files.
DIBSET	DS.L 1	Number of clusters desired for subsequent extents. User sets this field to $0000 0000 to use the volume default.

7.3.2 DIB OPTION WORD BIT DEFINITIONS

The option word is a bit significant word that can be used to select file control parameters at open time. If the option word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table (the defaults are bit = 0).

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word (use defaults)
	1	Use option word
14		This bit used by the system. User sets this bit to zero.
	0	
13		
	0	Truncate extra blocks at CLOSE
	1	Don't truncate blocks at CLOSE
		(If contiguous file, user may not want to truncate it)
12		
	0	Normal file structure (extendable)
	1	Contiguous file structure
11		File

	0	Open existing file
	1	Create new file
10		This bit used by the system.
	0	User sets this bit to zero.
9-7	0	Unused. User sets these bits to zero.
6	0	This bit used by the system. User sets this bit to zero.
5-0	0	Unused. User sets these bits to zero.

7.3.3 DIBTYP DEFINITION

This byte is set by the user to specify the file type when creating a new file. For an OPEN on an existing file, the TYPE is returned in this field.

BYTE SET TO	MEANING
$00	Binary file
$01	Binary file with transfer address
$03	Text file (hex file)

7.3.4 DIBACS DEFINITION

This byte is set by the user to specify the file access attributes when creating a new file. For an OPEN on an existing file, the ACCESS attributes are returned in this field.

BIT NO.	SET TO	MEANING
7-3		Unused.
	0	User sets these bits to zero.
2		File access
	0	Read and write.

```
          1         Read only.

 1              File renaming
          0         File can be renamed.
          1         File cannot be renamed.

 0              File protection
          0         File can be deleted.
          1         File cannot be deleted.
```

Note: Read-only protection does not also imply delete protection. Both bits are required, if both are desired.

7.3.5 DIBRLG DEFINITION

Record length is specified in units of bytes. A user would specify record length only when planning to use fixed sequential or relative record access methods for reading a file.

7.3.6 DIBIET DEFAULTS

The primary extent default for a file on the following volumes is:

$5\frac{1}{4}$-inch diskette	--	40 clusters (cluster = 1 block)
8-inch diskette	--	50 clusters (cluster = 1 block)
hard disk	--	16 clusters (cluster = 8 blocks)

A block or sector is 256 bytes. Notice that each device has its own cluster size. A cluster equals one bit in the bitmap and is the smallest unit that can be allocated on the device.

7.3.7 DIBSET DEFAULTS

The volume defaults for secondary extensions to the file are the same as the primary extent (see above).

7.4 FILE ACCESS DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand for all SYSIO synchronous and asynchronous I/O commands. The application program uses it to provide information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

7.4.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for transfer termination characters for variable length access method. Ignored by other access methods.
DTCTBL	DS.B 1	User puts lower limit to be used for transfer termination characters for variable length access method. Ignored by other access methods
DTCRS0	DC.B 0	This field is reserved. User sets this field to 0.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	Buffer length. 1 to 65535 bytes is value range. (When using fixed sequential, it can be different for each SYSIO.)
DTCBPT	DS.W 1	Byte offset into buffer. It indicates the byte in buffer to start transfer and is updated by the file access. Typically, the user would zero it before each READ/WRITE.
DTCREC	DS.L 1	Record/Block number. Filled in only when using relative record access method, where 1 is the first record in the file.

7.5 FILE ACCESS FUNCTIONS

The Function Packet Control Block provides for special operations. It is required for the FUNCTION command and optional for the OPEN command.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more words or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

7.5.1 SUMMARY OF FILE ACCESS FUNCTIONS

The functions listed below indicate whether they are accepted with SYSIO-OPEN, SYSIO-FUNCTION, and/or SYSIO CLOSE.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
SET TRANSFER MODE (OPEN, FUNCTION)	1 ($0001)	($0021)	ONE WORD
GET DISK FORMAT LEVEL (OPEN, FUNCTION)	2 ($0002)	NONE	ONE WORD (RETURN)
DELETE FILE (CLOSE)	5 ($0005)	($008C)	NONE
GET DISK DEVICE NAME (OPEN, FUNCTION)	12 ($000C)	NONE	6 BYTES (RETURN)
*SET EOF OFFSET INTO LAST SECTOR (CLOSE)	14 ($000E)	NONE	ONE WORD
POSITION TO BEGINNING FILE (OPEN, FUNCTION)	17 ($0011)	NONE	NONE
POSITION TO END OF FILE (OPEN, FUNCTION)	18 ($0012)	NONE	NONE
CHANGE ACCESS ATTRIBUTES (OPEN, FUNCTION, CLOSE)	19 ($0013)	NONE	ONE WORD
RENAME FILE (CLOSE)	20 ($0014)	($008C)	ONE LONG WORD
SET EOF TO AFTER LAST RECORD WRITTEN (OPEN, FUNCTION, CLOSE)	21 ($0015)	NONE	NONE
BUFFER FLUSH (OPEN, FUNCTION)	22 ($0016)	($0089)	NONE

* This function is included for Release 1.0 compatibility of RSAM. It is not a valid function with the new access methods.

SET DEFAULT DRIVE/VOLUME	23 ($0017)	NONE	NONE
GET TRANSFER MODE (OPEN, FUNCTION)	41 ($0029)	NONE	ONE WORD

7.5.2 FILE ACCESS FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTIONS
1	SET TRANSFER MODE	This function sets the mode to be used for File I/O.
	Function Data	One word, integer
	Data = $0000 $0001	Fixed Length Record Variable Length Record
	Error Code	$0021 illegal data value
2	GET DISK FORMAT LEVEL NUMBER	This function is accepted at SYSIO-OPEN or SYSIO-FUNCTION. If formatted by 1.0 FORMAT, 1 is returned. If formatted 1.1 FORMAT, 2 is returned.
	Function Data	One word return area
	Error Code	None
5	DELETE FILE	This function packet is only accepted on a SYSIO-CLOSE. It causes the file that has been opened, to be deleted from the directory.
	Function Data	None
	Error Code	$008C file protected from delete.
12	GET DISK DEVICE NAME	This function returns the name of the physical device that contains the opened file, and its logical drive number.

	Function Data	Eight byte return area
	Error Code	None
14	STORE EOF OFFSET	This function is used with a SYSIO CLOSE of a file being accessed via the old access method (RSAM). It stores the end of the file offset for the last 252-byte block of the file. This is the number of valid bytes of data.
	Function Data	One word, integer.
	Error Code	None
17	POSITION TO BEGINNING OF FILE	This function causes a file to be "rewound" to the first record. It is accepted only through a SYSIO-FUNCTION call.
	Function Data	None
	Error Codes	None
18	POSITION TO END OF FILE	This function causes a file to be positioned just after the last record so subsequent sequential writes will be appended to the end of the file. It is accepted only through a SYSIO-FUNCTION call.
	Function Data	None
	Error Codes	None
19	CHANGE ACCESS ATTRIBUTES	This function changes the file access attribute word. It is accepted through SYSIO-OPEN, SYSIO-FUNCTION and SYSIO-CLOSE calls.
	Function Data	One word, integer.
	Data	See DIBACS

	Error Codes	None
20	RENAME FILE	This function changes the name of the opened file to the name supplied in the function packet.
	Function Data	One long word - address pointing to new filename which must be in the following format: 0-7 filename (left-justified and padded with blanks) 8-10 file extension (left-justified and padded with blanks) 11 0 in it 12-13 1 in it (version of file; only version in 1.1 is 1.) NOTE: This address is not checked. A DTAK trap error will occur if it is an invalid address.
	Error Codes	$008C file protected from rename.
21	SET EOF TO AFTER LAST RECORD WRITTEN	This function sets the end of file to the current position in the file, and performs a buffer flush function
	Function Data	None
	Error Codes	None
22	BUFFER FLUSH	This function writes out the current buffer if a change occurred, and updates the File Index and Backup File Index. It will not be accepted for Read-only files.
	Function Data	None
	Error Codes	$0089 Writing to read-only file

23	SET DEFAULT DRIVE/VOLUME	This function sets the default values for drive number and volume.
	Function Data	None
	Error Code	None

41	GET TRANSFER MODE	This function returns a code indicating the mode of file I/O transfers.
	Function Data	One word return area
	Data = $0000 $0001	Fixed Length Records Variable Length Records
	Error Code	None

7.6 SYSIO CALLS FOR ACCESSING FILES

1. SYSIO OPEN,lun,DIBname,errorlabel is the call to open a file. End of file, type and access information for an old file is returned in the DIB. If a file is to be created at Open and the same filename already exists, the existing copy is automatically deleted. Whether the records are fixed length or variable length is indicated by the user in DIBTRN.

2. SYSIO S/AREAD,lun,DTCBname,errorlabel is the call to read a record in the above named file. If the Data Transfer Control Block Record Number Field (DTCREC) is 0, the Sequential Access Method is used and the next record is transferred. If DTCREC is not zero, the Relative Record Access Method is invoked to read the indicated record and place it in the user's buffer which must be word aligned. The Relative Record Access Method is only for fixed length records with an even number of bytes and any attempt to use it for variable length records will result in an error return. The length of fixed length records must be equal to the Buffer Length field of the Data Transfer Control Block (DTCBFL). With either access method, an attempt to read past the end of the file will result in an error return. (Read "End-of-File Considerations" for specifics).

3. SYSIO S/AWRITE,lun,DTCname,errorlabel is the call to write a record in the above named file. Either the Sequential Access Method is used to extend the file and write another record or the Relative Record Access Method is to write over the specified record (extending the file if necessary).

4. SYSIO FUNCTION,lun,FKPTname,errorlabel is the call to execute a function packet.

5. SYSIO CLOSE,lun,FKPTname,errorlabel is the call to close the above opened file. FKPT name can be either 0 for no function packet or a function packet list.

7.7 END-OF-FILE CONSIDERATIONS

Recognizing the end-of-file when reading a file or insuring a correct end-of-file is established when writing a file are very important.

7.7.1 VARIABLE LENGTH SEQUENTIAL ACCESS METHOD

When reading a file in variable length sequential access method, the end-of-file indicator is a "read beyond end of file" (#RDBY equate in DKMCLBxx.INC or $85) error. See the coding example in appendix.

7.7.2 FIXED LENGTH ACCESS METHODS

When reading a file in Relative Record or Sequential fixed length a partial last record is possible. Some possible cases are:

a) accessing a file of unknown created record length or unknown access method

b) accessing a file with a different record length than was used when the file was created

c) accessing a file with sequential fixed length and changing the DTCBFL from transfer to transfer

d) accessing a file that was created with a partial last record (see note below)

When a partial record is transferred, no error is returned to the access method user. Therefore, the user cannot assume that a full buffer was transferred.

There are two methods to detect end-of-file involving partial records:

1. For each read:

   a) clear DTCBPT

   b) after the read is completed, use only the DTCBPT number of bytes in the buffer (do not assume DTCBPT equals DTCBFL)

   This method is the most universal and allows the "read beyond end of file" error to accurately indicate the end-of-file.

2. DTCBFL must remain constant. The DTCBFL value is placed into DIBRLG (record length) at OPEN. DIBNMR (number of full or partial records of DIBRLG length in the file) and DIBOFR (number of bytes in the last record) are returned by OPEN. Count the records as they are read sequentially, and when the last one (DIBNMR) is processed, only the number of good bytes (DIBOFR) should be processed. Note that if DTCBPT were zeroed before this read, DTCBPT would equal DIBOFR after the read completed.

   If reading randomly in Relative Record access method, if the record number being processed were equal to DIBNMR (last record in file) it would have to be handled as above.

   Note: To accomplish creating a partial last record while using Relative Record Access Method, the second-to-last record must be read or written, and then the last record is written using Sequential (fixed length) Access Method. So, the last record is written with DTCREC = 0 and DTCBFL = number of bytes.

7.8 ERROR CODES

The following codes are returned in hexadecimal to register D7.W.

CODE	MEANING
$0080	NO WRITING TO OLD STRUCTURED DISKETTE
$0081	FILE ALREADY OPEN FOR WRITE
$0082	OLD FILE NOT FOUND
$0083	ILLEGAL FILE ACCESS METHOD
$0084	INVALID FILENAME
$0085	READ BEYOND END OF FILE
$0086	BUFFER SIZE INCORRECT (RSAM ONLY)
$0087	FILE NOT EXTENDABLE
$0088	DISK FULL
$0089	WRITING TO READ ONLY FILE
$008A	WRONG FILE TYPE FOR THIS ACCESS
$008B	READ OR WRITE ILLEGAL SECTOR FOR FILE (RSAM ONLY)
$008C	FILE PROTECTED FROM DELETE/RENAME
$008D	READ ONLY ON NEW FILE
$008E	DIRECTORY & FILE INDEX HAVE INCORRECT NAME
$008F	NOT ENOUGH SPACE FOR CONTIGUOUS FILE

See disk driver chapters for additional I/O error codes.

7.9 EXAMPLES OF CONTROL BLOCKS

The following DIB is used to open an old file on volume "TEST01" for read/write and record length 256. To open the same file as new, DIBOPT would be $8800 and DIBRLG would be 0.

```
        DS.W        0
        DC.B        'TEST01'    DIBVOL
        DC.B        2           DIBDTD (read/write)
        DC.B        0           DIBTRN
        DC.L        0,0         DIBRSO
        DC.W        0           DIBOPT
        DC.L        0           DIBFCN
        DC.L        0           DIBBIO
        DC.W        0           DIBNMS
        DC.W        0           DIBOFS
        DC.L        0,0         DIBCAT
        DC.B        'MERRILY '  DIBNAM  Eight character file name,
```

```
                                        including spaces
DC.B            'ROW'       DIBEXT      Three character file name
                                        extension
DC.B            3           DIBTYP
DC.B            0,0         DIBACS, DIBRS1
DC.W            0,0,0       DIBVER, DIBFID
DC.W            256         DIBRLG
DC.W            0,0,0       DIBNMR, DIBOFR
DC.L            0,0         DIBIET, DIBSET
```

DTCB to read next variable length record from file into memory starting at $40000.

```
DS.W            0
DC.B            0           DTCSTA
DC.B            $0D         DTCTBU (upper delimiter)
DC.B            $04         DTCTBL (lower delimiter)
DC.B            0           DTCRSO
DC.L            $40000      DTCBFS (buffer address)
DC.W            150         DTCBFL (buffer length)
DC.W            0           DTCBPT (updated by file access)
DC.L            0           DTCREC (zero if variable length)
```

7.10 DATA STRUCTURE ON DISK

Both the hard and soft disks will be formatted to have 256 byte physical sectors with the first sector used for a Volume Label Sector.

The Volume Label contains pointers to a Bad Sector Table, Bit Map, Diagnostic Test Areas, File Index and Backup File Index. The File Index contains an entry (one 256-byte sector) for each file on the volume. Each entry contains the file name, descriptive information, and a list of extents used by the file. The first entry in the File Index describes the File Index itself, the second entry is for the Directory of the Volume and the third entry is for the Backup File Index.

The Master Directory is a file which contains filenames and pointers into the File Index.

For more information on these data structures, see Appendix E.

7.11 SPACE ALLOCATION ON THE DISK

Space is allocated to files in units called clusters which are groups of one or more contiguous sectors. The number of sectors in each cluster is determined when a volume is initialized and can vary from volume to volume. The system maintains a bit map on each volume to indicate which clusters are available and which are allocated.

7.11.1 DISK SPACE ALLOCATION ALGORITHMS

When a file has to be extended, the system will allocate the requested space from the first contiguous free space on the disk that is large enough to fulfill the request.

If the request cannot be satisfied, the largest available space will be assigned, in the case of an extendable file. For a contiguous file, the request will be aborted with an error code of $008D.

If the initial space request for an extendable file is not met and it becomes necessary to extend the file for the first time before it is closed, the system will attempt to allocate the larger of the amount needed to fulfill the initial request or the normal amount for additional extents.

When extending a file, the system attempts to enlarge the file's last extent before allocating a new discontinuous extent even if this means extending the file by less than the requested amount.

8.0 DISKETTE DRIVER

8.1 DRIVER DESCRIPTION

The diskette driver provides the user with a means of accessing or storing data on a diskette. Each diskette device can be accessed through a file I/O structure, or physically, for direct sector I/O and track formatting. This driver is sharable and supports asynchronous requests and an alternate device ID (volume identifier). To use the file structure, see File Access (Chapter 7).

The physical sector access scheme uses a Logical Sector Number which can range from 0 through the total number of available sectors on the diskette.

One to four diskette drives are supported, in any combination of 8-inch $5\frac{1}{4}$-inch type drives. All sector transfers use Direct Memory Access (DMA), and interrupts to provide a true asynchronous data transfer between diskette and memory. Before any I/O is started, a hardware 'door-open' status bit is tested. This provides the driver with a means of knowing whenever a diskette has been changed. If a diskette has been changed, or at start up time, the volume information sector is read. It contains data that describes physical characteristics of the diskette, including the volume identifier, sector size, density and number of sides. The 8-inch diskette volume information sector follows the IBM Diskette Standard.

8.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the Diskette Driver for physical access, the user must create a DIB using one of the device names specified in "DIBVOL". During an initialization routine a SYSIO-OPEN is performed to the device. Upon

successful completion all standard SYSIO operations described in Chapter 1 are allowed.

8.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #FD00, #FD01, #FD02, or #FD03.
DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional.
DIBTRN	DS.B 1	Enter 0 for fixed length transfers.
DIBRS0	DC.L 0,0	User sets this field to 0.
DIBOPT	DC.W 0	User sets this field to 0.
DIBFCN	DS.L 1	Insert pointer to function packet or set this field to $0000 0000 to select the default mode.
DIBBIO	DC.L 0	Not used by this driver. User sets this field to 0.

8.3 DISKETTE DRIVER DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand for all SYSIO synchronous and asynchronous I/O commands. The application program uses it to provide information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

8.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DC.B 0	Not used by this driver. User sets this field to 0.
DTCTBL	DC.B 0	Not used by this driver. User sets this field to 0.
DTCRS0	DC.B 0	This field is reserved. User sets this field to 0.
DTCBFS	DS.L 1	User puts Buffer starting address here (must be on a word boundary).
DTCBFL	DS.W 1	Buffer length. User puts in the sector size of the diskette (minimum buffer length accepted is 256).
DTCBPT	DS.W 1	Byte offset into buffer. DTCBPT will be incremented by the sector size after the I/O completes successfully.
DTCREC	DS.L 1	Sector number. Specifies the logical sector number being accessed.

8.4 DISKETTE DRIVER FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like setting the density or setting the number of sides on a diskette. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure the device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more words or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

8.4.1 SUMMARY OF DISKETTE FUNCTIONS

The functions listed below can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets. Many of these functions are used by programs attempting to access diskettes that are not recorded in the supported formats.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
GET LEVEL NUMBER	2 ($0002)	NONE	WORD
WRITE TRACK	3 ($0003)		WORD AND LONGWORD POINTER
SET SECTOR ORIGIN	6 ($0006)	$0026	WORD
SET SECTORS PER DISKETTE	7 ($0007)	NONE	LONGWORD
SET BYTES PER SECTOR	8 ($0008)	NONE	WORD
SET SECTORS PER TRACK	9 ($0009)	NONE	WORD
SET NUMBER OF SIDES	10 ($000A)	$002A	WORD
SET DENSITY	11 ($000B)	$002B	WORD
SET TRACKS PER DISKETTE	13 ($000D)	NONE	WORD
SET VOLUME NAME	15 ($000F)		THREE WORDS
REINITIALIZE	16 ($0010)		NONE
GET DISK TYPE	32 ($0020)	NONE	WORD
GET SECTOR ORIGIN	33 ($0021)	NONE	WORD
GET SECTORS PER DISKETTE	34 ($0022)	NONE	LONGWORD
GET BYTES PER SECTOR	35 ($0023)	NONE	WORD
GET SECTORS PER TRACK	36 ($0024)	NONE	WORD
GET NUMBER OF SIDES	37 ($0025)	NONE	WORD
GET DENSITY	38 ($0026)	NONE	WORD
GET TRACKS PER DISKETTE	39 ($0027)	NONE	WORD
GET VOLUME NAME	40 ($0028)	NONE	THREE WORDS

8.4.2 DISKETTE FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None

2	GET LEVEL NUMBER	This function returns the file structure level number of the diskette.
	Function Data	Leave space for one word, integer.
	Data = $0001	Linked Sectors (old 1.0 file access method)
	$0002	Bit Map Sector Allocation (new 1.1 file access method)
	Error Code	None

3	WRITE TRACK	This function is used for formatting a track of a diskette. It calculates the physical cylinder and side number from the logical track number (see Section 8.5) specified, and writes the data from the user buffer to that track.
	Function Data	One word and one longword, integer.
	Word 1 = $NNNN	Specifies logical track number.
	Longword = $NNNN NNNN	Points to user data buffer; maximum length 10500.
		Note: This address is not checked. A DTAK TRAP error will occur if it is an invalid address.
	Error Code	See Section 8.5

6	SET SECTOR ORIGIN	This function is used to specify the cylinder number containing the first available logical sector number on a diskette.
	Function Data	One word, integer.
	Data = $0000	Sets origin to cylinder 0.
	$0001	Sets origin to cylinder 1.

	Error Code=$0026	Data out of limits.
	Default Setting	$0000 5¼-inch diskette $0001 8-inch diskette
7	SET SECTORS PER DISKETTE	This function is used to specify the total number of accessible sectors on the diskette. (The IBM standard format for an 8-inch diskette reserves the sectors located on cylinders 0, 75 and 76.)
	Function Data	One long word, integer.
	Data = $NNNN NNNN	Specifies number of sectors on the diskette.
	Error Code	None
	Default Setting	2560 5¼-inch diskette 3848 8-inch diskette
8	SET BYTES PER SECTOR	This function is used to specify the number of bytes per sector.
	Function Data	One word, integer.
	Data = $NNNN	Specifies number of bytes per sector.
	Error Code	None.
	Default Setting	256
9	SET SECTORS PER TRACK	This function is used to specify the number of sectors per track.
	Function Data	One word, integer.
	Data = $NNNN	Specifies number of sectors per track.
	Error Code	None
	Default Setting	16 5¼-inch diskette 26 8-inch diskette

10	SET NUMBER OF SIDES	This function is used to specify the number of sides on a diskette.
	Function Data	One word, integer.
	Data = $0000	Specifies single-sided diskette.
	$0001	Specifies double-sided diskette.
	Error Code=$002A	Data out of limits.
	Default Setting	$0001
11	SET DENSITY	This function is used to specify the density of a diskette.
	Function Data	One word, integer.
	Data = $0000	Specifies single-density.
	$0001	Specifies double-density.
	Error Code=$0028	Data out of limits.
	Default Setting	$0001
13	SET TRACKS PER DISKETTE	This function sets the number of tracks on a diskette. The value is calculated by multiplying the number of cylinders by the number of sides on the diskette.
	Function Data	One word, integer.
	Data = $NNNN	Specifies logical number of tracks.
	Error Code	None.
	Default Setting	160 $5\frac{1}{4}$-inch diskette 154 8-inch diskette

15	SET VOLUME NAME	This function writes the specified Volume Identifier into the Diskette Volume Label Sector (LSN 0). It is written to the Backup Volume Label Sector if a file structure 2 format. If an 8" diskette, the IBM Standard Volume Sector (track 0, Sector 7) is also updated.
	Function Data	Six ASCII Bytes.
Data =	$NN NN NN NN NN NN	Volume identifier
	Error Code	See Section 8.5

16	REINITIALIZE	This function causes the driver to reconfigure the device according to the Volume Information Sector on the Diskette.
	Function Data	None
	Error Code	See Section 8.5.

32	GET DISK TYPE	This function returns a code word that indicates the type of drive/size of diskette.
	Function Data	Leave space for one word, integer.
	Data = $0001	$5\frac{1}{4}$-inch drive type
	$0002	8-inch drive type
	Error Code	None

33	GET SECTOR ORIGIN	This function returns the cylinder number containing the first available logical sector number on a diskette.
	Function Data	Leave space for one word, integer.
	Data = $0000	Origin is cylinder 0
	$0001	Origin is cylinder 1

	Error Code	None
34	GET SECTORS PER DISKETTE	This function returns the number of sectors on the diskette.
	Function Data	Leave space for one long word, integer.
	Data = $NNNN NNNN	
	Error Code	None
35	GET BYTES PER SECTOR	This function returns the number of bytes per sector.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	
	Error Code	None
36	GET SECTORS PER TRACK	This function returns the number of sectors on a track of a diskette.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	
	Error Code	None
37	GET NUMBER OF SIDES	This function returns the number of sides on a diskette.
	Function Data	Leave space for one word, integer.
	Data = $0000	Single-sided
	$0001	Double-sided
	Error Code	None

38	GET DENSITY	This function returns the density of a diskette.
	Function Data	Leave space for one word, integer.
	Data = $0000 $0001	Single-density Double-density
	Error Code	None

39	GET TRACKS PER DISKETTE	This function returns the total number of tracks on a diskette.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	
	Error Code	None

40	GET VOLUME NAME	This function returns the volume identifier of the diskette (one to six ASCII characters).
	Function Data	Leave space for 3 words
	Data = 'NNNNNN'	
	Error Code	None

8.5 DEFINITIONS

The LOGICAL SECTOR NUMBER is converted to a physical address according to the following algorithm; the Logical Sector Number is divided by the number of sectors per track. The remainder plus one becomes the physical sector number. The quotient is divided by the number of heads (sides). The remainder here gives the physical side (0 or 1), and the quotient plus the second origin (0 or 1) becomes the physical track number.

The LOGICAL TRACK NUMBER is converted to a physical cylinder and side according to the following algorithm; the Logical Track Number is divided by the number of heads (sides). The remainder gives the physical side (0 or 1), and the quotient is the track number.

8.6 ERROR CODES

The following codes are returned in hexadecimal to register D7.W. The number specified as $********* is the logical sector number being accessed when the error occurred.

CODE	MEANING
$0010	SEEK TRACK -- an error occurred while executing a track seek or restore (seek track 0) command.
$0011	LOGICAL SECTOR OR TRACK NUMBER TOO BIG $********* -- the logical sector or track number is larger than what the disk is configured for. This can occur if there is bad data in the Volume Information Section (LSN=0).
$0012	VOLUME CHANGED -- an I/O request was made and the volume identifier had changed since the file or device was opened.
$0013	Reserved
$0014	I/O REQUEST TIMED OUT -- the diskette sector transfer did not complete (the DMA end interrupt was not received.)
$0015	Reserved.
$0016	ILLEGAL BUFFER ADDRESS -- the address must be an even word boundary, and it cannot cross a 64K address boundary.
$0017	DISK FORMAT NOT RECOGNIZED -- the CS 9000 FORMAT program was not used on this diskette, or if an 8-inch diskette, the IBM Standard label does not exist.
$0018	DISK WRITE PROTECTED -- the notch is uncovered on an 8-inch diskette, or the notch is covered on a 5¼-inch diskette.
$0019	SECTOR BUFFER TOO SMALL -- the minimum length of a sector I/O buffer (DTCBFL) is 256 bytes.
$001A	Reserved
$001B	CRC ERROR $********* -- a CRC error was found in the ID or data field of the sector indicated.
$001C	SECTOR NOT FOUND $********* -- the sector indicated was not found.
$002F	ILLEGAL VOLUME IDENTIFIER

8.7 EXAMPLES OF CONTROL BLOCKS

DIB is used to open the diskette in drive 0 for read/write

```
DS.W        0
DC.B        '#FD00 '        DIBVOL
DC.B        2               DIBDTD (read/write)
DC.B        0               DIBTRN
DC.L        0,0             DIBRSO
DC.W        0               DIBOPT
DC.L        0               DIBFCN
DC.L        0               DIBBIO
```

DTCB to read logical sector 0 into memory starting at $40000

```
DS.W        0
DC.B        0               DTCSTA
DC.B        0               DTCTBU
DC.B        0               DTCTBL
DC.B        0               DTCRS0
DC.L        $40000          DTCBFS (buffer address)
DC.W        256             DTCBFL (buffer length)
DC.W        0               DTCBPT (updated by driver)
DC.L        0               DTCREC (logical sector number)
```

9.0 HARD DISK DRIVER

9.1 DRIVER DESCRIPTION

The hard disk driver provides the user with a means of accessing or storing data on a disk. Each disk drive can be accessed through a file I/O structure, or physically, for direct sector I/O. This driver is sharable and supports asynchronous requests and an alternate device ID (volume identifier). To use the file structure, see File Access (Chapter 7).

The physical sector access scheme uses a Logical Sector Number which can range from 0 through the total number of available sectors on the disk.

One to four disk drives are supported. All sector transfers use Direct Memory Access (DMA), and interrupts to provide a true asynchronous data transfer between disk and memory. At start up time, the volume information sector is read. It contains data that describes physical characteristics of the disk, including the volume identifier, sector size, and number of cylinders and heads.

9.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the Hard Disk Driver for physical access, the user must create a DIB using one of the device names specified in "DIBVOL". During an initialization routine a SYSIO-OPEN is performed to the device. Upon successful completion all standard SYSIO operations described in Chapter 1 are allowed.

9.3 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION
DIBVOL	DS.B 6	Device name. Use #HD00, #HD01, #HD02, or #HD03
DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional.
DIBTRN	DS.B 1	Enter 0 for fixed length transfers.
DIBRSO	DC.L 0,0	User sets this field to 0.
DIBOPT	DC.W 0	User sets this field to 0.
DIBFCN	DS.L 1	Insert pointer to function packet or set this field to $0000 0000 to select the default mode.
DIBBIO	DC.L 0	Not used by this driver. User sets this field to 0.

9.4 DISK DRIVER DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand for all SYSIO synchronous and asynchronous I/O commands. The application program uses it to provide information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

9.4.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DC.B 0	Not used by this driver. User sets this field to 0.

DTCTBL	DC.B 1	Not used by this driver. User sets this field to 0.
DTCRS0	DC.B 0	This field is reserved. User sets this field to 0.
DTCBFS	DS.L 1	User puts Buffer starting address here (must be on a word boundary).
DTCBFL	DS.W 1	Buffer length. User puts in the sector size of the disk (minimum buffer length accepted is 256).
DTCBPT	DS.W 1	Byte offset into buffer. DTCBPT will be incremented by the sector size after the I/O completes successfully.
DTCREC	DS.L 1	Sector number. Specifies the logical sector number being accessed.

9.5 DISK DRIVER FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. It is required for the FUNCTION command and optional for the OPEN command.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more words or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

9.5.1 SUMMARY OF DISK FUNCTIONS

The functions listed below can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
GET LEVEL NUMBER	2 ($0002)	NONE	WORD
GET DISK TYPE	32 ($0020)	NONE	WORD
GET SECTORS PER DISK	34 ($0022)	NONE	LONGWORD

GET BYTES PER SECTOR	35 ($0023)	NONE	WORD
GET VOLUME NAME	40 ($0028)	NONE	THREE WORDS

9.5.2 DISK FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
2	GET LEVEL NUMBER	This function returns the file structure level number of the disk.
	Function Data	Leave space for one word, integer.
	Data = $0002	Bit Map Sector Allocation (new 1.1 access method)
	Error Code	None
32	GET DISK TYPE	This function returns a code word that indicates the type of drive/size of disk.
	Function Data	Leave space for one word, integer.
	Data = $0004	10 MB hard disk
	Error Code	None
34	GET SECTORS PER DISK	This function returns the total number of sectors on the disk.
	Function Data	Leave space for one long word, integer.

	Data = $NNNN NNNN	
	Error Code	None
	Default Setting	39168
35	GET BYTES PER SECTOR	This function returns the number of bytes per sector.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	
	Error Code	None
	Default Setting	256
40	GET VOLUME NAME	This function returns the volume identifier of the disk (one to six ASCII characters).
	Function Data	Leave space for 3 words
	Data = 'NNNNNN'	
	Error Code	None

9.6 ERROR CODES

The following codes are returned in hexadecimal to register D7. The number specified as $********* is the Logical Sector Number being accessed when the error occurred.

CODE	MEANING
$0010	SEEK TRACK -- an error occurred during a Restore command.
$0011	LOGICAL SECTOR OR TRACK NUMBER TOO BIG $********* -- the Logical sector number is larger than what the disk is configured for. This can occur if there is bad data in

Volume Information Section (LSN=0).

$0017 DISK FORMAT NOT RECOGNIZED -- the CS 9000 FORMAT Program was not used on this disk.

