IBM Instruments Inc.

Computer System
Operating System Reference Manual
Part 2. Logical I/O and System Services

Release 1.0

GC22-9200

Preliminary Edition (February 1983)

The contents of this preliminary 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.

00207830

01234567890

PREFACE

This manual describes the programming interfaces to the operating system of the IBM Instruments Computer System. It consists of eleven chapters and an appendix.

- Chapter 1 -- "Logical I/O, Event Posting, and Timer Services"-- introduces the general featues of the programming interfaces.

Chapters 2 through 10 discuss the programming of the various interfaces. These are:

- Chapter 2 -- "Keyboard, Keypad, and Softkey Driver"
- Chapter 3 -- "CRT Display Driver"
- Chapter 4 -- "Printer/Plotter Driver"
- Chapter 5 -- "RS-232 Asynchronous Communications Driver"
- Chapter 6 -- "IEEE-488 Interface Driver"
- Chapter 7 -- "User Parallel Port Driver"
- Chapter 8 -- "Diskette Driver"
- Chapter 9 -- "Sensor I/O Driver"
- Chapter 10-- "CRT Graphics Driver"
- Chapter 11-- "System Services" -- describes the system call routines available to the programmer.
- Appendix -- "Auxiliary Software -- describes several kinds of files that can be used as programming aids.

Publications that discuss related aspects of the Computer System are:

Computer System Product Description, GC22-9183

BASIC Programming Manual, GC22-9184 GC22-9184

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

Computer System Problem Isolation, GC22-9192

The U.S. Federal Government requires that the following statement be printed in operating manuals for all Class A computing devices:

"WARNING - This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with its associated publications, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference."

CONTENTS

1.0 LOGICAL I/O, TIME SERVICES, AND EVENT POSTING

1.1 INTRODUCTION

1.1.1 DEVICE-INDEPENDENT I/O

Data transfers involving specific physical devices can be initiated by instructions within applications programs, but this would require rewriting the applications program when a need arose to redirect data to or from alternate devices. To a great extent this effort can be avoided if device can be referred to in a generic sense or by use of logical labels rather than physical ones. Then the association between logical and physical labels can be deferred until runtime either using command line parameters or user input in response to application program menus. Thus, references within applications programs to logical label X need not be changed if, for example, one desired to redirect printed output from the CRT display to a printer. This could be accomplished by simply changing the definition of X to mean printer rather than CRT just before running a program.

The utility of this device independence to sequential data transfer. For example, data intended for a random access read/write device cannot be redirected to a standard line printer.

1.1.2 LOGICAL UNIT NUMBERS

The Computer System associates or "binds" logical device names, called logical unit numbers, with physical device names by use of a system OPEN command. The syntax of this and other commands is given below. A task may open up to 127 LUNs, limited in addition by system space requirements. Any number of LUNs may be assigned to a single device. However, only a single opening of a given LUN is allowed.

Futher references to an open device need only be by LUN. The LUN is only known to the task that opened it, in other words, logical unit numbers are "local" variables; other tasks may open identical LUN's against an entirely different set of physical devices.

After the task opens the device and anytime before closing, it may change the device state to something other than the default by issuing a FUNCTION command. This command requires a pointer to a data structure called a FUNCTION PACKET. The format of this structure is identical for all devices, with individual fields of the packet containing commands and data unique to an individual device.

1.1.3 SYNCHRONOUS AND ASYNCHRONOUS I/O

Tasks may select to wait for completion of an I/O request (synchronous I/O) by being blocked from access the CPU resorce. Alternatively, a task may select to gain access to the CPU on the basis of priority after initializing an I/O request (asynchronous I/O). This task may check on the completion status of the request by polling, or conserve the CPU resource by suspending its operation until awakened by the request's completion.

1.1.4 ASYNCHRONOUS EVENT POSTING

Asynchronous event posting allows tasks to synchronize their activities with the detection of external interrupt-producing events. System commands allow system recognition of user area data structures containing event flags and status fields allowing a task to either poll or suspend operations until awakened by occurrences of the specified event.

1.1.5 TIMER SERVICES

Users may allocate a timer to start and stop it and conduct inquiries of its current state. In addition, a user may "arm" the timer so that resumption of a suspended task's execution can occur upon expiration of a specified time period.

1.1.6 COMMAND SUMMARY

COMMAND	DESCRIPTION
SYS I/O OPEN	Assigns a logical unit number (LUN) to a file or device specified in the Device Initialization Block.
SYS I/O CLOSE	Designs a LUN. In addition, device specific actions may be taken such as cleanup, return memory to the free pool, etc.
SYS I/O FUNCTION	This command handles device specific requests defined in a Function Packet.
SYS I/O AREAD	Initiates an asynchronous or non-blocking transfer of a record or block from a device or device buffer into a user buffer. The parameters defining the transfer conditions are contained in a Data Transfer Control Block. Execution of the task's instruction stream continues upon completion of querying of the request.
SYS I/O SREAD	Initiates a synchronous transfer of a record or block from a device or device buffer into a user buffer. The parameters defining transfer conditions are contained in a Data Transfer Control Block. Execution of the task's instruction stream is blocked until completion of the transfer is signaled by the device driver.
SYS I/O AWRITE	Initiates an asynchronous or non-blocking transfer of a record or block from a user buffer into a device handler's buffer or directly to the device. The parameters defining the transfer conditions are contained in a Data Transfer Control Block. Execution of the task's instruction stream continues upon completion of querying of the request.
SYS I/O SWRITE	Initiates a synchronous transfer of a record or block from a user buffer into a device handler's buffer or directly to the device. The parameters defining the transfer conditions are contained in a Data Transfer Control Block. Execution of the task's instruction stream is blocked until completion of the transfer is signaled by the device driver.

SUSPEND TILL ANY	Suspend the execution of the invoking task's instruction stream until the completion of any prior asynchronous I/O, time out request or armed event posting.
SUSPEND TILL ALL	Suspends the execution of the invoking task's instruction stream until the completion of all prior asynchronous I/O time out requests and armed event postings.
SUSPEND SYNCH	Returns with the current number of outstanding asynchronous event postings, I/O requests of time out requests.
SYS I/O OPNEVELK	Associates an Event Block in user task space with a specific event which has been defined to the system by a device driver.
SYS I/O CLSEVBLK	Disassociates an Event Block with any event.
SYS I/O	If a user task is suspended, posting of specified event will resume the task.
TMER OPEN	Allocates a user timer.
TIMER CLOSE	Deallocates a user time.
TIMER SET	Initializes a timer but does not start it.
TIMER START	Allows either blocked or unblocked task execution pending expiration of a specified interval.
TIMER STOP	Freezes the current state of the timer.
TIMER STATUS	An inquiry concerning the current timer status.

1.2 APPLICATION PROGRAM I/O INTERFACE

The interface consists of two imperative macros: SYSIO and SUSPEND, and three system control blocks: DTCB, DIB, and FPKT.

- SYSIO

  This macro issues commands through the I/O manager by use of a TRAP #6 instruction. I/O commands include:

  OPEN - establish logical unit
  INIT - configure device
  CLOSE - release logical unit
  FUNCTION - set device option
  CANCEL - reset device driver

  SREAD - synchronous read
  SWRITE - synchronous write
  AREAD - asynchronous read
  AWRITE - asynchronous write
  BREAD - byte read
  BWRITE - byte write
  BTEST - byte test

- SUSPEND

  This macro invokes I/O manager WAIT functions by use of a TRAP #9 instruction. Functions include:

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

SYSIO makes use of the following control blocks. The one that you use depends on the I/O command.

- DIB

  This block is somewhat device dependent, therefore it is given a different name for each device category. For communications it is called a Port Initialization Block (PIB). For disk it is called a Disk Initialization Block (DIB). For graphics it is called a Graphics Initialization Block (GIB).

  The information in the DIB is used by the I/O manager at OPEN time. First it is checked for validity, then it is copied from user space into the appropriate control block in system space.

- DTCB

  The Data Transfer Control Block (DTCB) holds I/O status and buffer information during reads and writes.

- FPKT

  The Function Packet control block provides for device specific operations not necessarily involving data transfer. This would include things like setting the baud rate, reading the cursor position, repositioning a file pointer, etc. It is required for the FUNCTION command and optional for the OPEN command.

See the macro and control block sections for details on structure and usage.

1.2.1 EXCEPTION HANDLING

Application programs receive a status code in register D7 to inform them of exceptional conditions that occurred during I/O operations that they had initiated via SYSIO.

If D7 is zero then the operation concluded without exception; otherwise D7.L contains an exception code which can be interrogated by the caller.

Register bytes are numbered 3, 2, 1, and 0.

Byte 0: contains exception codes from the device driver. Values from $01 through $0F have common meanings across all device drivers. See "COMMON EXCEPTION CODES" below. Values from $10 through $7F can be assigned as needed by the device driver developer for device specific exceptions. These are indexed in "DEVICE EXCEPTION CODES" below.

Byte 1: If the sign bit is on, then this byte contains an exception code from the I/O manager. See "I/O Manager Codes" below. If the sign bit is off then this byte contains a PDB reference number which can be used to access the PDB involved in the exception for constructing error messages, etc.

Bytes 2 and 3 are reserved.

1.2.2 BYTE I/O FACILITIES

An efficient mechanism for transferring data on a byte basis has been provided. Upon opening a device for byte I/O (DIBBIO=-1) and a specific

direction (DIBTO - 0 for write, 1 for read), 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 and is returned with a byte on test or reads. Register D7.B is returned with a status code. (See codes for asynchronous request). No scheduling is done for these requests and if the device is busy with another request, the code returned in D7 so indicates. This facility, then, is best used for application programs using nonshareable devices (the first task to open a nonshareable 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 at open time.

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

1.2.3 EXCEPTION CODES

COMMON EXCEPTION CODES

$0001 reserved
$0002 reserved
$0003 reserved
$0004 reserved
$0005 READ or WRITE error occurred.
$0006 reserved
$0007 reserved
$0008 End Of File
$0009 Input record larger than buffer length, truncated.
$000A Device not ready.
$000B Byte I/O - Buffer Full on Write
$000C Byte I/O - Buffer Empty on Read
$000D not assigned.
$000E Invalid CODE in Function Packet.
$000F reserved

COMMAND EXCEPTION CODES

$0060 INIT
$0061 CANCEL
$0062 READ
$0063 WRITE

$0064 FUNCTION
$0065 WTBYTE
$0066 RDBYTE
$0067 CLOSE
$0068 TSTBYTE
$0069 DTACHDVR
$006A ATACHDEV
$006B DTACHDEV
$006C reserved
$006D reserved
$006E reserved
$006F reserved

I/O MANAGER EXCEPTION CODES

$8200 Invalid I/O Manager call.
$8300 Logical Unit Number not opened.
$8400 Inadequate system space for control blocks.
$8500 Duplicate Logical Unit Number.
$8600 Named Device Not Found.
$8700 Not Device Owner
$8800 Still Pending Opens Against Device.
$8900 Device Does Not Support Byte I/O
$8A00 Non-Null request Queue for Byte I/O
$8B00 Not Open For Byte I/O
$8C00 No Suspended "FOR" Task Found
$8D00 Named Event Not Found
$8E00 Event Not Opened

CODES FOR ASYNCHRONOUS REQUESTS

SYSIO will return a -1 in register D7 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 exception, and a positive number indicates completion with an exception code.

1.2.4 FILE NAME FORMAT

A fully qualified filename consists of four fields: A volume label, a catalog name, a filename, and an extension. When file names are specified within system commands you must use specific delimiters to separate the fields.

STANDARD FILENAME FORMAT FOR USE IN SYSTEM COMMANDS

<volume:><catalog.>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 and 4-7 for hard disk drives) opens the volume name that is mounted in the specified physical drive.

catalog (not supported in current operating system) is one to eight alphanumeric characters with the leading character alphabetic and always terminated with a period. This field can be omitted in which case the default catalog is used.