$0019 SECTOR BUFFER TOO SMALL -- minimum length of a Sector I/O buffer (DTCBFL) is 256 bytes.

$001A WRITE FAULT $********* -- a Write Fault condition exists on the selected drive. It may be removed by de-selecting the drive and then re-selecting it.

$001B CRC ERROR $********* -- a CRC error was found in the ID field of the sector indicated.

$001C SECTOR NOT FOUND $********* -- an ID field for the sector indicated was not found.

$0070 DATA ADDRESS MARK NOT FOUND $********* -- during a Read Sector command, after successfully identifying the ID field, the Data Address mark was not detected within 16 bytes of the ID field.

$0072 ABORTED COMMAND -- a valid command was received that could not be executed based on status information from the drive. For example, if a write sector command had been issued while a Write Fault condition exists.

$0076 UNCORRECTABLE DISK ERROR $********* -- while reading the data field or the ECC check bits, an error was detected that the hard disk controller was not able to correct.

$0077 BAD SECTOR DETECTED $********* -- a Bad Block mark was detected in the indicated sector ID field.

$0078 HARD DISK CONTROLLER NOT PRESENT -- the hard disk controller board is not plugged into the system.

9.7 EXAMPLES OF CONTROL BLOCKS

DIB is used to open the hard disk drive 0 for read/write

```
DS.W        0
DC.B        '#HD00 '        DIBVOL
DC.B        2               DIBDTD (read/write)
DC.B        0               DIBTRN
DC.L        0,0             DIBRSO
DC.W        0               DIBOPT
DC.L        0               DIBFCN
DC.L        0               DIBBIO
```

DTCB to read logical sector 0 into memory starting at $40000

```
DS.W        0
DC.B        0               DTCSTA
DC.B        0               DTCTBU
```

```
DC.B        0           DTCTBL
DC.B        0           DTCRS0
DC.L        $40000      DTCBFS (buffer address)
DC.W        256         DTCBFL (buffer length)
DC.W        0           DTCBPT (updated by driver)
DC.L        0           DTCREC (logical sector number)
```

10.0 RS-232 ASYNCHRONOUS COMMUNICATIONS DRIVER

10.1 DRIVER DESCRIPTION

The RS-232 Asynchronous communications driver provides support for multiple RS-232 ports. Independent control of each port is achieved via separate control blocks and status save areas. The asynchronous port software is interrupt-driven. This allows I/O transactions to be of the "call and proceed" type, decoupling the calling task from the I/O rate of the port. Programmable time-out periods are provided for transmission and reception. The time-out will generate an interrupt which is reported to the user, thus preventing the possible "hanging" of the port from tying up the port beyond the time-out period. (See also Figure 10-1.)

The driver provides resident circular buffer space for both input and output. The buffers have a length of 256 characters. The input buffer will wrap-around if the buffer capacity is exceeded. This condition can be avoided by using either the XON/XOFF protocol or the DTR line enable/disable feature to halt input. These features must be activated via function calls after the port is opened. The user may specify buffers in his own space to be used instead of the default buffers, if buffers of a greater length or with different XON and XOFF limits are required.

All ports default to operate at 9600 Baud, with framing of 8 data bits and no parity and have a time-out of 30 seconds. Ports #SER00 and #SER01 are configured as Data Communications Equipment (DCE) and Terminals may be attached directly to the connectors. Port #SER02 is configured as a Data Terminal Equipment (DTE) and a Controller may be attached directly to the connector. Port #SER00 has Direct Memory Access (DMA) capability. DMA is invoked if #SER00 is in fixed-length transfer mode and if the record length is greater than a programmable limit called the DMA LIMIT. This limit is a port variable which can be set by function 36, and which has a default value of 256. There is no discernible difference to the user if DMA is used or not, but DMA dramatically reduces system overhead.

The port default characteristics are set following a system warmstart. OPENs and CLOSES do not effect the port characteristics or the buffers; a newly opened port may have junk (or valid) characters in the input buffer. A PORT RESET function 39 exists which resets the dynamic state of the port, including emptying the buffers, but does not alter the device characteristics. Any changes made to the port must be "unmade" if the standard configuration is to be restored. Any parameter that may be

changed (via a FUNCTION call) may also be queried. This allows a function packet to be built which contains the default characteristics.

For data input or output, the user must set up a Data Transfer Control Block (DTCB). See paragraph 10.3. This block indicates the start and length of the user buffer, Transfer Termination character range for variable length transfers, and a buffer offset pointer that indicates the current position within the user buffer. The port can send and receive records up to a length of 32767 bytes. When receiving records longer than the receiver circular buffer (256 character default length) data loss may occur as a function of other system activity. This is because the task-time portion of the driver is invoked to move characters into the user's buffer when the port circular buffer is at the XOFF point. (Default circular buffer XOFF point is when there is only room for 64 more characters.) Depending on the baud rate and other system activity, the port may be overrun. This can be prevented by using XON/XOFF or the DTR line to halt input.

SYSIO SWRITE, AWRITE, SREAD, and AREAD requests for the port are kept in a single queue - reads and writes are not queued separately. Therefore, a read request by one task may keep the port tied up while waiting for data to come in, and writes requests from another task will pile up behind it. (This does not occur for byte I/O because the requests do not go through SYSIO queuing.) There are several ways to avoid blocking the queue: one is to put the port in the character mode for the receiver, as opposed to the block (record) mode (see function 2), so that no waiting for an end-of-record is done. Another method is to monitor the receiver character count (see function 27) and not do an SREAD until a certain number of characters have come in. A third method involves activation of the RSC_ event(function 46), which is posted when the receiver buffer character count goes from 0 to 1 character. Using this method a task may SUSPEND itself and be awakened when the first character of a record comes in.

10.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver uses at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open an RS-232 port the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to one of the ports using one of the Device Names specified under DIBVOL. When this is done all standard SYSIO operations described in Chapter 1 are allowed.

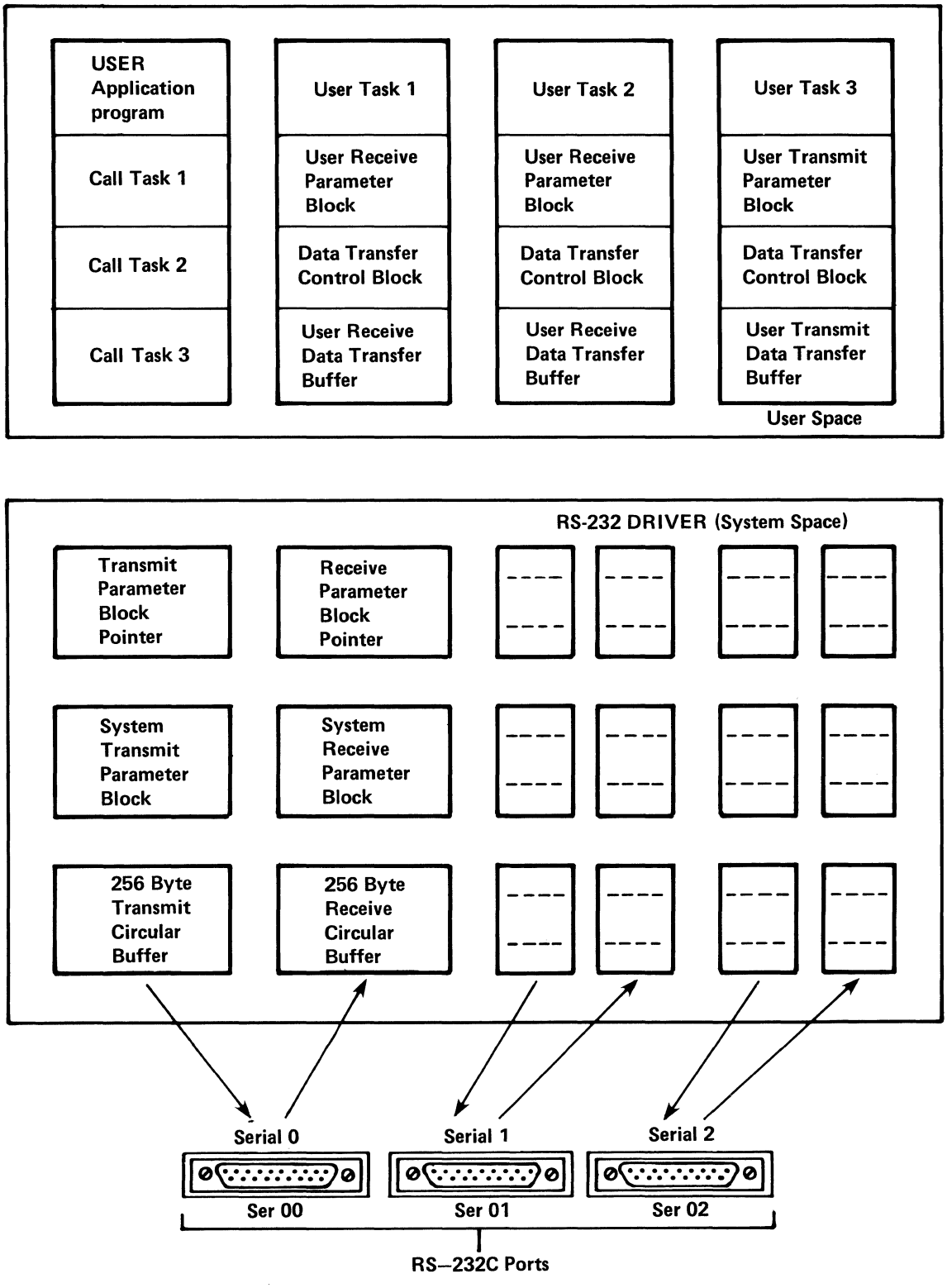


Figure 10-1. RS-232 Driver Block Diagram

10.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #SER00. #SER01, #SER02, #SER10, #SER11, #SER12 or #SER13.
DIBDTD	DS.B 1	Data Transfer Direction. Enter 0 for WRITE, 1 for READ or 2 for bidirectional.
DIBTRN	DS.B 1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRS0	DC.L 0,0	Not used. User sets this field to 0.
DIBOPT	DC.W 0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L 1	Insert pointer to function packet; null for default.
DIBBIO	DS.L 1	System used Byte I/O field. To OPEN the RS-232 driver for Byte I/O enter -1($FFFF FFFF), otherwise set it to zero. After OPEN the I/O manager fills this field with an identifier which is used for SYSIO-BREAD, SYSIO-BWRITE, and SYSIO-BTEST.

10.3 RS-232 DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READ's and WRITE's. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

10.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ/WRITE.

10.3.2 RETURN STATUS CODES

The following codes are returned in register D7.W and the low byte is placed in the DTCSTA field for transactions involving a DTC.

Code	Meaning
$0009	Variable length record longer than user buffer.
$000A	Device time-out
$000B	Transmit busy on BWRITE - transmit not done.
$000C	No character in receive buffer on BREAD.
$000D	Operation canceled.
$000E	Error in function call.
$0020	DTC buffer full before read (a request for 0 bytes).
$0030	Data suspect: parity error detected.
$0031	Data suspect: framing error detected.
$0032	Data lost: circular buffer overrun.Overrun can be prevented

by using XON/XOFF or the DTR line to regulate input.

$0033 Data lost: hardware overrun - too many hardware interrupts are occurring while receiving data.

$0034 BREAK received.

Note: Error conditions $0030 - $0034 are reset after being reported to the user.

10.4 RS-232 FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more word or longwords that send or receive the immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

10.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets. The error code mechanism is the common system code $000E, which provides the offset of the bad argument in the function packet.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	NONE	WORD
SET MODE BITS	2 ($0002)	NONE	TWO WORDS
GET MODE BITS	3 ($0003)	NONE	TWO WORDS RETURNED
SET TRANSMIT TIMEOUT	4 ($0004)	NONE	WORD
GET TRANSMIT TIMEOUT	5 ($0005)	NONE	WORD RETURNED
SET RECEIVE TIMEOUT	6 ($0006)	NONE	WORD
GET RECEIVE TIMEOUT	7 ($0007)	NONE	WORD RETURNED
SET TRANSMIT CONTROL	8 ($0008)	NONE	WORD

CHARACTER IN TABLE				
GET TRANSMIT CONTROL CHARACTER FROM TABLE	9	($0009)	NONE	WORD RETURNED
SET RECEIVE CONTROL CHARACTER IN TABLE	10	($000A)	NONE	WORD
GET RECEIVE CONTROL CHARACTER FROM TABLE	11	($000B)	NONE	WORD RETURNED
SET TRANSMIT CONTROL CHARACTER TABLE POINTER	16	($0010)	NONE	LONGWORD POINTER
SET RECEIVE CONTROL CHARACTER TABLE POINTER	18	($0012)	NONE	LONGWORD POINTER
SET BAUD	20	($0014)	NONE	WORD
GET BAUD	21	($0015)	NONE	WORD RETURNED
SET FRAMING	22	($0016)	NONE	WORD
GET FRAMING	23	($0017)	NONE	WORD RETURNED
SET TRANSMIT BUFFER PARAMETER BLOCK POINTER	24	($0018)	NONE	LONGWORD POINTER
GET TRANSMIT BUFFER PARAMETER BLOCK POINTER	25	($0019)	NONE	LONGWORD POINTER RETURNED
SET RECEIVER BUFFER PARAMETER BLOCK POINTER	26	($001A)	NONE	LONGWORD POINTER
GET RECEIVE BUFFER PARAMETER BLOCK POINTER	27	($001B)	NONE	LONGWORD POINTER RETURNED
SET TRANSMIT CHARACTER TRANSLATION SUBROUTINE POINTER	32	($0020)	NONE	LONGWORD POINTER
GET TRANSMIT CHARACTER TRANSLATION SUBROUTINE POINTER	33	($0021)	NONE	LONGWORD POINTER RETURNED
SET RECEIVE CHARACTER TRANSLATION SUBROUTINE POINTER	34	($0022)	NONE	LONGWORD POINTER
GET RECEIVE CHARACTER TRANSLATION SUBROUTINE POINTER	35	($0023)	NONE	LONGWORD POINTER RETURNED
SET DMA LIMIT	36	($0024)	NONE	WORD
GET DMA LIMIT	37	($0025)	NONE	WORD RETURNED
TRANSMIT BREAK	38	($0026)	NONE	WORD
RESET PORT	39	($0027)	NONE	NONE
SET MODEM CONTROL LINES	40	($0028)	NONE	WORD
GET MODEM CONTROL LINES	41	($0029)	NONE	WORD RETURNED
ATTACH BRK_ EVENT	44	($002C)	NONE	NONE
DETACH BRK_ EVENT	45	($002D)	NONE	NONE
ATTACH RSC_ EVENT	46	($002E)	NONE	NONE
DETACH RSC_ EVENT	47	($002F)	NONE	NONE
GET DIBTRN	48	($0030)	NONE	WORD RETURNED

10.4.2 RS-232 FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	Activates Transfer Termination Character checking to delimit a record that is being transferred.
	Function Data	One Word, integer.
	Data = $0000	Selects Fixed length transfer mode.
	$0001	Selects Variable length transfer mode.
	Error Code	None.
2	SET MODE BITS	Allows the user to modify the characteristics of the RS-232 port. The first word acts as a mask on the second word. The second word is formed using the word bit definitions. Since the first word acts as a mask, only the bits representing the parameters that it is desired to change are set. For example if you wish to enable the XON/XOFF mode for both transmit and receive bits 3 and 11 must be set to 1 in the mask word since these are the bits it is desired to change. The second word will have the same bits set to 1 to enable the mode, or to 0 to disable the mode.
	Function Data	One word integer
	Data Example	

```
          Word 1 - $0808   Bit mask     0000 1000 0000 1000

          Word 2 - $0808   Enable data  0000 1000 0000 1000
                   $0000   Disable data 0000 0000 0000 0000

        Error Code=$0022   Data out of limits.
```

RECEIVE MODE BITS

BIT NO.	SET TO	OPTION USE
15		Not used - set to zero.
14		Echo Enable
	0	Default - Do not echo character.
	1	Transmit echo of each received character.
13		Not used - set to zero.
12		Character Filter (Table)
	0	Default - Do not check for special characters.
	1	Check for special characters as defined by current entries in the receiver control character table.
11		XON/XOFF character checking
	0	Default - No XON/XOFF character will be sent.
	1	XON/XOFF characters will be sent to control input. The defined XOFF character will be sent when the high limit is reached in the receiver circular buffer. The defined XON character will be sent when the character count is at the low limit. These XON/XOFF limits are set by entries in the circular buffer parameter block. They cannot be changed when the default system circular buffer is used. (See function 26.)
10		Character or Block Mode Select
	0	Selects Character Mode - no wait for end-of-record will occur; the calling task will never be SUSPENDED. An incomplete record may be returned. The user must check the DTC upon return.
	1	Default - Selects Block Mode: the driver will

SUSPEND the calling task on an SREAD while waiting for a complete record.

9 Data Terminal Ready Control
- 0 Default - DTR line held active.
- 1 DTR line regulates input in the same way that XON/XOFF regulates it with character handshaking.

8 Record length
- 0 Fixed length records
- 1 Default - Variable length records

TRANSMIT MODE BITS

7 Not used
- 0 This bit must be set to zero.

6 Echo Enable
- 0 Default - No echoes expected.
- 1 Get echo after each character transmitted. The echoed character is placed in the receiver buffer. No comparison is made between the character sent and received.

5 Not used - set to zero.

4 Character Filter (Table)
- 0 Default - Do not check for special characters.
- 1 Check for special characters as defined by current entries in the transmit control character table.

3 XON/XOFF Enable
- 0 Default - The transmitter is not regulated by incoming XON/XOFF characters.
- 1 The transmitter is regulated by incoming XON/XOFF characters as defined by the current table entries.

2 Character or Block Mode Select
- 0 Default - Selects Character Mode
- 1 Selects Block Mode. Transmit option bit 4 must also be turned on to use this feature. Record transmission does not start until a "SEND" control character has been received for each record sent. Block mode cannot be used concurrently with the

		XON/XOFF transmitter mode.
1		Not used - set to zero.
0		Record Length
	0	Fixed length records
	1	Default - Variable length records.

3	GET MODE BITS	Returns the current option word in the second DATA word.
	Function Data	Leave space for two words, integer.
	Word 1 - $XXXX	Driver will set this word to $FFFF
	Word 2 - $NNNN	Specifies current option word. See paragraph 10.2.1 for bit definitions.
	Error Code	None

4	SET TRANSMIT TIMEOUT	Allows the user to specify the transmit time-out period in 50-ms increments. Period can be any positive number between $0001 and $7FFF or -1($FFFF) to specify an infinite time-out period.
	Function Data	One word, integer.
	Data - $NNNN	Specifies number of 50-ms periods.
	$FFFF	Specifies infinite time-out.
	Error Code	None.

5	GET TRANSMIT TIMEOUT	Returns the current Transmit time-out period. The DATA word specifies the number of 50-ms periods. $FFFF specifies an infinite time-out.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	Specifies number of 50-ms periods.
	$FFFF	Specifies infinite time-out.

	Error Code	None.
6	SET RECEIVE TIMEOUT	Allows the user to specify the Receive time-out period in 50-ms increments. Period can be any positive number between $0001 and $7FFF or -1($FFFF) to specify an infinite time-out period.
	Function Data	One word, integer.
	Data = $NNNN	Specifies number of 50-ms periods.
	$FFFF	Specifies infinite time-out.
	Error Code	None.
7	GET RECEIVE TIMEOUT	Returns current Receive time-out period. The DATA word specifies the number of 50-ms periods. $FFFF indicates an infinite time-out.
	Function Data	Leave space for one word, integer.
	Data = $NNNN	Specifies number of 50-ms periods.
	$FFFF	Indicates infinite time-out.
	Error Code	None.
8	SET TRANSMIT CONTROL CHARACTER IN TABLE	Allows the user to change entries in the TRANSMIT CONTROL CHARACTER TABLE. (See Table 10-2.) The most significant byte of the DATA word specifies the entry number in the table and the least significant byte specifies the character to be entered at that location.
	Function Data	One word, integer.
	Data Example $040D	Enter $0D at location 04 in the table.
	Error Code	None.

9	GET TRANSMIT CONTROL CHARACTER FROM TABLE	Returns the current entry in the TRANSMIT CONTROL-CHARACTER TABLE at the entry number in the table specified in the most significant byte of the DATA word.
	Function Data	Leave space for one word, integer.
	Data = $NNXX	$NN specifies table entry number. $XX is the current character at that location.
	Error Code	None.
10	SET RECEIVE CONTROL CHARACTER IN TABLE	Allows the user to change entries in the RECEIVE CONTROL-CHARACTER TABLE. (See Table 10-1.) The most significant byte of the DATA word specifics the entry number in the table and the least significant byte specifies the character to be entered at that location.
	Function Data	One word, integer.
	Example Data $0104	Enter $04 at location 01 in the table.
	Error Code	None.
11	GET RECEIVE CONTROL CHARACTER FROM TABLE	Returns the current entry in the RECEIVE CONTROL-CHARACTER TABLE at the entry number in the table specified in the most significant byte of the DATA word.
	Function Data	One word, integer.
	Data = $NNXX	$NN specifies the table entry number. $XX is the current character at that location.
	Error Code	None.
16	SET TRANSMIT CONTROL-CHARACTER TABLE POINTER	Allows the user to create and point to a user created TRANSMIT CONTROL-CHARACTER TABLE. This table will be copied from user

		space into the SYSTEM TRANSMIT CONTROL-CHARACTER TABLE and each entry number will have the meaning specified in Table 10-2. The longword pointer points to the beginning of the 16 byte user created table. If the pointer is set to zero then the default values shown in Table 10-2 will be copied back into the table.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to the beginning of the 16 byte table.
	$00000000	Copies default values into the system table.
	Error Code	None.
18	SET RECEIVE CONTROL-CHARACTER TABLE POINTER	Allows the user to create and point to a user created RECEIVE CONTROL-CHARACTER TABLE. This table will be copied from user space into the SYSTEM RECEIVE CONTROL-CHARACTER TABLE and each entry number will have the meaning specified in Table 10-1. The longword pointer points to the beginning of the 16 byte user created table. If the pointer is set to zero then the default values shown in Table 10-1 will be copied back into the table.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to the beginning of the 16 byte table.
	$00000000	Copies default values into the system table.
	Error Code	None.
20	SET BAUD	Allows the user to change the BAUD rate after OPEN time.
	Function Data	One word, integer.
	Data = $0000	45.5 Baud for ports #SER00,#SER01 and

		#SER02.
	$0000	3600 Baud for ports #SER10, #SER11, #SER12 and #SER13.
	$0001	50 Baud
	$0002	75 Baud
	$0003	110 Baud
	$0004	134.5 Baud
	$0005	150 Baud
	$0006	300 Baud
	$0007	600 Baud
	$0008	1200 Baud
	$0009	1800 Baud
	$000A	2000 Baud
	$000B	2400 Baud
	$000C	4800 Baud
	$000D	9600 Baud
	$000E	19200 Baud
	$000F	38400 Baud for ports #SER00, #SER01 and #SER02.
	$000F	7200 Baud for ports #SER10, #SER11, #SER12, and #SER13.
Error Code=$0034		Data out of limits.

21	GET BAUD	Returns a value representing the current baud rate in the DATA word. See Function 20 above for the baud rate equivalent of this value.
	Function Data	One word returned, integer.
	Data = $000N	Current setting of baud rate.
	Error Code	None

22	SET FRAMING	Allows the user to specify framing by creating a bit significant DATA word.
	Bits 15-8:	Not used. Set these bits to zero.
	Bits 7-6:	Specifies number of stop bits.
	01	1 stop bit.
	10	1.5 stop bits.
	11	2 stop bits. Default.

	Bits 5-4:	Specifies generated Parity bit
	00	No parity bit generated. Default.
	01	Odd parity generated.
	11	Even parity generated.
	Bits 3-2:	Specifies number of data bits.
	00	5 data bits
	01	6 data bits
	10	7 data bits
	11	8 data bits. Default.
	Bits 1-0:	Specifies Mark or Space Parity use with 8 bit data only. This option forces data bit 7 (highest order bit) to 0 for space parity and to 1 for mark parity. 8 data bits (bits 5-4 = 00) and no parity (bits 3-2 = 11) should also be specified.
	00	No parity. Default.
	01	Space parity (On transmit bit 7 is set to 0).
	11	Mark parity (On transmit bit 7 is set to 1) On Receive bit 7 is forced to 0.
	Data = $00FC	Specifies 2 stop bits, even parity and 8 data bits.
	Error Code	None.
23	GET FRAMING	Returns a bit significant word which indicates the current framing in use. See Function 22 for the significance of the bit combinations.
	Function Data	Leave space for one word, integer.
	Data = $00NN	Specifies current framing.
	Error Code	None
24	SET TRANSMIT BUFFER PARAMETER BLOCK POINTER	Allows the user to point to a Transmit Circular Buffer Parameter Block in user space. This parameter block will be pointed to by the driver and be used in the creation of a new circular buffer. If the DATA longword pointer is set to zero the

		driver points to the SYSTEM TRANSMIT CIRCULAR BUFFER PARAMETER BLOCK. See paragraph 10.4.3 for a description of the circular buffer parameter block.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to the User Transmit Circular Buffer Parameter Block.
	$0000 0000	Points to the System Transmit Circular Buffer.
	Error Code	None.
25	GET TRANSMIT BUFFER PARAMETER BLOCK POINTER	Returns the pointer to the System Transmit Buffer Parameter Block.
	Function Data	Leave space for one longword, integer.
	Data $NNNN NNNN	Points to System Transmit Buffer Parameter Block.
	Error Code	None.
26	SET RECEIVE BUFFER PARAMETER BLOCK POINTER	Allows the user to point to a Receive Circular Buffer Parameter Block in user space. This parameter block will be pointed to by the driver and be used in the creation of a new circular buffer. If the DATA longword pointer is set to zero the driver points to the System Receive Circular Buffer Parameter Block. See paragraph 10.4.3 for a description of the Circular Buffer Parameter Block.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to the User Receive Circular Buffer Parameter Block.
	$0000 0000	Points to the System Receive Circular Buffer Parameter Block.
	Error Code	None.

27	GET RECEIVE BUFFER PARAMETER BLOCK POINTER	Returns the pointer to the System Receive Buffer Parameter Block.
	Function Data	Leave space for one longword, integer.
	Data=$NNNN NNNN	Points to System Receive Buffer Parameter Block. This block can be read by a task in order to monitor the received character count.
	Error Code	None.
32	SET TRANSMIT CHARACTER TRANS-LATION SUBROUTINE POINTER.	Allows the user to point to a Transmit Character Translation Subroutine. See paragraph 10.4.4 for guidance in the creation of this user subroutine and particulars on how this driver will interact with it.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to a user created Transmit Character Translation Subroutine.
	Error Code	None.
33	GET TRANSMIT CHARACTER TRANS-LATION SUBROUTINE POINTER	Returns the pointer to the Transmit Character Translation Subroutine currently in use.
	Function Data	Leave space for one long word, integer.
	Data=$NNNN NNNN	Points to Transmit Character Translation Subroutine.
	Error Code	None
34	SET RECEIVE CHARACTER TRANS-LATION SUBROUTINE	Allows the user to point to a Receive Character Translation Subroutine. See paragraph 10.4.4 for guidance in the

	POINTER	creation of this user subroutine and particulars on how this driver will interact with it.
	Function Data	One longword, integer.
	Data=$NNNN NNNN	Points to a user created Receive Character Translation Subroutine.
	Error Code	None.
35	GET RECEIVE CHARACTER TRANS-LATION SUBROUTINE POINTER	Returns the pointer to the Receive Character Translation Subroutine currently in use.
	Function Data	One longword returned, integer.
	Data=$NNNN NNNN	Points to Receive Character Translation Subroutine.
	Error Code	None
36	SET DMA LIMIT	Allows the user to specify the buffer length, above which Serial Port #SER00 will use DMA to do fixed length transfers from the User Data Transfer Buffer to Serial Port #SER00. During DMA transfers character checking and character filtering are not in effect. The default length is 256 bytes.
	Function Data	One word, integer.
	Data = $NNNN	Specifies buffer length in bytes.
	Error Code	None.
37	GET DMA LIMIT	Returns the current buffer length above which DMA transfers will occur to Serial Port #SER00.
	Function Data	One word returned, integer.

	Error Code	None
38	TRANSMIT BREAK	Allows the user to specify the duration in 50-ms periods of the break condition. When using modems a break condition of this length may cause the modem to hang up. A break of shorter duration may be sent by dropping the transmitter baud rate and sending a NUL character.
	Function Data	One word, integer.
	Data = $NNNN	Specifies the number of 50-ms periods that the break condition will last.
	Error Code	None.
39	RESET PORT	This function clears the circular buffers and error conditions. If the XON/XOFF option is enabled, an XON will be sent. If the DTR option is enabled, the DTR line will be set to an active condition.
	Function data	None
	Error Code	None
40	SET MODEM CONTROL LINES	Allows the user to set the modem control lines in order to signal "Clear to Send (CTS)" or "Data Terminal Ready (DTR)". Only bits 0 and 1 in this word have assigned meaning.
	Function Data	One word, integer.
	Bit 0 = 1($0001)	Data Terminal Ready
	Bit 1 = 1($0002)	Clear to Send
	Error Code	None.

41	GET MODEM CONTROL LINES	Returns a bit significant word in which bits 0 and 1 signify "Clear to Send (CTS)", "Data Terminal Ready (DTR)" or "Data Carrier Detect (DCD)".
	Function Data	Leave space for one word, integer.
	Bit 0 = 1	Data Carrier Detect
	Bit 1 = 1	Data Set Ready for ports #SER00, #SER01 and #SER02.
	1 = 1	Clear to Send for ports #SER10, #SER11, #SER12 and #SER13.
	Error Code	None.

44	ATTACH BRK_ EVENT	Allows the user to use asynchronous event posting as described in paragraph 10.1.5. The events for each port have the names specified below. Each event is posted when a break condition is detected at the port.
	Function Data	None
	Error Code	None
	Port #SER00	BRK0
	#SER01	BRK1
	#SER02	BRK2
	#SER10	BRK3
	#SER11	BRK4
	#SER12	BRK5
	#SER13	BRK6

45	DETACH BRK_ EVENT	Detaches the break event named above from the port.
	Function Data	None
	Error Code	None

46	ATTACH RSC_ EVENT	Allows the user to use asynchronous event posting as described in paragraph 10.1.5. The events for each port have the names specified below. Each event is posted when the character count in the receiver circular buffer goes from 0 to 1 (empty to not empty).
	Function Data	None
	Error Code	None
	Port #SER00	RSC0
	#SER01	RSC1
	#SER02	RSC2
	#SER10	RSC3
	#SER11	RSC4
	#SER12	RSC5
	#SER13	RSC6

47	DETACH RSC_ EVENT	Detaches the break event named above from the port.
	Function Data	None
	Error Code	None

48	GET DIBTRN	The current transfer mode is returned.
	Function Data	Word returned.
	$0000	In fixed-length mode
	$0001	In variable-length mode
	Error Code	None

10.4.3 CIRCULAR BUFFER PARAMETER BLOCK

This parameter block format is used to create both transmit and receive User Circular Buffer Parameter Blocks which are used by the user transmit and receive buffers. The location of these blocks is specified with Function 24 (Transmit) and Function 26 (Receive). The location of the System Circular Buffer Parameter Blocks can be obtained with Function 25 (Transmit) and Function 27 (Receive). See paragraph 10.4.2 for a description of these functions.

Circular Buffer Parameter Block Format

DS.L 1 Buffer starting address.

DS.W 1 Buffer length.

DS.W 1 Extract pointer. (Note 1). Set to 0 by PORT RESET.

DS.W 1 Insert pointer. (Note 2). Set to 0 by PORT RESET.

DS.W 1 Character counter. Set to 0 by PORT RESET.

DS.W 1 The XON limit: the character count at which an XON will be sent if currently in an XOFF state.

DS.W 1 The XOFF limit: the character count at which an XOFF will be sent if currently in an XON state.

DS.B 1 This byte is set by the system to indicate buffer status. (See Note 3). Set to 0 by PORT RESET.

DC.B 0 Reserved byte.

Note 1. This pointer points to the next available character in the buffer. User initializes this pointer to $0000.
Note 2. This pointer points to the character beyond the last character in the buffer. User initializes this pointer to $0000.
Note 3. This bit significant byte indicates buffer status. The bit significance is shown below.

BIT NO. 7 Reserved

6 Set if counter is at high limit and bit 3 has just changed state.

5 Set if counter is at low limit and bit 3 has just

changed state.

4 Reserved

3 Set when counter is at high limit. Cleared when counter is at low limit. This is an XON or XOFF state indicator.

2 Set when buffer is full. 0 when buffer is not full.

1 Set when buffer is overrun. 0 when buffer not overrun.

0 Set when buffer contains characters. 0 when buffer is empty.

10.4.4 CHARACTER TRANSLATION SUBROUTINE

The character translation subroutine provides a place to perform general character processing of the input or output character stream. The subroutine is called as characters are moved between the user's data buffer and the port circular buffer. The user specifies the subroutine via FCN 34 for the receive side, and FCN 32 for the transmit side. Once the user subroutine has been specified, the driver will continue to call it as each character is transferred. Thus, it must remain in memory until the driver is informed (via data 0 longword for FCN 32 or 34) that it should not call the subroutine.

Useful examples of the subroutine would be to change lower case ASCII character to upper case, to delete or to insert linefeeds following carriage returns, to split 8-bit data characters into two printable ASCII characters, or to translate a non-standard end-of-record sequence into a single carriage return.

At entry into the subroutine D1.B contains the character from the data stream, and the higher order byte of D1.W is set to 0. If simple character substitution is desired, D1.B can be changed by the subroutine then a return-from-subroutine performed. If it is desired to delete the current character from the data stream, bit #8 of D1.W must be set (OR.W #$0100,D1) in the subroutine. If it is desired to insert an extra character, or characters, into the data stream after the current character, Bit #9 must be set (OR.W #$0200,D1). The subroutine must corrupt no other register than D1.

10.4.5 CONTROL CHARACTER TABLES

The RS-232 driver must know about several "special" characters during certain modes of operation. These characters are all kept together in lists called control character tables. There is a separate table for transmit characters and for receive characters. The operations associated with these control characters are usually performed just prior to character transmission, or just after character reception.

As an example, the XON and XOFF characters are kept at offsets 2 and 3, respectively, into the tables. If the XON/XOFF feature is enabled with FCN 2, bits 11 and 3, then the characters located at those positions in the tables will be sent (transmit table) or matched against incoming characters (receive table).

The rest of the characters in the table are activated by FCN 2, bits 12 and 4. These characters perform a few port control and data stream editing operations. For instance, some entries in the receive control character table specify characters which will be "thrown away" as they are received. This is a handy way of getting rid of "troublesome" characters.

The tables are searched sequentially by the driver in order to find matches. Searching stops after the first match is found, so the character cannot perform multiple functions. The user may specify new characters with FCNs 8 and 10.

Table 10-1. Receive-Control Characters

Table Entry Number	Character (Default Value is shown)	Meaning
0	$7F	Delete this character from the stream.
1	$7F	Delete this character from the stream.
2	$11	Receive this character for XON.
3	$13	Receive this character for XOFF.
4	$18	Clear the Transmit circular buffer.
5	$12	Clear the Receive circular buffer.
6	$04	Delete the last character from the Transmit buffer.
7	$7F	Delete the last character from the Receive buffer.
8	$01	Enable transmit to send record (Block Mode)
9		Unassigned
A	$14	Line turn-around character in half-duplex.
B		Unassigned
C	$7F	Delete this character from the stream.
D	$7F	Delete this character from the stream.
E		Unassigned
F		Unassigned

Table 10-2. Transmit-Control Characters

0		Unassigned.
1		Unassigned.
2	$11	Transmit this character for XON.
3	$13	Transmit this character for XOFF.
4		Unassigned.
5	$12	Clear the Receive circular buffer.
6	$04	Delete last character from the Transmit buffer.
7	$7F	Delete last character from the Receive buffer.
8		Unassigned.
9		Unassigned.
A	$14	Line turn around character in half-duplex.
B		Unassigned.
C	$7F	Delete this character from the stream.
D	$7F	Delete this character from the stream.
E		Unassigned.
F		Unassigned.

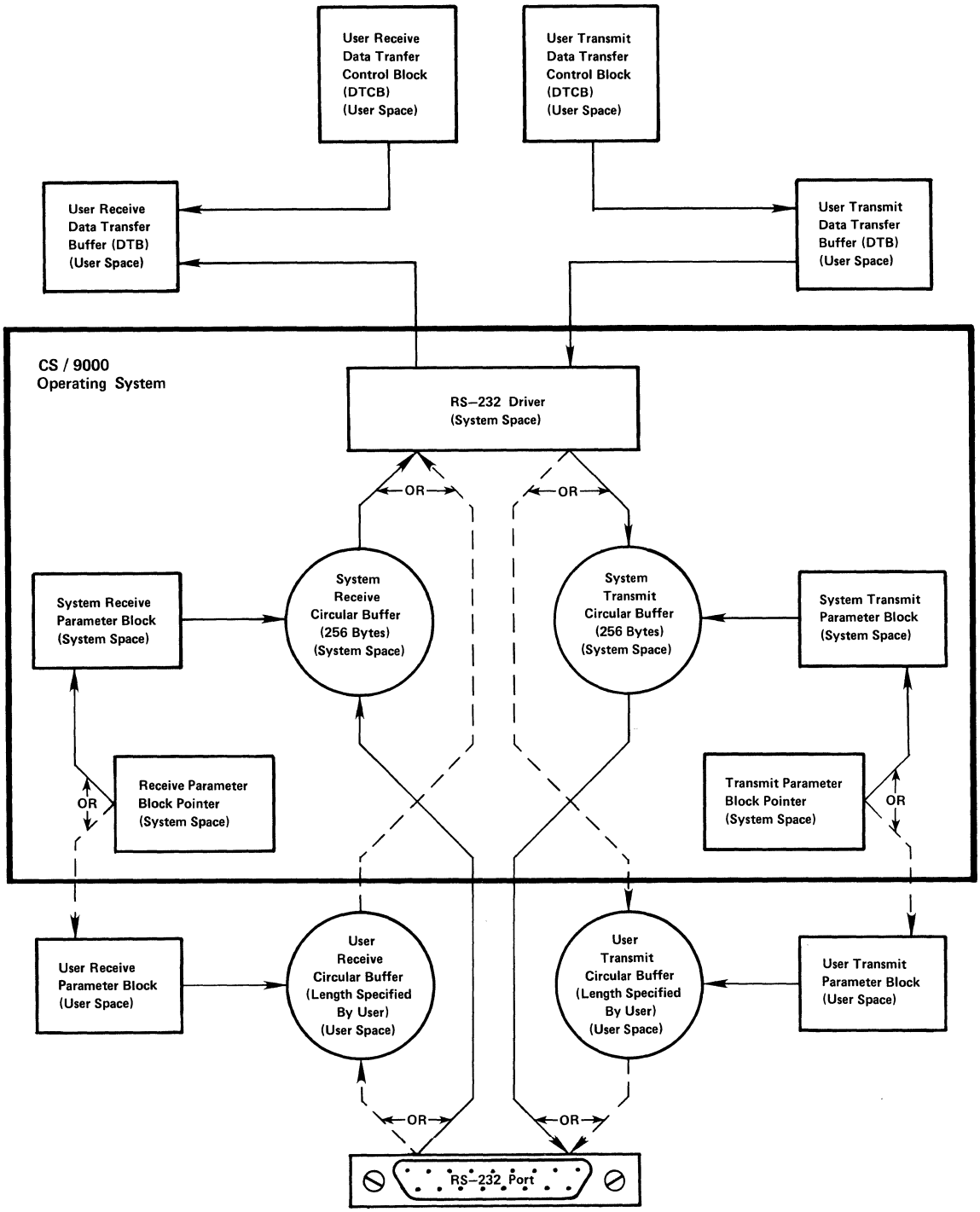


Figure 10-2. RS-232 Driver Overview Diagram

11.0 IEEE-488 INTERFACE DRIVER

11.1 DRIVER DESCRIPTION

The IEEE-488 Interface Driver acts as a single controller or talker/listener on the IEEE-488 General Purpose Interface Bus. This bus consists of 16 signal lines which are used to carry INTERFACE MESSAGES and DEVICE DEPENDENT MESSAGES among interconnected devices. The bus structure is organized into three groups of signal lines, consisting of 8 data lines, 5 control lines and 3 handshake lines. See Figure 11-1. This structure provides an unambiguous and effective communication link between interconnected devices. INTERFACE MESSAGES are used to manage the interface system and DEVICE DEPENDENT MESSAGES are used by the devices interconnected by the bus. Three organizational elements manage the flow of information exchanged among devices. This first element is a device with the ability to CONTROL. This device can command other devices to LISTEN or to TALK and can send INTERFACE MESSAGES which command specified actions to be taken by specified devices. The device is specified by means of an unique address. A device which has only the ability to control does not send or receive device dependent messages. The second element is a device with the ability to LISTEN. This device can be addressed by an INTERFACE MESSAGE and commanded to receive DEVICE DEPENDENT MESSAGES from another device connected to the bus. The third element is a device with the ability to TALK. This device can be addressed by an INTERFACE MESSAGE and commanded to send DEVICE DEPENDENT MESSAGES to another device connected to the bus.

As shown in Figure 11-1 these three elements of CONTROLLER, TALKER and LISTENER can occur individually or in combination in devices connected to the bus. In addition to the DEVICE DEPENDENT MESSAGES and INTERFACE MESSAGES mentioned above the bus may carry INTERFACE MESSAGES to accomplish serial polling, triggering or clearing. A serial poll sequence is initiated when a TALKER device requires some action by the CONTROLLER device. The TALKER transmits a service request message and the CONTROLLER will then obtain the "Status Byte" of all devices in sequence to ascertain which device requested service. Device triggering is the ability of a device to be TRIGGERED on command by the CONTROLLER and CLEARing (Function 19) initializes the device.

The IEEE-488 Interface Driver allows the user to implement these actions through the use of Function Commands. See Paragraph 11.4.1 for a summary of the available functions.

Each device of the IEEE-488 bus has a unique bus address, determined by 5 bits of the 8 bit wide data bus. Sometimes a device may have what is called extended addressing, in which an address is comprised of two successive 5 bit addresses. At any particular time one device is designated the bus controller. Other devices will have the ability to talk on the bus, or to listen on the bus, or both. Customarily the bus controller has both talker and listener capability in addition to the controller capability.

Each transaction on the IEEE-488 bus has two phases: the command phase, when the ATN line is active, and the data phase, when ATN is inactive. It is the bus controller's job to address the bus talker and the bus listeners while it drives ATN low. It then releases the ATN line, which allows the talker to place data on the bus, to be received by each actively addressed bus listener. The rate of data transmission over the bus is determined by the slowest listener, via the 3-wire handshake mechanism. There is no possibility of data overrun at the receiver end.

The bus controller must detect the end of the data transmission in order to regain control of the bus. Normally, this is done in one of two ways. In the first method, the talker drives the EOI line active along with the last data byte. In the second method the bus controller monitors the data and watches for a special EOS character. It then regains control of the bus after the EOS character. The bus controller also has the ability to seize control after any data byte (called taking control synchronously, TCS), or even mid-way in the data handshake (called taking control asynchronously, TCA).

Typically, the CS 9000 is a bus controller, which includes the ability to talk and to listen. Controlling, talking, and listening are three independent functions. The CS 9000 switches from bus controller to bus talker or listener by addressing itself during the bus command phase. Then, during the bus data phase, it sends or receives data. Because of this, it must have a bus address. The CS 9000 may also be just a talker/listener on the bus, with no controller capability. This also requires that the device have an address on the bus. The address is supplied when the CS 9000 is attached to the bus via the "ATCHDRV BUS" command, which requires one argument which is the device address, and a second argument which determines controller or talker/listener only configuration.

A program running on the CS 9000 may want to do I/O to an IEEE-488 device on the bus. In order to do this, the program must inform the operating system that there is such a device on the bus. This is done from the command line by using the ATCHDEV command. Then, the program is free to do OPENs, CLOSEs, SREADs, AREADs, SWRITEs, AWRITEs, and FUNCTIONs against the device. Byte I/O is not supported. When doing I/O to a remote bus device, the CS 9000, as controller, serves as a transparent node through which program tasks talk to the bus.

Because IEEE-488 device addresses are 5 bits only, printable ASCII characters are used as arguments in ATCHDRV and ATCHDEV, but are stripped down to 5 bits when placed on the bus. Thus, the ASCII letter 'A' or 'a' becomes $01, 'B' becomes $02, etc. Extended addressing is usually not used, and the extension address must be set to an invalid address as a flag to that effect. The character '?' does this.

```
#BUSA?          address $01, no extension address
#BUSB?          address $02, no extension address
#BUSCD          address $03, extension address $04
```

NOTE: For hardware implementation, see the "Computer System Technical Reference Manual."

11.1.1 BUS SEQUENCES

When the computer system is the bus controller, the following bus sequence is performed during a SYSIO SWRITE or AWRITE:

Universal Untalk
Universal Unlisten
My Talk Address (address of the computer system)
Other Listen Address (address of some other device)
Data transfer from user's Data Transfer Buffer, up to last byte
The EOI line is made active by the sender and the last data byte is sent. (Computer system regains control of the bus)

When the computer system is the bus controller, the following bus sequence is performed during an SREAD or SWRITE:

Universal Unlisten
Other Talk Address (Address of some other device)
My Listen Address (Address of computer system).
Data Transfer into user's Data Transfer Buffer, up to last byte
The EOI line is made active by the sender and the last byte is received.(Computer system regains control of the bus.)

The following Function Packet, issued to the CS 9000 as bus controller, will enable two TALKER/LISTENER devices to transfer data to each other:

COMMAND WORD	COMMAND DATA	FUNCTION PURPOSE
22 ($0016)	-	Send Unlisten Command
23 ($0017)	-	Send Untalk Command
16 ($0010)	$NNNN	Send other Talker Address

15 ($000F)	$NNNN	Send other Listener Address
25 ($0019)	-	Take control on end
0 ($0000)	-	Endlist

Note: In order for the computer system to regain control of the bus the TALKER must transfer the last data byte with the EOI line driven low (by the talker). The time-out interval for the controller must be long enough to allow the entire data transfer from device to device to occur. This time interval is a determined by the slowest device involved in the handshaking.

11.1.2 ASYNCHRONOUS EVENTS

As a CONTROLLER the IEEE-488 Interface driver supports asynchronous events that indicate the occurrence of an SERVICE REQUEST (SRQ) on the bus. When a TALKER/LISTENER forces an SRQ on the bus, the controller performs a parallel, or serial, poll sequence. It then signals the event as 'SQ--' where the last two characters are the device address. 'SQB?' is the event which indicates TALKER/LISTENER device 'B?' has issued a service request. In order for the SRQ event to the posted, the TALKER/LISTENER must be opened with bit 4 in the DIBOPT field set to 1, and Function 39, "Activate SRQ Event", must be executed to activate the event.

As a TALKER/LISTENER the IEEE-488 Interface driver supports two distinct events. One indicates being addressed as a TALKER (Function 35 'ACTIVATE TALKER') and the other indicates being addressed as a listener (Function 37 'ACTIVATE LISTENER'). The event names are 'TLKR' and 'LSNR' respectively.

11.1.3 REQUEST QUEUING

With multiple devices attached to the bus, access to the bus through the controller is on a first-come first-serve basis; the queuing is done by the I/O manager. The length of any particular bus transaction is a function of that device's response time, the length of the transfer, and eventually the time-out period specified for the device. Service requests are handled after the completion of the current transaction, even though other transactions may be waiting in the I/O queue. Byte I/O is not supported by the driver.

11.1.4 DEVICE DETACHMENT

The DTCHDEV command removes a device from the bus. No control structures are required from the user.

11.1.5 SERVICE REQUESTS

Service requests are handled by the controller, supporting both parallel and serial polling. The controller configures a device for parallel poll response if bits #5 and #4 are set in FUNCTION 42 data. If only bit #4 is set, the device is serviced by serial polling. If parallel polling is specified for the device, but all bit positions for parallel poll response are taken, the polling method will default to serial and an error code will be returned.

After the controller identifies the source of the service request and fetches the status byte, it will post the service request event for the task owning the device. The poll status byte will be transferred to the event block for use by the task.

11.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time. To open the IEEE-488 General Purpose Interface Bus the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. When this is done all standard SYSIO operations described in Chapter 1 are allowed.

11.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #BUS00 for IEEE-488 driver.
DIBDTD	DS.B 1	Data transfer direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional.

DIBTRN	DS.B 1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRS0	DC.L 0,0	Not used. User sets this field to 0.
DIBOPT	DC.W 0	Not used, set to 0.
DIBFCN	DS.L 1	Insert pointer to function packet or null for default.
DIBBIO	DC.L 0	Not used by this driver. User sets this field to 0.

11.3 IEEE-488 DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

The DTCTBU and DTCTBL fields are only used for determining the length of transmitted records while in variable-length mode. Received record lengths are determined by the bus protocol. The sender either drives EOI active on the last byte, or a single EOS character is detected by the receiver. Functions 32 and 42 contain additional information about the EOS character.

11.3.1 DTCB FORMAT

DATA MNEMONIC	LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.

DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ/WRITE.

11.4 IEEE-488 FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like issuing an interface clear command or disabling serial polling. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicated END-OF-LIST. The COMMAND word is followed by zero or more bytes, words or longwords that send or receive the immediate DATA for the command, or a long word that points to the DATA for that COMMAND.

All functions return the standard system error code of $000E, which also contains the offset to the bad element in the function packet.

11.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets. The (C) suffix in the following list indicates that the function is valid only for the CS 9000 as bus controller. The (T) and (L) suffixes show that the function is valid only for talkers or listeners, respectively.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	NONE	WORD
SEND UNLISTEN AND DEVICE CLEAR (C)	2 ($0002)	NONE	NONE
SEND EOI	3 ($0003)	NONE	NONE
SEND EOS	4 ($0004)	NONE	NONE
FINISH HANDSHAKE	5 ($0005)	NONE	NONE
GROUP EXECUTE TRIGGER (C)	6 ($0006)	NONE	NONE
GO TO LOCAL	7 ($0007)	NONE	NONE
GO TO STANDBY	8 ($0008)	NONE	NONE
SET HANDSHAKE MODE	9 ($0009)	NONE	WORD
SEND INTERFACE CLEAR (C)	10 ($000A)	NONE	NONE
SEND LOCAL LOCKOUT (C)	11 ($000B)	NONE	NONE
RESET REN LINE (C)	12 ($000C)	NONE	NONE
SEND MY LISTENER ADDRESS (C)	13 ($000D)	NONE	NONE
SEND MY TALKER ADDRESS (C)	14 ($000E)	NONE	NONE
SEND OTHER LISTENER ADDRESS (C)	15 ($000F)	NONE	WORD
SEND OTHER TALKER ADDRESS (C)	16 ($0010)	NONE	WORD
SET REN LINE (C)	17 ($0011)	NONE	NONE
SEND BYTE COMMAND (C)	18 ($0012)	NONE	WORD
SEND SELECTED DEVICE CLEAR (C)	19 ($0013)	NONE	NONE
SEND SERIAL POLL DISABLE (C)	20 ($0014)	NONE	NONE
SEND SERIAL POLL ENABLE (C)	21 ($0015)	NONE	NONE
SEND UNLISTEN COMMAND (C)	22 ($0016)	NONE	NONE
SEND UNTALK COMMAND (C)	23 ($0017)	NONE	NONE
GET DATA BYTE	24 ($0018)	NONE	WORD RETURNED
TAKE CONTROL ON END (C)	25 ($0019)	NONE	NONE
TAKE CONTROL SYNCHRONOUSLY (C)	26 ($001A)	NONE	NONE
SEND SERIAL POLL (C)	27 ($001B)	NONE	SEVERAL WORDS
TAKE CONTROL ASYNCHRONOUSLY (C)	28 ($001C)	NONE	NONE
DISABLE TIME-OUTS	29 ($001D)	NONE	NONE
SET TIME-OUT	30 ($001E)	NONE	WORD
SPECIFY EOS CHARACTER	32 ($0020)	NONE	WORD
SEND DATA BYTE	33 ($0021)	NONE	WORD
SET DMA LIMIT	34 ($0022)	NONE	WORD
ACTIVATE TALKER EVENT (T)	35 ($0023)	NONE	NONE
DEACTIVATE TALKER EVENT (T)	36 ($0024)	NONE	NONE
ACTIVATE LISTENER EVENT (L)	37 ($0025)	NONE	NONE
DEACTIVATE LISTENER EVENT (L)	38 ($0026)	NONE	NONE
ACTIVATE SRQ EVENT (C)	39 ($0027)	NONE	NONE
DEACTIVATE SRQ EVENT (C)	40 ($0028)	NONE	NONE
SRQ (REQUEST SERVICE) (T)	41 ($0029)	NONE	WORD
SET OPTIONS	42 ($002A)	NONE	NONE
GET OPTIONS	43 ($002B)	NONE	WORD RETURNED
GET TRANSFER MODE	44 ($002C)	NONE	WORD RETURNED
GET EOS CHARACTER	45 ($002D)	NONE	WORD RETURNED

GET DMA LIMIT	46 ($002E)	NONE	WORD RETURNED
GET HANDSHAKE MODE	47 ($002F)	NONE	WORD RETURNED
GET TIME-OUT	48 ($0030)	NONE	WORD RETURNED

11.4.2 IEEE-488 FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	This function allows the user to select fixed or variable length record transfers.
	Function Data	One word, integer.
	Data = $0000	Selects fixed length transfer mode.
	$0001	Selects variable length transfer mode.
	Error Code	None
2	SEND UNLISTEN AND DEVICE CLEAR	This function instructs the driver to issue an UNLISTEN followed by a universal DEVICE CLEAR.
	Function Code	None
	Error Code	None
3	SEND EOI	This function instructs the driver to send EOI on the next data byte transmitted but the driver must first be in the talker mode.
	Function Data	None

	Error Code	None
4	SEND EOS	This function instructs the driver to send the EOS character specified with Function 288
	Function Data	None
	Error Code	None
5	FINISH HANDSHAKE	This function instructs the driver to issue a local FINISH HANDSHAKE command.
	Function Data	None
	Error Code	None
6	GROUP EXECUTIVE TRIGGER	This function instructs the driver to issue a GROUP EXECUTE TRIGGER command to the bus.
	Function Data	None
	Error Code	None
7	GO TO LOCAL	This function instructs the driver to issue a GO TO LOCAL command to the bus.
	Function Code	None
	Error Code	None
8	GO TO STANDBY	This function instructs the driver to issue a local GO TO STANDBY.
	Function Data	None
	Error Code	None

9	SET HANDSHAKE MODE	This function instructs the driver to set the handshake mode to that specified by the DATA word. This function is used for debugging.
	Function Data	One word, integer.
	Data = $0000	Selects normal handshake mode.
	$0001	Selects RFD holdoff on all data mode.
	$0010	Selects RFD holdoff on END mode.
	$0011	Selects continuous mode.
	Error Code	None.
10	SEND INTERFACE CLEAR	This function instructs the driver to issue an INTERFACE CLEAR command to the bus for a duration of approximately 200 microseconds.
	Function Data	None
	Error Code	None
11	SEND LOCAL LOCKOUT	This function instructs the driver to issue a LOCAL LOCKOUT command to the bus. This will effect all listeners which are in the REMOTE state.
	Function Data	None
	Error Code	None
12	RESET REN LINE	This function instructs the driver to reset the REN line on the bus.
	Function Data	None
	Error Code	None
13	SEND MY LISTENER ADDRESS	This function instructs the driver to send the address of the "driver' to the bus as a listener device.

	Function Data	None
	Error Code	None

14	SEND MY TALKER ADDRESS	This function instructs the driver to send the address of the driver to the bus as a talker device.
	Function Data	None
	Error Code	None

15	SEND OTHER LISTENER ADDRESS	This function instructs the driver to send an OTHER LISTENER ADDRESS to the bus. Extended addressing is supported.
	Function Data	One word, integer.
	Data = $NNNN	Specifies the other listener address
	Error Code	None

16	SEND OTHER TALKER ADDRESS	This function instructs the driver to send an OTHER TALKER ADDRESS to the bus. Extended addressing is supported.
	Function Data	One word, integer.
	Data = $NNNN	Specifies the other talker address.
	Error Code	None

17	SET REN LINE	This function instructs the driver to set the REN line on the bus.
	Function Data	None
	Error Code	None

18	SEND BYTE COMMAND	This function instructs the driver to send a SEND BYTE command to the bus. The driver must be the bus controller. The driver will wait for previous handshaking to be completed before transmitting the command byte.
	Function Data	One word, integer.
	Data = $NNNN	Command byte.
	Error Code	None
19	SEND SELECTED DEVICE CLEAR	This function instructs the driver to send a SELECTED DEVICE CLEAR command to the bus.
	Function Data	None
	Error Code	None
20	SEND SERIAL POLL DISABLE	This function instructs the driver to send a SERIAL POLL DISABLE command to the bus.
	Function Data	None
	Error Code	None
21	SEND SERIAL POLL ENABLE	This function instructs the driver to send a SERIAL POLL ENABLE command to the bus.
	Function Data	None
	Error Code	None
22	SEND UNLISTEN COMMAND	This function instructs the driver to send an UNLISTEN command to the bus.
	Function Data	None
	Error Code	None

23	SEND UNTALK COMMAND	This function instructs the driver to send an UNTALK command to the bus.
	Function Data	None
	Error Code	None
24	GET DATA BYTE	This function instructs the driver to read a single byte from the bus. The bus talker and listeners including the driver must have been previously configured. If time-outs are enabled a time-out will be started.
	Function Data	One word returned.
	Data =$00NN	Space into which the driver places the byte.
	Error Code	None
25	TAKE CONTROL ON END	This function instructs the driver to take control of the bus following the next END message.
	Function Data	None
	Error Code	None
26	TAKE CONTROL SYNCHRONOUSLY	This function instructs the driver to TAKE CONTROL SYNCHRONOUSLY after the next data byte transfer.
	Function Data	None
	Error Code	None
27	SEND SERIAL POLL	This function instructs the driver to send a SERIAL POLL to a list of devices. The list is terminated with an illegal device address

		such as the printable characters "??". The device response is entered into the location in the Function Packet following the device address.
	Function Data	Several words following the COMMAND number.
	Word 1 $NNNN Word 2 $XXXX Word 3 $NNNN Word 4 $XXXX	Device address. Space for device status response. Device address. Space for device status response.
	Word N $3F3F	?? terminates the list.
	Error Code	None

28	TAKE CONTROL ASYNCHRONOUSLY	This function instructs the driver to TAKE CONTROL ASYNCHRONOUSLY regardless of the current bus state.
	Function Data	None
	Error Code	None

29	DISABLE TIME-OUTS	This function instructs the driver to disable time-outs.
	Function Data	None
	Error Code	None

30	SET TIME-OUT	This function instructs the driver to enable time-outs and specifies the time-out interval in the DATA word.
	Function Data	One word, integer.
	Data = $NNNN	Time-out period, number of 50-ms intervals.
	Error Code	None

32	SPECIFY EOS CHARACTER	This function allows the user to specify the EOS character for EOS generation and detection. The character must be in the least significant byte of the DATA word.
	Function Data	One word, integer.
	Data = $NN00	Specifies EOS character.
	Error Code	None.
33	SEND DATA BYTE	This function instructs the driver to send a data byte to the bus. The computer system must be configured as a talker. If time-outs are enabled a time-out will be started. The DATA word specifies the data byte to be transmitted.
	Function Data	One word, integer.
	Error Code	None.
34	SET DMA LIMIT	This function tells the driver the limit at which data transfers switch from direct I/O to DMA. If time-outs ae enabled while DMA is progress the time-out period is 50-ms multiplied by the record length.
	Function Data	One word, integer.
	Data = $NNNN	Specifies the limit at which DMA is in effect
	Error Code	None
35	ACTIVATE TALKER	This function instructs the driver to activate the "addressed-as-talker" event when the computer system is a talker/listener. The event name is 'TLKR'.
	Function Data	None

	Error Code	None
36	DEACTIVATE TALKER	This function instructs the driver to de-activate the "addressed as talker" event.
	Function Data	None
	Error Code	None
37	ACTIVATE LISTENER	This function instructs the driver to activate the "addressed as listener" event when the computer system is a talker/listener. The event name is 'LSNR'.
	Function Data	None
	Error Code	None
38	DEACTIVATE LISTENER	This function instructs the driver to de-activate the "address as listener" event.
	Function Code	None
	Error Code	None
39	ACTIVATE SRQ EVENT	This function instructs the driver to activate the SRQ event when the computer system is the bus controller. The event name is "SQ----" where the last two characters are filled in with the device bus address.
	Function Data	None
	Error Code	None
40	DEACTIVATE SRQ EVENT	This function instructs the driver to de-activate the SRQ event for a particular device.

	Function Data	None
	Error Code	None

41	SRQ (SERVICE REQUEST)	This function instructs the driver to send an SRQ to the bus while the computer system is talker/listener. The least significant byte of the DATA word is sent to the bus controller during SRQ polling.
	Function Data	One word, integer.
	Data = $00NN	Specifies status.
	Error Code	None.

42	SET OPTIONS	This function sets driver options.

Function Data

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word.
	1	Use option word.
14-6		Not used at present.
	0	User sets these bits to zero.
5		Parallel Polling
	0	No parallel polling.
	1	Configures device for parallel polling.
4		Service Request
	0	Device does not have service request capability.
	1	Device has service request capabilities.
3		Time-outs
	0	Time-outs are enabled.
	1	Time-outs are disabled.
2		EOS character size.
	0	7 bit EOS character

	1	8 bit EOS character.
1		EOS auto generation.
	0	EOS character is not auto generated.
	1	EOS character is auto generated.
0		EOS auto detection
	0	EOS auto detection disabled.
	1	EOS auto detection enabled.

Error Code None

---

43 GET OPTIONS Return data as specified above.

Function Data One word returned, integer.

Data = $00NN As above.

Error Code None.

---

44 GET DIBTRN Get the current transfer mode.

Function Data One word returned, integer.

Data = $0000 In fixed-length mode
= $0001 In variable-length mode

Error Code None.

---

45 GET EOS CHARACTER Get the current EOS character.

Function Data One word returned, integer.

Data = $000N The EOS character.

Error Code None.

---

46 GET DMA LIMIT Get the current DMA limit.

Function Data One word returned, integer.

	Data = $NNNN	Character count at which DMA is invoked.
	Error Code	None.
47	GET HANDSHAKE MODE	Get the current handshake mode.
	Function Data	One word returned, integer.
	Data = $000N	
	Error Code	None.
48	GET TIME-OUT	Get the current time-out interval.
	Function Data	One word returned, integer.
	Data = $NNNN	
	Error Code	None.