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:COLORS.BLUE.SRC          CLYDE:TESTCASE.BIN
VOL7:Z1.JERRYPGM.REL            DOREEN1.REL
```

WILDCARD FEATURE

This feature is the same as described earlier; however, it applies also to catalog names but not to volume names.

1.2.5 DEVICE NAMING

Physical device names are three to six characters in length. The first character is always "#". The next two to four characters are a device mnemonic. A two digit suffix is used to distinguish between multiple devices of the same kind in the system. The first digit is a board number where "0" means the planar board and the numbers "1" through "9" and the

letters "A" through "Z" refer to plug-in boards. (This leaves plenty of room for expansion.) The second digit allows selection of multiple devices on the same board.

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. Device names are bound to application program logical unit numbers with the SYSIO-OPEN command and a Device Initialization Block (DIB) supplied by an application program. Task logical unit assignments can be displayed with the SHOW command.

ASSIGNED DEVICE NAMES

Floppy Disk	#FD00 through #FD03
Hard Disk	#HD00 through #HD03
IEEE-488 Bus	#BUS00 (additional devices would be #BUS01, etc.)
Keyboard/Keypad	#CON
Keypad/control	#KPD
Null Device	#NULL (not implemented)
Printer	#PR
RS-232 Asynch	#SER00, #SER01
RS-232 Bisynch/Asynch	#SER02
CRT (see note)	#SCRN0 for page 0 #SCRN1 for page 1 #CNSL0 for page 0. There is no page 1. note: CRT has ability to display one of the two memory pages at any given time. The page to be displayed can be set by function call.

SENSOR I/O PLUG IN BOARD (Board 1)

Analog/Dig Conv	#ADC10 through #ADC03
Clock/Timer Ch	#CTC10 thorugh #CTC01
Switches	#SWI10 through #SWI07
LEDs	#LED10 through #LED07
Parallel ports	#PPA10 through #PPA11 #PPB10 through #PPB11

1.3 APPLICATION MACROS

1.3.1 SYSIO

GENERAL INFORMATION

Application programs will use SYSIO to perform all Logical I/O operations. SYSIO allows you to issue one of twelve commands as detailed in the examples below. There are two mechanisms used by SYSIO. One is a fast path for Byte operations (BREAD, BWRITE, and BTEST), the other is a block oriented path for record I/O, Open, Close, and Function calls.

1. ALL COMMANDS EXCEPT BYTE I/O

   SYSIO 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 long PC relative branch to your exception handling routine. If the command is unsuccessful then the branch is taken and register D7.W will contain an error code. If the call is successful then the branch is not taken and control will continue with the next sequential instruction following the macro.

   CODING

   The commands have a common form for coding. First is the COMMAND, next is the Logical Unit Number, next is the name of a control block, and finally is the address of an exception handling routine. You may specify the Logical Unit Number either in immediate notation using a # sign, i.e. #1 or #INUNIT, or as the label of a byte containing the value, i.e. LOGUNIT or D3. If register D5 already contains the Logical Unit Number then code D5 and no load instruction will be generated.

   SYNCHRONOUS/ASYNCHRONOUS OPERATIONS

   The AREAD and AWRITE commands are asynchronous operations while the other commands are synchronous.

   Synchronous operations execute in the invoking program's instruction stream. This means that you will be blocked from continued execution until the operation is complete. When you get control back from SYSIO register D7 will contain the completion status.

   Asynchronous operations such as Direct Memory Access are initiated in the invoking program's instruction stream with an interrupt being

used to signal completion of data transfer. The invoking program gets control back immediately with a -1 in register D7 to indicate that the operation has started successfully. Any other status in register D7 means that an exception had occurred and that the operation had never gotten started.

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. Otherwise it can use the SUSPEND macro to wait for completion.

SYSIO COMMANDS WITH EXAMPLE CODE

a. SYSIO OPEN,#1,INPIB,ERRORLAB

   INPIB is the "Port Initialization Block" for RS232 input which configures the port to user specified characteristics and enables the device as an interrupt source. This example OPENs the port as Logical Unit Number 1.

b. SYSIO CLOSE,#1,0,ERRORLAB

   This example shows the CLOSE for Logical Unit Number 1. It disables interrupts from the device and resets buffer pointers. Note that no control block need be specified for CLOSE.

c. SYSIO SREAD,#1,INDTCB,ERRORLAB

   This is a synchronous READ from Logical Unit Number 1. The buffer address and termination characters are passed using Data Transfer Control Block INDTCB.

d. SYSIO SWRITE,#2,OUTDTCB,ERRORLAB

   Synchronous WRITE to Logical Unit Number 2. This request uses the Data Transfer Control Block OUTDTCB.

e. SYSIO AREAD,#INUNIT,INDTCB,ERRORLAB

   Asynchronous READ from the Logical Unit Number equated to INUNIT.

f. SYSIO AWRITE,OUTLUN,OUTDTCB,ERRORLAB

Asynchronous WRITE to Logical Unit Number contained in the byte labeled OUTLUN.

g. SYSIO FUNCTION,#1,FPKT1,ERRORLAB

This is a FUNCTION command which sends the Function Packet FPKT1 to the device driver associated with Logical Unit Number 1.

h. SYSIO INIT,#1,SCRNGIB,ERRORLAB

SCRNGIB is a "Graphics Initialization Block" which is used to configure the CRT window to user specified characteristics.

i. SYSIO CANCEL,#1,0,ERRORLAB

The CANCEL command causes the device driver to disable interrupts, reset device buffer variables, and reset current default conditions for Logical Unit Number 1. Note that no control block specification is necessary for CANCEL.

2. 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 SYSIO and entry to the driver.

Unlike the block oriented requests, the Byte I/O requests will 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 the byte. Register A6 is loaded with the system identifier obtained from DIBBIO. Register D7.W contains the completion status or return.

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.

BYTE I/O COMMANDS WITH EXAMPLE CODE

a. SYSIO BREAD,BYTEID
-- or --
SYSIO BREAD,A6

This is a Byte I/O Read. The macro will load register A6 with the system identifier obtained from DIBBIO at open time. You should code A6 as the system identifier if it already contains the correct value.

b. SYSIO BWRITE,ID
-- or --
SYSIO BWRITE,A6

This is a Byte I/O Write.

c. SYSIO BTEST,IDFIELD
-- or --
SYSIO BTEST,A6

This is a Test Byte call. It will return a byte from the input buffer in register D0.B without moving the buffer pointer. If there is no byte to read then it will set register D7.B to]0C, Buffer Empty on Read.

SYSIO is contained in the IOMGMAC2.SA macro include file on sign-on 2000. Specify INCLUDE SYS:2000..IOMGMAC2.SA.

1.3.2 SUSPEND

Application programs will use SUSPEND to wait for the completion of outstanding I/O and timer requests. The application may elect to wait for the completion of one request or all requests.

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. Application programs that are interested in the completion of one specific request out of several outstanding requests must check the DTCSTA field of the Data Transfer Control Block.

If the application program uses decision logic based on the DTCSTA fields then there is a possibility that the status will change between the time that the application 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. See the examples below.

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 COMMANDS WITH EXAMPLE CODE

1. SUSPEND SYNCH,D0

   The number of outstanding requests for the current task is returned in register D0. The macro always returns the count in register D0, however it is made a required operand for documentation purposes.

2. SUSPEND TILLALL

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

3. SUSPEND TILLANY,D0

   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 must contain the number of requests that were outstanding before it 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.

4. SUSPEND TILLANY,NOSYNCH

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

EXAMPLE CODE USING SUSPEND-SYNCH

We will assume that there are three outstanding requests which use Data Transfer Control Blocks AAADTCB, BBBDTCB, and CCCDTCB. The program wishes to wait until AAADTCB is complete, the other requests don't matter. We know that the fields AAASTA, BBBSTA, and CCCSTA are all set to -1 indicating request not yet complete.

```
WAITAAA  SUSPEND  SYNCH,D0            Get current number of requests.
         IF AAASTA <EQ> -1 THEN      If the one you want is not done
```

```
SUSPEND  TILLANY,DO          Then wait for a change in status
  GOTO WAITAAA               and recheck if it is done
ENDIF
```

1.3.3 ASYNCHRONOUS EVENT POSTING

The Asynchronous Event Posting facility provides a mechanism for user tasks to detect the occurrence of specific events. By associating a data structure in user space with a predefined event the user can either poll the data structure to determine when the event occurs, or suspend himself until the event occurs. Polling the data structure is analogous to monitoring a Data Transfer Control Block (DTCB) for completion of an asynchronous I/O request. Monitoring for an event consists of performing a Test-and-Set (TAS) instruction on a specific byte in the data structure. The TAS sets one of three condition codes. This indicates whether or not the event occurred, occurred more than once, or did not occur at all. Because it is an indivisible instruction, the TAS also prevents the operating system from modifying the byte while the user examines it.

In addition to the event indicator byte, the event data structure also contains a 32-bit longword which may be used to pass status from the driver to the user. Its meaning is completely device-dependent. This field could be used to transfer things like A/D status registers, or pointers to blocks containing additional data about the event.

In addition to polling, the user can suspend his task until the event occurs. This is analogous to suspending a task until an asynchronous I/O request completes. Both cases make use of operating system services to resume a task when the I/O request count for the task falls below a certain value. Interrupt handlers within the operating system are responsible for decrementing the I/O request count when events occur or I/O operations complete.

The interface to Asynchronous Event Posting consists of three SYSIO functions. For a detailed description of each function, the following section illustrates SYSIO event functions with specific examples.

a. SYSIO OPNEVBLK,0,EVENTBLK,ERRORLAB

   The OPNEVBLK function associates an Event Block in user space with a specific event which has been defined to the system by a device driver. The association is by event name, which is four characters long. The format of the Event Block is as follows:

   EVBSEM DS.B 1 TAS byte

EVBRS0 DS.B 1 Reserved

EVBSTA DS.B 4 Status Word

EVBNAM DS.B 4 ASCII name of the event

The user should use the TAS (Test and Set) instruction to interrogate EVBSEM. TAS sets the condition code in the following manner:

MI - No event has occurred since the last TAS

EQ - One event has occurred since the last TAS

PL - More than one event has occurred since the last TAS

When more than one event has occurred since the last TAS the user can determine the event count from the low-order seven bits of EVBSEM (TAS sets the most significant bit).

A sequence of operations might look like this:

```
        LEA         EVBLK,A0        ;point A0 to event block
        TAS         EVSEM(A0)       ;TAS the semaphore
        BMI         EXIT            ;nothing happened
        BPL         OVRRUN          ;more than one happened
ONEEVT  EVTMOVE.L   EVBSTA(A0),D0   ;just one happened
        BRA         EXIT            ;return
OVRRUN  MOVE.B      EVSEM(A0),D0    ;pick up event count
        AND.B       #]7F,D0         ;turn off MSB
        MOVE.B      D0,OVRCNT       ;save overrun count
        BRA         EXIT
        .
        .
        .
```

b. SYSIO CLSEVBLK,0,EVENTBLK,ERRORLAB

The CLSEVBLK function removes the association between the Event Block and the event. No further references to this Event Block are valid.

c. SYSIO ARMEVENT,0,EVENTBLK,ERRORLAB

The ARMEVENT function specifies that the user task is to be resumed from the suspended state when the event occurs. This allows a task to later suspend itself until the event occurs via

the SUSPEND macro. ARMEVENT increments the I/O request count the same as an asynchronous I/O call does. When the event occurs the request count is decremented by one. If the new count is less than the trigger count and the task is suspended it is placed back on the ready queue.

1.3.4 TIMER

GENERAL INFORMATION

Application programs use the TIMER macro to perform logical timer operations. Timers are resources which must be allocated by an OPEN function and deallocated by a CLOSE function. Examples of these and other functions are detailed in subsequent paragraphs.

Once opened, the timeout period and mode of use is specified by the SET function. The timeout period must be given as a multiple of tenths of a second. The timer is activated by the START function and deactivated by the STOP function. The current state and value of the timer is obtained from the STATUS function.

When setting the timeout period, the user has a choice of one of three modes in which to use the timer. Polled mode is used in situations where he periodically wants to query the timer to determine elapsed time, without blocking his task. Blocked mode suspends his task until the timer expires. Asynchronous mode treats timer expiration as an ordinary I/O request awaiting completion. It is useful when used in conjunction with the SUSPEND macro.

TIMER generates in-line code to carry out its functions. It loads register A6 with the address of a control block, then it loads register D5.B with a logical timer number. Next it issues a TRAP #10 instruction followed by the function word. This is followed by a long PC relative branch to the exception handling routine. If the function is unsuccessful the exception branch is taken and register D7.W contains an error code. If the call is successful the exception branch is not taken and control continues with the next sequential instruction following the macro.

CODING

Timer functions have a common coding form. First is the function word, next is the logical timer number (obtained from the OPEN command), next is the name of a control block, and finally is the address of an exception handling routine. Logical timer numbers are specified as either immediate values or as the address of a byte of storage containing a value.

TIMER FUNCTIONS WITH EXAMPLE CODE

a. TIMER OPEN,TMRID,ERRORLAB

This example allocates a timer and sets its state to OPENed. The logical timer number is returned in the least significant byte of TMRID. No control block is required.

b. TIMER CLOSE,TMRID,,ERRORLAB

This example frees the timer for use by other tasks. No further references can be made to this logical timer number. No control block is required.

c. TIMER SET,TMRID,SETBLK,ERRORLAB

This example sets the state of the timer to INIT, and sets the MODE attribute and initial countdown from the values specified in the control block labelled SETBLK. This command does not activate the timer.

d. TIMER START,TMRID,,ERRORLAB

This example sets the state of the timer to ACTIVE, and starts the countdown. If the mode is BLOCKed, the calling task is suspended until the timer expires, regardless of any pending I/O requests. If the mode is ASYNCHRONOUS, no control block is required.

e. TIMER STOP,TMRID,,ERRORLAB

This example sets the state of the timer to STOPPED, and freezes its current value. No control block is required.

f. TIMER STATUS,TMRID,STATBLK,ERRORLAB

This example fetches a snapshot of the timer's current value and state, and returns them in the control block labelled STATBLK.

The format of the SET Timer Conrol Block is as follows:

MODE	DS.W	1	Mode of Use
VALUE	DS.L	1	Interval

Mode of use is specified as polled, blocked or asynchronous via the equates MODPOL, MODBLK or MODASY, respectively.

Value is a 32-bit value specifying the number of tenths of seconds the timer is to run.

The format of the STATUS Timer Control Block is as follows:

```
STATUS      DS.W        1       Timer Status
VALUE       DS.L        1       Timer Value
```

Status is a 16-bit work indication the current state of the timer. Its values are as follows:

```
NOTSET          Timer has not been initialized
OPENED          Timer has been opened
INITED          Timer has been initialized
ACTIVE          Timer is currently counting down to zero
STOPPED         Timer has been stopped by the user
EXPIRED         Timer has counted down to zero
```

Value is a 32-bit word indicating the number of tenths of seconds the timer has left before expiring.

1.4 APPLICATION CONTROL BLOCKS

DIB

"DIB" refers to a form of control block to be used by an application program at OPEN time which can specify a non-default mode of operation for the device.

Each device driver has a version of "DIB" in a device support macro library. There is unique information that the device driver needs to know at OPEN time, so the customized DIB meets these needs. The macro library also provides instructions and examples to go along with the customized DIB.