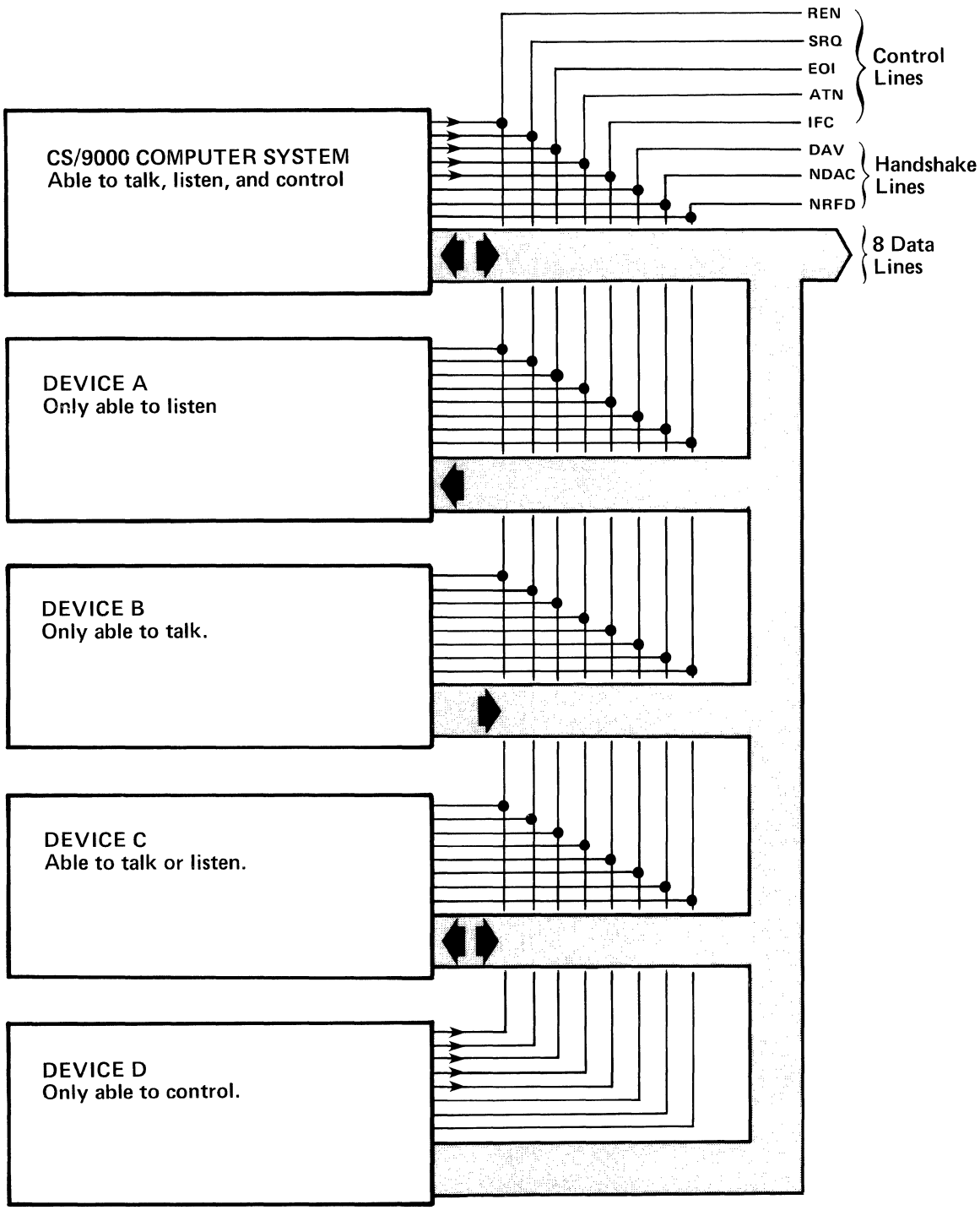


Figure 11-1. IEEE-488 Bus

12.0 PARALLEL PORT DRIVER

12.1 DRIVER DESCRIPTION

The parallel port is a bidirectional 8-bit interface with 2-line handshaking in each direction. The driver supports both fixed and variable length WRITEs and READs. Both asynchronous and synchronous operations are supported as well as byte I/O. The transfer mode may be changed from fixed or variable length with a standard Function packet. Standard Error Codes are returned by the driver for invalid control blocks or unsupported operations. See paragraph 12.5. The default device timeout period is five seconds. For specific hardware implementation information, see the "Computer System Technical Reference Manual".

The parallel port can be opened by only one task at a time.

The attributes of the parallel port driver are as follows:

(1) It is non-sharable
(2) It does not support asynchronous event posting
(3) It supports byte reads and byte writes but not byte tests
(4) It supports asynchronous I/O
(5) It does not support attach or detach device

The default characteristics of the parallel port at open time are as follows:

(1) Parallel printer mode (output strobed)
(2) Auto linefeed insertion on output
(3) Five second timeout
(4) Fixed-length transfers

12.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to the control block which is used at OPEN time to specify a non-default mode of operation for the device.

12.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #PPU.
DIBDTD	DS.B 1	Data Transfer Direction.
DIBTRN	DS.B 1	Use 0 for fixed length or 1 for variable length transfers.
DIBRS0	DC.L 0,0	User sets this field to 0.
DIBOPT	DC.W 0	Not supported by this driver. User sets this field to 0.
DIBFCN	DS.L 1	Insert pointer to function packet; null for default.
DIBBIO	DS.L 1	System used Byte I/O field. To open the Parallel Port driver for byte I/O enter -1 ($FFFF FFFF), otherwise set it to zero. After open the I/O manager fills this field with an identifier which is used for SYSIO-BWRITE, SYSIO-BTEST and SYSIO-BREAD.

12.3 PARALLEL PORT DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during transfers. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

12.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User monitors this field for status on I/O operation.
DTCTBU	DS.B 1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B 1	User puts lower limit to be used for Transfer Termination characters in Variable length transfers here.
DTCRS0	DC.B 0	This field is reserved. User sets this field to $00.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver after each byte transmitted. It should be reset after every WRITE.

12.4 PARALLEL PORT FUNCTIONS

The Function Packet Control Block provides for device specific operations. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The Function Packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero, indicating END-OF-LIST. The COMMAND word is followed by a word that sends the immediate DATA for the COMMAND.

12.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	$0021	WORD
GET TRANSFER MODE	2 ($0002)	$0022	WORD
SET PARALLEL PORT MODE	3 ($0003)	$0023	WORD
GET PARALLEL PORT MODE	4 ($0004)	$0024	WORD
SET TIMEOUT	5 ($0005)	$0025	WORD
GET TIMEOUT	6 ($0006)	$0026	WORD
ENABLE/DISABLE AUTO LINEFEED	7 ($0007)	$0027	WORD
GET AUTO LINEFEED	8 ($000F)	$0008	WORD

12.4.2 PARALLEL PORT FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates function packet processing.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	Activates Transfer Termination checking to delimit a buffer that is being transferred.
	Function Data	One word, integer.
	Data = 0	Fixed-length transfer mode.
	1	Variable-length transfer mode.
	Error Code $0021	Invalid transfer mode.

	Default =	Fixed-length transfers.
2	GET TRANSFER MODE	Get transfer mode, either fixed or variable. Leave space for 1 word following command.
	Data = 0	Fixed.
	1	Variable.
3	SET PARALLEL PORT MODE	Adjust hardware to support unidirectional or bidirectional mode.
	Data = 0	Parallel printer mode.
	1	Bidirectional.
	Error Code=$23	Invalid Parallel Port Mode.
	Default =	Parallel printer mode.
4	GET PARALLEL PORT MODE	Get hardware mode of port, either unidirectional or bidirectional. Leave space for 1 word following command.
	Data = 0	Parallel printer mode.
	1	Bidirectional.
5	SET TIMEOUT	Set number of 50 millisecond intervals driver will wait before assuming a timeout has occurred on the device.
	Data = $XXXX	Number of 50 millisecond intervals.
	$FFFF	Infinite timeout.
	Error Code=$25	Invalid timeout specified.
	Default =	$64 (5 seconds).

6	GET TIMEOUT	Gets timeout. Leave space for 1 word following command
	Data = $XXXX	Number of 50 millisecond intervals.
	$FFFF	Infinite timeout.
7	ENABLE/DISABLE AUTOLINEFEED	Enables or disables automatic linefeed insertion into output streams following carriage return ($0D characters.)
	Data = 0	Disable autolinefeed.
	1	Enable autolinefeed.
	Error Code=$27	Invalid autolinefeed specifier.
	Default =	Autolinefeed enabled.
8	GET AUTOLINEFEED	Gets autolinefeed setting, either 0 or 1. Leave space for 1 word following command.
	Data = 0	Autolinefeed disabled.
	1	Autolinefeed enabled.

Note that when the parallel port is opened for the parallel printer mode read operations are not permitted. In addition, the parallel printer output will not work in bidirectional mode because of timing characteristics of the port.

12.5 ERROR CODES

The following error codes are returned in hexadecimal to register D7.

Code	Meaning
$0005	READ ERROR (trying to read in the parallel printer mode).
$000A	DEVICE NOT READY.
$000B	BUFFER FULL ON WRITE BYTE.
$000E	INVALID CODE USED IN FUNCTION PACKET.
$000F	INVALID TRANSFER MODE SPECIFIED IN DIB.
$0021	INVALID VALUE FOR SET TRANSFER MODE DATA.
$0023	INVALID VALUE FOR SET PARALLEL PORT MODE DATA.
$0025	INVALID VALUE FOR SET TIMEOUT DATA.
$0027	INVALID VALUE FOR SET AUTOLINEFEED DATA.
$0068	BTEST IS NOT SUPPORTED BY THIS DRIVER.
$006A	ATTACH DEVICE IS NOT SUPPORTED BY THIS DRIVER.
$006B	DETACH DEVICE IS NOT SUPPORTED BY THIS DRIVER.

13.0 INTERTASK COMMUNICATION CHANNELS DRIVER

13.1 DRIVER DESCRIPTION

The Intertask Communication Channels are provided as a means of exchanging data between two concurrent tasks consistent with the I/O structure of the computer system. Each channel consists of two software (pseudo) devices, a Read Channel and Write Channel. Device names are #ITRXX for Write Channels and ITWXX for Read Channels where xx is the channel ID number. Communication can only occur between devices with the same channel ID number. For example device #ITW01 can write to device #ITR01. At coldstart the system attaches intertask channel 01 consisting of devices #ITR01 and #ITW01. Additional channels may be added using the ITCGEN System Call #58 and providing a unique two character printable ASCII channel ID number in register D0 and clearing register D1.W. Variable and fixed length block transfer requests in synchronous or asynchronous mode are supported. Time-out periods may be set via function packets. If a request cannot be completed, it will be suspended until it is completed or until it times out. Requests are queued on a first-in-first-out basis.

13.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open an intertask channel the user must create a DIB and within an initialization routine perform a SYSIO-OPEN using one of the Device Names specified under DIBVOL.

13.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION
DIBVOL	DS.B	6	Device name. Use #ITWXX or #ITRXX where XX represents the channel identifier.
DIBDTD	DS.B	1	Data Transfer Direction. Enter 0 for WRITE, 1 for READ.
DIBTRN	DS.B	1	Enter 0 for Fixed length or 1 for Variable length transfers.
DIBRSO	DC.L	0,0	User sets this field to 0.
DIBOPT	DC.W	0	Not used by this driver. User sets this field to 0.
DIBFCN	DS.L	1	Insert pointer to function packet; null for default.
DIBBIO	DC.L	0	Not used by this driver. User sets this field to 0.

13.3 DATA TRANSFER CONTROL BLOCK (DTCB)

The Data Transfer Control Block (DTCB) holds I/O status and buffer information during READ's and WRITE's. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer request after the request has been made.

13.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH		DESCRIPTION OF USE
DTCSTA	DS.B	1	User monitors this field for status on I/O operation. NOTE: During asynchronous operations, a DTSTA field greater than $2F indicates that the DTC buffer has been emptied and that the task is free to re-use the data area.
DTCTBU	DS.B	1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCTBL	DS.B	1	User puts upper limit to be used for Transfer Termination characters in Variable length transfer here.
DTCRSO	DC.B	0	This field is reserved. User puts zero here.
DTCBFS	DS.L	1	User puts Buffer starting address here.
DTCBFL	DS.W	1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W	1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ/WRITE.

13.4 INTERTASK COMMUNICATION CHANNEL FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more word or longwords that send or receive the

immediate DATA for the command, or a longword that points to the DATA for that COMMAND.

13.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET TRANSFER MODE	1 ($0001)	$0021	WORD
GET TRANSFER MODE	2 ($0002)	NONE	WORD
SET TIMEOUT	3 ($0003)	$0023	WORD
GET TIMEOUT	4 ($0004)	NONE	WORD

13.4.2 FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET TRANSFER MODE	Activates Transfer Termination Character checking to delimit a record that is being transferred.
	Function Data	One Word, integer.
	Data = $0000	Selects Fixed length transfer mode.
	$0001	Selects Variable length transfer mode.
	Error Code=$0021	Data out of limits.

2	GET TRANSFER MODE	Returns the current transfer mode in the data word.
	Function Data	One Word, integer.
	Error Code	None
3	SET TIMEOUT	Sets timeout period in units of 50 milliseconds.
	Function Data	One Word, integer.
	Data = $XXXX	Specifies number of 50 millisecond periods.
	Error Code=$0023	
4	GET TIMEOUT	Returns the current timeout period.
	Function Data	One Word, integer.
	Data = $XXXX	Specifies the current timeout period in 50 millisecond units.
	Error Code	None

13.5 ITC ERROR CODES

```
$0000     NO ERROR
$0006     ILLEGAL DATA TRANSFER DIRECTION IN DIB
$0009     VARIABLE RECORD EXCEEDS BUFFER LENGTH
$000D     REQUEST CANCELLED
$000E     FUNCTION NUMBER NOT SUPPORTED
$000F     ILLEGAL DATA TRANSFER MODE

$0010     DUPLICATE ITC IDENTIFIER
$0011     INVALID ITC IDENTIFIER
```

$0015	LOWER TRIGGER BYTE EXCEEDS UPPER TRIGGER BYTE
$0016	BUFFER OFFSET EXCEEDS BUFFER LENGTH
$0017	EITHER BUFFER LENGTH OR OFFSET ILLEGAL
$0018	NO AVAILABLE SYSTEM MEMORY
$0019	RETURN OF SYSTEM MEMORY FAILED
$0021	INVALID TRANSFER MODE
$0023	INVALID TIME OUT PARAMETER
$0062	READ NOT SUPPORTED FOR THIS DEVICE
$0063	WRITE NOT SUPPORTED FOR THIS DEVICE
$0065	WTBYTE NOT SUPPORTED FOR THIS DEVICE
$0066	RDBYTE NOT SUPPORTED FOR THIS DEVICE
$0068	TSTBYTE NOT SUPPORTED FOR THIS DEVICE
$006A	ATACHDEV NOT SUPPORTED FOR THIS DEVICE
$006B	DTACHDEV NOT SUPPORTED FOR THIS DEVICE
$0070	READ WAIT FAILED
$0071	WRITE WAIT FAILED
$0072	READ SIGNAL FAILED
$0073	WRITE SIGNAL FAILED
$0075	REQUEST TIMED OUT
$0076	CHANNEL(S) IN USE DRIVER WILL NOT BE DETACHED
$0077	ERROR IN COLDSTART SEQUENCE
$0078	ERROR IN DETACH SEQUENCE-DETACH INCOMPLETE

14.0 A/D CONVERTER DRIVER

14.1 DRIVER DESCRIPTION

The A/D converter driver allows a user to write application programs which collect data from A/D converter channels on the sensor I/O board. NOTE: The sensor I/O board option must be installed and certain hardware jumper options must be made to the board. The "ATCHDRV SENSOR" command must have been issued in order to use this driver. For hardware implementation, see the "Computer System Technical Reference Manual."

The A/D converter driver supports synchronous and asynchronous read requests. The Data Transfer Control Block (DTCB) specifies the buffer address and buffer length in bytes. The buffer address must be word aligned. The data returned to the buffer is in a scaled 32-bit integer format. The relationship between the returned value and the actual voltage is:

$$\text{Actual voltage (V)} = \frac{\text{returned value}}{1048576}$$

1,048,576 = 1 volt

The specified channels are sampled and stored in the buffer in sequential order. If the returned value is $7FFF FFFF an overrange condition has occurred in the positive direction. $10000001 indicates a negative overrange.

The default settings are 30 samples per second, unity gain, non-shared access, no attenuation, no alternate channels, no bias, no summing and no averaging.

14.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open an A/D Converter Channel the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. When this is done all standard SYSIO operations described in Chapter 1 are allowed, except byte I/O.

14.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #ADC00, #ADC01, #ADC02 or #ADC03.
DIBDTD	DC.B 1	Data Transfer Direction. Use 1. This device is READ only.
DIBTRN	DC.B 0	Enter 0 for fixed length.
DIBRS0	DC.L 0,0	User set this field to 0.
DIBOPT	DS.W 1	Insert option word described in paragraph 14.2.2.
DIBFCN	DS.L 1	Insert pointer to function packet; null for default.
DIBBIO	DC.L 0	Not used. User sets this field to 0.

14.2.2 DIB OPTION WORD BIT DEFINITIONS

If the option word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table.

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word
	1	Use option word
14-10		Not currently used
	0	Set to zero

9		Attenuation (Note: A hardware jumper must be installed)
	0	No attenuation
	1	Attenuation on
8		Shared Access
	0	Only one "OPEN" at a time
	1	More than one "OPEN" at a time.
7-0		Not currently used.
	0	Set to zero.

14.3 A/D CONVERTER DATA TRANSFER CONTROL BLOCK (DTCB)

The data transfer control block (DTCB) holds I/O status and buffer information during READs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User looks here for status on I/O operation.
DTCTBU	DC.B 0	Not used by A/D Driver.
DTCTBL	DC.B 0	Not used by A/D Driver.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.L 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ.

14.4 A/D CONVERTER FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like specifying alternate channel sampling or shared access. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more bytes, words or longwords that send or receive the immediate DATA for the command, or a long word that points to the DATA for that COMMAND.

14.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET GAIN	1 ($0001)	$0021	WORD
SET SAMPLERATE	2 ($0002)	$0022	LONG WORD
AVERAGING	3 ($0003)	$0023	WORD
ALTERNATE CHANNELS	4 ($0004)	$0024	SEVERAL WORDS
PASS SUM	5 ($0005)	$0025	WORD
SET BIAS	6 ($0006)	NONE	LONG WORD

14.4.2 A/D CONVERTER FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET GAIN	Allows the user to select AUTORANGE mode in which the A/D converter self-adjusts its gain to an optimum scale to make the reading or to FIXED GAIN mode in which the gain may be fixed at 1, 4, 32 or 256.
	Function Data	One word, integer.
	Data =$0000	Specifies AUTO RANGE
	$0001	Gain of 1
	$0004	Gain of 4
	$0020	Gain of 32
	$0100	Gain of 256
	Error Code=$0021	Data out of limits.
2	SET SAMPLE RATE	Allows the user to specify the sample rate in microseconds (up to 33,333ms). The hardware has a maximum sample rate of 30 samples per second.
	Function Data	One long word, integer
	Error Code $0022	Data out of limits.
3	AVERAGING	If a sample rate slower than 30 samples per second is selected the user may use this function to specify that the data returned to the application program be the average of all the data converted during the sample period.

	Function Data	One word, integer.
	Data = $0000	Averaging is not used. The value returned is that of the last sample taken in the period.
	$0001	Averaging is in effect.
	Error Code=$0023	Data out of limits.

4	ALTERNATE CHANNELS	This function allows data to be collected from more than one A/D converter on the same Sensor I/O Board. The DATA is a list of words, each word specifies the channel that is to read during that sample period.
	Function Data	Several words ending with $FFFF
	Data = $0003	Read channel 3 in this time period.
	$0000	Read channel 0 in this time period.
	$0002	Read channel 2 in this time period.
	$0001	Read channel 1 in this time period.
	$FFFF	Specifies end of list.
	Error Code $0024	Data out of limits.

5	PASS SUM	This function allows the user to specify that the sum of all the data converted during a sample period is to be returned to the application program. This function requires that averaging be specified also.
	Function Data	One word, integer.
	Data = $0000	Data is not summed.
	$0001	Data is summed.
	Error Code $0025	Data out of limits.

6	BIAS	This function causes a fixed value specified by the DATA long word to be added as a BIAS to the data returned. The value must be a scaled value. One volt is 1,048,576 or

	$00100000. Two volts bias would be $00200000, three volts $00300000, etc.
Function Data	One long word, integer.
Data $NNNN NNNN	Specifies value of the BIAS.
Data Example =	$00A00000 is a bias of 10 volts.
Error Code	None

14.5 ERROR CODES

The following error codes have been defined and may be returned by this driver.

ERROR CODE	MEANING
16 ($0010)	DEVICE LOCKED. (OPEN ATTEMPTED AGAINST EXISTING NON-SHARED OPEN.)
17 ($0011)	DEVICE ALREADY OPEN. (NON-SHARED OPEN ATTEMPTED AGAINST AN EXISTING SHARED OPEN.)
18 ($0012)	A/D OVERRANGE.

15.0 SWITCH INPUT DRIVER

15.1 DRIVER DESCRIPTION

The switch input driver allows an application program to detect the momentary closure of one of the switch inputs on the sensor I/O board. NOTE: The sensor I/O board option must be installed and the "ATCHDRV SENSOR" Command must have been issued in order to use this driver. (For hardware implementation, see the "Computer System Technical Reference Manual.") Each switch is viewed as a separate device with device names of #SWI00 thru #SWI07. Figure 15-1 shows the relationship between channel numbers, start-stop designations and device names. Also note from Figure 15-1 that the status registers are all reset after every READ. This means that the READ will be destructive to the data contained in other switch registers. In addition the switched input should be a momentary closure as shown in the "Typical Switch Input" in Figure 15-1. The momentary closure is detected and latched by the data latch. The output of the data latch will be a steady high or low which will be passed through the data register when read and reset by the READ. The driver supports byte, synchronous and asynchronous READ requests but only one value is returned, 0 for OPEN or 1 for CLOSED. For synchronous and asynchronous READ requests the driver will wait for the next time that a switch is pressed and at that time the driver puts a 1 in the buffer and the request is terminated. Specifying #SWI08 at OPEN causes the driver to return a bit significant byte in which the bits numbers correspond directly with the switch numbers. 11110110 ($F6) would indicate that all switches except #SWI00 and #SWI03 were closed. Asynchronous I/O is only allowed on #SWI08.

15.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open a switch input the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the

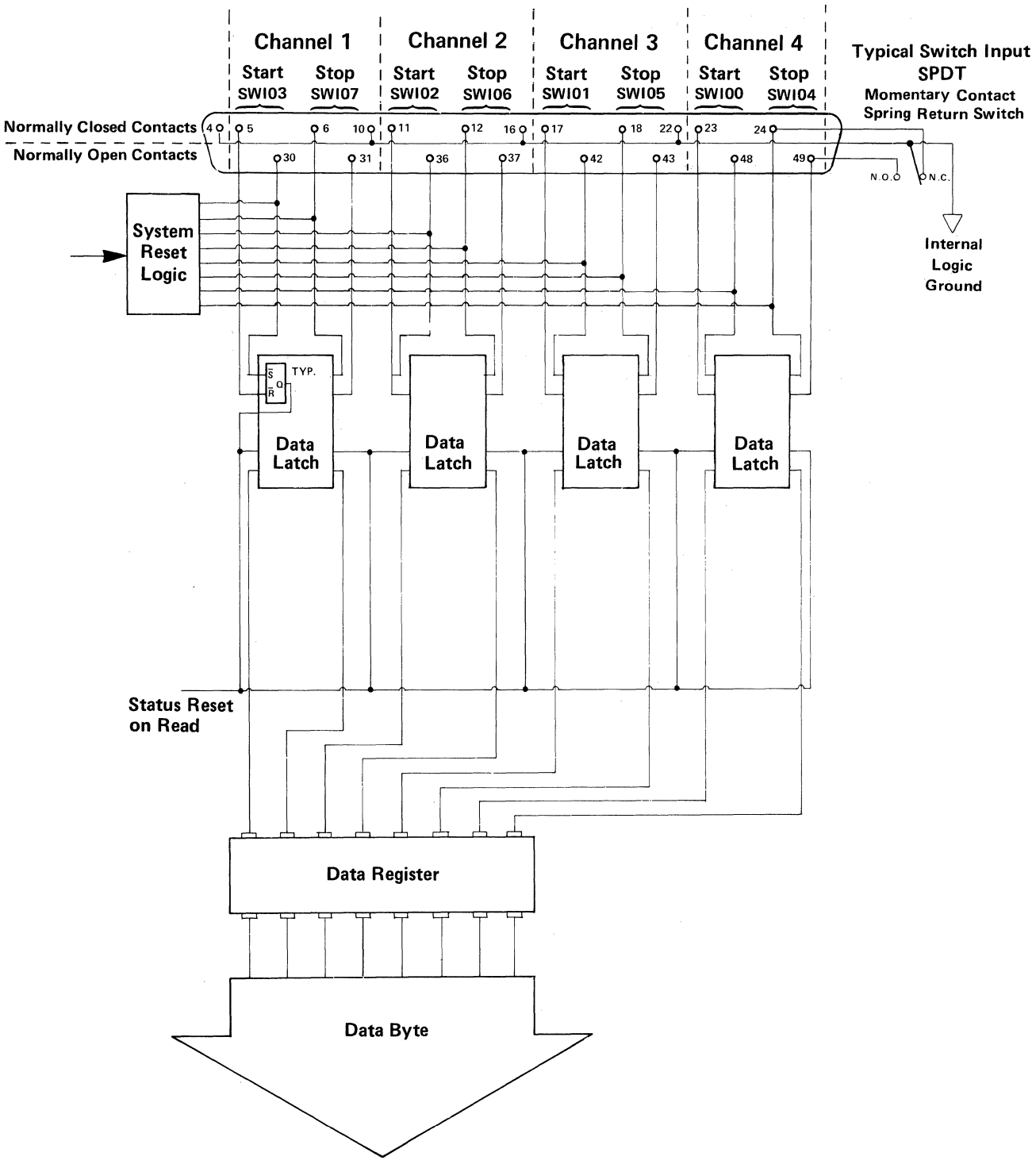

Channel 1
Channel 2
Channel 3
Channel 4
Start
Stop
SWI03
SWI07
SWI02
SWI06
SWI01
SWI05
SWI00
SWI04
Typical Switch Input
SPDT
Momentary Contact
Spring Return Switch
Normally Closed Contacts
Normally Open Contacts
4
5
6
10
11
12
16
17
18
22
23
24
30
31
36
37
42
43
48
49
N.O.
N.C.
System
Reset
Logic
Internal
Logic
Ground
S
Q
R
TYP.
Data
Latch
Status Reset
on Read
Data Register
Data Byte

Figure 15-1.

15.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #SWI00, #SWI01, #SWI02, #SWI03, #SWI04, #SWI05, #SWI06 or #SWI07 or #SWI08.
DIBDTD	DC.B 1	Data transfer direction. Use 1. This driver is READ only.
DIBTRN	DC.B 0	Enter 0 for Fixed length.
DIBRS0	DC.L 0,0	Set this field to 0.
DIBOPT	DC.W 0	Insert option word described in paragraph 15.2.2.
DIBFCN	DC.L 0	Not used by this driver. Fill space with null.
DIBBIO	DS.L 1	System used Byte I/O field. To open the switch input for Byte I/O enter -1 ($FFFF FFFF), otherwise set it to zero. After open the I/O manager fills this field with an identifier which is used for SYSIO-BREAD.

15.2.2 DIB OPTION WORD BIT DEFINITIONS

If the option word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table.

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word
	1	Use option word
14-9		Not currently used
	0	Set to zero
8		Shared Access
	0	Only one "OPEN" at a time

	1	More than one "OPEN" at a time
7-0		Not currently used
	0	Set to zero

15.3 SWITCH INPUT DATA TRANSFER CONTROL BLOCK (DTCB)

The data transfer control block (DTCB) holds I/O status and buffer information during READs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

15.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User looks here for status on I/O operation.
DTCTBU	DC.B 0	Not used by this driver. Set to 0.
DTCTBL	DC.B 0	Not used by this driver. Set to 0.
DTCRS0	DC.B 0	This field is reserved. User sets this field to 0.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ.

15.4 SWITCH INPUT FUNCTIONS

This driver does not use Function Packets.

15.5 ERROR CODES

The following error codes have been defined and may be returned by this driver.

ERROR CODE	MEANING
16 ($0010)	DEVICE LOCKED. (OPEN ATTEMPTED AGAINST EXISTING NON-SHARED OPEN.)
17 ($0011)	DEVICE ALREADY OPEN. (NON-SHARED OPEN ATTEMPTED AGAINST AN EXISTING SHARED OPEN.)

16.0 LED OUTPUT DRIVER

16.1 DRIVER DESCRIPTION

The LED output driver allows the user to write application programs which switch TTL level outputs on eight output lines which appear on connector JA3 of the sensor I/O board.

Note: The sensor I/O board option must be installed and the "ATCHDRV SENSOR" command must be invoked in order to use this driver. For hardware implementation, see the "Computer System Technical Reference Manual."

Only Byte I/O output using SYSIO BWRITE is supported by this driver. Each output line is viewed by the driver as a separate device with device names of #LED00 through #LED07. A byte output of zero is used to output a logic 0 and any non-zero byte output will output a logic 1 (TTL levels). Each of the outputs is latched by a 74LS174 Data Latch and driven by an Am2946 Bus Driver. See Figure 16-1 for the specific pinouts on JA3 and note that eight 270 ohm pullup resistors have been made available to nearby pins on JA3. These resistors are provided for LED current limiting. Four pins are also provided to connect logic ground with an external logic ground so that an external 5 volt voltage source can be used to drive a 5 volt relay. This source must be capable of providing enough current to drive the relay and the relay must be diode shunted as shown or damage to the bus driver may occur.

16.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open a LED-OUTPUT Channel the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. When this is done the SYSIO BWRITE operation may be used.

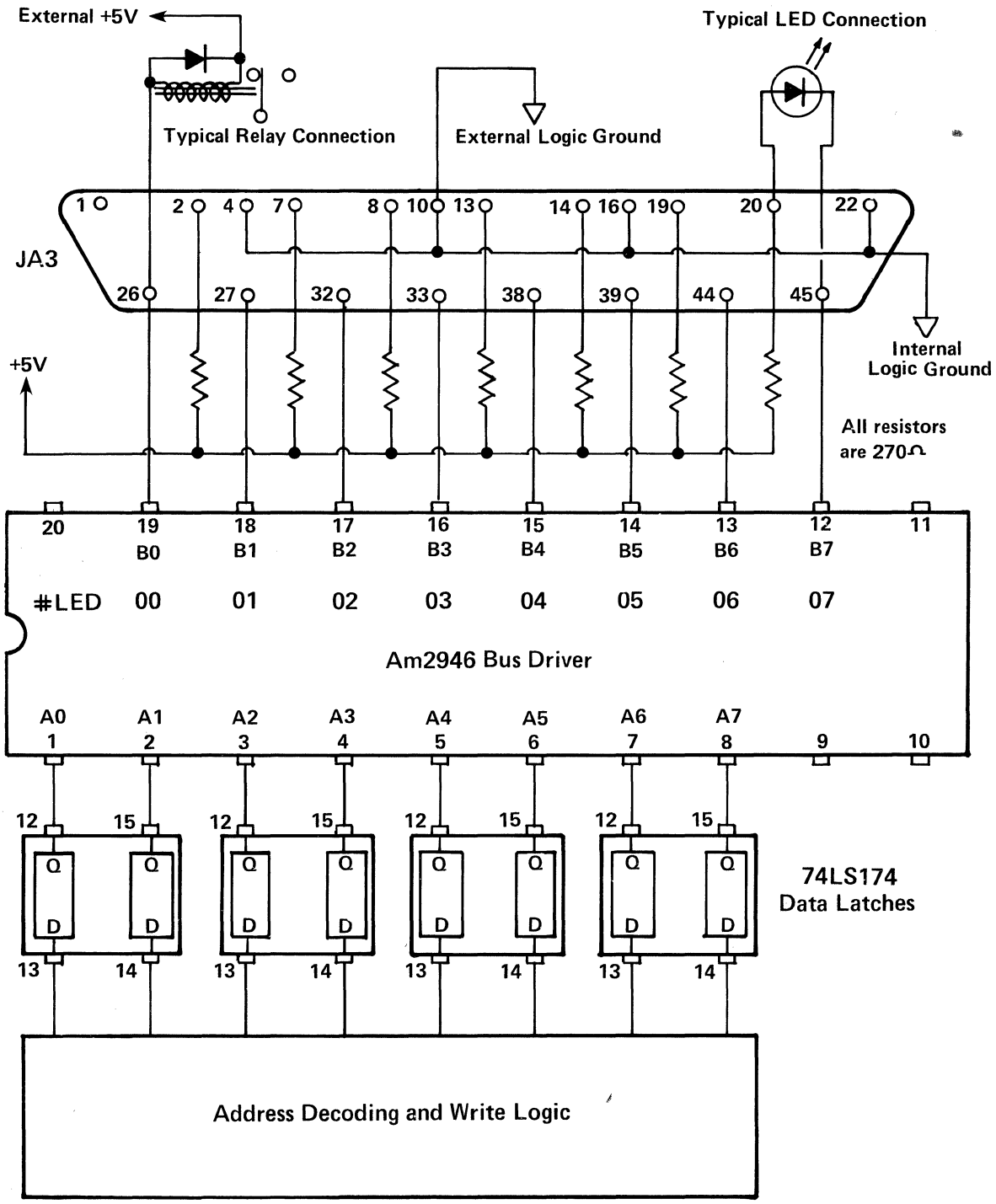


Figure 16-1. LED Output Circuits

16.2.1 DIB FORMAT

MENMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 1	Device name. Use #LED00, #LED01, #LED02, #LED03, #LED04, #LED05, #LED06 or #LED07.
DIBDTD	DS.B 1	Data Transfer Direction. Use 0. This device is WRITE only.
DIBTRN	DC.B 0	Enter 0 for fixed length.
DIBRS0	DC.L 0,0	Set this field to 0.
DIBOPT	DS.W 1	Insert option word described in paragraph 16.2.2.
DIBFCN	DC.L 0	Not used by this driver. Set this field to 0.
DIBBIO	DS.L 1	System used Byte I/O field. To open the LED OUTPUT driver for Byte I/O enter -1 ($FFFF FFFF). After open the I/O manager fills this field with an identifier which is used for SYSIO-BWRITE.

16.2.2 DIB OPTION WORD BIT DEFINITIONS

If the option word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table.

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word
	1	Use option word
14-9		Not currently used
	0	Set to zero
8		Shared Access
	0	Only one "OPEN" at a time
	1	More than one "OPEN" at a time.

7-0		Not currently used
	0	Set to zero

16.3 LED OUTPUT DATA TRANSFER CONTROL BLOCK (DTCB)

This driver does not use a DTCB. Output bytes are passed to the driver in register D0.B.

16.4 LED OUTPUT FUNCTIONS

This driver does not use Function Packets.

16.5 ERROR CODES

The following error codes have been defined and may be returned by this driver.

ERROR CODE	MEANING
16 ($0010)	DEVICE LOCKED. (OPEN ATTEMPTED AGAINST EXISTING NON-SHARED OPEN.)
17 ($0011)	DEVICE ALREADY OPEN. (NON-SHARED OPEN ATTEMPTED AGAINST AN EXISTING SHARED OPEN.)

17.0 SENSOR BOARD PARALLEL PORTS DRIVER

17.1 DRIVER DESCRIPTION

The sensor board parallel ports driver allows the user to write application programs which interface with the parallel ports on the sensor I/O board. NOTE: The sensor I/O board option must be installed and the "ATCHDRV SENSOR" Command must have been issued in order to use this driver. There are four parallel ports each with a separate device name. These ports appear on connector JA1. See Figure 17-1 and refer to the Computer System Technical Reference Manual for further details. The ports can be opened in units of bits, bytes or words. When bit I/O is desired the bit number must be specified. See paragraph 17.2.2. This driver supports synchronous and byte I/O only.

17.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open the Parallel Ports Driver the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device.

17.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 1	Device name. Use #PPA00, #PPA01, #PPB00 or #PPB01.
DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, 1 for READ or 2 for bidirectional. Bidirectional transfers must be done in strobed-mode.

```
DIBTRN    DC.B 0    Enter 0 for fixed length, or 1 for variable length.
                    Variable length is used only in byte mode.
```

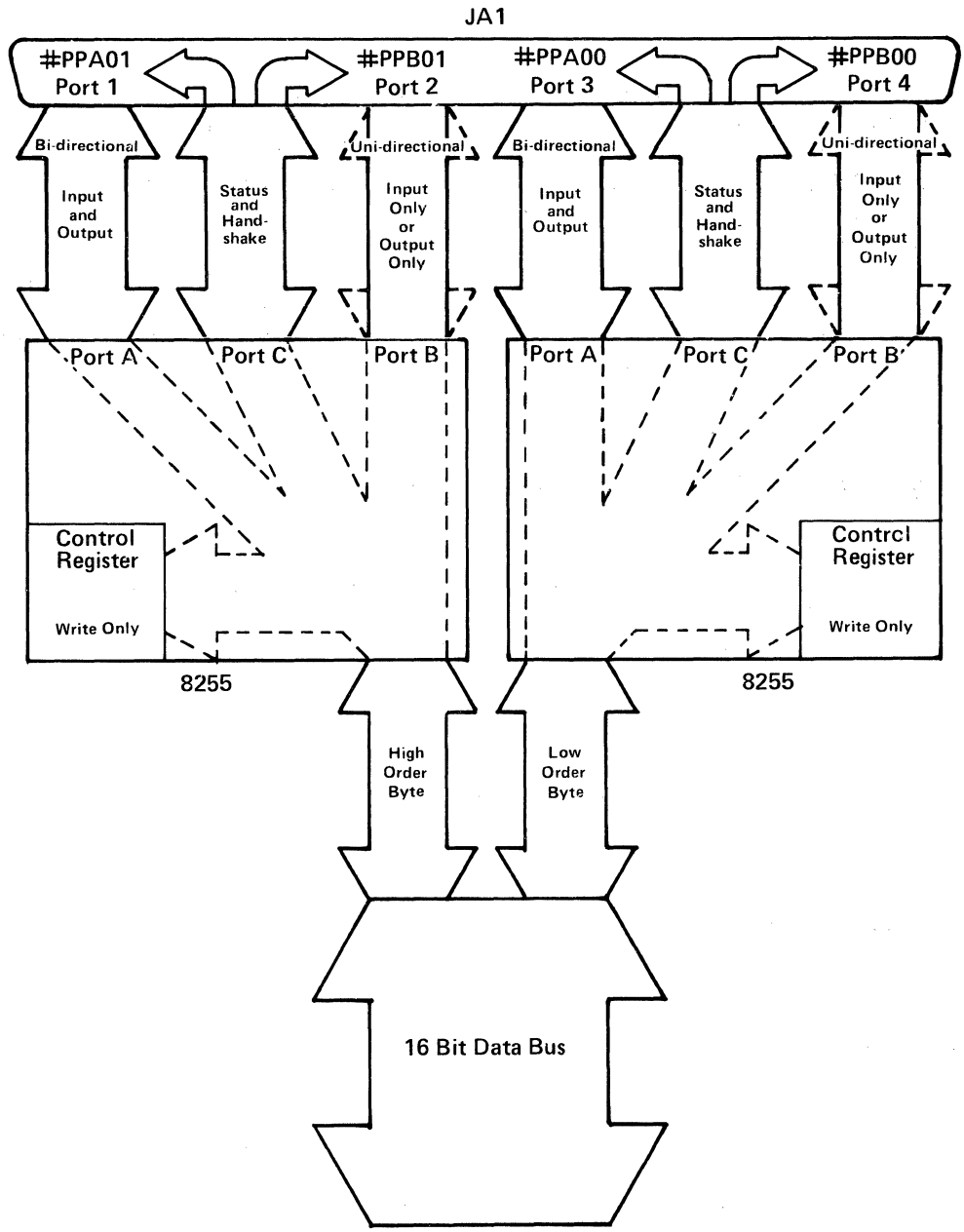


Figure 17-1. Sensor Board Parallel Ports

```
DIBRS0    DC.L 0,0 User sets this field to 0.

DIBOPT    DC.W 0   Insert option word described in paragraph 17.2.2.

DIBFCN    DC.L 1   Not used by this driver.  Set to zero.

DIBBIO    DS.L 1   System used Byte I/O field.  To open the Parallel
                   Ports driver for Byte I/O enter -1 ($FFFF
                   FFFF), otherwise set it to zero.  After open the I/O
                   manager fills this field with an identifier which is
                   used for SYSIO-BREAD, SYSIO-BWRITE.  Byte I/O only
                   allowed with non-strobed mode.
```

17.2.2 DIB OPTION WORD BIT DEFINITIONS

If the option word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table.

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word
	1	Use option word
14		Non-strobed I/O
	0	Strobed I/O (Not legal for bit I/O).
	1	Non-strobed I/O
13-11		Bit number for bit I/O
	NNN	Specifies bit number (0-7)
10-9		Size
	00	Byte I/O.
	01	Bit I/O.
	10	Word I/O.
8		Shared Access
	0	Only one "OPEN" at a time.
	1	More than one "OPEN" at a time.
7-0		Not currently used.
	0	Set to zero.

17.3 SENSOR BOARD PARALLEL PORTS DATA TRANSFER CONTROL BLOCK (DTCB)

The data transfer control block (DTCB) holds I/O status and buffer information during READs and WRITEs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

17.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User looks here for status on I/O operation.
DTCTBU	DC.B 0	Upper limit termination character for variable length transfers. User puts zero here for other transfers.
DTCTBL	DC.B 0	Lower limit termination character for variable length transfers. User puts zero here for other transfers.
DTCRS0	DC.B 0	This field is reserved. User puts zero here.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ/WRITE.

17.4 SENSOR BOARD PARALLEL PORTS FUNCTIONS

This driver does not use Function Packets.

17.5 ERROR CODES

The following error codes have been defined and may be returned by this driver.

ERROR CODE	MEANING
16 ($0010)	DEVICE LOCKED. (OPEN ATTEMPTED AGAINST EXISTING NON-SHARED OPEN.)
17 ($0011)	DEVICE ALREADY OPEN. (NON-SHARED OPEN ATTEMPTED AGAINST AN EXISTING SHARED OPEN.)
20 ($0014)	ILLEGAL OPEN MODE.

18.0 COUNTER DRIVER

18.1 DRIVER DESCRIPTION

The sensor I/O board has an Intel 8253 programmable counter chip which is made available to the user by means of the COUNTER driver. The sensor I/O board option must be installed and the "ATCHDRV SENSOR" command must be used to "attach" and use this driver. (For hardware implementation, see the "Computer System Technical Reference Manual.") The 8253 contains three individually programmed 16-bit counters, two of which are available to the user by means of the COUNTER driver, the third being used by the system. The hardware outputs and inputs are made available on RS-232 connector JA2 in accordance with Table 18-1 below.

Counter Number	1	2
Output Pin Number	22	49
External Clock Input Pin Number	24	47
Gate Input Pin Number	23	48

Table 18-1. JA2 Pin Assignments for the Programmable Counters

The two counters may be started or stopped and have their mode of operation changed through the use of function commands. Six modes of operation may be specified as follows:

MODE 0 -- This mode generates an interrupt which allows an application program to continue. The user programs the counter for MODE 0, sets the count and starts the countdown via function packets using the SYSIO FUNCTION macro. The next instruction in the application program must be a SYSIO WRITE macro which is used as a pseudo instruction to "wait" for the counter to count down. When the counter reaches zero, the instruction following the WRITE will be executed.

MODE 1 -- This mode allows the counter to be used as a programmable one-shot. The output on the pin specified in Table 18-1 will go low on the count following the rising edge of the gate input on the pin specified in Table 18-1. When the count reaches zero the output returns high. The counter may be read at any time without affecting the one-shot pulse by using the SYSIO READ macro. If a new count value is issued via

function packet and the SYSIO FUNCTION macro while the output of the counter is low it will not affect the duration of the one-shot pulse until the next trigger pulse appears on the gate input. The one-shot is retriggerable, therefore the output will remain low for the full count after any raising edge of the gate input.

MODE 2 -- This mode allows the counter to be used as a Rate Generator or Divide by N counter. When the mode is set the output on the pin specified in Table 18-1 will remain high until the count register is loaded via the START COUNT Function, at which time the output will go low unless the gate input is low, therefore the counter may be started in two ways. The first is by software and requires that the gate input be high. The counter is loaded via the SET COUNT function and then started via the START COUNT function. The second way is by releasing the gate input from a held low state and bringing it high. While it is low the output pin specified in Table 18-1 will stay high even if the START COUNT function is issued. The counter will start from the initial count loaded when the gate input can be used to synchronize the counter. The output of the counter will be low for one period of the input clock. If the internal 2 MHz clock input is jumper selected this period will be .5 µs. The period from output pulse to output pulse will equal the counts loaded via SET COUNT.

MODE 3 -- This mode is similar to MODE 2 except that the output remains high until one half of the count has been completed (for even numbers) and go low for the remainder of the count, therefore the counter can be programmed to be a Square Wave Rate Generator. If the count is set with an odd value the output on the pin specified in Table 18-1 is high for that value plus one divided by two or in formula notation (N+1)/2 and low for (N-1)/2 counts.

MODE 4 -- This mode allows the counter to be used as a Software Triggered Strobe. When the mode is set the output goes high and stays there until the counter reaches zero, at which time the output goes low for one clock period, and then returns high again. The count is set and the counter started via function

packets using the SYSIO FUNCTION macro.

MODE 5 -- This mode allows the counter to be used as a Hardware Triggered Strobe. The mode is set, the count is loaded and the counter started via function packet using the SYSIO FUNCTION macro. The counter will not start counting until after the rising edge of a trigger pulse on the gate input pin specified in Table 18-1. When the terminal count is reached the output will go low for one clock period and return high again. The counter is retriggerable. The output will not go low until the full count after the last trigger pulse (rising edge).

Each counter is viewed as a separate device with a unique device name (#CTC00 for counter 1 and #CTC01 for counter 2).

The sensor I/O board has an on board 2 MHz clock and a 30 Hz clock which can be jumpered to the clock input of the counter chip. If an external clock input is desired, this jumper must be switched. See the Computer System Technical Reference Manual for further details.

18.2 DEVICE INITIALIZATION BLOCK (DIB)

"DIB" refers to a form of control block that is used at OPEN time which can specify a non-default mode of operation for the device. There is unique information that the device driver needs to know at OPEN time. This information is used by the I/O manager. It is copied from user space into the appropriate control block in system space.

To open a Counter the user must create a DIB and within an initialization routine perform a SYSIO-OPEN to the device. When this is done then you may issue the SYSIO SREAD, SWRITE, FUNCTION, and CLOSE macros. This driver does not support Byte I/O or asynchronous operations. AREADs and AWRITEs will be performed as SREADs and SWRITEs.

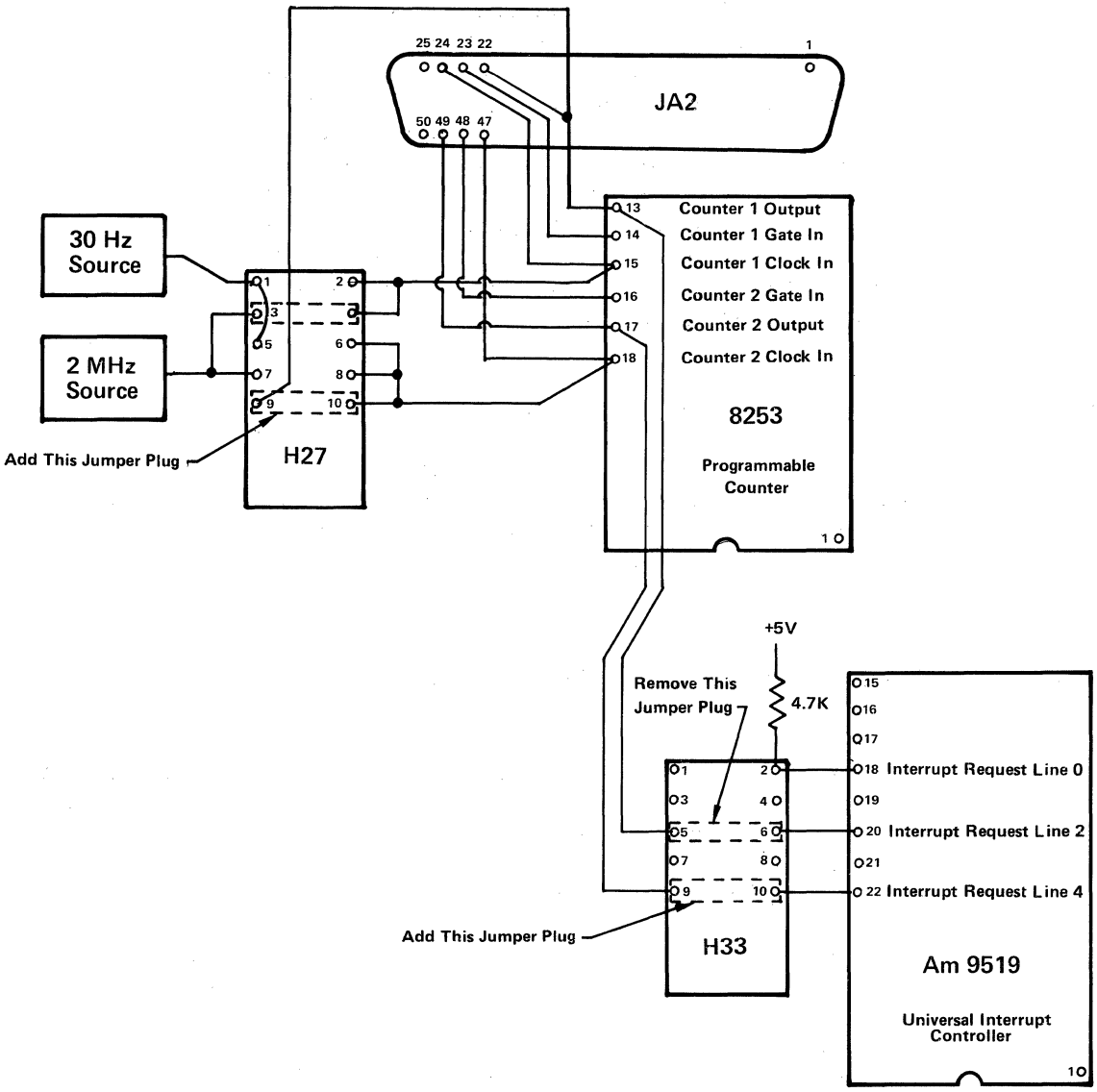


Figure 18-1. Cascading the Counters

18.2.1 DIB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DIBVOL	DS.B 6	Device name. Use #CTC00 or #CTC01.
DIBDTD	DS.B 1	Data Transfer Direction. Use 0 for WRITE, and 1 for READ.
DIBTRN	DC.B 0	Enter 0 for fixed length.
DIBRS0	DC.L 0,0	User sets this field to 0.
DIBOPT	DC.W 0	Insert option word described in paragraph 18.2.2.
DIBFCN	DS.L 1	Insert pointer to function packet; null for default.
DIBBIO	DS.L 1	System used Byte I/O field. Use 0. This driver does not support byte I/O.

18.2.2 DIB OPTION WORD BIT DEFINITIONS

If the options word field is nulled at open time the system will use the "set to 0" condition specified for each bit as described in the following table.

BIT NO.	SET TO	OPTION USE
15		Option word select
	0	Ignore option word
	1	Use option word
14-9		Not currently used
	0	Set to zero
8		Shared Access
	0	Only one "OPEN" at a time
	1	More than one "OPEN" at a time.
7-0		Not currently used
	0	Set to zero

18.3 COUNTER DATA TRANSFER CONTROL BLOCK (DTCB)

The data transfer control block (DTCB) holds I/O status and buffer information during SREADs and SWRITEs. It is a required operand of the SYSIO macro. The application program uses it to determine information required in completing each data transfer request, and to monitor the status of the transfer after the request has been made.

18.3.1 DTCB FORMAT

MNEMONIC	DATA LENGTH	DESCRIPTION OF USE
DTCSTA	DS.B 1	User looks here for status I/O operation.
DTCTBU	DC.B 0	Not used by this driver.
DTCTBL	DC.B 0	Not used by this driver.
DTCRS0	DC.B 0	This field is reserved. User sets this field to 0.
DTCBFS	DS.L 1	User puts Buffer starting address here.
DTCBFL	DS.W 1	User puts count of number of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into buffer (if any) to the first byte of the record. This pointer will be incremented by the driver for every byte transmitted. It must be reset after every READ.

18.4 COUNTER FUNCTIONS

The Function Packet Control Block provides for device specific operations not necessarily involving data transfer. This would include things like specifying shared access. It is required for the FUNCTION command and optional for the OPEN command. It is used by the application program to configure a device to something other than its default mode.

The function packet is a list of COMMAND-DATA pairs terminated by a terminating COMMAND of zero indicating END-OF-LIST. The COMMAND word is followed by zero or more bytes, words or longwords that send or receive the immediate DATA for the command, or a long word that points to the DATA for that COMMAND.

18.4.1 SUMMARY OF FUNCTIONS

The functions listed in this section can be used with the SYSIO-FUNCTION command of the I/O manager using FUNCTION packets.

FUNCTION PURPOSE	COMMAND WORD	ERROR CODE	DATA REQUIRED
ENDLIST	0 ($0000)	NONE	NONE
SET COUNT	1 ($0001)	$0021	LONG WORD
SET MODE	2 ($0002)	$0022	WORD
START COUNT	3 ($0003)	$0023	WORD

18.4.2 COUNTER FUNCTION DESCRIPTIONS

COMMAND	FUNCTION PURPOSE	FUNCTION DESCRIPTION
0	ENDLIST	Terminates processing of the function packet.
	Function Data	None
	Error Code	None
1	SET COUNT	This function is used to set the initial count of the counter. If the counter is being used as a 16-bit counter only the least significant word is used to load the counter with the initial count and the most significant word is not used. Count is actually loaded into counter by SYSIO SWRITE or start count function.

	Function Data	One long word, integer.
	Data $0000 NNNN	Specifies initial count for 16-bit counters.
	Error Code	None

2	SET MODE	This function is used to select the mode of operation of the counter. See Paragraph 18-1 for a description of the available modes.
	Function Data	
	Data $0000	Set MODE to Mode 0.
	$0001	Set MODE to Mode 1.
	$0002	Set MODE to Mode 2.
	$0003	Set MODE to Mode 3.
	$0004	Set MODE to Mode 4.
	$0005	Set MODE to Mode 5.
	Error Code	$0022

3	START COUNT	
	Data = 0	No operation performed.
	= 1	Count loaded.

18.5 ERROR CODES

The following error codes have been defined and may be returned by this driver.

ERROR CODE	MEANING
16 ($0010)	DEVICE LOCKED. (OPEN ATTEMPTED AGAINST EXISTING NON-SHARED OPEN.)
17 ($0011)	DEVICE ALREADY OPEN. (NON-SHARED OPEN ATTEMPTED AGAINST AN EXISTING SHARED OPEN.)
19 ($0013)	COUNTER OVERFLOW.

19.0 SEMAPHORE MANAGER

19.1 MANAGER DESCRIPTION

The counting semaphore provides a means of synchronization in a multitask environment. It is useful to think of the semaphore as a container of resource units. In a typical application one task (producer) will add units to the semaphore while other tasks (consumers) will receive units from the semaphore. For example a data acquisition task may use a semaphore to communicate the number of data points it has acquired to other processing tasks who will use the data after some number of points have been acquired.

The available SEMMGR functions include:

ATTACH	Used to attach or "open" a semaphore
DETACH	Used to detach or "close" a semaphore
SEND	Used to send units to a semaphore
RECV	Used to receive units from a semaphore (synchronous)
RECVIF	Used to receive units from a semaphore (asynchronous)
CANCEL	Used to cancel any pending asynchronous requests for the calling task.
QUERY	Used to obtain information about a semaphore's status

The semaphore manager uses a system identifier as an index to locate a particular semaphore (once it has been attached). This system identifier is returned to the owner task after an ATTACH command, and it must be used in a subsequent reference to the semaphore for SEND, RECV, and RECVIF operations.

If other tasks wish to know a semaphore's system identifier, they must use the QUERY command and the 4-character name used when the semaphore was first attached. The system identifier and the maximum semaphore count will both be returned to the querying task.

Users gain access to the semaphore facilities by making calls to the semaphore manager using the commands on the following pages:

SEMMGR ATTACH,,APKPOINT,ERRORLABEL

The ATTACH function creates a semaphore and attaches it to the operating system. The task that performs the attach is responsible for detaching the semaphore, no other task can perform this function. Three parameters must be supplied at ATTACH time using the semaphore attach packet:

APKNAM	The semaphore name (4chars)	-long word
APKICT	The semaphore initial count	- word
APKMAX	The maximum semaphore count	- word

The semaphore attach packet is described below.

APKNAM is a four character ASCII name for the semaphore. APKICT is the initial count to be loaded in the semaphore and must be less than or equal to the maximum semaphore count APKMAX.

Register Usage:

A6.L - Points to an Attach Packet
D4.L - Function number = 1
D5.L - Contains system identifier upon return
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

The semaphore attach packet (APK) is used with the attach function to supply open time information to the semaphore manager. The fields in the APK are as follows:

FIELD	OFFSET	VALUE	DESCRIPTION	SIZE	TYPE
APKNAM	EQU	0	SEMAPHORE NAME	4 BYTES	ASCII
APKICT	EQU	4	SEMAPHORE COUNT	2 BYTES	INTEGER
APKMAX	EQU	6	MAXIMUM COUNT	2 BYTES	INTEGER
APKQMD	EQU	8	QUEUEING MODE	2 BYTES	INTEGER
APKLEN	EQU	10	CONTROL BLOCK LENGTH IN BYTES		

Comments:

APKNAM is the four character ASCII name of the semaphore. This field must be nonzero. Tasks wishing to use the semaphore will supply this name with a query function call to find the system identifier for the semaphore and the semaphore maximum count.

APKICT is the initial count to be loaded into the semaphore. This field is a positive 16 bit two's complement integer.

APKMAX is the maximum count the semaphore is allowed to have. This field is a nonzero positive 16 bit two's complement integer.

APKQMD is the desired queueing mode for the semaphore. This field may assume the 16 bit values (0,1) where:

0 denotes FIFO request queueing
1 denotes task priority FIFO request queueing

APKLEN is the total control block length (not a data field).

Example:

```
675 0 0000017A                    SEMMGR   ATTACH,,APKBLK,ERROR
    0 0000017A 7801               MOVE.L   #ATTACH,D4          + LOAD ATTACH FUNCTION CODE
    0 0000017C 4DF90000022C       LEA      APKBLK,A6           + LOAD ATTACH PACKET ADDRESS
    0 00000182 4E40               TRAP     #0                  + TRAP CALL TO SEMAPHORE MANAGER
    0 00000184 0038               DC.W     56                  +
    0 00000186 4EF900000210       JMP.L    ERROR               + ERROR EXCEPTION BRANCH
```

SEMMGR DETACH,SYSID,,ERRORLABEL

The DETACH function removes the semaphore from the system and terminates any pending requests against the semaphore. The DETACH function must be performed by the task who initially opened the semaphore.

Register Usage:

D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

None

Comments:

If other tasks have made unsatisfied requests, they will be resumed with the appropriate completion code (See RECV and RECVIF completion codes) returned in either D7.W or USBSTA.

Example:

```
717 0 0000018C                    SEMMGR   DETACH,SYSID,,ERROR
    0 0000018C 7802               MOVE.L   #SEMDETACH,D4          + LOAD DETATCH FUNCTION CODE
    0 0000018E 2A3900000234       MOVE.L   SYSID,D5               + LOAD IDENTIFIER
    0 00000194 4E40               TRAP     #0                     + TRAP CALL TO SEMAPHORE MANAGER
    0 00000196 0036               DC.W     56                     +
    0 00000198 4EF900000210       JMP.L    ERROR                  + ERROR EXCEPTION BRANCH
```

SEMMGR RECV,SYSID,USBPOINT,ERRORLABEL

The RECV function is a synchronous call to the semaphore manager. If more units are requested than the semaphore has in its possession at the time of the call, the calling task is suspended until the request can be satisfied, the request times out, or the semaphore is detached. The request time out period is specified in the USBTIM field of the user semaphore block. A minus one in this field signifies an infinite time out. Numbers greater than or equal to zero signify the time out period in units of 50 milliseconds.

Register Usage:

A6.L - Points to a User Semaphore Block
D5.L - System identifier returned either by ATTACH or QUERY command returns
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

The Fields in the user semaphore block are as follows:

FIELD	OFFSET	VALUE	DESCRIPTION	SIZE	TYPE
USBSTA	EQU	0	REQUEST STATUS	1 BYTE	INTEGER
USBRS1	EQU	1	RESERVED BYTE	1 BYTE	
USBRS2	EQU	2	RESERVED WORD	2 BYTES	
USBCNT	EQU	4	REQUEST COUNT	2 BYTES	INTEGER
USBTIM	EQU	6	REQUEST TIME OUT	2 BYTES	INTEGER
USBLEN	EQU	8	CONTROL BLOCK LENGTH IN BYTES		

Comments:

USBSTA is a one byte integer status field (see semaphore manager error codes -- the $30 Semaphore Manager prefix is not loaded; only the error suffix is).

USBRS1 is a reserved byte

USBRS2 is a reserved word

USBCNT is a count to be exchanged between the task issuing the request and the semaphore manager. This field is a two byte two's complement positive integer.

USBTIM is a time out value (in 50 msec units) for the request. This field is only used in the RECV and RECVIF functions. This field is a 2 byte two's complement integer which is greater than or equal to -1. If a -1 is entered in USBTIM an infinite time out will be used.

Example:

```
724
725 0 0000019E                    SEMMGR   RECV,SYSID,USBBLK,ERROR
    0 0000019E 7804               MOVE.L   #SEMRECV,D4          + LOAD RECV FUNCTION CODE
    0 000001A0 2A3900000234       MOVE.L   SYSID,D5             + LOAD IDENTIFIER
    0 000001A6 4DF900000230       LEA      USBBLK,A6            + LOAD USB ADDRESS
    0 000001AC 4E40               TRAP     #0                   + TRAP CALL TO SEMAPHORE MANAGER
    0 000001AE 0038               DC.W     56                   +
    0 000001B0 4EF900000210       JMP.L    ERROR                + ERROR EXCEPTION BRANCH
```

SEMMGR RECVIF,SYSID,USBPOINT,ERRORLABEL

RECVIF is an asynchronous call to the semaphore manager to receive units from a asemaphore. The RECVIF function is the same as the RECV function except that it is performed asynchronously. If the request for units cannot be satisfied, the request is queued. Control always returns immediately to the calling task. The request status is monitored in the USBSTA field of the user semaphore block. A minus one in this field indicates that the request has been queued. A zero indicates that the request completed. A positive value in this field indicates that an error has occurred and the request has been terminated (i.e. time out or detach of the semaphore).

Register Usage:

A6.L - Points to a User Semaphore Block (See SEMMGR RECV for details)
(NOTE: The USBCNT field will contain the remaining count in custody of this semaphore upon satisfaction of this request.)
D5.L - System identifier returned either by ATTACH or QUERY commands
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

User Semaphore Block (See RECV)

Comments:

This is a non-blocking request. A -1 in the USBSTA area indicates a successful queuing of the request. The time count will be updated and is available for testing by the task. The requested count must not exceed the maximum specified by the owner task at ATTACH time.

Example:

```
733 0 000001B6                  SEMMGR  RECVIF,SYSID,USBBLK,ERROR
    0 000001B6 7805             MOVE.L  #SEMRECVIF,D4       + LOAD RECVIF FUNCTION CODE
    0 000001B8 2A3900000234     MOVE.L  SYSID,D5            + LOAD IDENTIFIER
    0 000001BE 4DF900000230     LEA     USBBLK,A6           + LOAD USB ADDRESS
    0 000001C4 4E40             TRAP    #0                  + TRAP CALL TO SEMAPHORE MANAGER
    0 000001C6 0038             DC.W    56                  +
    0 000001C8 4EF900000210     JMP.L   ERROR               + ERROR EXCEPTION BRANCH
```

SEMMGR SEND,SYSID,USBPOINT,ERRORLABEL

This command can be used to send units to a semaphore.

Register Usage:

A6.L - Points to a User Semaphore Block (See RECV for format) Note that USBCNT shall be filled with the count to be sent to the semaphore and will return with the new TOTAL semaphore count!!!
D5.L - System identifier obtained from a QUERY or ATTACH command return.
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

User Semaphore Block (See RECV)

Comments:

None.

Example:

```
740 0 000001CE                  SEMMGR   SEND,SYSID,USBBLK,ERROR
    0 000001CE 7803             MOVE.L   #SEMSEND,D4          + LOAD SEND FUNCTION CODE
    0 000001D0 2A3900000234     MOVE.L   SYSID,D5             + LOAD IDENTIFIER
    0 000001D6 4DF900000230     LEA      USBBLK,A6            + LOAD USB ADDRESS
    0 000001DC 4E40             TRAP     #0                   + TRAP CALL TO SEMAPHORE MANAGER
    0 000001DE 0038             DC.W     56                   +
    0 000001E0 4EF900000210     JMP.L    ERROR                + ERROR EXCEPTION BRANCH
```

SEMMGR QUERY,SEMNAM,USBPOINT,ERRORLABEL

This command returns the semaphore system identifier and the maximum semaphore count into the User Semaphore Block.

Register Usage:

A6.L - Points to a User Semaphore Block. Maximum count is returned in USBCNT.
D5.L - Upon entry, this contains the left-justified ASCII name of the semaphore to be queried. Upon return, this contains the system identifier to be used in RECV, RECVIF, and SEND semaphore calls.
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

User Semaphore Block (See RECV)

Comments:

None.