```
SYSIO  OPEN, #5,INPUTDIB,OPENERR
```

DIB FORMAT

```
DIB     DS.W   0
DIBVOL  DS.B   6    (U) Volume or device name
DIBDTD  DS.B   1    (U) Device transmit direction (see below)
DIBFOR  DS.B   1    (U) 'FOR' Task number
DIBOPT  DS.W   1    (U) Device Configuration Options
```

```
DIBFCN  DS.L   1    (U) Configuration Function Packet Pointer
DIBBIO  DS.L   1    (S) For BYTE I/O into manager
DIBLEN  EQU    *-DIB   Current length of DIB = 18
---------- DIB EXTENSION AREA -----------------------------
DIBxxx  EQU    0+DIBLEN   Device specific things may be defined by the
                          device support macro library.
```

Note: (U) means application sets this field.
(S) means I/O manager sets this field.

EXPLANATION OF DIB FIELDS

- DIB EXTENSION

  The DIB extension area is designed to allow device dependent information to be passed from the application to the device driver.

- DIBOPT - Device Configuration Options

  This defines to the driver at open time the kind of operation that we expect to be doing. Bits 0-7 are reserved for common definition across all device drivers.

- DIBFCN - Configuration Function Packet Pointer

  Optional address of list of function/data words to set up port at OPEN time.

- DIBDTD - Data Transfer Direction

  0 = Out (Write)
  1 = In (Read)
  2 = Both (Read/Write)

EQUATES

DTCB

Application programs will use equates provided in a macro support library.

The Data Transfer Control Block is a required operand of the SYSIO macro for READs or WRITEs. The application program uses it to supply

information for each data transfer request, and to monitor status after the request has been issued.

DTCB FORMAT

```
DTCSTA  DS.L  1  (S) Completion or current status
DTCTBU  DS.B  1  (U) Buffer termination byte range - upper
DTCTBL  DS.B  1  (U) Buffer termination byte range - lower
DTCBFS  DS.L  1  (U) Buffer start address
DTCBFL  DS.W  1  (U) Buffer length
DTCPTR  DS.L  1  (S) Buffer Pointer
```

Note: (U) means Application sets this field.
(S) means I/O manager sets this field.

EQUATES

Code "INCLUDE IOMGR.MAK" for equates to the above labels.

FPKT

The Function Packet is a list of COMMAND-DATA word pairs that an application program uses to configure a device to something other than its default mode. The SYSIO-FUNCTION macro uses a Function Packet as a required operand. The SYSIO-OPEN macro allows optional use of Function Packets to configure the device to desired characteristics.

Typically the format of a Function Packet is a word representing the COMMAND followed by a word that represents the DATA for that command. Several pairs of COMMAND-DATA make up the Function Packet, with a terminating COMMAND of zero indicating end-of-list. Appendix A contains a summary of the functions and function codes for all standard drivers.

SAMPLE FUNCTION PACKET

The following is an actual Function Packet from the RS-232 driver program. It is used to set the Baud rate to 9600 and to set character framing. Note that the driver developer has provided equates for each COMMAND and DATA parameter.

```
    DC.W  SETBAUD
    DC.W  B9600
    DC.W  SETFRM
    DC.W  DATA8+PARNONE+STOP2
    DC.W  ENDLIST
```

In the above example, SETBAUD, SETFRM, and ENDLIST are commands while B9600, DATA8, PARNONE, and STOP2 are data. The equates become binary numbers or bit coded words that the driver can interpret.

2.0 KEYBOARD, KEYPAD, AND SOFTKEY DRIVER

2.1 INTRODUCTION

The IBM Instruments Computer System can be configured with keyboard, keypad, and soft keys, or with just keypad and soft keys. Application programs looking for console input need not be aware of the input device; a logical unit reference is the only thing necessary.

Two Physical Device Blocks are available for console access: #CON for keyboard and console-oriented keypad entry, and #KPD for device oriented keypad and softkey entry.

2.2 FEATURES

1. The logical unit number for CRT character echoing can be specified by the caller. This allows for multiple windowing.

2. Optional prompt string for input lines.

3. Read Byte or Read Line operation.

4. I/O and proceed for Read Line.

5. Editing options may be specified by the caller.

6. Keyboard control-function keys

Ctrl-Break	Sets globally accessable exception byte. Normally used to break out of a programmed loop.
Ctrl-NumLock	Sets globally accessible exception byte, reset by any other key. Normally used to start/stop programmed operations.
Shift-PrtSc	Copy CRT screen to printer.
Ctrl-Alt-Del	System warmstart.

7. All keystrokes (except above control-function) are available to the caller via Byte I/O. Each key has a unique scancode.

8. Optional table lookup of control and function keys in line input.

   - ASCII strings can be generated from keystrokes.
   - Task Wakeup or Suspend from keystroke.

9. Caller can specify whether non-ASCII control and function keys should be ignored in line input or returned as exceptions.

10. Optional command parsing of #CON line input.

2.3 METHOD OF OPERATION

The keyboard, keypad, and soft keys share a common interrupt. Upon receiving an interrupt, the interrupt handler reads one of two parallel ports to retrieve the scancode of the key.

2.4 KEYBOARD SCANCODE DEFINITIONS

Figure 2-1 shows the key positions of the keyboard. The corresponding scan codes are listed in Table 2-1.

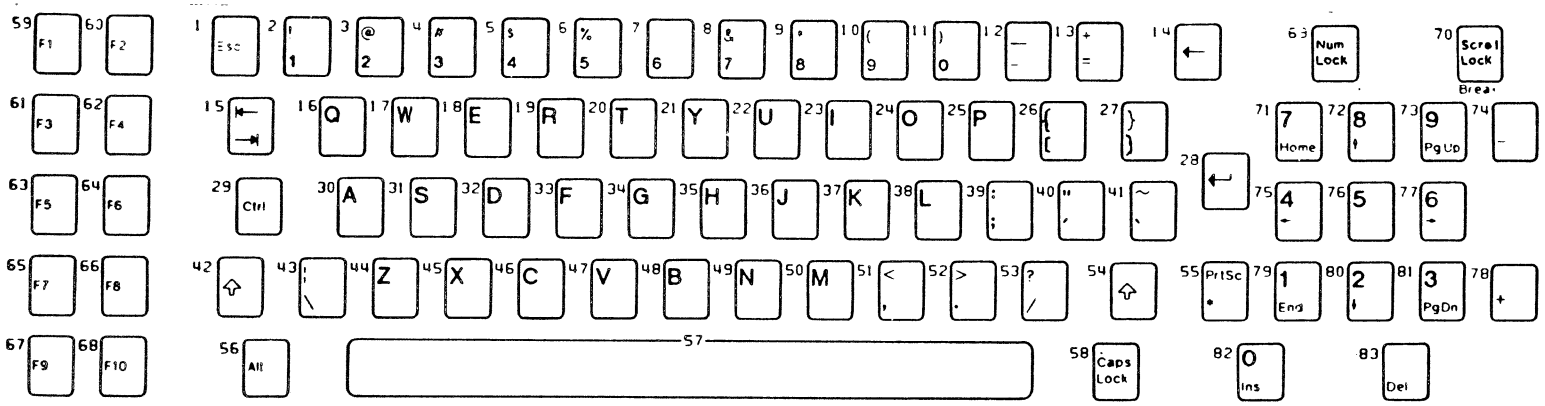


Figure 2-1. Keyboard. The number in the upper left-hand corner of each key is the key position (see Table 2-1).

Table 2-1. Keyboard Scan Codes

Key Position	Scan Code in Hex	Key Position	Scan Code Hex
1	01	43	2B
2	02	44	2C
3	03	45	2D
4	04	46	2E
5	05	47	2F
6	06	48	30
7	07	49	31
8	08	50	32
9	09	51	33
10	0A	52	34
11	0B	53	35
12	0C	54	36
13	0D	55	37
14	0E	56	38
15	0F	57	39
16	10	58	3A
17	11	59	3B
18	12	60	3C
19	13	61	3D
20	14	62	3E
21	15	63	3F
22	16	64	40
23	17	65	41
24	18	66	42
25	19	67	43
26	1A	68	44
27	1B	69	45
28	1C	70	46
29	1D	71	47
30	1E	72	48
31	1F	73	49
32	20	74	4A
33	21	75	4B
34	22	76	4C
35	23	77	4D
36	24	78	4E
37	25	79	4F
38	26	80	50
39	27	81	51
40	28	82	52
41	29	83	53
42	2A		

2.4.1 KEYBOARD ENCODING AND USAGE

2.4.1.1 Encoding

The keyboard software routine converts the keyboard scan codes into a form of ASCII called "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 by the keyboard software routine or through interrupts.

2.4.1.2 Character Codes

The character codes in Table 2-2 are passed through the keyboard routine to the system or application program. A "-1" means the combination is suppressed in the keyboard routine. The codes are returned in AL. (See Table 2-1 for scan codes).

Table 2-2. Character Codes

KEY #	BASE CASE	UPPER CASE	CTRL	ALT
1	ESC	ESC	ESC	-1
2	1	1/2	-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
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
29CTRL	-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
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 1)	NUL (Note 1)	NUL (Note 1)	NUL (Note 1)
69NUM LOCK	-1	-1	Pause (Note 2)	-1
70SCROLL LOCK	-1	-1	Break (Note 2)	-1

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

Note 1: Refer to Extended Codes(2.4.2).
Note 2: Refer to Special Handling(2.4.3).

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

2.4.2 EXTENDED CODES

2.4.2.1 Extended Functions

For certain functions that cannot be represented in the standard ASCII code, an extended code is used. A character code of 000 (NUL) is returned in the low-order byte of register D0. 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. This code is returned in the second byte (bits 15-8) of register D0.

Table 2-3. Keyboard Extended Functions

SECOND CODE	FUNCTION
3	NUL Character
15	←
16-25	ALT Q, W, E, R, T, Y, U, I, O, P
30-38	ALT A, S, D, F, G, H, J, K, L
44-50	ALT Z, 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 (Start/Stop Echo to Printer) 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)
132	CTRL PG UP TOP 25 Lines of Text & Home Cursor

2.4.2.2 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 calling an entry point in the keyboard routine. 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 2.4.3.1. Used with SCROLL LOCK to cause "break" function described in Section 2.4.3.2. Used with NUMLOCK to cause "pause" function described in Section 2.4.3.3.

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 2.4.3.1.

ALT has a special use to allow the user to enter any characer code (0-255) into the system from the keyboard. The user holds down the ALT key and types the decimal value of characters using the numeric keyboard (keys 71-73, 75-77, 79-82). The ALT key is then released. If more than three digits are typed, a modulo 256 result is created. These three keys are interpreted as a character code (000-255) and are transmitted through the keyboard routine to the system or application program. ALT is handled by the to keyboard routine.

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

NUM LOCK - Shifts keys 71-73, 75-77, 79-83 to numeric state. A second depression of NUM LOCK reverses the action. NUM LOCK IS handled by the 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.

2.4.2.3 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.

2.4.3 SPECIAL HANDLING

2.4.3.1 System Reset

The combination of ALT CTRL DEL (Key 83) will result in the keyboard routine initiating the equivalent of a system reset/reboot (handled by the keyboard routine).

2.4.3.2 Typematic Action Syppressed

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

2.4.3.3 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, with unrecognizable characters printing as blanks.

The keyboard routine does its own buffering. The buffer is big enough to support a fast typist. If a key is entered when the buffer is full, the key will be ignored.

2.4.4 KEYBOARD USAGE

This section outlines a set of guidelines for key use when performing commonly used functions.

Table 2-4. Keyboard -- Commonly Used 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, command entry
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
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	Extra function keys if

	CTRL F1-F10 ALT F1-F10	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

2.4.5 LOOKUP TABLES

The scancode is used as an argument in lookup tables that may be set by the user. These tables can perform translations on the scancode:

- Separation of keypad #KPD device commands from #CON console input.
- Scancode to ASCII character conversion.
- Scancode to ASCII string conversion.
- Scancode to immediate program function (task WAKEUP or SUSPEND)

These tables are initially set to empty. Application programs may issue a function call to set or clear a lookup entry.

The general format of the following three tables is the same. Each table starts with a long-word displacement to the next free table entry. This would initially be zero. An entry consists of the scan code, followed by an operation code, followed either by a character string or a one-byte task number. A character string can be one to N characters followed by a carriage return ($0D) or EOF ($04).

Keyboard Table

This works with scan codes from the keyboard function keys F1-F10. Possible operation codes include WAKEUP and DELAY to the task number that follows, or insertion of an ASCII string into the #CON buffer.