Example:

```
748 0 000001E6                  SEMMGR   QUERY,SEMNAM,USBBLK,ERROR
    0 000001E6 7806             MOVE.L   #SEMQUERY,D4          + LOAD QUERY FUNCTION CODE
    0 000001E8 2A3900000238     MOVE.L   SEMNAM,D5             + LOAD IDENTIFIER
    0 000001EE 4DF900000230     LEA      USBBLK,A6             + LOAD USB PACKET ADDRESS
    0 000001F4 4E40             TRAP     #0                    + TRAP CALL TO SEMAPHORE MANAGER
    0 000001F6 0038             DC.W     56                    +
    0 000001F8 4EF900000210     JMP.L    ERROR                 + ERROR EXCEPTION BRANCH
```

SEMMGR CANCEL,SYSID,USBPOINT,ERRORLABEL

This command is used to cancel all asynchronous requests by the calling task to the semaphore identified in the cancel call.

Register Usage:

A6.L - Points to a User Semaphore Block (See RECV for format of this block.)
D5.L - System identifier obtained by QUERY or ATTACH commands.
D7.W - Completion code upon return

Completion Codes:

$0000	NO ERROR
$3001	DUPLICATE SEMAPHORE NAME
$3002	ALL SEMAPHORES IN USE
$3003	ILLEGAL MAXIMUM COUNT
$3004	COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3005	ILLEGAL QUEUING MODE
$3006	ILLEGAL SEMAPHORE NAME OR INVALID SYSTEM I.D.
$3007	NOT OWNER TASK-CANNOT DETACH SEMAPHORE
$3008	REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009	INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A	SEMAPHORE HAS BEEN DETACHED-REQUEST TERMINATED
$300B	INVALID COUNT
$300C	SEMAPHORE NOT FOUND
$300D	INVALID TIME OUT VALUE
$300E	REQUEST TIMED OUT
$300F	FUNCTION NOT SUPPORTED
$3010	INITIAL COUNT EXCEEDS MAXIMUM COUNT-WILL BE TRUNCATED
$3011	REQUEST CANCELLED

Data Structures:

User Semaphore Block (See RECV)

Comments:

This call cancels all pending RECVIF requests for the Calling Task!

Example:

```
757 0 000001FE                    SEMMGR   CANCEL,SYSID,USBBLK,ERROR
    0 000001FE 7807               MOVE.L   #SEMCANCEL,D4          + LOAD CANCEL FUNCTION CODE
    0 00000200 2A3900000234       MOVE.L   SYSID,D5               + LOAD IDENTIFIER
    0 00000206 4E40               TRAP     #0                     + TRAP CALL TO SEMAPHORE MANAGER
    0 00000208 0038               DC.W     56                     +
    0 0000020A 4EF900000210       JMP.L    ERROR                  + ERROR EXCEPTION BRANCH
```

19.2 EXAMPLE PROGRAM WITH SEMAPHORES

```
ASSEMBLER LISTING          SEMDEMO.ASM   09/26/83  14:46:01   PAGE    1


  1                        SEMDEMO  IDNT     0,0                 SEMDEMO   SEMAPHORE DEMONSTRATION PROGRAM  26SEP83
  2                        ********************************************************************************
  3                        * SEMTEST-A MULTITASKING DEMONSTRATION PROGRAM USING SEMAPHORES      *
  4                        *        AND GRAPHICS 9/26/83                                       *
  5                        *                                                                   *
  6                        *        THREE TASKS ARE CREATED FOR THIS DEMONSTRATION:            *
  7                        *                PRODUCR1  PRODUCES (SENDS) UNITS TO THE SEMAPHORE *
  8                        *                          ONE AT A TIME.                           *
  9                        *                CONSUMR1  CONSUMES (RECEIVES) UNITS FROM THE       *
 10                        *                          SEMAPHORE 350 AT A TIME.                 *
 11                        *                CONSUMR1  CONSUMES (RECEIVES) UNITS FROM THE       *
 12                        *                          SEMAPHORE 350 AT A TIME.                 *
 13                        *       THE PRODUCTION AND CONSUMPTION OF UNITS IS MADE VISABLE      *
 14                        *       THROUGH THE USE OF BAR CHARTS. THE BAR CHART SERVES THE      *
 15                        *       ADDITIONAL PURPOSE OF SLOWING THE PROCESS DOWN FOR VIEWING  *
 16                        ********************************************************************************
319                        ********************************************************************************
320                        *                                                                   *
321                        *       WARNING ON SYSMAC50                                         *
322                        *                                                                   *
323                        *       If you are assembling this INCLUDE file into existing code  *
324                        *       it is possible that a comment may unwittingly become a      *
325                        *       positional parameter and cause unwanted results.            *
326                        *                                                                   *
327                        *       There should not be anything in the comment field of the    *
328                        *       following macros:                                           *
329                        *       CMWC      GETPCB     GSTAT      SETPRI                      *
330                        *       DELAY     GIVPCB     PRTMSG                                 *
331                        *                                                                   *
332                        ********************************************************************************
333                        * SYSMAC50.INC REV 11-AUG-83 HAS BEEN INCLUDED
334
576                        * IOMCLB50.INC REV 26-AUG-83 HAS BEEN INCLUDED
577
721                        * SEMMAC50.INC REV 07-SEP-83 HAS BEEN INCLUDED
722
768                        * GRMCLB50.INC REV 08-AUG-83 HAS BEEN INCLUDED
769
770                        ********************************************************************************
771                        *  SEMTEST-SPAWNS PRODUCR1 TASK AND RETURNS TO THE OPERATING SYSTEM  *
772                        ********************************************************************************
773 0          00000000  SEMTEST  EQU      *
774 0 00000000 700A                MOVE.L   #10,D0              LOAD TASK PRIORITY
775 0 00000002 223C50524F44        MOVE.L   #'PROD',D1          LOAD TASK NAME
776 0 00000008 243C55435231        MOVE.L   #'UCR1',D2
777 0 0000000E 4DF900000001A       LEA      PRODUCR1,A6         LOAD TASK START ADDRESS
778 0 00000014                     GETPCB
779 0 00000018 4E75                RTS
780                        ********************************************************************************
781                        * PRODUCR1-PRODUCER TASK                                              *
782                        *          ATTACHES AND PRODUCES UNITS FOR THE SEMAPHORE              *
783                        *          SPAWNS CONSUMR1 AND CONSUMR2 TASKS                         *
784                        ********************************************************************************
785 0          0000001A  PRODUCR1 EQU      *
786 0 0000001A 4FF900000484        LEA      P1STACK,A7          ALLOCATE STACK AREA FOR TASK
787                        *
788                        * OPEN GRAPHICS WINDOW FOR BARCHART DISPLAY
789                        *
```

```
790 0 00000020 41F9000001F4          LEA     P1GRDIB,A0          LOAD DIB POINTER
791 0 00000026 701A                  MOVE.L  #DIBLEN,D0          LOAD BYTE COUNT
792 0 00000028 10FC0000      P1LOOP0 MOVE.B  #0,(A0)+            CLEAR DIB
793 0 0000002C 51C8FFFA              DBF     D0,P1LOOP0
794 0 00000030 41F9000001F4          LEA     P1GRDIB,A0          RELOAD DBB POINTER
795 0 00000036 20BC23475220          MOVE.L  #'#GR ',DIBVOL(A0)  LOAD DEVICE NAME IN DIB
796 0 0000003C 317C20200004          MOVE.W  #'  ',DIBVOL+4(A0)
797 0 00000042 217C0000020E          MOVE.L  #P1OPNPKT,DIBFCN(A0) LOAD POINTER TO FPKT
            0012
798 0 0000004A                       SYSIO   OPEN,#10,P1GRDIB,P1ERROR2 OPEN THE WINDOW
799                          *
800                          * ATTACH SEMAPHORE CALLED 'TANK'
801                          *
802 0 00000062 41F9000001E0          LEA     P1APK,A0            LOAD POINTER TO APK
803 0 00000068 20BC54414E4B          MOVE.L  #'TANK',APKNAM(A0)  LOAD SEMAPHORE NAME IN APK
804 0 0000006E 317C00000004          MOVE.W  #0,APKICT(A0)       LOAD INITIAL COUNT IN APK
805 0 00000074 317C01900006          MOVE.W  #400,APKMAX(A0)     LOAD MAX. COUNT IN APK
806 0 0000007A 317C00000008          MOVE.W  #0,APKQMD(A0)       SELECT FIFO QUEUEING
807 0 00000080                       SEMMGR  ATTACH,#0,P1APK,P1ERROR1
808 0 00000092 23C5000001DC          MOVE.L  D5,P1SYSID          SAVE SYSTEM I.D. FOR SEMAPHORE
809                          *
810                          *  IF ATTACH IS SUCCESSFUL THEN SPAWN CONSUMER TASKS 1 AND 2
811                          *
812 0 00000098 700A                  MOVE.L  #10,D0              LOAD TASK PRIORITY
813 0 0000009A 223C434F4E53          MOVE.L  #'CONS',D1          LOAD TASK NAME
814 0 000000A0 243C554D5231          MOVE.L  #'UMR1',D2
815 0 000000A6 4DF900000486          LEA     CONSUMR1,A6         LOAD TASK START ADDRESS
816 0 000000AC                       GETPCB
817                          *
818 0 000000B0 700A                  MOVE.L  #10,D0              LOAD TASK PRIORITY
819 0 000000B2 223C434F4E53          MOVE.L  #'CONS',D1          LOAD TASK NAME
820 0 000000B8 243C554D5232          MOVE.L  #'UMR2',D2
821 0 000000BE 4DF900000852          LEA     CONSUMR2,A6         LOAD TASK START ADDRESS
822 0 000000C4                       GETPCB
823                          *
824                          * ADD UNITS TO THE SEMAPHORE ONE AT A TIME AND SHOW  URRENT CONTENTS
825                          * USING THE BAR CHART
826                          *
827 0 000000C8 323C06D6              MOVE.W  #1750,D1            LOAD NUMBER OF CYCLES
828 0 000000CC 41F9000001EA          LEA     P1USB,A0            LOAD POINTER TO USB
829 0 000000D2 317C00000004          MOVE.W  #0,USBCNT(A0)       INITIALIZE COUNT IN USB
830 0 000000D8 33C100000484 P1LOOP1  MOVE.W  D1,CYCLES           SAVE CYCLE COUNT
831 0 000000DE 41F9000001EA          LEA     P1USB,A0            LOAD POINTER TO USB
832 0 000000E4 0C6801900004 P1IF1    CMPI.W  #400,USBCNT(A0)     IF COUNT (=400 THEN
833 0 000000EA 6C000024              BGE     P1ELSE1
834 0 000000EE 317C00010004          MOVE.W  #1,USBCNT(A0)       ADD MORE UNITS
835 0 000000F4                       SEMMGR  SEND,P1SYSID,P1USB,P1ERROR1
836 0 0000010C 60000020              BRA     P1ENDI1
837 0 00000110 317C00000004 P1ELSE1  MOVE.W  #0,USBCNT(A0)       ELSE
838 0 00000116                       SEMMGR  SEND,P1SYSID,P1USB,P1ERROR1
839 0 0000012E 30280004      P1ENDI1 MOVE.W  USBCNT(A0),D0       ENDI
840 0 00000132 4EB90000016C          JSR     P1BAR               DRAW BAR CHART
841 0 00000138 323900000484          MOVE.W  CYCLES,D1           RESTORE CYCLE COUNT
842 0 0000013E 51C9FF98              DBF     D1,P1LOOP1          REPEAT
843 0          00000142      P1ERROR1 EQU     *
844 0 00000142                       SEMMGR  DETACH,P1SYSID,P1USB,P1ERROR2
845 0 00000154               P1ERROR2 SYSIO   CLOSE,#10,P1CLSPKT,P1ERROR3
846 0 00000166 4280          P1ERROR3 CLR.L   D0
```

```
847 0 00000168                    GIVPCB
848                       ********************************************************************************
849                       * P1BAR-ROUTINES TO DISPLAY A BAR INDICATOR ON THE CRT                         *
850                       ********************************************************************************
851 0          0000016C   P1BAR    EQU     *                   D0 CONTAINS BAR HEIGHT
852 0 0000016C 48E78482           MOVEM.L D0/D5/A0/A6,-(A7)
853 0 00000170 41F90000022C       LEA     P1GRFPKT,A0         LOAD FUNCTION PACKET BUFFER
854 0 00000176 30FC0005           MOVE.W  #GRSETPICTL,(A0)+   CLEAR PREVIOUS BAR USING
855 0 0000017A 30FC0000           MOVE.W  #0,(A0)+            RECTANGULAR FILL
856 0 0000017E 30FC0017           MOVE.W  #GRSETCOP,(A0)+
857 0 00000182 30FC0064           MOVE.W  #100,(A0)+
858 0 00000186 30FC0028           MOVE.W  #40,(A0)+
859 0 0000018A 30FC001C           MOVE.W  #GRBLKFIL,(A0)+
860 0 0000018E 30FC0078           MOVE.W  #120,(A0)+
861 0 00000192 30F9000001F2       MOVE.W  P1LSTVAL,(A0)+
862 0 00000198 06400028           ADD.W   #40,D0
863 0 0000019C 33C0000001F2       MOVE.W  D0,P1LSTVAL
864 0 000001A2 30FC0005           MOVE.W  #GRSETPICTL,(A0)+   DRAW NEW BAR USING RECTANGULAR
865 0 000001A6 30FC0001           MOVE.W  #1,(A0)+            FILL
866 0 000001AA 30FC0017           MOVE.W  #GRSETCOP,(A0)+
867 0 000001AE 30FC0064           MOVE.W  #100,(A0)+
868 0 000001B2 30FC0028           MOVE.W  #40,(A0)+
869 0 000001B6 30FC001C           MOVE.W  #GRBLKFIL,(A0)+
870 0 000001BA 30FC0078           MOVE.W  #120,(A0)+
871 0 000001BE 30F9000001F2       MOVE.W  P1LSTVAL,(A0)+
872 0 000001C4                    SYSIO   FUNCTION,#10,P1GRFPKT,P1FUNCER
873 0          000001D6   P1FUNCER EQU    *
874 0 000001D6 4CDF4121           MOVEM.L (A7)+,D0/D5/A0/A6
875 0 000001DA 4E75               RTS
876                       *
877                       * STORAGE FOR TASK PRODUCR1
878                       *
879 0 000001DC 00000004   P1SYSID  DS.L    1
880 0 000001E0 0000000A   P1APK    DS.B    APKLEN
881 0 000001EA 00000008   P1USB    DS.B    USBLEN
882 0 000001F2 00000002   P1LSTVAL DS.W    1
883 0 000001F4 0000001A   P1GRDIB  DS.B    DIBLEN
884 0 0000020E 001E       P1OPNPKT DC.W    GRCLRWIN            CLEAR DISPLAY
885 0 00000210 001700320102       DC.W    GRSETCOP,50,450     DISPLAY BAR CHART HEADER
886 0 00000216 001D               DC.W    GRSETCHR
887 0 00000218 00000222           DC.L    P1MSG
888 0 0000021C 0000               DC.W    0
889 0 0000021E 001E       P1CLSPKT DC.W    GRCLRWIN            CLEAR DISPLAY
890 0 00000220 0000               DC.W    0
891                       *
892 0 00000222 0008       P1MSG    DC.W    8                   BAR CHART HEADER
893 0 00000224 50524F445543       DC.B    'PRODUCER'
894                       *
895 0 0000022C 00000064   P1GRFPKT DS.B    100
896 0 00000290 000001F4           DS.B    500
897 0          00000484   P1STACK  EQU     *                   TASK STACK AREA
898 0 00000484 00000002   CYCLES   DS.W    1                   CYCLE COUNTER
899                       ********************************************************************************
900                       * CONSUMR1 CONSUMES UNITS FROM A SEMAPHORE CALLED TANK IN 350 CHUNKS *
901                       *          AND EATS THEM UP 1 A A TIME   THE NUMBER OF UNITS ARE     *
902                       *          DISPLAYED ON THE CRT                                      *
903                       *                                                                    *
904                       *                                                                    *
```

```
905                             *                                                              *
906                             ********************************************************************************
907 0          00000486     CONSUMR1 EQU     *
908 0 00000486 4FF900000852             LEA     C1STACK,A7              ALLOCATE STACK FOR CONSUMR1
909                             *
910                             * OPEN GRAPHICS WINDOW FOR BARCHART DISPLAY
911                             *
912 0 0000048C 41F900000SC2             LEA     C1GRDIB,A0              LOAD DIB POINTER
913 0 00000492 701A                     MOVE.L  #DIBLEN,D0              LOAD BYTE COUNT
914 0 00000494 10FC0000     C1LOOP1     MOVE.B  #0,(A0)+                CLEAR DIB
915 0 00000498 51C8FFFA                 DBF     D0,C1LOOP1
916 0 0000049C 41F9000005C2             LEA     C1GRDIB,A0              RELOAD DIB POINTER
917 0 000004A2 20BC23475220             MOVE.L  #'#GR ',DIBVOL(A0)      LOAD DEVICE NAME IN DIB
918 0 000004A8 317C20200004             MOVE.W  #'  ',DIBVOL+4(A0)
919 0 000004AE 217C000005DC             MOVE.L  #C1OPNPKT,DIBFCN(A0)    LOAD OPEN FUNCTION PACKET
                0012
920 0 000004B6                          SYSIO   OPEN,#10,C1GRDIB,C1ERROR1 OPEN THE WINDOW
921                             *
922                             * QUERY THE SEMAPHORE FOR THE SYSTEM IDENTIFIER
923                             *
924 0 000004CE                          SEMMGR  QUERY,C1SEMNAM,C1USB,C1ERROR1
925 0 000004E6 23C5000005B4             MOVE.L  D5,C1SYSID
926                             *
927                             * IF SEMAPHORE IS FOUND THEN RECIEVE UNITS FROM IT
928                             *
929 0 000004EC 41F9000005B8 C1LOOP3     LEA     C1USB,A0                LOAD POINTER TO USB
930 0 000004F2 317CFFFF0006             MOVE.W  #-1,USBTIM(A0)          LOAD INFINITE TIME OUT
931 0 000004F8 317C015E0004             MOVE.W  #350,USBCNT(A0)         LOAD UNIT REQUEST COUNT
932 0 000004FE                          SEMMGR  RECV,C1SYSID,C1USB,C1ERROR1
933 0 00000516 203C0000015E             MOVE.L  #350,D0                 BEGIN CONSUME LOOP
934 0 0000051C 4EB900000540 C1LOOP2     JSR     C1BAR                   DISPLAY BAR CHART
935 0 00000522 51C8FFF8                 DBF     D0,C1LOOP2              DECREMENT COUNT AND LOOP
936 0 00000526 60C4                     BRA     C1LOOP3                 GET MORE UNITS
937 0 00000528              C1ERROR1    SYSIO   CLOSE,#10,C1CLSPKT,C1ERROR3
938 0 0000053A 4280         C1ERROR3    CLR.L   D0
939 0 0000053C                          GIVPCB
940                             ********************************************************************************
941                             * C1BAR-ROUTINES TO DISPLAY A BAR INDICATOR ON THE CRT                         *
942                             ********************************************************************************
943 0          00000540     C1BAR    EQU     *                       D0 CONTAINS BAR HEIGHT
944 0 00000540 48E78482                 MOVEM.L D0/D5/A0/A6,-(A7)
945 0 00000544 41F9000005FA             LEA     C1GRFPKT,A0             LOAD POINTER TO FUNCTION PACKET
946 0 0000054A 30FC0005                 MOVE.W  #GRSETPICTL,(A0)+       CLEAR PREVIOUS BAR USING
947 0 0000054E 30FC0000                 MOVE.W  #0,(A0)+                RECTANGULAR FILL
948 0 00000552 30FC0017                 MOVE.W  #GRSETCOP,(A0)+
949 0 00000556 30FC012C                 MOVE.W  #300,(A0)+
950 0 0000055A 30FC0028                 MOVE.W  #40,(A0)+
951 0 0000055E 30FC001C                 MOVE.W  #GRBLKFIL,(A0)+
952 0 00000562 30FC0140                 MOVE.W  #320,(A0)+
953 0 00000566 30F9000005C0             MOVE.W  C1LSTVAL,(A0)+
954 0 0000056C 06400028                 ADD.W   #40,D0
955 0 00000570 33C0000005C0             MOVE.W  D0,C1LSTVAL
956 0 00000576 30FC0005                 MOVE.W  #GRSETPICTL,(A0)+       DRAW NEW BAR USING RECTANGULAR FILL
957 0 0000057A 30FC0001                 MOVE.W  #1,(A0)+
958 0 0000057E 30FC0017                 MOVE.W  #GRSETCOP,(A0)+
959 0 00000582 30FC012C                 MOVE.W  #300,(A0)+
960 0 00000586 30FC0028                 MOVE.W  #40,(A0)+
961 0 0000058A 30FC001C                 MOVE.W  #GRBLKFIL,(A0)+
```

```
 962 0 0000058E 30FC0140            MOVE.W  #320,(A0)+
 963 0 00000592 30F9000005C0        MOVE.W  C1LSTVAL,(A0)+
 964 0 00000598                     SYSIO   FUNCTION,#10,C1GRFPKT,C1FUNCER
 965 0          000005AA    C1FUNCER EQU    *
 966 0 000005AA 4CDF4121            MOVEM.L (A7)+,D0/D5/A0/A6
 967 0 000005AE 4E75                RTS
 968                        *
 969                        * STORAGE FOR TASK CONSUMR1
 970                        *
 971 0 000005B0 54414E4B    C1SEMNAM DC.B   'TANK'              SEMAPHORE NAME
 972 0 000005B4 00000004    C1SYSID  DS.L   1                   SAVE AREA FOR SYSTEM IDENTIFIER
 973 0 000005B8 00000008    C1USB    DS.B   USBLEN              USER SEMAPHORE BLOCK
 974 0 000005C0 00000002    C1LSTVAL DS.W   1
 975 0 000005C2 0000001A    C1GRDIB  DS.B   DIBLEN
 976 0 000005DC 001700FA01C2 C1OPNPKT DC.W  GRSETCOP,250,450    DISPLAY BAR CHART HEADER
 977 0 000005E2 001D                DC.W    GRSETCHR
 978 0 000005E4 000005EA            DC.L    C1MSG
 979 0 000005E8 0000                DC.W    0
 980 0 000005EA 000A        C1MSG   DC.W    10                  BAR CHART HEADER
 981 0 000005EC 434F4E53554D        DC.B    'CONSUMER 1'
 982 0 000005F6 001E        C1CLSPKT DC.W   GRCLRWIN            CLEAR DISPLAY
 983 0 000005F8 0000                DC.W    0
 984 0 000005FA 00000064    C1GRFPKT DS.B   100
 985 0 0000065E 000001F4            DS.B    500
 986 0          00000852    C1STACK  EQU    *
 987                        ************************************************************************
 988                        * CONSUMR2 CONSUMES UNITS FROM A SEMAPHORE CALLED TANK IN 350 CHUNKS *
 989                        *         AND EATS THEM UP 1 A A TIME   THE NUMBER OF UNITS ARE     *
 990                        *         DISPLAYED ON THE CRT                                      *
 991                        *                                                                   *
 992                        *                                                                   *
 993                        *                                                                   *
 994                        ************************************************************************
 995 0          00000852    CONSUMR2 EQU    *
 996 0 00000852 4FF900000C1E        LEA     C2STACK,A7          ALLOCATE STACK FOR CONSUMR2
 997                        *
 998                        * OPEN GRAPHICS WINDOW FOR BARCHART DISPLAY
 999                        *
1000 0 00000858 41F90000098E        LEA     C2GRDIB,A0          LOAD DIB POINTER
1001 0 0000085E 701A                MOVE.L  #DIBLEN,D0          LOAD BYTE COUNT
1002 0 00000860 10FC0000    C2LOOP1 MOVE.B  #0,(A0)+            CLEAR DIB
1003 0 00000864 51C8FFFA            DBF     D0,C2LOOP1
1004 0 00000868 41F90000098E        LEA     C2GRDIB,A0          RELOAD DIB POINTER
1005 0 0000086E 20BC23475220        MOVE.L  #'#GR ',DIBVOL(A0)  LOAD DEVICE NAME IN DIB
1006 0 00000874 317C20200004        MOVE.W  #'  ',DIBVOL+4(A0)
1007 0 0000087A 217C000009A8        MOVE.L  #C2OPNPKT,DIBFCN(A0) LOAD POINTER TO FPKT
                0012
1008 0 00000882                     SYSIO   OPEN,#10,C2GRDIB,C2ERROR1 OPEN THE WINDOW
1009                       *
1010                       * QUERY THE SEMAPHORE FOR THE SYSTEM IDENTIFIER
1011                       *
1012 0 0000089A                     SEMMGR  QUERY,C2SEMNAM,C2USB,C2ERROR1
1013 0 000008B2 23C500000980        MOVE.L  D5,C2SYSID
1014                       *
1015                       * IF SEMAPHORE IS FOUND THEN RECIEVE UNITS FROM IT
1016                       *
1017 0 000008B8 41F900000984 C2LOOP3 LEA     C2USB,A0            LOAD POINTER TO USB
1018 0 000008BE 317CFFFF0006        MOVE.W  #-1,USBTIM(A0)      LOAD INFINITE TIME OUT
```

```
1019 0 000008C4 317C015E0004          MOVE.W  #350,USBCNT(A0)       LOAD UNIT REQUEST COUNT
1020 0 000008CA                       SEMMGR  RECV,C2SYSID,C2USB,C2ERROR1
1021 0 000008E2 203C0000015E          MOVE.L  #350,D0               BEGIN CONSUME LOOP
1022 0 000008E8 4EB90000090C C2LOOP2  JSR     C2BAR                 DISPLAY BAR CHART
1023 0 000008EE 51C8FFF8              DBF     D0,C2LOOP2            DECREMENT COUNT AND LOOP
1024 0 000008F2 60C4                  BRA     C2LOOP3               GET MORE UNITS
1025 0 000008F4               C2ERROR1 SYSIO   CLOSE,#10,C2CLSPKT,C2ERROR3
1026 0 00000906 4280          C2ERROR3 CLR.L   D0
1027 0 00000908                        GIVPCB
1028                                   ********************************************************************
1029                                   * C2BAR-ROUTINES TO DISPLAY A BAR INDICATOR ON THE CRT             *
1030                                   ********************************************************************
1031 0          0000090C     C2BAR    EQU     *                     D0 CONTAINS BAR HEIGHT
1032 0 0000090C 48E78482              MOVEM.L D0/D5/A0/A6,-(A7)
1033 0 00000910 41F9000009C6          LEA     C2GRFPKT,A0           LOAD POINTER TO FUNCTION PACKET
1034 0 00000916 30FC0005              MOVE.W  #GRSETFXCTL,(A0)+     CLEAR PREVIOUS BAR USING
1035 0 0000091A 30FC0000              MOVE.W  #0,(A0)+              RECTANGULAR FILL
1036 0 0000091E 30FC0017              MOVE.W  #GRSETCOP,(A0)+
1037 0 00000922 30FC01F4              MOVE.W  #500,(A0)+
1038 0 00000926 30FC0028              MOVE.W  #40,(A0)+
1039 0 0000092A 30FC001C              MOVE.W  #GRBLKFIL,(A0)+
1040 0 0000092E 30FC0208              MOVE.W  #520,(A0)+
1041 0 00000932 30F90000098C          MOVE.W  C2LSTVAL,(A0)+
1042 0 00000938 06400028              ADD.W   #40,D0
1043 0 0000093C 33C00000098C          MOVE.W  D0,C2LSTVAL
1044 0 00000942 30FC0005              MOVE.W  #GRSETFXCTL,(A0)+     DRAW NEW BAR WITH RECTANGULAR FILL
1045 0 00000946 30FC0001              MOVE.W  #1,(A0)+
1046 0 0000094A 30FC0017              MOVE.W  #GRSETCOP,(A0)+
1047 0 0000094E 30FC01F4              MOVE.W  #500,(A0)+
1048 0 00000952 30FC0028              MOVE.W  #40,(A0)+
1049 0 00000956 30FC001C              MOVE.W  #GRBLKFIL,(A0)+
1050 0 0000095A 30FC0208              MOVE.W  #520,(A0)+
1051 0 0000095E 30F90000098C          MOVE.W  C2LSTVAL,(A0)+
1052 0 00000964                       SYSIO   FUNCTION,#10,C2GRFPKT,C2FUNCER
1053 0          00000976     C2FUNCER EQU     *
1054 0 00000976 4CDF4121              MOVEM.L (A7)+,D0/D5/A0/A6
1055 0 0000097A 4E75                  RTS
1056                                   *
1057                                   * STORAGE FOR TASK CONSUMR2
1058                                   *
1059 0 0000097C 54414E4B     C2SEMNAM DC.B    'TANK'                SEMAPHORE NAME
1060 0 00000980 00000004     C2SYSID  DS.L    1                     STORAGE FOR SYSTEM ID
1061 0 00000984 00000008     C2USB    DS.B    USBLEN                USER SEMAPHORE BLOCK
1062 0 0000098C 00000002     C2LSTVAL DS.W    1
1063 0 0000098E 0000001A     C2GRDIB  DS.B    DIBLEN
1064 0 000009A8 001701C201C2 C2OPNPKT DC.W    GRSETCOP,450,450      DISPLAY BAR CHART HEADER
1065 0 000009AE 001D                  DC.W    GRSETCHR
1066 0 000009B0 000009B6              DC.L    C2MSG
1067 0 000009B4 0000                  DC.W    0
1068 0 000009B6 000A         C2MSG    DC.W    10                    BAR CHART HEADER
1069 0 000009B8 434F4E53554D          DC.B    'CONSUMER 2'
1070 0 000009C2 001E         C2CLSPKT DC.W    GRCLRWIN              CLEAR DISPLAY
1071 0 000009C4 0000                  DC.W    0
1072 0 000009C6 00000064     C2GRFPKT DS.B    100
1073 0 00000A2A 000001F4              DS.B    500
1074 0          00000C1E     C2STACK  EQU     *
1075                                   END

****** TOTAL ERRORS      0--   0
****** TOTAL WARNINGS    0--   0
```

20.0 SYSTEM CALLS

20.1 ISSUING SYSTEM CALLS (SC)

All system calls are performed by a TRAP #0 instruction followed by a command word. The Operating System automatically vectors the call to the appropriate address. The System Call handler saves the contents of all the registers that are not explicitly changed by the specific system function.

Since each System routine call is done in the same way, that is, TRAP #0 with a command number, they can be made macros and used like new instructions. For example, the Operating System's routine to print a message would be called as follows:

```
TRAP #0   System Call
DC.W 18   PRTMSG is no. 18
```

A macro could be written:

```
PRTMSG  MACRO
        TRAP #0
        DC.W 18
        ENDM
```

so that whenever a message is to be printed, a PRTMSG instruction can be given. The Computer System's operating system was written with the express purpose of providing a list of such useful "extended instructions."

20.2 SC ROUTINE INDEX

The following table lists the operating system's routines by their number. See the following section for a description of each routine.

* = System call usable only by the SYSTEM task.

1	reserved for users to implement customized system calls	
2	unused	
3	unused	
4	unused	
5	unused	
6	unused	
7	unused	
8	unused	
9	reserved	
10	DPRTERR	PRTERR to specific logical unit
11	CHAIN	load new transient and pass control to it
12*	PRTERR	print system error message
13	WRMST	warm restart of system
14	LOADB	load binary file into memory
15*	FMTDIB	parse command line data into DIB fields
16*	NXTOK	parse a token from command line
17*	GTCMD	get a command line and parse first token
18*	PRTMSG	print a message to #SCRN0
19*	PRTERMSG	print a message to #CNSL0
20	reserved	
21	CMWC	string compare with wildcards
22	SECSIZ	returns the sector size
23	GETTIM	get the current time-of-day/date
24	SETTIM	update the time-of-day/date
25	GETPCB	allocate PCB to task (start task)
26	GIVPCB	deallocate PCB (kill task)
27	GSTAT	get data from task PCB
28	SETPRI	change task priority
29	DELAY	set PCBDEL value in PCB (suspend task)
30	WAKEUP	clear PCBDEL value (resume task)
31*	GTCMDE	get a command line and parse first token; also display on #SCRN0 (screen)
32	SYSLEV	get system level indicator
33	reserved	
34	FMTTOD	format time for GETTIM into ASCII characters
35	reserved	
36	reserved	
37	unused	
38	unused	
39	unused	
40	SRCDIR	search directory
41*	GETCMD	get command
42*	GETCMDE	get command, echo to #SCRN0
43	EXIT	EXIT system call
44	unused	
45	unused	

```
46      unused
47      unused
48      unused
49      unused
50      unused
51      unused
52      GBASPTR     get the starting address of system common
53      reserved
53      unused
54      SETCRTCR    set CRT control register
55      INQCRTCP    inquire CRT control register
56      SEMMGR      Semaphore manager (see Chapter 19)
57      reserved
58      ITCGEN      generator for intertask channels
59      GPRTSCR     graphics print screen
60      reserved
61      unused
62      unused
63      unused
64      unused
65      unused
66      unused
67      reserved
68      reserved
69      reserved
```

The remainder of this chapter describes the TRAP #0 services handled by the system call handler.

20.3 COMMAND-PARSING ROUTINES

15 FMTDIB

This command is only available from the SYSTEM task. This routine parses the next token in the command line for a device name (e.g., #FD01) or a file specification ([VOLUME:]filename.ext) and puts the result in the DIBVOL, DIBNAM, DIBEXT fields of the DIB pointed to in register A6. Upon return from FMTDIB D7.W contains status or error indications as follows:

0 - device name was parsed
1 - volume name only was parsed
2 - unambiguous filename (filename and extension were completely specified)
3 - ambiguous filename (some part of filename/ext was wildcarded)

$104 - missing file extension (filename was unambiguous)
$110 - invalid device name
$184 - invalid file specification

With all results, except $110 and $184, the command line is now ready for the next token in the command line. If result is $110 or $184, the command line pointers are unchanged.

The results of FMTDIB must be interpreted by its user. If the user can only accept unambiguous filenames, then all other results are errors. If the user will default a file extension, then upon a $104 result, the user fills in DIBEXT.

16 NXTOK

This command is only available from the SYSTEM task. This routine breaks up a command line into "tokens." A token is a substring of the command line and is treated as a unit. The operating system defines the following tokens:

NAME: A name is a string of characters which begins with an alphabetic character and contains only alphanumeric characters (no imbedded spaces).

NAME WITH WILD-CARD CHARACTERS: A name which may include the special characters "*" and "?".

NUMBER: A string of digits which may be decimal or hexadecimal. Hexadecimal numbers must begin with a dollar sign ($).

DELIMITER: One of the special characters defined by the system. This includes the period (.), comma (,), colon (:), dollar sign ($), equals sign (=), semicolon (;), and the arithmetic symbols +,-, and / .

CARRIAGE RETURN: The ASCII carriage return character (0D hex).

ERROR: A token not falling into one of the above classes.

NXTOK uses SYSTEM common locations for its parameters. Scanning of the command line begins at the character whose address is in CUCHAR. The address of the first character of the token is returned in DESCRA. Note that spaces are not part of any token. Spaces are skipped over by NXTOK unless they are imbedded in a token. The count of the number of characters in a token is returned in DESCRC. The system common locations RC and CLASS return the classification of the token as follows:

```
NAME              RC=01          CLASS=02
NAME (WILDCARD)   RC=02          CLASS=02
NUMBER            RC=03          CLASS=02

DELIMITER     RC=ASCII code of character   CLASS =04

CARRIAGE RET. RC=0D hex      CLASS=0D hex

ERROR         RC=00          CLASS=00
```

When CUCHAR is returned, it points to a location one character beyond the end of the present token. If the token is a number (RC=03), its binary value is returned in the system common location VALUE. NXTOK will automatically convert unsigned decimal or hexadecimal numbers into binary form. The hex numbers must have a leading dollar sign ($). NXTOK will trap numbers that are too large (> 32 bits) as errors.

Example of use of NXTOK:

command line='LOAD 1:MYFILE.EXT' carriage return

```
first token='LOAD'     RC=01, CLASS=02
second token='1'       RC=03, CLASS=02, VALUE=00000001
third token=':'        RC=3A, CLASS=04
fourth token='MYFILE'  RC=01, CLASS=02
fifth token='.'        RC=2E, CLASS=04
sixth token='EXT'      RC=01, CLASS=02
seventh token=c.r.     RC=0D, CLASS=0D
```

20.4 FILENAME FORMATTING

21 CMWC

This routine compares two strings skipping over the wildcard character "?" which matches any character, including a space, when it occurs in the first string. Register A0 points to the first string, register A1 points to the second string. Register D0 contains a byte count. Strings with up to 255 characters may be compared. A string terminator (04 hex) in either string terminates the comparison at that point. The results of the comparison are returned in the condition codes.

20.5 INITIALIZATION AND WARMSTART

13 WRMST

This routine reinitializes CSOS. It is essentially the same as pressing CTRL-ALT-DEL.

32 SYSLEV

This returns the operating system release level number in D0.L, in the following format:

```
byte   byte     word
level revision internal
```

for example: Release 1.0 would appear as $01000001 in D0.L, with an internal level indicator of 1.

58 ITCGEN Intertask Channel Generator

The ITCGEN System Call is used to add or remove an intertask channel from the system. (Further documentation on intertask channels is available in Chapter 13.) This function expects the following arguments:

```
D1.W    OPTION SELECT
        0 : ATTACH INTERTASK CHANNEL
        1 : DETACH INTERTASK CHANNEL
        2 : SYSTEM USE
        3 : SYSTEM USE

D0.W    CHANNEL IDENTIFIER
        2 ASCII (PRINTABLE) CHARACTERS
```

20.6 DISPLAY CONTROL

Video Display Control

54 SETCRTCR

This routine loads the contents of D0.B into the CRT control register. The bits on the one-byte field have the following significance:

```
BIT 0     PAGE SELECT      0=PAGE 0; 1=PAGE 1
BIT 1     NOT USED
BIT 2     INVERSE VIDEO    0=NORMAL; 1=INVERSE
BIT 3     VIDEO BLANKING   0=BLANK; 1=NORMAL
BIT 4-7   NOT USED
```

55 INQCRTCR

INQCRTCR returns the one-byte value of the CRT control register in D0.B (See above).

20.7 UTILITY SYSTEM CALLS

10 DPRTERR

This routine prints a system error to any device, except disk, whose opened LUN is in register D5.B. When a SYSIO or other system manager, call returns with an error, issuing DPRTERR will result in an error message of the following format:

```
source ERROR #  message test  [TASK = taskname]
```

"TASK=" will only appear if the task is not the SYSTEM task.

12 PRTERR

This command is only available from the SYSTEM task. This routine is the same as DPRTERR without D5.B (LUN) required. (The SYSTEM window "CONSOLE BOX" is used.)

17 GTCMD

This command is only available from the SYSTEM task. This routine accepts a command line from the console or an open SUBMIT file. If in a SUBMIT file, GTCMD reads characters from the file and expands any macro parameters. If not in a SUBMIT file, the user is prompted and a new line may be typed in. GTCMD passes the line directly to NXTOK, so on return from GTCMD, the first token on the line has been parsed.

18 PRTMSG

This command is only available from the SYSTEM task.* This routine prints a string on the screen (#SCRN0). Register A6 is pointed to the start of the string. If the string terminates with a carriage return, a carriage return-linefeed is issued. If the string terminates with 04 hex, no carriage return linefeed is issued.

19 PRTERMSG

This command is only available from the SYSTEM task. This routine prints a string in inverse video on the CRT error line (#CNSL0). The string must end with a carriage return.

22 SECSIZ

This call returns in register D0.L the number of bytes in a disk sector. The current system default is 256.

31 GTCMDE

This command is only available from the SYSTEM task. Same as GTCMD with the additional feature of displaying the command line on the screen (#SCRN0).

41 GETCMD

This command is only available from the SYSTEM task. This routine is the same as GTCMD without the automatic NXTOK.

42 GETCMDE

This command is only available from the SYSTEM task. This routine is the same as GTCMDE without the automatic NXTOK.

52 GBASPTR

This routine returns the starting address of system common in register A5. This is needed by those programs that must examine variables in system common such as memory bounds. The long word located at 4(A5) contains APPBEGIN which is the start of user memory. The long word located at 12(A5) contains RAMEND which is the physical end of memory for the system.

*In MSPMAC50.INC is an assembler macro PRNTMSG that can be used by any task to print messages. It works with PRNTINIT macro and can be used as is or tailored to fit the programmer needs.

The long word located at 8(A5) contains APPEND which is the end of user memory. (This address may be altered through use of the SET command, see Part 1.) You use the include file COMDEF50.INC to reference these fields.

59 GPRTSCR Graphics Screen Dump

This system call will cause the current display to be copied to the printer.

CAUTION: If the printer is offline or in use, this function will fail!

No arguments are required for this function.

20.8 DIRECTORY SEARCH

40 SRCDIR

This routine searches the directory of a disk (of old or new file structure) for the filename requested. If the filename contains wildcarding, repeated SRCDIR uses will find the additional matches in the directory.

Upon input

D5.B = LUN of opened DIR.DIR file (a read-only file)

A6 = address of DTCB with the following contents:

DTCREC = 0 (on first call) left alone on additional calls
DTCBFS = address of 256-byte buffer
DTCBPT = left alone between calls
DTCTBL = left alone between calls

A0 = address of 11 character string containing the formatted. filename/extension to be found. The first 8 characters is the filename, left-justified and padded with blanks. The last 3 characters is the file extension left-justified and padded with blanks. If characters in the name are to be wildcarded a '?' replaces that character.

Upon return from SRCDIR, if register D7.W is zero (no SYSIO errors) register A1 will be valid. Register A1 points to the 11 character filename found. When A1 is zero, the end of the directory has been reached with no (additional) filename found.

20.9 TIME OPERATIONS

23 GETTIM

This system call retrieves the time-of-day and date and returns it in registers D1 and D2 as binary bytes. The format of the registers is as follows:

```
D1.L = X:S:M:H   where

       X = undefined
       S = seconds (0-59)
       M = minutes (0-59)
       H = hours (0-23)

D2.L = W:D:M:Y:  where

       W = day of week (1-7)
       D = day of month (1-31)
       M = month of year (1-12)
       Y = year (0-99)
```

24 SETTIM

This system call sets the time according to the values in registers D1 and D2. The register contents should be formatted as shown in GETTIM. If the time or date is invalid then register D0 will contain a 1. If no error occurred, D0 will be cleared.

34 FMTTOD

This routine takes the output of GETTIM (D1.L and D2.L) and formats it into an 18 or 22 character ASCII string pointed to by register A6. A6 is updated by FMTTOD to point to the next byte past the formatted time. The format of the ASCII string is as follows:

```
[WWW] DD MMM YY HH:MM:SS

WWW   day of the week
DD    numerical day
MMM   month of year
YY    year (83)
HH    hour (00-23)
MM    minute (00-59)
SS    second (00-59)
```

The 18 character conversion is done when the day of the week is missing (high-order byte of D2 is zero).

20.10 LOADING PROGRAMS

11 CHAIN

This routine is the same as LOADB with the additional feature of transferring control immediately to the transfer address of the program loaded. It is useful when you wish to overlay the current program module with another program module.

14 LOADB

This routine loads a program into RAM memory at the addresses specified during the ALINK or SAVE process. Input to LOADB is:

A6 = address of 74-byte DIB containing:

```
DIBVOL
DIBNAM
DIBEXT
DIBDTD = 1 (read only)
DIBxxx = 0 (all other fields cleared)
```

D5.B = LUN that LOADB can use to open the file (program) to be loaded

Upon return from LOADB, register D7.W will contain status and if no errors, register A2 will contain the transfer address of the program loaded. If no transfer address exists, A2 is 0.

The D7.W status maybe errors from SYSIO calls or ones specifically from LOADB:

```
$151   invalid file format
$152   transfer address too low
$153   not enough memory to complete LOADB
```

20.11 MULTITASK SYSTEM CALLS

There are 7 system calls devoted to task control functions. Most require a PCB number in register D0.B as a calling argument. The PCB number

identifies the task by providing internal addressing to the PCB allocated to the task. The user should normally not try to directly address a PCB.

25 GETPCB

This system call acquires a PCB and assigns it to a task. GETPCB starts all tasks under CS-OS. The desired task name is passed in registers D1.L and D2.L. The task's priority is passed in register D0.B. The starting address for the task is passed in register A6. GETPCB returns status in D0.B and the PCB-number allocated to the task in D1.B. The status can take on the following values in hex:

00 = good
01 = no available PCBs
02 = invalid priority
FF = duplicates an existing task name

The value in D1.B is valid only if the status is $00. Otherwise, the task did not start. An example of how a task might be started under CS-OS is illustrated below.

```
LOOP    MOVE.L #'NEWT',D1     name of task is NEWTASK
        MOVE.L #'ASK ',D2
        MOVE.B #100,D0        priority is 100
        LEA.L TASKCD,A6       starting address
        GETPCB                start task
        TST.B D0              good status?
        BNE.S LOOP            no, try again
*
        MOVE.B D1,MYPCB       save PCB number
```

NEWTASK is placed on the ready queue and will run when its time comes. Note that GETPCB affects only registers D0 and D1.

26 GIVPCB

This system call deallocates a PCB and thus "kills" the task associated with it. GIVPCB is called with the PCB-number of the task to be killed in register D0.B (if D0.B=0, then kill currently running task). You use this facility to kill tasks which you had started via GETPCB. If you want to terminate your own task (the current task) then you should use the EXIT system call. GIVPCB returns a status byte in D0.B. The values returned by GIVPCB are:

00 = good
01 = PCB-invalid number
04 = Trying to kill the system task.

If GIVPCB returns nonzero status, the task still exists. Note: It is possible for a task to kill itself, in which case GIVPCB does not return but instead dispatches a new task to run. GIVPCB affects only register D0.

27 GSTAT

This system call returns information from the PCB whose number is passed in D0.B. If D0.B is zero, GSTAT returns information on the currently-running task PCB, regardless of its actual number. (Note: PCB-1 is normally the "idle" task and PCB-2 is the SYSTEM task.) GSTAT returns the following information:

D0.B	task priority
D1.L, D2.L	task name
D3.L, D4.L	task time-of-generation (in GETTIM format)
D5.B	task status
A5	address of "token-processing" variables for SYSTEM task, zero for other tasks.

You use the INCLUDE file PARDEF50.INC to reference token processing fields. If the PCB-number specified is invalid, GSTAT returns task status (D5.B) as zero. (PCB not allocated.)

28 SETPRI

* Warning * This system call now sets a return code. In release 1.0 it did not set a return code.

This system call changes the priority value of the task whose associated PCB-number is passed in D0.B. If D0.B is zero, SETPRI will use the currently-running PCB regardless of its actual PCB-number. The priority value desired is passed in D1.B. If the priority is less than 1 or greater than 127, SETPRI returns a 2 in D7.W. If the PCB-number is less than 0 or greater than 15, SETPRI returns a 1 in D7.W. Otherwise, if the priority is adjusted correctly, SETPRI returns a 0 in D7.W.

29 DELAY

This system call suspends the execution of a task for a specified number of time-slices. DELAY is called with the PCB-number of the desired task in D0.B. If D0.B is zero, the currently-running task will be used regardless of its actual PCB-number. The desired number of time-slices to delay is passed in D1.L. DELAY requires that the task to be delayed have a name (so that "idle" cannot be delayed). DELAY does not change any registers.

30 WAKEUP

This system call clears the PCBDEL field in the PCB whose PCB-number is passed in D0.B. If D0.B is zero, currently-running task will be used regardless of its actual PCB-number. Otherwise, if the priority is adjusted WAKEUP overrides DELAY and the task associated with the PCB will resume its place on the ready-queue (or device queue). WAKEUP changes no registers.

43 EXIT

This system call replaces RTS/GIVPCB in user BINS and tasks . It releases resources the user may not have released, terminates the current task if it is not the SYSTEM task, and does an RTS if it is the SYSTEM task.

20.12 PROGRAM DEVELOPMENT

INVOKING PROGRAMS AND TASKS

CSOS allows you to "call" a program from the command line interpreter which runs under the SYSTEM task. It also allows you to spawn new tasks by use of the RUNTASK command or GETPCB system call.

20.12.1 PROGRAMS CALLED FROM THE SYSTEM TASK.

You may write programs which can be invoked by simply typing their name, like "HELP" and "COPY". Such programs are loaded into user memory by the command line interpreter with control transferred to them via a subroutine call to the program transfer address.

On entry, your program will be in user mode. There is a default user stack of 400 bytes. It can be used if large enough for your programs requirements, otherwise you must establish your own user stack of sufficient size.

On exit, your program will return to the command line interpreter via the EXIT System Call. If you had established your own user stack then you must restore the original pointer before returning.

Programs called from the SYSTEM task may use certain reserved logical unit numbers established at cold start time for communicating with the keyboard and CRT screen. LUN 249 is the output logical unit for #SCRN0 configured

as a 24x80 scroll display. LUN 246 is the input logical unit for #CON which echoes to #CNSL0, using logical unit 250.

SYSTEM task programs may also use some special system calls for handling command input. These include GETCMD, GETCMDE, NXTOK, PRTMSG, AND PRTERMSG. (See Command-Parsing routines.)

20.12.2 TASKS STARTED BY RUNTASK OR GETPCB

The GETPCB system call establishes a new task separate from the calling task, with its own task control block (PCB) and its own priority. You can write programs which perform a GETPCB to establish a second instruction stream, or you can use the RUNTASK system command which loads in a program from disk and then performs a GETPCB at the priority specified in the command syntax.

Tasks run in user mode. You must establish a user stack to allow for subroutine calls (BSR, JSR) and for register saving. Remember that the stack pointer should be set to the top of the stack.

You use the EXIT system call to terminate the task.

20.12.3 MEMORY AVAILABLE TO APPLICATION PROGRAMS

You may obtain the addresses of available memory by issuing the GBASPTR system call. This call returns the pointers APPBEGIN and APPEND which are the lower and upper bounds of unprotected user memory. The operating system reserves the memory below APPBEGIN for 68000 exception handling vectors and for system work areas. It also may reserve memory above APPEND to RAMEND for certain optional device driver requirements. Operating system memory is storage protected, therefore you cannot modify it from a program running in user mode. An attempt to do this, whether intentional or accidental, will result in a protect violation and the abnormal termination of your program.

20.12.4 MANAGING MEMORY AMONG SEVERAL TASKS

CSOS does not provide a facility for allocating memory dynamically to programs at execution time. It is the responsibility of the application programmer to pre-plan the use of memory for all programs and to establish

separate user stack areas for each task. Note that the memory values obtained from GBASPTR are globally available memory bounds, accessible to all tasks. The proper allocation of memory into separate areas for each task cannot be left to chance, but instead must be carefully thought out with regard to the requirements of each task.

20.12.5 PERFORMING SCREEN I/O

The application programmer is responsible for the layout of the window, setting of the various window attributes, and providing for keyboard echoing and prompting. There is no provision in CSOS for shared use of the same window by multiple tasks.

A.0 APPENDIX A - ERROR MESSAGES AND CODES

A.1 ERROR MESSAGES FROM OPERATING SYSTEM COMMANDS

The following messages are those that occur most frequently. For a complete list of errors related to a command, see the command description in Chapter 1.

General Messages

COMMAND FORMAT ERROR	Data entered cannot be processed as a command.
UPPER CASE REQUIRED	Command must be entered in uppercase.
CHECK SPELLING OF COMMAND OR PROGRAM NAME	Command not found.

Messages used by many commands.

SYNTAX ERROR	The command line does not conform to the syntax specified for the command.
BAD PARAMETER	A bad argument in the command line was encountered.

The following messages may occur during program loading --

LOAD ADDRESS TOO LOW	Starting address must be above APPBEGIN.
NOT ENOUGH MEMORY FOR LOAD	Ending address must be below.
INVALID LOAD FILE FORMAT	File type must be binary, type 00 or 01.

The next two error messages result from use of the RENAME command:

DUPLICATE NAME	Indicates that the new name already exists on the disk.

FILE PROTECTED	Indicates that the old file is protected from renaming (access code = 02, 03, 06, or 07).

The next error message is from the SET command

ILLEGAL VALUE	Value entered is unreasonable for the type of command

The next 3 messages come from the RUNTASK command:

NO TRANSFER ADDRESS	The binary load module specified in the RUNTASK command does not have a transfer address.
NO MORE TASKS MAY BE STARTED	There is no PCB available to start a task with the run command. All PCBs are in use.
DUPLICATE TASK NAME	There is already a PCB with the name specified in the RUNTASK command.

The next 3 messages are related to commands involving tasks (i.e., PRIority, DELAY WAKEUP, RUNTASK, KILL, DELAY, RESUME, SHOW)

NO SUCH TASK	There is no PCB with a task name that matches the one entered.
INVALID PRIORITY	A priority outside the range 1-27 was used in a RUNTASK or PRIORITY command.
SYSTEM TASK CANNOT BE KILLED	You cannot issue the KILL command for the system task.

The next two messages come from the SAVE command:

INVALID TRANSFER ADDRESS	Transfer address must be between the start address and end address and be on an even boundary.
ENDING ADDRESS IS TOO LOW OR TOO HIGH	Ending address must be between starting address and end of system memory.

The next two messages come from the SECURE command:

DIR.DIR FILE ACCESS CANNOT BE CHANGED	DIR.DIR is the directory file on the disk. Its access code is not to be changed.
VALUE RANGE ERROR	Access code specified is not acceptable.

The next two messagers come from the SUBMIT command:

SUBMIT FILE ERROR	An error occurred while trying to open the requested SUBMIT file or a command line exceeded 80 characters.
WRONG FILE TYPE	A submit file must be text or type 3.

File-dependent messages

WRITE TO OLD-STRUCTURED FILE
FILE ALREADY OPEN FOR WRITE
FILE NOT FOUND
ILLEGAL FILE ACCESS METHOD
INVALID FILENAME
READ BEYOND END OF FILE
BUFFER SIZE INCORRECT
FILE NOT EXTENDABLE
DISK FULL
WRITING TO READY-ONLY FILE
WRONG FILE TYPE FOR ACCESS
INVALID SECTOR SPECIFIED
FILE PROTECTED
NEW FILE BEING READ
BAD FILE -- SYSTEM ERROR
NO SPACE FOR CONTIGUOUS FILE

A.2 COMMON DEVICE AND MANAGER ERROR CODES

A.2.1 MESSAGE FORMAT

The Syntax for messages is:

component [ERROR=$NNNN][message text][TASK=taskname]

COMPONENT is the name of a device or operating system manager issuing the message.

ERROR=$NNNN is a four digit hexadecimal error code which identifies the error. Driver programs issue error codes in the range $0001 through $00FE. Manager programs issue the remaining error codes. It is possible for a device driver to issue a manager error return code. You can find the reason for this code by looking through the manager error codes section.

This field of the message display is always printed if no message text is available. If message text is available then this field is printed only if you have SET EC=YES.

MESSAGE TEXT is supplied by device drivers and system managers for many of the error codes listed in this section. Message text is printed with the message, if it is available.

TASK=TASKNAME is printed with the error message for all tasks other than the SYSTEM task.

A.2.2 COMMON DEVICE-DRIVER ERROR CODES

$0001 reserved
$0002 reserved
$0003 reserved
$0004 NO CONTROL BLOCK STORAGE
$0005 READ/WRITE ERROR
$0006 INVALID DATA TRANSFER DIRECTION IN DIB
$0007 ILLEGAL BUFFER ADDRESS IN DTCB
$0008 END OF FILE
$0009 RECORD LARGER THAN BUFFER; TRUNCATED - input record too big
$000A DEVICE NOT READY
$000B NON-ZERO BYTE I/O WRITE STATUS - buffer full
$000C NON-ZERO BYTE I/O READ STATUS - buffer empty
$000D SYSIO REQUEST CANCELLED
$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$000F INVALID TRANSFER MODE IN DIB

$0021 INVALID VALUE FOR SETTRANS FUNCTION PACKET
$0022-$005F have a common meaning for function packet data errors in all drivers. The error code is equal to the function code number plus $0020. The message produced by the system is as follows:

ERROR=$NNNN INVALID DATA IN FUNCTION $NNNN AT OFFSET $NNNN.

$0060 CANNOT READ - DIBDTD SPECIFIED OUTPUT
$0061 CANCEL FAILED
$0062 READ NOT SUPPORTED - by device driver
$0063 WRITE NOT SUPPORTED - by device driver
$0064 FUNCTION NOT SUPPORTED - by device driver
$0065 BWRITE NOT SUPPORTED - by device driver
$0066 BREAD NOT SUPPORTED - by device driver
$0067 CANNOT WRITE - DIBDTD SPECIFIED INPUT
$0068 TSTBYTE NOT SUPPORTED - by device driver
$0069 DTACHDVR NOT SUPPORTED - by device driver
$006A ATACHDEV NOT SUPPORTED - by device driver
$006B DTACHDEV NOT SUPPORTED - by device driver
$006C reserved
$006D reserved
$006E reserved
$006F reserved

A.2.3 MANAGER ERROR CODES

RTMMGR ERROR CODE

$2000 RTMMGR ERROR

SEMAPHORE ERROR CODES

$3001 DUPLICATE SEMAPHORE NAME
$3002 ALL SEMAPHORES IN USE
$3003 ILLEGAL MAXIMUM COUNT
$3004 COUNT EXCEEDS MAXIMUM COUNT - will be truncated
$3005 ILLEGAL QUEUING MODE
$3006 ILLEGAL SEMAPHORE NAME OR SYSTEM I.D.
$3007 NOT OWNER TASK - cannot detach semaphore
$3008 REQUESTED COUNT EXCEEDS MAXIMUM COUNT
$3009 INSUFFICIENT SYSTEM SPACE TO QUEUE REQUEST
$300A SEMAPHORE HAS BEEN DETACHED - request terminated
$300B INVALID COUNT
$300C SEMAPHORE NOT FOUND
$300D INVALID TIME OUT VALUE
$300E REQUEST TIMED OUT
$300F FUNCTION NOT SUPPORTED
$3010 INITIAL COUNT EXCEEDS MAXIMUM COUNT - will be truncated
$3010 REQUEST CANCELLED

MEMORY MANAGEMENT ERROR CODES

$6001 INVALID NAME IN MPD
$6002 DUPLICATE MEMORY POOL NAME
$6003 NO AVAILABLE FPD BLOCKS - internal storage
$6004 ILLEGAL NAME - passed as function argument
$6005 MEMORY POOL NOT FOUND
$6006 POOL STILL CONTAINS MEMORY - descriptor cannot be removed
$6007 ILLEGAL SUB POOL SIZE - must be greater than zero
$6008 ILLEGAL SUB POOL BOUNDARY - memory must be aligned on page boundary
$6009 SUB POOL OVERLAP - memory overlaps existing sub pool
$600A SUB POOL OVERLAPS SYSTEM AREA
$600B NO AVAILABLE SPD BLOCKS - internal storage
$600C SUB POOL MEMORY NOT FOUND
$600D ILLEGAL DMA MEMORY REQUEST - greater than 64K
$600E reserved
$600F FUNCTION NOT SUPPORTED
$6010 MEMORY NOT AVAILABLE - see Appendix F, of Operating System Reference Manual, Part 2 (GC22-9199)
$6011 reserved
$6012 MEMORY CANNOT BE RETURNED

I/O MANAGER ERROR CODES

$8100 DUPLICATE VOLUME IDENTIFIER
$8200 INVALID SYSIO CALL
$8300 LOGICAL UNIT NOT OPENED.
$8400 NO CONTROL BLOCK STORAGE - inadequate system space.
See Appendix F, "System Memory Consumption," in the Operating System Reference Manual, Part 1 (GC22-9199).
$8500 DUPLICATE LOGICAL UNIT
$8600 DEVICE NOT FOUND
$8700 NOT DEVICE OWNER
$8800 NON SHARABLE DEVICE ALREADY OPEN
$8900 BYTE I/O NOT SUPPORTED
$8A00 PROCESSING I/O REQUEST ALREADY
$8B00 NOT OPENED FOR BYTE I/O
$8C00 INVALID DIB FIELD (TRN, DTD, or RS0)
$8D00 EVENT NOT FOUND
$8E00 EVENT NOT OPENED
$8F00 ILLEGAL READ BUFFER ADDRESS
$9000 CONTROL BLOCK NOT WORD-ALIGNED
$9100 CONTROL BLOCK OR BUFFER OUT OF RANGE

A.2.4 CODES FOR ASYNCHRONOUS REQUESTS

SYSIO will return a -1 in register D7.W if the asynchronous operation has started successfully. Completion status is returned in the DTCSTA field of the Data Transfer Control Block. The convention is the same as for register D7 status. A -1 indicates that the operation is not yet complete; a zero indicates complete with no error; and a positive number indicates completion with an error.