```
Operation Code 01 = WAKEUP
               02 = SUSPEND
               03 = ASCII string insertion into the #CON buffer.
```

Keypad-Shifted Table

This works with scan codes from the keypad in situations where the SHIFT key is already being pressed. Three operation codes determine whether to send a character string to CONRBF or to KPDRBF or to let the destination of the previous keystroke determine which buffer should receive the character string.

```
Operation Code 00 = Use destination of previous keystroke
               01 = ASCII string insertion into #KPD buffer.
               02 = ASCII string insertion into #CON buffer.
```

Keypad-unshifted Table

This is the same as KPTBL1 except that the SHIFT key will not already have been pressed.

EXAMPLES

KBTBL code to assign the string "DIR" with carriage return to function key F10 and to WAKEUP task number 3 with function key F1.

```
DC.L    9           Displacement to next free entry
DC.B    $44         Scan code of function key F10
DC.B    $03         ASCII string insertion into #CON buffer
DC.B    'DIR'       ASCII string 'DIR'
DC.B    $0D         Carriage Return
DC.B    $3B         Scan code of function key F1
DC.B    $01         WAKEUP operation
DC.B    3           Task number to wake up
..........
..........
..........
```

#CON BYTE I/O

Each SYSIO-BREAD will retrieve a keystroke from the input ring buffer. Register D0.L will contain the keystroke data as follows:

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
- Bit 3: Suspend key has been toggled

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.

#CON LINE I/O

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 "#CON FUNCTION SUMMARY".

There are lookup tables that may be used to translate non-ASCII keystrokes into program control functions or into ASCII strings. The use of these tables is optional. Note that the translation is global, meaning that all programs using console input must agree on the same translations.

Function and control 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 address 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.

2.4.6 #CON FUNCTIONS

2.4.6.1 Summary of Functions

The SYSIO-FUNCTION command can perform the following operations to #CON.

DESCRIPTION	CODE	DATA FOLLOWING
SET TRANSFER MODE	1	WORD
SET EDIT OPTIONS	2	WORD
PROMPT ON	3	NONE
PROMPT OFF	4	NONE
ECHO ON	5	NONE
ECHO OFF	6	NONE
RESET CTRL-BREAK	7	NONE
GET CTRL-BREAK	8	LONGWORD ADR RETURNED
GET CTRL-NUMLOCK	9	LONGWORD ADR RETURNED
SET ECHO	10	LONGWORD
SET TAB AMOUNT	11	WORD
SET PROMPT	12	LONGWORD POINTER
PARSE NEXT TOKEN	13	NONE
FUNKY MODE ON	14	NONE
FUNKY MODE OFF	15	NONE
PARSING ON	16	NONE
PARSING OFF	17	NONE

2.4.6.2 #CON Function Descriptions

The functions listed in this section are all callable by the FUNCTION command of the I/O manager using FUNCTION packets.

SET TRANSFER MODE — Allows transfer mode to be set to either fixed length transfer or variable length transfer. Fixed length uses the DTCLEN field of the Data Transfer Control Block as the amount to transfer. Variable length uses the DTCTBU and DTCTBL fields as delimeters on the amount of data to transfer.

Function code: 1 (SETTRANS)

Function data: Word. 0=Fixed, 1=Variable length.

SET EDIT OPTIONS	Sets bits into the Edit Control Word.
Function code:	2 (SETEDIT)
Function data:	Word. Bit 8: 0=Enable backspace 9: 0=Enable forward tab 11: 0=Swallow nulls not in lookup table
PROMPT ON	Enables prompt before read. You use the SET PROMPT function to establish a prompt string.
Function code:	3 (PRMPTON)
Function data:	None
PROMPT OFF	Disable prompt before read.
Function code:	4 (PRMPTOFF)
Function data:	None
ECHO ON	Enable echo to the CRT. The CRT window on which to echo may be either the default window or else a window specified by
Function code:	5 (ECHOON)
Function data:	None
ECHO OFF	Disable echo to the CRT.
Function code:	6 (ECHOOFF)
Function data:	None
RESET CTRL-BREAK	Sets the CTRL-BREAK byte to zero. This byte is set to $FF when the CTRL-BREAK keys are depressed. It may be used within a program to determine when to leave a loop or abort a function. You would normally reset the byte to zero before starting a loop to be broken by CTRL-BREAK.
Function code:	7 (RESCTLBK)

Function data:	None

GET CTRL-BREAK	Returns the address of the CTRL-BREAK byte which will be set to $FF whenever CTRL-BREAK is pressed.
Function code:	8 (GETCTLBK)
Function data:	Longword to receive the address of the CTRL-BREAK byte.

GET CTRL-NUMLOCK	Obtains the address of a byte which is set to $FF whenever CTRL-NUMLOCK is pressed and is reset to $00 whenever any other key is pressed.
Function code:	9 (GETCTLNM)
Function data:	Longword to receive the address of the CTRL-NUMLOCK byte.

SETECHO	Set CRT echo reference word. The user can open a can open a window for Byte I O and use it for keyboard echoing.
Function code:	10 (SETECHO)
Function data:	Longword. Either the word returned in DIBBIO following an open for Byte I O to to a window, or else a -1 to specify that the default screen should be used.

SET TAB AMOUNT	Sets a value to be used as the tab amount. This is the conceptual space between tab columns. The cursor is moved to the next tab column when the tab key is pressed.
Function code:	11 (SETTAB)
Function data:	Word to contain tab amount.

SET PROMPT	Sets a prompt string up to eight characters in length which will be sent to the echo device before reading. If fewer than eight characters are needed for a prompt then the bytes should be padded on the right with blanks.

Function code:	12 (SETPRMPT)
Function data:	Longword. Points to prompt string.

PARSE NEXT TOKEN	Causes the next token in the input line to be parsed. See COMMAND PARSING.
Function code:	13 (PARSE)
Function data:	None

FUNKY MODE ON	This establishes a mode of operation called "Function Key Mode" where keystrokes that result in an ASCII byte of zero will cause an exception code of $10 to be returned to the caller. The input buffer # will contain the scan code of the key thatwas pressed, followed by the bytes KBFLAG and KBFLAG1, which represent the state of the keyboard at the time the key was pressed. See #CON Byte I 0 for KBFLAG and KBFLAG1 definitions. This information is placed in the buffer offset DTCBFS from the start. The buffer will contain ASCII codes for any key pressed before the ASCII code zero was pressed.
Function code:	14 (FUNKYON)
Function data:	None

FUNKY MODE OFF	This turns off Function Key Mode.
Function code:	15 (FUNKYOFF)
Function data:	None

PARSING ON	Establishes a mode of operation whereby the input line on each read is sent to the parsing routine which parses the first token in the line and makes the rest of the line available for parsing via the PARSE NEXT TOKEN function.
Function code:	16 (PARSEON)
Function data:	None

PARSING OFF	Turns off the mode of operation established by PARSING ON.
Function code:	17 (PARSEOFF)
Function data:	None

2.4.7 DEFINITION OF CTRL-BREAK

Many programs need a way to "break" out of a loop at the console operator's discretion. The Ctrl-BREAK feature provides a global byte that may be tested within an application program. The byte can be reset to zero by the RESCTLBK function call. It will be set to -1 when Ctrl-BREAK is pressed. You issue the GETCTLBK function call to get the address of the byte. Please note that this is a global byte, accessable to all users of #CON.

2.4.8 DEFINITION OF CTRL-NUMLOCK

Many programs need a way to temporarily halt their operation and then later resume via operator command. One example is screen scrolling. The CTRL-NUMLOCK feature provides a global byte that may be tested within an application program. The byte is set to -1 whenever CTRL-NUMLOCK is pressed and reset to zero when any other key is later pressed. You issue the GETCTLNM function call to get the address of CTRL-NUMLOCK.

EXAMPLE

This example shows an OPEN of #CON. A function packet is used at OPEN TIME to set the eight-character prompt string and to enable prompting.

```
....
....
SYSIO  OPEN,#2,#CONSOLE,#ERRORLAB
....
....
....
```

```
CONSOLE   DS.W    0
          DC.B    '#CON  '    DIBVOL
          DC.B    1           DIBDTD = INPUT
          DC.B    1           DIBTRN = VARIABLE
          DC.L    0,0         DIBFOR
          DC.W    $0000       DIBOPT
          DC.L    CONFPKT     DIBFCN
          DC.L    0           DIBBIO


CONFPKT   DC.W        SETPRMPT
          DC.L        PROMPTER
          DC.W        PRMPTON
          DC.W        0


PROMPTER  DC.B        ENTER:
```

2.4.9 COMMAND PARSING

This feature breaks up a command into "tokens." A token is a substring of a line that is treated as a unit. The following definitions apply to command parsing.

NAME A name is a string of characters that begins with an alphabetic character and containsonly alphanumeric characters, (i.e. imbedded spaces).

NAME WITH WILDCARD CHARACTERS
A name that may include the special characters "*" and "?".

NUMBER A string of digits that may be decimal or hexadecimal. Hexadecimal numbers must start with a dollar sign. ($)

DELIMETER
One of the following special characters: period (.), comma (,), colon (:), semicolon (;), and the arithmetic symbols +,-, and .

CARRIAGE RETURN
The ASCII carriage return character. ($0D)

ERROR A token not falling into one of the above classes.

COMMAND PROCESSING PARAMETER BLOCK

Command parsing makes use of a parameter block to hold token processing variables. This block is available in the PCB. Its address may be obtained from the GSTAT system call.

```
CUCHAR   DS.L  1   Address of the next character
DESCRA   DS.L  1   Address of the first character of a token
DESCRC   DS.W  1   Length in bytes of the current token
CLASS    DS.B  1   Class of the current token
RC       DS.B  1   Return code of the current token
VALUE    DS.L  1   Binary value of a numeric token (unsigned)
```

2.4.10 TOKEN CLASSIFICATION

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 unless they are imbedded in a token. The count of the number of characters in a token is returned in DESCRC. RC and CLASS return the classification of the token as follows:

```
NAME             CLASS = 02     RC = 01
NAME (wildcard)  CLASS = 02     RC = 02
NUMBER           CLASS = 02     RC = 03
DELIMETER        CLASS = 04     RC = ASCII code of character
CARRIAGE RETURN  CLASS = 0D     RC = 0D
ERROR            CLASS = 00     RC = 00
```

CUCHAR is returned pointing one character beyond the end of the present token. If the token is a number (RC=03) then its binary value is returned in VALUE. Numbers in hexadecimal or decimal are automatically converted to binary form. Numbers that are too large are classified as ERROR.

EXAMPLE

Input line is "LOAD 1:MYFILE.EXT" carriage return