A.3 DRIVER ERROR CODES

A.3.1 CRT GRAPHICS DRIVER (#GR)

$0021-$0041	Data out of limits for a function packet Function number = error number - $0020

A.3.2 CRT DISPLAY DRIVER (#SCRN, #CNSL)

$000E	INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0015	MAXIMUM NUMBER OF WINDOWS OPENED
$0016	ADDRESS BOUNDARY ERROR - OPEN FAIL
$0021	DATA OUT OF LIMITS FOR A FUNCTION PACKET
through	(Function number = error number - $0020)
$004B	
$0069	DTACHDRV NOT SUPPORTED
$006A	ATTACHDEV NOT SUPPORTED
$006B	DTACHDEV NOT SUPPORTED

A.3.3 KEYBOARD DRIVER (#CON)

$0009	RECORD LARGER THAN BUFFER; TRUNCATED
$000C	NON-ZERO BYTE I/O READ STATUS
$000D	REQUEST CANCELLED
$000E	INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0010	KEYBOARD FUNCTION KEY EXCEPTION
$0021	BAD DATA IN SET TRANSFER MODE FUNCTION PACKET
$002B	INVALID TAB AMOUNT
$002D	COMMAND PARSING NOT ENABLED
$0063	WRITE NOT SUPPORTED
$0065	BWRITE NOT SUPPORTED
$006A	ATCHDEV NOT SUPPORTED
$006B	DTACHDEV NOT SUPPORTED

A.3.4 KEYPAD DRIVER (#KPD)

$0009 RECORD LARGER THAN BUFFER; TRUNCATED
$000C NON-ZERO BYTE I/O READ STATUS
$000D REQUEST CANCELLED
$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0011 (FPKT 12, 19) SCANCODE ALREADY IN TABLE
$0012 (FPKT 12, 19) NOT ENOUGH SPACE IN TABLE
$0013 (FPKT 13) ERROR IN TABLE STRUCTURE
$0014 (FPKT 13) SCANCODE NOT FOUND IN TABLE
$0015 (FPKT 12, 19, 13) INVALID TABLE NUMBER
$0016 (FPKT 12) ILLOGICAL SCANCODE FOR TABLE
$0017 (FPKT 12) STRING LENGTH <GT> 20
$0018 (FPKT 12, 19) BAD TERMINATOR, NOT $OD OR $04
$0019 (FPKT 12, 19) INVALID BUFFER CODE
$001E (FPKT 17) TABLES NOT EMPTY, CAN'T REALLOCATE
$001F (FPKT 17) NOT ENOUGH MEMORY FOR TABLES
$0021 (FPKT 01) DIBTRN MODE NOT 0 OR 1
$0024 (FPKT 03) BAD TIME VALUE (MINUS OR >256)
$0026 (FPKT 06) INVALID LED NUMBER
$0027 (FPKT 07) INVALID LED NUMBER
$002E (FPKT 14) BAD TABLE #, MUST BE 1-4
$002F (FPKT 15) INVALID BUFFER CODE
$0031 (FPKT 17) REQUESTED TABLE SIZE NEGATIVE
$0032 (FPKT 19) STRING TOO LONG OR BAD TERMINATOR
$0035 (FPKT 21) INVALID ENTER/SHIFT KEY
$0063 WRITE NOT SUPPORTED
$0065 BWRITE NOT SUPPORTED
$006A ATCHDEV NOT SUPPORTED
$006B DTACHDEV NOT SUPPORTED

A.3.5 PRINTER DRIVER (#PR)

$0006 INVALID DATA TRANSFER DIRECTION IN DIB
$0009 RECORD LARGER THAN BUFFER; TRUNCATED
$000A DEVICE NOT READY
$000B NON-ZERO BYTE I/O - WRITE STATUS
$000D REQUEST CANCELLED
$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$000F INVALID TRANSFER MODE IN DIB
$0010 PRINTER DRIVER COLDSTART FAILED
$0012 PARAMETER ERROR IN DTCB
$0014 ONLY FIXED LENGTH TRANSFER ALLOWED IN GRAPHICS
$0019 DETACH DRIVER FAILED
$0021-$0045 DATA OUT OF LIMITS FOR A FUNCTION PACKET
FUNCTION NUMBER = ERROR NUMBER -$0020.
$0061 CANCEL FAILED

A.3.6 RS-232 DRIVER (#SER)

```
$0009  RECORD LARGER THAN BUFFER; TRUNCATED
$000A  DEVICE NOT READY
$000D  REQUEST CANCELLED
$000E  INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0020  DTCB BUFFER FULL BEFORE READ
$0022  INTMGR ERROR DURING DMA OPERATION
$0030  DATA SUSPECT:  PARITY ERROR DETECTED
$0031  DATA SUSPECT:  FRAMING ERROR DETECTED
$0032  DATA LOST:  CIRCULAR BUFFER OVERRUN
$0033  DATA LOST:  HARDWARE OVERRUN
$0034  BREAK RECEIVED
```

A.3.7 IEEE-488 DRIVER (#BUS)

```
$0009  RECORD LARGER THAN BUFFER; TRUNCATED
$000A  DEVICE NOT READY
$000D  REQUEST CANCELLED
$000E  INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0020  CONTROLLER NOT ATTACHED
$0021  CONTROLLER ALREADY ATTACHED
$0022  DUPLICATE OR INVALID BUS ADDRESS
$0023  ATTACH WOULD EXCEED MAXIMUM DEVICE COUNT
$0024  DRIVER CANNOT DETACH:  DEVICES ACTIVE
$0030  WRITE NOT STARTED: NOT ADDRESSED AS TALKER
$0031  WRITE ABORTED: TIMEOUT ON DMA DATA WRITE
$0032  WRITE ABORTED: ERROR DURING COMMAND SEQUENCE
$0033  WRITE ABORTED: TIMEOUT ON DATA
$0034  WRITE ABORTED:  LOST TALKER STATE
$0035  WRITE NOT STARTED: NULL RECORD
$0040  READ NOT STARTED: NOT ADDRESSED AS LISTENER
$0041  READ ABORTED: TIMEOUT ON DMA DATA READ
$0042  READ ABORTED: ERROR DURING COMMAND SEQUENCE
$0043  READ ABORTED: TIMEOUT ON DATA
$0044  READ ABORTED: LOST LISTENER STATE
```

A.3.8 ITC DRIVER (#ITC)

```
$0006  INVALID DATA TRANSFER DIRECTION IN DIB
$0009  RECORD LARGER THAN BUFFER; TRUNCATED
$000D  REQUEST CANCELLED
```

$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$000F INVALID TRANSFER MODE IN DIB
$0010 DUPLICATE ITC IDENTIFIER
$0011 INVALID ITC IDENTIFIER
$0012 INVALID ATTACH/DETACH CODE
$0015 LOWER TRIGGER BYTE EXCEEDS UPPER TRIGGER BYTE
$0016 BUFFER OFFSET EXCEEDS BUFFER LENGTH
$0017 EITHER BUFFER LENGTH OR OFFSET ILLEGAL
$0018 NO AVAILABLE SYSTEM MEMORY
$0019 RETURN OF SYSTEM MEMORY FAILED
$0021 INVALID TRANSFER MODE
$0023 INVALID TIME OUT PARAMETER
$0062 READ NOT SUPPORTED
$0063 WRITE NOT SUPPORTED
$0065 BWRITE NOT SUPPORTED
$0066 BREAD NOT SUPPORTED
$0068 TSTBYTE NOT SUPPORTED
$006A ATACHDEV NOT SUPPORTED
$006B DTACHDEV NOT SUPPORTED
$0070 WRITE WAIT FAILED
$0071 READ WAIT FAILED
$0072 READ SIGNAL FAILED
$0073 WRITE SIGNAL FAILED
$0075 REQUEST TIMED OUT
$0076 CHANNEL(S) IN USE DRIVER WILL NOT BE DETACHED
$0077 ERROR IN COLDSTART SEQUENCE
$0078 ERROR IN DETACH SEQUENCE - DETACH INCOMPLETE
$0079 DRIVER NOT ATTACHED
$007A DRIVER IS ATTACHED

A.3.9 PARALLEL PORT DRIVER (#PPU)

$0005 READ ERROR - trying to read in printer output mode
$000A DEVICE NOT READY
$000B NON-ZERO BYTE I/O WRITE STATUS - buffer full
$000C NON-ZERO BYTE I/O READ STATUS - buffer empty
$000D REQUEST CANCELLED
$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$000F INVALID TRANSFER MODE IN DIB
$0021 INVALID VALUE FOR SET TRANSFER MODE DATA
$0023 SET PARALLEL PORT MODE DATA INCORRECT
$0025 SET TIMEOUT DATA INCORRECT
$0027 SET AUTO LINEFEED DATA INCORRECT
$0068 BTEST IS NOT SUPPORTED BY THIS DRIVER
$006A ATCHDEV NOT SUPPORTED
$006B DTACHDEV NOT SUPPORTED

A.3.10 DISK DRIVERS (#FD0X OR #HD0X)

$0005 READ/WRITE ERROR
$0006 INVALID DATA TRANSFER DIRECTION IN DIB
$0007 ILLEGAL BUFFER ADDRESS IN DTCB
$000A DEVICE NOT READY
$000E INVALID FUNCTION PACKET CODE AT OFFSET $NNNN
$0010 SEEK TRACK
$0011 LOGICAL SECTOR OR TRACK NUMBER TOO BIG $**********
$0012 VOLUME CHANGED
$0013 reserved
$0014 I/O REQUEST TIMED OUT
$0015 reserved
$0016 ILLEGAL BUFFER ADDRESS
$0017 DISK FORMAT NOT RECOGNIZED
$0018 DISK WRITE PROTECTED
$0019 SECTOR BUFFER TOO SMALL
$001A WRITE FAULT $**********
$001B CRC ERROR $**********
$001C SECTOR NOT FOUND $**********
$002F ILLEGAL VOLUME IDENTIFIER
$0061 CANCEL FAILED
$0065 BWRITE NOT SUPPORTED
$0066 BREAD NOT SUPPORTED
$0067 CANNOT WRITE - DIBDTD SPECIFIED INPUT
$0068 TSTBYT NOT SUPPORTED
$006A ATCHDEV NOT SUPPORTED
$006B DTACHDEV NOT SUPPORTED
$0070 DATA ADDRESS MARK NOT FOUND $**********
$0071 reserved
$0072 ABORTED COMMAND
$0073 reserved
$0074 reserved
$0075 reserved
$0076 UNCORRECTABLE DISK ERROR $**********
$0077 BAD SECTOR DETECTED $**********
$0078 HARD DISK CONTROLLER NOT PRESENT

A.3.11 SENSOR I/O DRIVERS

$0010 DEVICE LOCKED
$0011 DEVICE ALREADY OPEN
$0012 A/D OVERANGE
$0013 CTC TIMER OVERRUN
$0014 ILLEGAL OPEN MODE
$0015 A/D TIME-OVERRUN

A.4 ABNORMAL-TERMINATION SCREEN

CSOS includes a facility for detecting processor TRAPS and providing a display of the pertinent information available as an aid in troubleshooting.

There are two types of TRAPS:

TYPE 1 Standard processor TRAP

includes
- OPCO invalid OP code trap
- DIVO divide by zero trap
- CHKC check instruction trap
- TRPV Trap V instruction
- PRIV privilege violation
- 1010 illegal instruction
- 1111 illegal instruction
- TR13 unexpected trap 13
- TR14 unexpected trap 14
- ?INT unexpected miscellaneous trap
- ABRT Abort button interrupt

TYPE 2 Extended information TRAPS

- ADDR illegal address trap; word operand on odd address
- SPUR spurious interrupt trap
- BUS bus error trap
- ABUS address bus error
- DBUS data bus error
- PROT memory protection error; attempt to store into system memory
- DTAK missing DTACK error
  (This message can result from an attempt to address memory or devices that are not implemented on your machine. Every access must terminate with "Data Transfer Acknowledge" or DTACK. If it does not, an error message is generated.)
- MPAR parity error
- POWR power failure error

TRAP DISPLAY FORMAT

```
FNC=XXXX  ADD=XXXXXXXX  INR=XXXX     } TYPE 2 ONLY

TASK=TASKNAME      XXXX TRAP ERROR

PC=XXXXXX      SR=XXXX        USP=XXXXXX     SSP=XXXXXX
```

```
D0=XXXXXXXX   D1=XXXXXXXX   D2=XXXXXXXX   D3=XXXXXXXX
D4=XXXXXXXX   D5=XXXXXXXX   D6=XXXXXXXX   D7=XXXXXXXX
A0=XXXXXXXX   A1=XXXXXXXX   A2=XXXXXXXX   A3=XXXXXXXX
A4=XXXXXXXX   A5=XXXXXXXX   A6=XXXXXXXX   A7=XXXXXXXX


PRESS ANY KEY TO REBOOT
```

NOTES

The extended information for type 2 traps is:

FNC = processor function code
ADD = access address at time of trap
INR = instruction register at time of trap

A.5 MACRO ASSEMBLER

A.5.1 ERROR MESSAGES

Error messages generated during an assembly may originate from the assembler or from Pascal or the operating system environment. Assembler-generated messages may be of two forms:

1. *******ERROR xxx -- nnnn

   where xxx is the number of the error (defined in the list in this appendix), and nnnn is the number of the line where the previous error occurred.

   Errors indicate that the assembler is unable to interpret or implement the intent of a source line.

2. *******WARNING xxx -- nnnn

   where xxx is the number of the error (defined in the list in this appendix), and nnnn is the number of the line where the previous error occurred.

   Warnings may indicate possible recoverable errors in the source code, or that a more optimal instruction format is possible.

ERROR CODE	MEANING OF ERROR
	SYNTACTIC ERRORS
200	ILLEGAL CHARACTER (IN CONTEXT)
201	SIZE CODE/EXTENSION IS INVALID
202	SYNTAX ERROR
203	SIZE CODE/EXTENSION NOT ALLOWED
204	LABEL REQUIRED
205	END DIRECTIVE MISSING
206	REGISTER RANGES FOR THE MOVEM INSTRUCTION MUST BE SPECIFIED IN INCREASING ORDER
207	A AND D REGISTERS CAN'T BE INTERMIXED IN A MOVEM REGISTER RANGE
	OPERAND/ADDRESS MODE ERRORS
210	MISSING OPERAND(S)
211	TOO MANY OPERANDS FOR THIS INSTRUCTION
212	IMPROPER TERMINATION OF OPERAND FIELD
213	ILLEGAL ADDRESS MODE FOR THIS OPERAND
214	ILLEGAL FORWARD REFERENCE
215	SYMBOL/EXPRESSION MUST BE ABSOLUTE
216	IMMEDIATE SOURCE OPERAND REQUIRED
217	ILLEGAL REGISTER FOR THIS INSTRUCTION
218	ILLEGAL OPERATION ON A RELATIVE SYMBOL
219	MEMORY SHIFTS MAY ONLY BE SINGLE BIT
220	INVALID SHIFT COUNT
221	INVALID SECTION NUMBER
	SYMBOL DEFINITION
230	ATTEMPT TO REDEFINE A RESERVED SYMBOL
231	ATTEMPT TO REDEFINE A MACRO; NEW DEFINITION IGNORED
232	ATTEMPT TO REDEFINE THE COMMAND LINE LOCATION
233	COMMAND LINE LENGTH MUST BE > 0; IGNORED
234	REDEFINED SYMBOL
235	UNDEFINED SYMBOL
236	PHASING ERROR ON PASS2
237	START ADDRESS MUST BE IN THIS MODULE, IF SPECIFIED
238	UNDEFINED OPERATION (OPCODE)
239	NAMED COMMON SYMBOL MAY NOT BE XDEF
	DATA SIZE RESTRICTIONS
250	DISPLACEMENT SIZE ERROR
251	VALUE TOO LARGE
252	ADDRESS TOO LARGE FOR FORCED ABSOLUTE SHORT
253	BYTE MODE NOT ALLOWED FOR THIS OPCODE

254 MULTIPLICATION OVERFLOW
255 DIVISION BY ZERO

MACRO ERRORS

260 MISPLACED MACRO, MEXIT, OR ENDM DIRECTIVE
261 MACRO DEFINITIONS MAY NOT BE NESTED
262 ILLEGAL PARAMETER DESIGNATION
263 A PERIOD MAY OCCUR ONLY AS THE FIRST CHARACTER IN A MACRO NAME
264 MISSING PARAMETER REFERENCE
265 TOO MANY PARAMETERS IN THIS MACRO CALL
266 REFERENCE PRECEDES MACRO DEFINITION
267 OVERFLOW OF INPUT BUFFER DURING MACRO TEXT EXPANSION

CONDITIONAL ASSEMBLY ERRORS

270 UNEXPECTED 'ENDC'
271 BAD ENDING TO CONDITIONAL ASSEMBLY STRUCTURE (ENDC EXPECTED)

STRUCTURED SYNTAX ERRORS

280 MISPLACED STRUCTURED CONTROL DIRECTIVE (IGNORED)
281 MISSING "ENDI"
282 MISSING "ENDF"
283 MISSING "ENDW"
284 MISSING "UNTIL"
285 UNRESOLVED SYNTAX ERROR IN THE PRECEDING PARAMETERIZED
STRUCTURED CONTROL DIRECTIVE; RECOVERY ATTEMPTED WITH
THE CURRENT LINE
286 "=" EXPECTED; CHARACTERS UP TO "=" IGNORED
287 "<" EXPECTED; CHARACTERS UP TO "<" IGNORED
288 ">" EXPECTED; CHARACTERS UP TO ">" IGNORED
289 "DO" EXPECTED: REMAINDER OF LINE IGNORED
290 "THEN" EXPECTED; REMAINDER OF LINE IGNORED
291 "TO" OR "DOWNTO" EXPECTED; "TO" ASSUMED
292 ILLEGAL CONDITION CODE SPECIFIED

MISCELLANEOUS

300 IMPLEMENTATION RESTRICTION
301 TOO MANY RELOCATABLE SYMBOLS REFERENCED
<LINKAGE EDITOR RESTRICTED>
302 RELOCATION OF BYTE FIELD ATTEMPTED
303 ABSOLUTE SECTION OF LENGTH ZERO DEFINED (LINK ERROR)
304 NESTED "INCLUDE" FILES NOT ALLOWED; IGNORED
305 FILE NAME REQUIRED IN OPERAND FIELD

INTERNAL ERRORS

400
•
•
•
499

SOURCE CODE NOT OPTIMAL OR RECOVERABLE ERRORS

500	THIS BYTE WILL BE SIGN-EXTENDED TO 32 BITS
501	MISSING PARAMETER REFERENCE IN MACTRO SOURCE
502	TOO MANY PARAMETERS IN THIS MACRO CALL
550	THIS BRANCH COULD BE SHORT
551	THIS ABSOLUTE ADDRESS COULD BE SHORT

NOTE: If more than 10 errors occur in one line, the message

****** TOO MANY ERRORS ON THIS LINE

will be generated.

B.0 APPENDIX B - COMMAND SUMMARY

I/O COMMAND	DESCRIPTION
SYSIO-OPEN	Assigns a logical unit number (LUN) to a file or device specified in the Device Initialization Block (DIB).
SYSIO-CLOSE	De-assigns a LUN.
SYSIO-FUNCTION	Handles device specific requests defined in a Function Packet.
SYSIO-AREAD	Initiates an asynchronous or non-blocking transfer from a device or device buffer into a user buffer using the Data Transfer Control Block (DTCB) of the request.
SYSIO-SREAD	Initiates a synchronous transfer from a device or device buffer into a user buffer using the Data Transfer Control Block (DTCB).
SYSIO-AWRITE	Initiates an asynchronous or non-blocking transfer from a user buffer to the device using the Data Transfer Control Block (DTCB).
SYSIO-SWRITE	Initiates a synchronous transfer from a user buffer directly to the device using the Data Transfer Control Block (DTCB).
SYSIO-CANCEL	Prematurely terminates and discards all I/O requests currently associated with the specified LUN.
SYSIO-BREAD	Performs a byte-read operation using the system identifier obtained at OPEN time.
SYSIO-BWRITE	Performs a byte-write operation using the the system identifier obtained at OPEN time.
SYSIO-BTEST	Performs a test to determine whether a specified buffer contains information.

SYSTEM SERVICE COMMAND	DESCRIPTION
SUSPEND-TILLANY	Suspends the execution of the invoking task's instruction stream until the completion of any outstanding asynchronous request.
SUSPEND-TILLALL	Suspends the execution of the invoking task's instruction stream until the completion of all outstanding asynchronous requests.
SUSPEND-SYNCH	Returns with the current number of outstanding asynchronous requests.
EVENTMGR-OPNEVBLK	Associates an Event Block in user task space with a specific event which has been defined to the system by a device driver.
EVENTMGR-CLSEBLK	Disassociates an Event Block with the event.
EVENTMGR-ARMEVENT	If a user task is suspended, posting of a specified event will resume the task.
RTMGR-OPEN	Seizes the time facility and designates the task data block that will be used for timer control and data/time transfer.
RTMGR-CLOSE	Release the timer facility associated with the RTT that was previously OPENed.
RTMGR-QTIME	Get current date/time.
RTMGR-QALRM	Get current alarm setting.
RTMGR-STOP	Terminate an active alarm.
RTMGR-ALRM	Set absolute date/time alarm.
RTMGR-ALRMR	Set relative date/time alarm.
RTMGR-SUM	Compute sum of date/time for RTT block input and output areas.

C.0 APPENDIX C - SAMPLE CODING

```
*****************************************************************************
*                                                                           *
*     SAMPLE PROGRAM                                                        *
*           COPIES FILE TESTER.LST ON DEFAULT DRIVE TO PRINTER              *
*                  USING SEQUENTIAL(VARIABLE LENGTH) ACCESS METHOD          *
*                                                                           *
*     REGISTERS:     USE                                                    *
*                              A3 - INPUT (READ) BLOCKS, DIB/DTCB           *
*                              A4 - OUTPUT (WRITE) BLOCKS, DIB/DTCB         *
*                                                                           *
*     LUN'S:         1 - INPUT            2 - OUTPUT                        *
*****************************************************************************
          INCLUDE   IOMCLE50.INC
          INCLUDE   DKMCLE50.INC
          INCLUDE   SYSMAC50.INC
SAMPLE    IDNT      0,0         COPY FUNCTION                      08-05-83
*
LINELEN   EQU       132         LINE LENGTH
*---------------------------------------------------------------------------*
*
* CLEAR CONTROL BLOCKS
*
          LEA       INDIB,A3
          MOVE.W    #BLKLEN-1,D6
CLEAR     CLR.B     (A3)+                CLEAR BLOCK BUFFERS
          DBF       D6,CLEAR             ----(DIB'S,DTCB'S)
*
          LEA       INDIB,A3
          MOVE.B    #1,DIBDTD(A3)                  INPUT
          MOVE.L    #'     ',DIBVOL(A3)            INIT DIBVOL TO BLANKS
          MOVE.W    #'   ',DIBVOL+4(A3)               (DEFAULT DRIVE)
          MOVE.B    #1,DIBDTD(A3)                  READ
          MOVE.B    #1,DIBTRN(A3)                  VARIABLE SEQUENTIAL ACCESS
          MOVE.L    #LEVFFK,DIBFCN(A3)             GET FORMAT LEVEL #
          MOVE.L    #'TEST',DIBNAM(A3)             FILE NAME
          MOVE.L    #'ER  ',DIBNAM+4(A3)
          MOVE.B    #'L',DIBEXT(A3)
          MOVE.B    #'S',DIBEXT+1(A3)
          MOVE.B    #'T',DIBEXT+2(A3)
*
          LEA       OUTDIB,A4
          MOVE.L    #'#FR ',DIBVOL(A4)             OUTPUT TO PRINTER
          MOVE.W    #'   ',DIBVOL+4(A4)
          MOVE.B    #1,DIBTRN(A4)                  VARIABLE LENGTH DATA
*
* SET UP DTCB'S
*
          LEA       INDTCB,A3
          MOVE.L    #LINEBUF,DTCBFS(A3)
          MOVE.W    #LINELEN,DTCBFL(A3)
          MOVE.B    ##0D,DTCTEU(A3)                TERMINATION RANGE
          MOVE.B    ##04,DTCTBL(A3)
*
          LEA       OUTDTCB,A4
          MOVE.L    #LINEBUF,DTCBFS(A4)
          MOVE.W    #LINELEN,DTCBFL(A4)
          MOVE.B    ##0D,DTCTEU(A4)
          MOVE.B    ##04,DTCTBL(A4)                TERMINATION RANGE
```

```
*-------------------------------------------------------------------------------*
* COPY FILE TO PRINTER                                                          *
*-------------------------------------------------------------------------------*
          SYSIO     OPEN,#1,INDIB,OPENERR1          OPEN FILE
* TEXT FILE ON A NEW FORMAT STRUCTURE     (DIB ADDR PUT IN A6)
          IF.B  DIBTYP(A6) <EQ> #3 AND.W
                LEVEL <EQ> #2 THEN.S
            SYSIO     OPEN,#2,OUTDIB,OPENERR2        OPEN DEVICE
            REPEAT                              TRANSFER DATA
              CLR.W     DTCBPT(A3)
              SYSIO     SREAD,#1,INDTCB,RDERR        READ RECORD
*
              CLR.W     DTCBPT(A4)
              SYSIO     SWRITE,#2,OUTDTCB,WRERR        WRITE LINE
*
              MOVEA.L   DTCBFS(A4),A1          ;
              MOVE.W    DTCBPT(A4),D0          ;
              MOVE.B    -1(A1,D0.W),D2        GET TERMINATING BYTE
           UNTIL.B    D2 <EQ> #$04          EXIT IF END-OF-FILE MARKER
*
CLOSE     SYSIO     CLOSE,#2,0,CLOSE1               OUTPUT
          ENDI                          TEXT FILE ON NEW FORMAT DISK
CLOSE1    SYSIO     CLOSE,#1,0,CLOSE2               INPUT
CLOSE2    EXIT
*
* ERROR HANDLING
*
OPENERR1  PRTERR
          EXIT
*
OPENERR2  PRTERR
          BRA       CLOSE1
*
*    NORMAL END-OF-FILE WILL BE DISCOVERED HERE UNLESS FILE HAS
*       USED THE UNNECESSARY "$04" END-OF-FILE MARKER
RDERR     IF.W  D7 <NE> #RDBY THEN.S TEST FOR EOF (READ BEYOND END ERROR)
            PRTERR                          ERROR, NOT EOF
          ENDI
          BRA       CLOSE
*
WRERR     PRTERR
          BRA       CLOSE
*
*-------------------------------------------------------------------------------*
* FUNCTION PACKET
*
LEVFPK    DC.W      GETLEV      GET FILE STRUCTURE LEVEL #
LEVEL     DS.W      1
          DC.W      ENDLIST
*-----------------------------------------------------
LINEBUF   DS.B      LINELEN   LINE BUFFER
*-----------------------------------------------------
*  FOLLOWING IS STORAGE CLEARED AT BEGINNING OF PROGRAM
*
BLKLEN    EQU       (DIBEND+DIBLEN+2*DTCLEN)
*
INDIB     DS.B      DIBEND    DEVICE INITIALIZATION BLOCKS
OUTDIB    DS.B      DIBLEN
INDTCB    DS.B      DTCLEN    DATA TRANSFER CONTROL BLOCKS
OUTDTCB   DS.B      DTCLEN
*
          END
```

D.0 APPENDIX D - INCLUDE FILES ON EXTENSION DISKETTE

These files are for use with the Assembler. You use the Assembler INCLUDE statement to merge them with your source statements. These files include EQUATE statements and/or MACRO definitions.

COMDEF50.INC	Equates for use with GBASPTR System Call.
PARDEF50.INC	Equates for use with GSTAT System Call. Used for parsing the command line.
DKMCLB50.INC	Equates for files and disk/diskette. Note: This include must follow IOMCLB50.INC because it references DIB fields.
GRMCLB50.INC	Equates for #GR Graphics driver.
IOMCLB50.INC	Equates for EVB, DIB, DTCB, and common FPKT's. Macros for SYSIO and SUSPEND.
KBMCLB50.INC	Equates for #CON keyboard driver.
KPMCLB50.INC	Equates for #KPD keypad driver.
PPMCLB50.INC	Equates for #PPU parallel port driver.
PRMCLB50.INC	Equates for #PR Printer Driver.
RSMCLB50.INC	Equates for #SERnn Serial Port Driver.
SCMCLB50.INC	Equates for #SCRNO, #SCRN1, #CNSLO Character Screen Driver.
SIMCLB50.INC	Equates for Sensor I/O.
RTMMAC50.INC	Macro and Equates for RTMGR Timer Facility.
SEMMAC50.INC	Macro and Equates for Semaphore Facility.
SYSMAC50.INC	Macros for CS 9000 System Calls. This INCLUDE file contains two sets of macros for displaying messages from your program. One set is TYPE and TYPES, which display messages

from programs running under the SYSTEM task to #SCRN0 via the PRTMSG SYSTEM Call.

You code TYPE <MESSAGE STRING>
To display a message with carriage return-line feed.

You code TYPES <MESSAGE STRING>
To display a message without carriage return-line feed.

There is also a macro set called PRNTINIT and PRNTMSG which can display messages to any open window or device from any task.

You must perform a SYSIO-OPEN to the desired window or device. You then issue PRNTINIT once in your program to generate a DTCB for the message. Code PRNTMSG <message text> for each message. Register D5.B must contain the LUN.

E.0 APPENDIX E - DISK STRUCTURE

This appendix describes the internal disk structure that can be examined with the REPAIR command.

E.1 VOLUME LABEL SECTOR

The Volume Label Sector stores information about the physical characteristics of the disk, and the file structure used on it. It is contained in Logical Sector Number (LSN) 0 on CSOS formatted disks.

The Volume Label Sector is formatted as shown below. (Values in parenthesis are hexadecimal offsets.)

BYTES		DESCRIPTION
0-3	(0)	0 (reserved)
4-9	(4)	Volume identifier in ASCII
10	(A)	Surface indicator (diskette only) ' '=1, '1'=2, 'M'=2D, 'N'=1D
11	(B)	Sector size (diskette only) ' '=128, '1'=256, '2'=512, '3'=1024
12-15	(C)	0 (reserved)
16-19	(10)	'LEVL'
20-22	(14)	0 (reserved)
23	(17)	File structure Level # 1=Linked Sectors (old), 2=Bit Mapped Sectors (new)
24-27	(18)	Pointer to Bad Sector Table (LSN)
28-31	(2C)	Pointer to Bit Map (LSN)
32-35	(20)	Pointer to File Index (LSN)
36-39	(24)	Pointer to Backup File Index (LSN)
40-43	(28)	Pointer to First Diagnostic Area (LSN)
44-45	(2C)	Length First Diagnostic Area in Sectors
46-49	(2E)	Pointer to Second Diagnostic Area (LSN)
50-51	(32)	Length Second Diagnostic Area in Sectors
52-53	(34)	Number of Bytes per Sector (=256)
54-55	(36)	Number of Sectors per Track
56-57	(38)	Number of Heads

58-59	(3A)	Number of Cylinders useable for File System
60-61	(3C)	Disk Type 1=5¼-inch diskette 2=8-inch diskette 3=Reserved 4=10MB Hard Disk
62-69	(3E)	Date and Time Formatted/Initialized
70-71	(46)	Number of Sectors Per Cluster
72-73	(48)	Default Number of 256 Byte Blocks Per Extent
74-77	(4A)	Number of Clusters
78-255	(4E)	Not Used

E.2 BACKUP VOLUME LABEL

A backup copy of the Volume Label is stored in the last LSN on a diskette. For the hard disk, it is stored in the first good LSN before the second diagnostic area.

E.3 BAD SECTOR TABLE

The Bad Sector Table contains a list of defective sectors on the disk media. The LSN of the Bad Sector Table is stored in the Volume Label Sector, and is one sector in length. Each entry in the list is a 32 bit LSN, and the list is terminated with an entry of -1 ($FFFFFFFF). It is only supported on the hard disk.

E.4 BIT MAP

The Bit Map contains one bit for each cluster on the disk. A cluster is a fixed number of contiguous sectors. The number of sectors per cluster is stored in the Volume Label Sector along with the LSN of the Bit Map. A 0 indicates a cluster is in use, and a 1 indicates a cluster is available for assignment. The MSB (most significant bit) of the first byte corresponds to the first cluster on the disk and the LSB (least significant bit) of the first byte corresponds to the eighth cluster on the disk. Unused bits at the end of the last sector of the Bit Map are set to zero.

E.5 FILE INDEX

The File Index is itself a file which contains a 256 byte entry for each file on the volume. The LSN of the File Index is stored in the Volume Label Sector. The first entry is for the File Index itself, the second entry is for the Directory (DIR.DIR), and the third entry is for the Backup File Index. The fourth and successive entries are for user created files.

BYTES		DESCRIPTION
0	(0)	Type of this block (0=Unused, 1=Extension, 2=Defines Contiguous file, 3=Defines Extendable File)
1	(1)	Type of data (0=Binary, 1=Binary with Transfer Address, 2=ASCII Text)
2-3	(2)	(1= file is in directory, 0= file is not in directory)
4-11	(4)	File Name, ASCII
12-14	(C)	Extension, ASCII
15	(F)	Reserved, (=0)
16-17	(10)	Version, Binary (=1)
18-19	(12)	Revision Number
20-27	(14)	Date and Time Created
28-35	(1C)	Date and Time Revised
36-43	(24)	Date and Time Backed Up
44-45	(2C)	Reserved
46-49	(2E)	Reserved for Owner Information
50-53	(32)	Access Control Information
54-57	(36)	User Area
58-61	(3A)	Number of 256 Byte Blocks used in File
62-63	(3E)	Number of Bytes used in Last Block
64-67	(40)	Number of Blocks Desired in each Extent (0=Use Volume Default)
68-72	(44)	Number of Blocks Allocated to File
73	(49)	Number of Extents Defined in Rest of This Block
74-75	(4A)	Pointer to First Extension Entry, 0 if None
76-77	(4C)	Number of Blocks in First Extent
78-81	(4E)	Pointer to First Extent
.		.
.		.
.		.
250-251	(FA)	Number of Blocks in 30th Extent
252-255	(FC)	Pointer to 30th Extent

Note: For the file index of the File Index, the following fields are specially defined:

8-9	(8)	Number of free blocks in File Index
10-11	(A)	Next free File Index entry (block number)

As shown above, the first block allocated for a file contains descriptive information and pointers to up to 30 extents. If a file expands past 30 extents then additional blocks are allocated in the File Index as required. These Extension Blocks are formatted as follows:

BYTES		DESCRIPTION
0	(0)	Type (=1, Extension)
1-6	(1)	Not Used
7	(7)	Number of Extents Defined in the Rest of this Block
8-9	(8)	Pointer to Next Extension Entry, 0 if None
10-11	(A)	Number of Blocks in Next Extent
12-15	(C)	Pointer to Next Extent
.		.
.		.
.		.
250-251	(FA)	Number of Blocks in Last Extent (for this File Index Block)
252-255	(FC)	Pointer to Extent

E.6 DIRECTORY FILE

The Directory is a file which contains a 16 byte entry for each file on the volume. The first entry is for the Directory itself.

BYTES		DESCRIPTION
0-7	(0)	Name in ASCII. First Character Blank if Unused, 0 if End of Directory
8-10	(8)	Extension in ASCII
11	(B)	Reserved (=0)
12-13	(C)	Version Number
14-15	(E)	File Index entry/block number

E.7 BACKUP FILE INDEX

The Backup File Index is a copy of the File Index that is maintained to allow recovery of file information if a defect develops in the File Index. The LSN of the Backup File Index is stored in the Volume Label Sector.

E.8 DIAGNOSTIC AREAS

There are two separate diagnostic areas supported on the hard disk. (They are not supported on diskettes.) Each area is reserved space for use by diagnostic programs only. The LSN's and the length of the First and Second Diagnostic Areas are stored in the Volume Label Sector.

READER'S COMMENT FORM

This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM Instruments, Inc. shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM Instruments, Inc. publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM Instruments, Inc. product to your IBM Instruments, Inc. representative or to the IBM Instruments, Inc. office serving your locality.*

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.

GC22-9200-1

Please do not staple

Fold and Tape

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Instruments, Inc.
P.O. Box 332
Danbury, Ct. 06810

Please do not staple

Fold and tape

IBM Instruments, Inc.
P.O. Box 332
Danbury, Ct. 06810

GC22-9200-1

IBM Instruments, Inc.
P.O. Box 332
Danbury, Ct. 06810