```
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
```

2.4.11 ERROR CODES RETURNED BY #CON

$09	Input record longer than buffer length
$0C	Buffer empty on byte I O BTEST or BREAD
$0E	Invalid code in function packet
$10	Keyboard function key exception
$2D	Command parsing not enabled
$61	Invalid operation CANCEL
$63	Invalid operation WRITE
$65	Invalid operation WRITE BYTE
$6A	Unimplemented entry - ATACHDEV
$6B	Unimplemented entry - DTACHDEV

2.4.12 KEYPAD AND SOFTKEY DESCRIPTION

The keypad is confirgured as a 3 row X 19 column tactile switch matrix and is located in front of the printer. The softkey assembly is a 1 X 10 matrix and is located directly beneath the CRT screen.*(1) Each key actuation produces a unique 1-byte scan code*(2) which can be read and decoded by the keypad driver routine.

The system user has the ability to define through a function call to the keypad driver key depression. The user established entry in a table (Figure 2-1) which can be decoded by the keypad driver to place a string of data in either the console buffer (#CNIN) or the keypad (#KPD). Date placed in the console buffer is treated just as though it were typed in through they keyboard. Data placed in the keypad buffer can be accessed via SYSIO calls to the keypad and used in an application as required.

To open the keypad, the user must create a DIB (See Figure 2-2) and within an initialization routine perform a SYSIO-OPEN to the device (see Figure 2-3). Afther this is performed, all standard SYSIO operations are allowed, with the exception of CANCEL, WRITE, and WRITE BYTE, to access the keypad and softkeys.

*(1) soft-key definition should be user displayed on the CRT above above the respective key position.

*(2) one duplication os scan code on the keypad. See list of keypad scan-codes.

2.4.13 KEYPAD FUNCTIONS

SYSIO-FUNCTION calls are defined in Figure 2-4. The following list gives a basic definition of each function:

SET TRANSFER MODE: See section 2.4.6.2 for definition

BEEPER ON: Turns system audible alarm on continuously

BEEPER OFF: Turns system audible alarm off

TIMED BEEP: System audible alarm sounds for a period defined in the time out word following the CODE word

BEEP SINGLE SHOT: System alarm sounds for approximately20 milli-20 milli-seconds

LED ON: Turns on LED number defined in the word following the CODE word. LED's are numbered as follows:

	LED NUMBER	KEYPAD
LEFT TOP	1 6	
	2 5	
	3 4	RIGHT BOTTOM

LED OFF: Turns off defined LED above

CLEAR LEDS: Turns off all LEDS

SET LEDS: Turns on all LEDS

ENABLE KEYBOARD: Allows keypad to interrupt and be read by operating by operating system.

DISABLE KEYPAD: Disables keypad interrupt to system.

ADD TO TABLE:
SCANCODE - See table of scan codes
TABLE TYPE - 0 = keypad shift table
1 = keypad table
2 = function key table (see 2.4.5)
3 = soft-key table
TASK CODE - 0 = put data in last used buffer
1 = put data in #KPD buffer

2 = put data in #CNIC

STRING LENGTH - length of message string, max 20 bytes

KEYCODE - user defined string to be presented in the defined buffer upon key actuation

EOT - string termination character

DELETE FROM TABLE: SCANCODE - as above
TABLETYPE - as above
TASK CODE - as above

2.4.14 ERROR CODES

Error return codes for the keypad driver are defined in Figure 2-5. Additional codes returned from table build delete functions are as follows:

HEX CODE	FUNCTION	ERROR
$11	BUILD	scan-code entry exists, must delete first
$12	BUILD	total table limits exceeded
$13	BUILD DELETE	error in table organization,
$14	DELETE	number not in table
$15	BUILD	invalid task or table code
$16	BUILD	invalid wake-up code (function-keys)

3.0 CRT DISPLAY DRIVER

3.1 PROGRAMMING SYSTEMS

The information contained in this section on the CRT display applies only to assembler language programming. It will be expanded at a future date to include higher level programming languages, specifically Basic, Fortran, and Pascal.

3.2 CHARACTERISTICS AND FEATURES

The CRT screen has a display capacity of 2240 characters in a format of 28 lines of 80 characters each. This format will normally consist of a user character-display area of 25 lines and a "console box" of 3 lines at the bottom of the screen. The latter displays system messages (e.g. prompts and error messages). The console box cannot be disabled, though it can be programmed by the user to display material other than system messages. The user implements such programming at his own risk, however.

The CRT display also functions as a graphics device, providing a high resolution of 760 by 480 pixels. All points on the screen are addressable. The character block is 9 x 16; the character size is 7 x 12.

Character generation is implemented by software mapping of standard fonts, but the display will also support user-defined fonts (see Figure 3-1). Each word in the CRT memory is 12 bits long. The image buffer uses the most significant four bits for attribute storage. Figure 3-2 shows a map of the CRT memory.

To Be Supplied

Figure 3-1. Standard Font

To Be Supplied

Figure 3-1. Standard Font

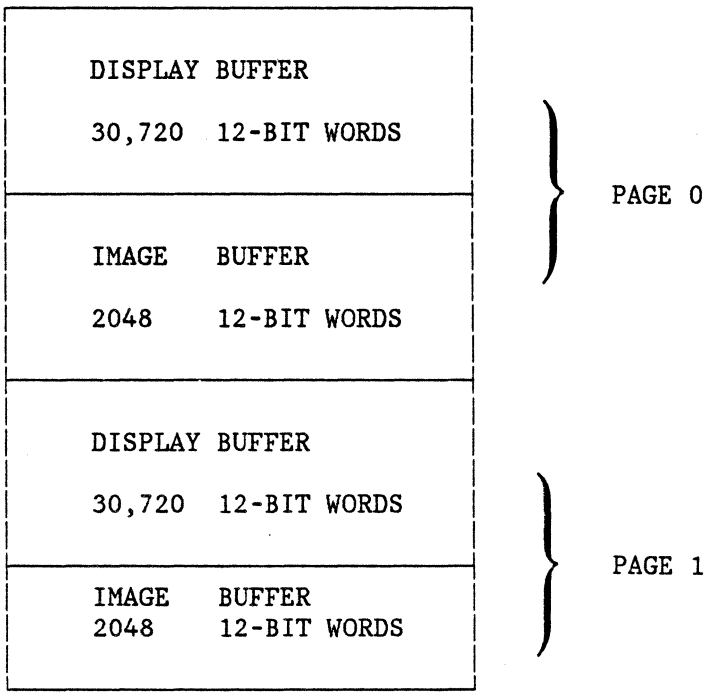


Figure 3-2. CRT Memory Map

The normal user area may be divided into up to five separate display areas, or "windows." These windows can be opened from within the user's program by using logical I/O facilities, each display area is then associated with a unique logical unit number. The default window position and dimensions are the entire 25x80 character user area. Subsequent Function calls allow redefinition of the window's position and dimensions without closing and reopening the logical unit.

Each window is opened with a one-character border around the perimeter. This border is not included in the dimension specified by the user, but must be taken into account in designing screen usage. No space management is in effect and overlapping windows are allowed.

Windows may be "framed" at the option of the user at any time. The pixels forming the frame are taken from those in the character border.

Information within each window may be updated in several modes. Default mode processing includes upward scrolling of the characters within the window when the current character position is incremented past the lower right-hand corner. The cursor symbol position is updated after each character write, and will be the underline character.

Carriage return ($0D) and line feed ($0H) character are used in formatting text when the control-character ($00-$19) filter is enabled. A carriage return places the cursor in column 0 of the current line, while a line feed places the cursor in the next line with the column unchanged. If the control character filter is disabled, all characters in the range $00-$FF are displayed and carriage return and line feed characters do not produce the above cursor movements.

If scrolling of the display is not desired, the window may be placed in the page mode. Receipt of carriage-return/line-feed pairs (in either order) increments the cursor to column 0 of the next line with subsequent clearing of this line except when the cursor initially resides on the last line. In this case, the cursor is placed in column 0 of line 0 and the line is cleared. The above actions also result if the cursor is incremented past the end of the current line.

In addition to high-performance transfers of the image buffer contents to the display buffer, users may specify dumps from specified buffers in application memory.

Since font generation is accomplished in software, the user may specify any of the available fonts as a cursor indicator. Thus different windows can use different cursor symbols.

Performance can be enhanced somewhat by forcing the cursor to be updated as infrequently as possible, or disabled entirely. Thus FUNCTION calls allow the user to choose the cursor update mode.

Attribute nibbles can be used to modify the display characteristics of individual characters; however, these modifiers must accompany the character when the display buffer is the target and attribute decoding has been activated by a prior FUNCTION call.

Hardware switching between two different display buffers is available, allowing a coordinating task control over the currently viewed page. Each page is considered to be a separate physical device for logical I/O use. Similarly, there are two separate complimentary image buffers, typically used as repositories of the ASCII image of the window contents. These areas are used for screen dumps to the printer, and a SYSIO SREAD will result in a transfer of a line from this bufffer to the user buffer specified in the DTCB. Optionally, keyboard echoing can take place to both the display and image buffers if fast dumping of the screen contents to a printer is anticipated; otherwise, a dump to the printer can be done in the graphics mode.

The default font table consists of a total of 256 characters. In addition, users may specify their own font table arranged in a linear 14-byte-per-character array.

3.2.1 SCRIB (SCREEN INITIALIZATION BLOCK)

Characteristics of the display area or window can be specified at open time. The following fields describe SCRIB usage.

```
SCRIVOL   DS.B   6   #CNSC0,#CNSC1
SCRIBDTD  DS.B   1   Bidirectional (=2)
SCRIBFOR  DS.L   1   "FOR" Task PCB
SCRIBOPT  DS.W   1   See mode word description below; uses default
                     if null.
SCRIBFCN  DS.L   1   Pointer to function packet; uses default if null.
SCRIBBIO  DS.L   1   Byte I/O field
```

3.2.2 MODE OR OPTION WORD DESCRIPTION

If the mode word field is nulled at open time, then default conditions are used. Bit assignments are as follows:

BIT NO. (DEFAULT)	OPTION/MODE
15 (0)	Mode word active 0 -- ignore mode word 1 -- mode word significant
14-13 (0)	Cursor update mode (applies to display buffer) 0 -- Single byte increment 1 -- Stationary cursor 2 -- Record/Block update
12 (0)	User font enable (must supply table pointer)
11 (0)	Scroll/Page mode
10 (0)	Attribute decode enable
9 (0)	System use (cleared by user)
8 (0)	System use (cleared by user)
7 (0)	Target buffer 0 -- Display 1 -- Image
6 (0)	Truncate enable flag
5 (0)	System use (cleared by user)
4 (0)	Auto line feed 0 -- no line feed generated on carriage return 1 -- line feed generated on carriage return
3 (0)	System use (cleared by user)
2-1(0)	Character write mode 0 -- overwrite existing data 1 -- exclusive OR with existing data 2 -- OR with existing data 3 -- inverted overwrite
0 (0)	System use (cleared by user)

Note that if the above conditions are to be modified after OPEN time, FUNCTION commands must be issued against the logical limit number specified in the OPEN command.

3.2.3 CHARACTER ATTRIBUTES

Special attributes that apply to the initial data transfer (record or block) may be selected using the FUNCTION command of SISIO. Display attributes of individual characters can be applied by activating attribute decode using a FUNCTION command, then sending full-words (2 bytes) to the display buffer. The LSB of the word contains the character code, while the low order four bits of the MSB contain the attributes that apply to the character. If attribute decoding is deselected, successive

bytes in the user buffer are interpreted as successive character codes for transfer to the display buffer. For transfers to the image buffer, activating attribute decode simply allows the attribute nibble to be stored along with the character code, then subsequent dumps to the display buffer can be made with the desired attribute decode state. In addition, attributes only can be modified if the image buffer is the selected target by using the ATTRIBUTE UPDATE FUNCTION.

3.2.3.1 Attribute Code

Since character fonts are software generated, transfers to the display buffer must be accompanied by the corresponding attribute byte. Only the least significant nibble is used.

Attribute Byte Bit &	Attribute
0	User font (USER FONT SELECT FUNCTION command must have been issued to set user font table address)
1	Underline
2	Inverse
3	Non - display
4	Not Used

3.2.4 CALL REQUIREMENTS OF THE CRT CHARACTER INTERFACE

Any user program involving the character CRT interface must contain system calls, if the program language is assembler; procedure calls, if the program language is Pascal; and subroutine calls if the program language is Basic or Fortran.

3.3 SUMMARY OF CRT FUNCTIONS

The functions listed in this section are all callable by the FUNCTION command of the I/O manager (SYSIO), using FUNCTION packets.

CODE	FUNCTION	DESCRIPTION
1	VARIABLE/FIXED TRANSFER SELECT:	activates terminal character checking to delimit a record that is being transferred. See a description of the Data Transfer Control Block (DCTB) and the mode word field in the SCRIB.
	Function data:	one word integer 0 -- fixed length 1 -- variable length
2	USER FONT SELECT/DESELECT:	permits specification of a user-defined table for a specific window, making it possible to display output in a foreign language or an APL character set. This font will be the one displayed in output until it is deselected. The address pointing to the user's font table must appear in the long word following the function code. This same table will be used if the user font bit is set in the attrubute nubble on a character-by-character basis.
	Function data:	long word pointer to user table; if null, font is deselected
3	FLOOD WINDOW:	modifies information displayed in an entire window (information displayed outside the window remains unaffected.) This command will not affect any frame around the window.

	Function data:	0 - clear window 1 - fill with current fill word 2 - exclusive OR fill word with previous contents 3 - OR fill word with previous contents
5	FLOOD LINE:	deletes displayed output from the cursor position to the end of the line on which the cursor appears.
	Function data:	0 - clear line 1 - fill with current fill word 2 - exclusive OR fill word with previous content 3 - OR fill word with previous contents
6	CLEAR PAGE:	is a "global" function (i.e., it affects the entire screen) and "wipes" the screen clean of all displayed output by changing the contents of the display buffer in the CRT memory (instead of disabling the screen). Affects all windows, including those of other tasks so must be used with caution.
	Function data:	0 - page 0 1 - page 1
13	READ CURSOR:	asks for the position of the cursor relative to the upper left-hand corner (X = 0, Y = 0) of window.
	Function data:	returns with long-word response-- most significant word: Y (0 ≤ Y ≤ line count -1) least significant word: X (0 ≤ X ≤ column count -1)

14	WRITE CURSOR:	XORS the cursor font with exisiting data at a position defined by the X-Y coordinates specified in the function (i.e., relative to the upper left-hand corner of window). Does not erase the cursor at the previous position. (See Function Code 40.)
	Function data:	long word MSW: Y (0 ≤ Y ≤ line count -1) LSW: X (0 ≤ X ≤ column count -1)
15	READ POINTER:	used in stationary cursor mode. X and Y coordinates are relative to the upper lefthand corner of the window; subsequent writes to the window increment this pointer.
	Function data:	long word MSW: Y(0 ≤ Y ≤ line count -1) LSW: X(0 ≤ X ≤ column count -1)
16	WRITE POINTER:	positions current pointer. Used in stationary mode. X and Y coordinates are relative to the upper left-hand corner of the window; subsequent writes to the window increment this pointer.
	Function data:	returns in long word MSW: Y (0 ≤ Y ≤ line count -1) LSW: X (0 ≤ X ≤ column count -1)
17	CURSOR FONT SELECT:	allows the user to select any symbol from the default font table or from a user defined table. This function allows the user to specify a unique symbol for each window.

	Function data:	one-word integer character code (see Figure 3-1)
18	DUMP IMAGE BUFFER:	dumps the contents of the image buffer (treated as a circular buffer of lines) of the CRT memory into the display buffer, which displays the dump on the screen. The dump begins with the first character of the current top line of the image buffer. The top line may be changed by using the SET TOP LINE function command.
	Function data:	none
19	SCROLL/PAGE MODE SELECT:	allows the user to control the screen display update mode. In scroll mode, either line feeds beyond the end of the window or attempts to write beyond the last character position result in an upward movement of the screen contents by one line with clearing of the last line and positioning of the cursor at column 0 of the next line and subsequent clearing of the line. This action also results if attempts to write beyond the end of the current line are made. The cursor is positioned at column 0, line 0 if attempts are made to write beyond the end of the last line.
	Function data:	one-word integer 0 -- scroll 1 -- page
21	ATTRIBUTE ENABLE/DISABLE:	can be used to enable or disable attributes characters displayed on the screen. The four attributes are user font,

		underline, display inversion, and blanking (i.e., not displaying the character). Attributes are decoded upon output to the display buffer (2 bytes per transfer) or are stored in the image buffer for decoding upon dump.
	Function data:	one-word integer 0 -- disable attribute decode 1 -- enable attribute decode
23	FRAME ENABLE/DISABLE:	can be specified for individual windows. This command results in exclusive OR of the frame pixels with the previous data. Successive function calls will erase the frame pixels.
	Function data:	none
24	TARGET BUFFER SELECT:	sends display information to one of two buffers in the CRT memory -- the display buffer or the image buffer. Also identifies source buffer for SREAD operations.
	Function data:	one-word integer 0 -- display buffer 1 -- image buffer
25	SET TOP LINE:	can be used to specify the that is to be displayed at the top of the screen when the image buffer is dumped to the display buffer. In scroll mode, this line is incremented modulo window line count -1. Bottom of window is then previous line.
	Function data:	one-word integer Y-coordinate of line relative to top of window.

26	READ TOP LINE:	can be used to return Y-coordinater of top line in image buffer.
	Function data:	returns one-word integer Y-coordinate of top line relative to top of window.
27	SCROLL UP N LINES:	specifies the number of lines the screen display is to be scrolled upward. The top n lines are lost; the bottom n lines are cleared to blanks.
	Function data:	one-word integer = n
28	SCROLL DOWN N LINES:	specifies the number of lines the screen display is to be scrolled downward. The bottom n lines are lost; the top n lines are cleared to blanks.
	Function data:	one-word integer = n
29	TRUNCATE LINES ENABLE/DISABLE:	allows the user to choose beween displaying the first n characters of a line or displaying the "overrun" on the next line. For example, if this function is enabled for the display if a directory in a window space 10 lines deep and 20 characters wide, that space will show the first 20 characters of each of 10 successive lines of the directory. If the function is is disabled, the CRT will display the first 20 characters of line 1 on the top line of the window space, the second 20 characters on line 2, and so on until the space is completely filled.

	Function data:	one-word integer 0 -- disable 1 -- enable
30	CLEAR ATTRIBUTES:	nulls the attribute bits in the image buffer
	Function data:	none
31	CONTROL CHARACTER FILTER ENABLE DISABLE:	is used if an "image mode" is desired whereby ASCII control will be displayed (utilizing fonts described in the accompanying font table description). If the filter is disabled, carriage returns (/0D) and line feeds (/0A) will not be used for format control. Otherwise, if the filter is enabled, no characters in the range will be displayed; line feeds and carriage returns are in effect.
	Function data:	one-word integer 0 -- enable filter 1 -- disable filter
32	CURSOR UPDATE MODE:	is used to control the frequency of cursor update. If this function is disabled, the next character is placed at the pointer position (see READ POINTER and WRITE POINTER)
	Function data:	0 -- single-character update 1 -- disable cursor 2 -- block record update
33	ATTRIBUTE UPDATE: ENABLE/DISABLE	allows a user to update attribute bytes only in image buffers rent image buffer pointer.

	Function data:	one word integer 0 -- disable 1 -- enable
34	CHARACTER OVERWRITE MODE SELECT:	allows user control over the treatment of existing data during character generation
	Function data:	one word 0 -- overwrite existing data 1 -- exclusive OR with existing data 2 -- logical OR with existing data 3 -- inverted overwrite
35	XOR CURSOR:	exclusive-OR's cursor font with existing data at current cursor position. If cursor font pixels are illuminated, this function will rease the cursor.
	Function data:	none
36	FILL WORD UPDATE:	allows a user to define a fill word to be used in FLOOD WINDOW, FLOOD LINE, and SCROLL/UP SCROLL DOWN and FILL BORDER function calls.
	Function data:	one word fill pattern
38	FILL BORDER:	allows user to XOR fill word in one-character border around window using preset fill word defined by FUNCTION 32.
	Function data:	none
39	SET/RESET AUTO LINE FEE MODE:	allows every carriage return to be followed by a line feed.

	Function data:	none
40	MOVE CURSOR:	unlike write cursor, this function assumes existence of cursor font at current position and erases it before rewriting at specified locations.
	Function data:	long word Y (0 ≤ Y ≤ line count -1) X (0 ≤ X ≤ column count -1)
41	WINDOW SET:	allows user to set non-default window position and dimensions without reopening logical unit.
	Function data:	one-word integers X upper left-hand corner column (0 ≤ X ≤ column count -1) Y upper left-hand corner line (0 ≤ Y ≤ line count -1) Column count Line count
42	IMAGE BUFFER ADDRESS:	allows a user to set the image-buffer in FUNCTION 18.
	Function data:	long word buffer address or, if null, resets default image buffer
43	IMAGE POINTER WRITE:	sets current pointer in image buffer. Subsequent writes update this pointer. X-Y coordinates are relative to top line.
	Function data:	Long word MSW--Y (0 ≤ Y ≤ line count -1) LSW--X (0 ≤ X ≤ column count -1)
44	IMAGE POINTER READ:	reads current pointer coordinates relative to top

	line. X-Y coordinates are relative to current top line.
Function data:	Long word MSW--Y (0 ¬ Y ¬ line count -1) LSW--X (0 ¬ X ¬ column count -1)

4.0 PRINTER/PLOTTER DRIVER

4.1 DESCRIPTION OF THE USER INTERFACE

INTRODUCTION

Access to the printer is provided through the general purpose system calls to the I/O manager ---> OPEN, CLOSE, AWRITE, SWRITE, FUNCTION, CANCEL and INIT. Note that SREAD and AREAD are not among the above since they are not valid for the printer. Calls to the I/O manager are performed using the SYSIO Macro. In order to make effective use of the information that follows the reader should review to documentation on the I/O manager thoroughly.

4.1.1 PRINTER DEVICE INFORMATION BLOCK (DIB)

The Device Information Block (DIB) for the printer is used to describe the state of the printer at OPEN TIME and is used in conjunction with the SYSIO MACRO call to open a device. The DIB for the printer is composed of the following.

```
PRDIB    EQU  *
         DC.B 'PR        Printer Mnemonic
         DC.B 0          Xmit Direction-Out
         DS.B 8          "For" Task Number
         DS.W 1          Configuration Options
         DS.L 1          Pointer to Function Packet
         DS.L 1          Table offset for byte I/O
                         returned by I/O manager
```

WHERE:

'FOR' TASK NUMBER:	Is the number of the task for whom the device is to be opened if different from the calling task
CONFIGURATION OPTIONS:	There are two options selectable through this field

BIT 0
Fixed Variable block length (0 1)

BIT 1
Enable Suppress (0 1)

Bit 2-14
Not used at present

Bit 15
Option select enable-if set the option bits above will be decoded

FUNCTION PACKET POINTER: Points to a function packet containing the printer options desired at open time. If no function packet is needed then enter 0 in this field. See below for description of the printer function packet.

TABLE OFFSET FOR BYTE I/O: At open time the I O manager passes a table offset value to the user in this field. If the user wishes to use byte I/O he must pass this parameter back to the I/O manager in the SYSIO call

4.1.2 PRINTER DATA TRANSFER CONTROL BLOCK (DTCB)

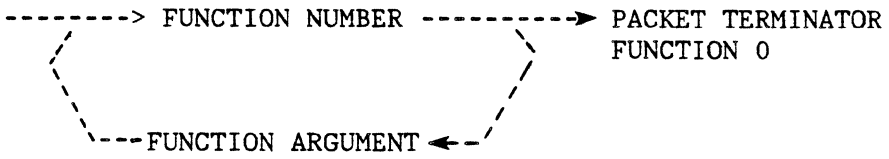
The Printer Data Transfer Control Block (DTCB) is identical to other devices and consists of the following fields:

```
PRDCTB    DS.L 1      Current Status of I O Transfer
          DS.B 1      Termination Byte (Upper)
          DS.B 1      Termination Byte (Lower)
          DS.L 1      Buffer Start Address
          DS.W 1      Buffer Length
          DS.L 1      Buffer Offset Pointer
```

See Logical I O Documentation For Further Details Explanation.

4.1.3 PRINTER FUNCTION PACKET (FPKT

The Printer Function Packet is the vehicle through which a user can change the operation of the printer. This data structure is composed entirely of single word integers in sequence. The sequence consists of function numbers followed by parameters (if needed) and terminated with A (i.e. function 0 is interpreted as the end of the function packet).



FUNCTION PACKET SYNTAX DIAGRAM

4.2 PRINTER MODES

4.2.1 MODES OF PRINTER OPERATION-GENERAL INFORMATION

There are three modes of printer operation - one alphanumeric mode and two graphics modes. The alphanumeric mode is used to output text character while the graphics modes are used to output graphics information.

4.2.2 ALPHANUMERIC MODE

Alphanumeric mode is used to output text either byte at a time or in blocks of fixed or variable length. A variety of functions exist to support alphanumeric output including:

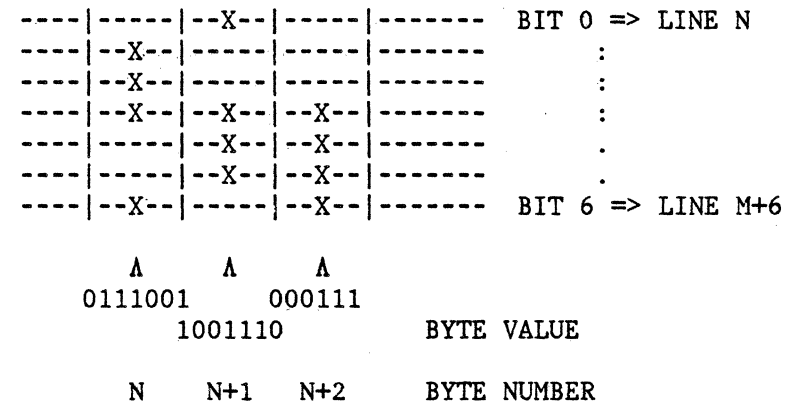
Multiple Colors Proportional Spacing Text Justification Character Enchancement Subscript Superscript

Each of these and more are described below. Printing in alphanumeric mode is bidirectional.

4.2.3 GRAPHICS MODE

There are two modes of graphics output to the printer - 100 dots per inch and 200 dots per inch. Both are otherwise identical. When operating in graphics mode only fixed length block output is permitted (note: successive blocks may vary in length!!!). Bytes received by the printer in graphics mode are interpreted as the firing pattern for the first seven wires on the print head for successive horizontal positions (see diagram). The wires are spaced 4 336's of an inch apart. If desired graphics output lines can be interlaced one or three times to achieve increased vertical resolution. The amount of vertical advance which takes place can be controlled with the vertical advance function or the absolute position function. In general, vertical advance 1 is used for this purpose. An horizontal return is automatically performed following each block of graphics data.

Illustration of pin firing in successive horizontal locations corresponding to graphic data.



4.3 PRINTER SUPPORT FUNCTIONS

4.3.1 SUMMARY OF PRINTER CONTROL FUNCTIONS

The following is a list of available printer control functions:

TABLE 3.0 PRINTER FUNCTIONS

FUNCTION NUMBER	MNEMONIC	DESCRIPTION OF FUNCTION
1	SETWRMOD	Set data transfer mode fixed horizontal
2	SETMODE	Set printer mode
3	INQMODE	Inquire mode
4	SETFONT	Set font
5	INQFONT	Inquire font
6	SETCOLOR	Select ribbon setting (color)
7	INQCOLOR	Inquire ribbon setting
8	SETDENS	Set character density-10 12 16.8 CPI
9	INQDENS	Inquire character density
10	SETFORM	Set form length field
11	INQFORM	Inquire form length
12	SETMARGN	Set right left margins
13	INQMARGN	Inquire right left margins
14	SETNHANC	Set character enhance on off
15	INQNHANC	Inquire character enhance
16	SETPRPSP	Set proportional spacing on off
17	INQPRPSP	Inquire proportional spacing
18	SETNTRSP	Set inter character spacing
19	INQNTRSP	Inquire inter character spacing
20	SETTAB	Set vertical horizontal tab stops
21	INQTAB	Inquire tab stops

22	SETJSTFY	Set text justification on off
23	INQJSTFY	Inquire text justification
24	SETVADV	Set vertical advance distance
25	INQVADV	Inquire vertical advance
26	VERTADV	Perform vertical advance
27	SETPOS	Move head to abs. X Y position
28	SETCHPOS	Move head to abs. col. row position
29	SETRHPOS	Set relative horizontal position
30	PKSET	Printer reset-hard reset+soft reset

4.3.2 SETWRMODE - SET DATA TRANSFER MODE

This function enables the user to change the data transfer mode following open. A value of 0 is used for fixed block transfers while a value of one is used for variable length transfers.

4.3.3 SETMODE - SET PRINTER MODE

SETMODE places the printer in either alphanumeric mode or graphics mode. Mode 0 (or alphanumeric mode) configures the printer to produce alphanumeric output from ASCII character input. The normal alphanumeric control characters are accepted in this mode:

BS	BACK SPACE	08
HT	HORIZONTAL TAB	09
LF	LINE FEED	0A
VT	VERTICAL TAB	0B
FF	FORM FEED	0C
CR	CARRIAGE RETURN	0D

Other ASCII control characters are ignored.

MODE 1 (or graphics mode) configures the printer to produce 100 dot per inch graphics output from Byte width integer input. (See section 2.2 on Graphics for details)

MODE 2 Configures the printer to produce 200 dot per inch graphics output from byte wide integer input. (See section 2.2 on Graphics for details)

Control functions may be performed in either alphanumeric or graphics mode

Note: Only fixed length block transfers are allowed in graphics mode.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	ALPHANUMERIC MODE	0
	100 DPI GRAPHICS	1
	200 DPI GRAPHICS	2
	DEFAULT = 0	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.4 INQMODE - INQUIRE MODE

INQMODE returns the present printer mode in the word following the function in the function packet.

Number of arguments in this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN VALUE OF PRINTER MODE	0-2

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.5 SETFONT - SET FONT

Set font allows the user to select the desired character font to be used for alphanumeric output

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	CHARACTER FONT	0-2
	CORRESPONDENCE FONT	0
	-90 DEGREE ROTATED GRAPHICS FONT	1
	0 DEGREE ROTATED GRAPHICS FONT	2
	DRAFT QUALITY FONT	3

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.6 INQFONT - INQUIRE FONT

INQFONT returns the current character font number in the word following the function number in the function packet.

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN VALUE OF CURRENT FONT SELECTION	0-2

Note: All arguments whether function input or return information occupy the 16-bit word each following the function number in the function packet.

4.3.7 SETCOLOR - SELECT RIBBON SETTING (COLOR)

SETCOLOR shifts the printer ribbon to cause a different color to be output. The color number corresponds to one of the four possible ribbon positions numbered 0-3 starting at the top of the ribbon. Typically the order from the top will be red green blue black.

Number of arguments for this function

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RIBBON POSITION	0-3
	DEFAULT = 3	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.8 INQCOLOR - INQUIRE RIBBON SETTING

INQCOLOR returns the current ribbon position number in the word following the function number in the function packet.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN VALUE OF RIBBON POSITION	0-3

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.9 SETDENS - SET CHARACTER DENSITY-10 12 16.8 CPI

SETDENS allows the user to select the character density to be used for alphanumeric output. Three densities are available - 10, 12 and 16.8 characters per inch.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	OUTPUT CHARACTER DENSITY	0-2
	10 CPI	0
	12 CPI	1

	16.8 CPI	2
	DEFAULT = 0	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.10 INQDENS - INQUIRY CHARACTER DENSITY

INQDENS returns the number corresponding to the output character density in the word following the function number in the function packet.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN CHARACTER DENSITY NUMBER	0-2

Note: All arguments wheter function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.11 SETFORM - SET FORM LENGTH FIELD

SETFORM allows the user to select the form length and print field length. The first argument is the form length and the second argument is the form field length. The form length must be greater than the form field. Each argument is expressed in 1 336 of an inch.

Number of Arguments for this Function = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	FORM LENGTH IN 1 336 IN DEFAULT = 3696	0-9999
2	FORM FIELD IN 1 336 IN DEFAULT = 3996	0-9999

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number

in the function packet.

4.3.12 INQFORM - INQUIRE FORM LENGTH

INQFORM returns the current form length and field length in the two words following the function number in the function packet.

Number of arguments for this function = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN FORM LENGTH	0-9999
2	RETURN FORM FIELD LENGTH	0-9999

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.13 SETMARGN - SET RIGHT LEFT MARGINS

SETMARGIN sets the right and left margins. Argument 1 is left margin and argument 2 is the right margin. Both are expressed in 1 120 inch units. The right margin must exceed the left margin.

Number of arguments for this function = 2

ARGUMENT #	DESCRPTION	RANGE VALUE
1	LEFT MARGIN POSITION IN 1 120 IN. DEFAULT = 0	0-960
2	RIGHT MARGIN POSITION IN 1 120 IN DEFAULT = 960	0-960

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.14 INQMARGN - INQUIRE RIGHT LEFT MARGINS

INQMARGIN returns the right and left margin positions in the two words following the function number in the function packet.

Number of arguments for this function = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN POSITION OF THE LEFT MARGIN	0-960
2	RETURN POSITION OF THE RIGHT MARGIN	0-960

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.15 SETNHANC - SET CHARACTER ENHANCE ON OFF

SETNHANC selects or deselects character enhancement. If enhancement is selected then characters will be printed with double width.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	CHARACTER ENHANCEMENT SELECT	0-1
	ENHANCEMENT OFF	0
	ENHANCEMENT ON	1
	DEFAULT = 0	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.16 INQNHANC - INQUIRE CHARACTER ENHANCE

INQNHANC returns a zero in the word following the function number in the function packet if enhancement is not selected otherwise a one is

returned.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	RETURN ENHANCEMENT STATE (ON OFF)	0-1

NOTE: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet

4.3.17 SETPRPSP - SET PROPORTIONAL SPACING ON OFF

SETPRPSP selects deselects proportional spacing of output text. If a value of 0 is entered in the argument field then proportional spacing is deselected. If one is entered then proportional spacing is selected.

NOTE:Proportional spacing may not be used if the inter-character spacing is >0.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	DESELECT SELECT PROPORTIONAL SPACING	0-1
	PROPORTIONAL SPACING OFF	0
	PROPORTIONAL SPACING ON	1
	DEFAULT = 0	

NOTE: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.18 INQPRPSP - INQUIRE PROPORTIONAL SPACING

INQPRPSP returns A 0 or A 1 according to whether proportional spacing is on of off. The argument is returned in the word following the function number in the function packet.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	Proportional spacing off on	0-1

NOTE: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.19 SETNTRSP - SET INTER CHARACTER SPACING

SETNTRSP allows the user to adjust the inter-character spacing. The argument of this function is the inter-character spacing in 1 24's of a character width.

NOTE: Inter-character width >0 cannot be used with proportional spacing.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	Intercharacter spacing in 1 24's of a character width	0-999
	DEFAULT = 0	

NOTE: All arguments whether function input or return information occupy one 16 bit word each followint the function number in the function packet.

4.3.20 INQNTRSP - INQUIRE INTER CHARACTER SPACING

INQNTRSP returns the value of the inter-character spacing increment in the word following the function number in the function packet.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	Return inter-character space value	0-999

expressed in 1 24's of a character width

NOTE: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.21 SETTAB - SET VERTICAL HORIZONTAL TAB STOPS

SETTAB allows the user to set the horizontal vertical tab stops for the printer. The first argument selects either horizontal or vertical tabs with A 0 or 1. The second argument is the number of tabs the user wishes to set. The following arguments are the positions of the tab stops in order from left to right or top to bottom in ascending numerical order. Thirty-two (32) horizontal tabs may be specified in units of 1 120's of an inch. Eight vertical tabs may be specified in units of 1 336's of an inch.

Number of arguments for this function = 2-34

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	HORIZONTAL VERTICAL TAB SELECT	0-1
2	NUMBER OF HORIZONTAL OR VERTICAL TABS TO BE SELECTED	0-32 HOR.
3-35	TAB POSITIONS	0-8 VERT.
	UP TO 32 HORIZONTAL TAB POSITIONS 1 120,	0-960
	UP TO 8 VERTICAL TAB POSITIONS 1 336,	0-9999

Note: All arguments whether function input or return informatio occupy one 16 bit word each following the function number in the function packet.

4.3.22 INQTAB - INQUIRE TAB STOPS

INQTAB returns the current horizontal vertical tab positions. The first argument selects vertical or horizontal tabs. The second argument selects the number of tabs the user wishes to inquire. The printer driver will return the umber of tabs selected in the second argument in successive

words in the function packet. If a tab has not been set the driver will return a zero in the cooresponding location.

Number of arguments for this function = 2-34

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	HORIZONTAL VERTICAL TAB SELECT	0-1
2	NUMBER OF HORIZONTAL OR VERTICAL TABS TO BE RETURNED	0-32 HOR. 0-8 VERT.
3-35	RETURNED TAB POSITIONS	
	UP TO 32 HORIZONTAL TAB POSITIONS 1 120,	0-960
	UP TO 8 VERTICAL TAB POSITIONS 1 336,	0-9999

NOTE: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packets.

4.3.23 SETJSTFY - SET TEXT JUSTIFICATION ON OFF

SETJSTFY turns justification off or on according to whether the function argument is 0 ir 1. While in this mode, the printer will 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.

Number of Agruments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	DESELECT SELECT TEXT JUSTIFICATION	0-1
	TEXT JUSTIFICATION OFF	0
	TEXT JUSTIFICATION ON	1
	DEFAULT = 0	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.24 INQJSTFY - INQUIRE TEXT JUSTIFICATION

Returns an argument value of 0 or 1 in the word following the function number according to whether text justification is off or on.

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE

Number of arguments for this function = 1

ARGUMENT	DESCRITPION	RANGE VALUE
1	RETURN ARGUMENT	0-1
	TEXT JUSTIFICATION OFF	0
	TEXT JUSTIFICATION ON	1

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.25 SETVADV - SET VERTICAL ADVANCE DISTANCE

SETVADV allows the user to set one of the three programmable Vertical Advance lengths.

Note: Vertical Advance 0 is the same as a Line Feed.

The first argument selects the Vertical Advance to be programmed. The second argument is the desired length of the Vertical Advance. Vertical Advance 0 and 1 are forward advances while Vertical Advance 2 is a Reverse Advance. Vertical Advance lengths are in units of 1 336's of an inch.
Advance 0 is equivalent to a line feed and is used for this purpose in alphanumeric mode
Advance 1 is typically used as a line feed in Graphics Mode.
Advance 2 is typically used for super-scripting in alphanumeric mode.

Number of arguments for this function = 2

ARGUMENT	DESCRITPION	RANGE VALUE
1	VERTICAL ADVANCE SELECT	0-2
	VERTICAL ADVANCE 0	0
	VERTICAL ADVANCE 1	1
	VERTICAL ADVANCE 2	2
2	VERTICAL ADVANCE LENGTH IN 1 336's OF AN INCH	0-9999
	DEFAULT = 56 ADVANCE 0 28 ADVANCE 1 28 ADVANCE 2	

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.26 INQVADV - INQUIRE VERTICAL ADVANCE

INQVADV returns the current length of the vertical advance selected in the first argument in the second argument.

Number of arguments for this function = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	VERTICAL ADVANCE SELECT	0-2
2	RETURN LENGTH OF THE VERTICAL ADVANCE SELECTED IN ARGUMENT 1 IN 1 336's OF AN INCH	0-9999

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.27 VERTADV - PERFORM VERTICAL ADVANCE

VERTADV performs a vertical advance 0,1 or 2 according to the value of the function argument. Vertical Advance 0 or 1 will move the paper forward while Vertical Advance 2 will move the paper backward. (See above for defaults etc.)

Number of arguments for this function = 1

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	SELECT VERTICAL ADVANCE 0,1,2	0-2

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.28 SETPOS - MOVE HEAD TO ABS. HORIZONTAL VERTICAL POSITION

SETPOS moves the print head to an absolute horizontal or vertical position. The first argument selects the horizontal or vertical axis, while the second argument specifies the absolute coordinate to be moved to on that axis. Horizontal units are in 1 120's of an inch while vertical units are in 1 336's of an inch.

Number of arguments for this section = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	HORIZONTAL VERTICAL AXIS SELECT	0-1
2	POSITION VALUE	0-960 HOR. 0-9999 VERT.

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.29 SETCHPOS - MOVE HEAD TO ABS. COLUMN ROW POSITION

Moves the print head to an absolute horizontal or vertical position expressed in column or row units. Argument 1 selects either the horizontal or vertical axis while argument 2 specifies the position to be moved to on the axis.

Note: The column row unit dimension correspond to the current character width line height.

Number of arguments for this function = 2

ARGUMENT #	DESCRIPTION	RANGE VALUE
1	SELECTING HORIZONTAL OR VERTICAL AXIS	0-1
2	COORDINATE ON HORIZONTAL OR VERTICAL AXIS	0-120 HOR. 0-9999 VERT.

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.3.30 SETRHPOS - SET RELATIVE HORIZONTAL POSITION

SETRHPOS move the print head an incremental amount in the horizontal axis. Units for relative horizontal movement are in 1 120's of an inch.

Number of arguments for this function = 1

ARGUMENT #	DESCRITPION	RANGE VALUE
1	HORIZONTAL INCREMENT IN 1 120's OF AN INCH	-120 to +120

Note: All arguments whether function input or return information occupy one 16 bit word each following the function number in the function packet.

4.4 PRSET HARD PRINTER RESET

PRSET performs a hard and soft printer reset. There are no arguments for this function.

5.0 RS-232 ASYNCHRONOUS COMMUNICATIONS DRIVER

5.1 INTRODUCTION

The RS-232 Asynchronous communications driver provides support for multiple RS-232 ports from a single ROM'd version of code. Port vectoring upon entry to the driver enables independent control via separate control blocks and status save areas for each port. Default configuration mechanisms free the user from the burden of setting up the port for normal system I/O. Alternatively, the driver allows a large degree of user programmability for customization needs in constructing high level protocols.

5.2 FEATURES

The driver is position-independent and can be moved in memory prior to opening the port.

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 actual I/O rate of the port.

The driver provides resident buffer space for output and for accumulating unsolicited input data. These are circular buffers with overrun indication or lockout on input. Alternate buffers can be specified by the user if desired, such as during block I/O or DMA transfers.

Read and write operations through the port are potentially independent, and may be concurrent. (The ACIA provides a hardware restriction that baud and framing be the same for input and output.) Some protocols may require that the port be restricted to input only or output only at any one time. The choice is user-selectable.

Programmable timeout periods are provided for transmission and reception. The possible "hanging" of the port will create an interrupt which is reported to the user.

There is separate status reporting to the I/O manager and to the user. This provides the calling task with interrupt-time updating of the state of the port, and allows for task switching based upon I/O completion or other state-change of the port.

User customization routines may be attached to the driver which allow the driver to continue providing interrupt service and system maintenance, but which pass control to the user routines for character processing.

5.3 OVERVIEW - GENERAL DESCRIPTION

User data associated with an entry into the driver is passed in a couple of ways. Usually the driver is given a pointer to the request packet formed by the I/O manager. This packet has the location of all control blocks and information required to perform the operation. For byte I/O operations, the system overhead associated with the control blocks and request packet is circumvented by passing data directly in registers.

5.3.1 USER INITIALIZATION OF THE PORT

Attaching the port notifies the system of the existence of the RS-232 driver. A user (it may be the system) must initialize the port for a specific protocol.

At OPEN time the user prepares a device initialization block (DIB), which allows user configuration of the port. Default data is supplied by the RS-232 driver if the user supplies a blank data field (all zeros).

The data word at DIB location "DIBOPT" is used for mode specification. The format of the DIBOPT is driver specific, and is listed under "MODES".

Other port parameters may be selectively changed from default values by specifying a pointer in the "DIBFCN" location. This identifies the start of a list of items to be changed in the protocol. (As an example, the user may want the default configuration except for the receive timeout interval.) The format of this list is described in the "FUNCTIONS" text.

5.3.2 WRITE RECORD

For record input or output, the user must set up a data transfer control block (DTC). This block indicates the start and length of the user

buffer, end-of-record indicators for variable length records, and a buffer-pointer that indicates the current position within the user buffer.

The RS-232 driver moves the user data into the driver's internal buffer before transmission. The buffer-start address and buffer-pointer are added together to identify the start of the data. Transfer will continue until an end-of-record character or end of user buffer is encountered.

The "MODES" description further on in this document provides details of fixed versus variable-length record writes and end-of-record decision making.

5.3.3 READ RECORD

Reading a record is similar to writing. The user must set up a DTC with buffer start, length, pointer, and a termination characters if variable-length reads are in effect.

Non-DMA transfers move data into the user's buffer starting at the pointer offset, and ending at buffer-end or with the end-of-record terminator.

See the "MODES" description in this document for details of the various read-record formats.

5.3.4 BYTE I/O

Byte I/O avoids the overhead of creating control structures in order to access the driver. Byte I/O is always synchronous, and proceeds only if the driver is idle at the time of the call. Three functions are provided for byte I/O: write, read and test. The test function reads a character but does not increment buffer pointers; the character is left there for the next test or read.

5.3.5 FUNCTION ENTRY

The functions entry into the RS-232 driver provides a means of changing or querying a driver parameter, without having to completely re-initialize the port.

Associated with a SYSIO FUNCTION call is a pointer to a function packet. The function packet is a list of commands and associated data which the driver processes. The commands allow specification or inspection of certain device parameters, such as buffer lengths or timeout periods. Function calls are always unique to the driver.

5.3.6 TRANSMIT MODES

Fixed or Variable Length Records

Fixed: No checking for record termination characters is performed. The output transfer starts at DTCBFS+DTCPTR, and stops at the end of the DTC buffer, at DTCBFS+DTCBFL. The buffer point is updated by the driver as characters are moved from the DTC buffer to the driver's circular buffer, and will have a value equal to DTCBFL at the end of the transfer.

Variable: An interval of record termination characters is supplied by the user via DTCTBU (upper termination byte) and DTCTBL (lower termination byte). The driver will check characters as it moves them from the DTC to the circular ouput buffer. After a character is moved which is in the range of termination characters data transfer stops. The buffer pointer, DTCPTR, will point to the termination character encountered. Note that termination character checking is done after calling the user-supplied character processing subroutine.

Method of specification: via bit in DIBOPT word, FCN CODE 1 or 2.

Character or Block Mode

Character: This mode should be the one used in most cases. The data characters are moved into the circular output buffer, then transmission starts. If a is record longer than the circular buffer, the buffer will be filled up once, transmitted, then refilled as necessary. The default length of the circular output buffer is 256 bytes. Long records may be transmitted more efficiently via the #SER02 port DMA, but this allows no character processing.

Block: This mode should only be used if the port is a dedicated device. Transmission does not necessarily start after the record is moved into the circular buffer, but is subject to a 'prompt' or 'send' character to start transmission. The port will stay in a busy state until the transmit buffer is empty. Timeouts are in effect.

Method of specification: bit in DIBOPT word or FCN CODE 2.

I/O Time Character Processing Enabled/Disabled

I/O time character processing refers to checks done as data is moved between the circular buffer and the port hardware. Output characters are compared against characters in a control-character table, and certain actions are taken if a match is found. An example would be to have the first character of an output record reset the input buffer in order to synchronize record transfers. Local control characters are not transmitted. Characters in the control-character table are user programmable.

Method of specification: bit in DIBOPT word or FCN CODE 2

XON/XOFF Enable/Disabled

XON/XOFF enabled on output allows the transmission of an XON or XOFF character according to the number of characters in the input circular buffer. The length of the circular buffer is 256 bytes. An XOFF will be transmitted when the character count is 192, an XON when the count is 64. Note that the port owner may replace the default circular buffer with one in his own memory space. This is done via a function call.

Method of specification: bit in DIBOPT word or FCN CODE 2.

Half/Full Duplex

Full duplex: The port transmitter and receiver are independent (except for baud and framing). The I/O manager implements separate queus for transmit and receive. The user can choose to loosely couple the transmitter by allowing recognition of XON/XOFF characters by the receiver, which in turn controls the transmission rate. This coupling would not effect the receiver and would only show up as an increased delay before transmission for the output. Transmit and receive can be operating concurrently.

Half duplex: The port may only transmit or receive at any one time. The port must be in the transmit mode before starting record output. The port switches from receive to transmit modes upon the receipt of a line-turnaround character (receive-side control character checking must be enabled). The port switches from transmit to receive modes after transmitting a turn-around character (transmit control character checking must be enabled).

Method of specification: bit in DIBOPT word or FCN CODE 2.

Echoplex Enabled/Disabled

The port will wait for an echo after the transmission of each data character if Echoplex is enabled for output. The echoed character is not compared to the character sent, and is treated as per a normal received data character.

Method of specification: bit in DIBOPT word or FCN CODE 2.

5.3.7 RECEIVE MODES

Fixed or Variable Length Records

Fixed: No checking for record termination characters is performed. The input transfer starts at DTCBFS+DTCPTR, and ends at DTCBFS+DTCBFL. The buffer pointer is at the value DTCBFL.

Variable: An interval of record termination characters is supplied by the user via DTCTBU (upper termination byte) and DTCTBL (lower termination byte). The driver will check characters as it moves them to the DTC buffer. After a character is moved which is in the range of termination characters data transfer stops. At termination the buffer pointer, DTCPTR, points to one beyond the last character. This allows the pointer to be used as a record length indicator, or for contiguous records to be input without having to adjust the buffer pointer.

Method of specification: bit in DIBOPT word or FCN CODE 2.

Character or Block Mode

Character: This is an asynchronous read. If a complete record is not in the circular input buffer at the time of call, the incomplete record is moved to the DTC buffer. The transaction is then regarded as complete. The return status should be checked for indication of a complete record.

Block: In this mode a complete record is read into the DTC buffer. If a complete record is not available in the circular receive buffer at the time of call a receive timeout is started. If no new character is received in the timeout interval, the transaction is ended with a timeout return status. If no timeout occurs, reception will continue until a termination character is detected.

Method of specification: bit in DIBOPT word or FCN CODE 2.

Transfer-Time Character Processing Enabled/Disabled

Character processing at transfer timer refers to the transfer of data between the circular input buffer and the DTC buffer. This is a point at which a user may provide a custom subroutine. The variable-length record termination byte checking is effected by the processing routine, in that termination characters are checked after the routine.

Method of specification: bit in DIBOPT word or FCN CODE 2.

I/O Time Character Processing Enabled/Disabled

I/O time character processing refers to checks done as data is moved between the port hardware and the circular buffer. Processing at this point causes state changes to the port at reception time. Examples would be the reception of line turnaround characters.

Method of specification: bit in DIBOPT word or FCN CODE 2.

Half/Full Duplex

The discussion under Transmit Modes is applicable to both transmit and receive.

Echo Enabled/Disabled

If echo is enabled, the received character is echoed to the output side of the port. The character will intervene in any output in progress.

5.3.8 FUNCTIONS

The Function Packet is a list of FCN CODE and FCN DATA words. The start of the list is passed to the driver via the SYSIO FUNCTION call. The meaning of the codes, and the data associated with them, is unique for each driver in the system. The following list displays FCN CODE and associated FCN DATA for the RS232 driver. Some functions allow setting or querying of the driver configuration. If the driver is being queried, space must be provided in the function list for the return data.

CODE	Description and DTA
0	End of function list.

No data.

1 Transfer Mode: Fixed or Variable Length Records
Word: 0= fixed length, 1=variable length

2/3 Set/Get Mode Bits.
Word: Mask, a '1' indicates bit in next word is active.
Word: Bits same as DIBOPT specification.

4/5 Set/Get Transmit Timeout Period (50ms Intervals)
Word: -1 is infinite timeout period. Positive number must be between 1 and $7FFF.

6/7 Set/Get Receive Timeout Period (50ms Intervals)
Word: same as transmit

8/9 Set/Get Transmit Control Character in Current Table
Byte: character offset in table
Byte: data character

10/11 Set/Get Receive Control Character in Current Table
Byte: character offset in table
Byte: data character

12/13 Not used

14/15 Not used

16/17 Set/Get Transmit Control Character Table
Longword: Pointer to start of 16 byte character table.

18/19 Set/Get Receive Control Character Table
Longword: Pointer to start of 16 byte character table.

20/21 Set/Get Baud
Word: value as per table in appendix

22/23 Set/Get Framing
Word: value as per table in appendix

24/25 Set/Get Transmit Buffer Parameter Block
Longword: a pointer to a parameter block in user space, used to create a new circular buffer. If the pointer is a 0 value the driver defaults back to its internal circular buffer.

26/27 Set/Get Receive Buffer Parameter Block
Longword: same as transmit

28/29 Not used

30/31 Not used

32/33 Set/Get Transmit Character Translation Subroutine
Longword: the address of a user subroutine to process characters. A zero value defaults to the driver dummy subroutine.

34/35 Set/Get Receive Character Translation Subroutine
Longword: same as transmit

36/37 Set/Get DMA Limit
Word: buffer length at which #SER02 uses DMA.

38 Transmit Break
Word: duration (in 50ms intervals) of the break condition

5.3.9 RETURN STATUS CODES

During a record transaction the driver returns status to the user in the DTCSTA byte in the user's DTC control block. The status information is given below; the periods are don't care bits, the numerals are values for various bit fields.

```
    Bit Number
 7 6 5 4 3 2 1 0
|-|-----|-------|  Transaction complete/incomplete:
 0 . . . . . . .       port operation complete
 1 . . . . . . .       port operation incomplete
|-|-----|-------|  Record transfer to/from DTC buffer:
 .   0   . . . .       no data in bits 6-4
 .   1   . . . .       transfer suspended: circular buffer full
 .   2   . . . .                           circular buffer empty
 .   3   . . . .       transfer ended: term character found
 .   4   . . . .                       term character not found
 .   5   . . . .                       circular buffer empty
 .   6   . . . .                       circular buffer full
 0   7   . . . .       data transfer aborted or not started:
 0   7     0                 port already active
 0   7     1                 port half-duplex, contention problem
 0   7     2                 timeout
|-|-----|-|-----|  Codes specific to read operations:
 .  0-6  0  0-7         unused
 .  0-6  1 . . 1        receiver parity error
 .  0-6  1 . 1 .        receiver overrun
 .  0-6  1 1 . .        receiver framing error
|-|-----|-|-----|  Codes specific to write operations:
 .  0-6  0 . . .        transmit circular buffer empty
 .  0-6  1 . . .        transmit circular buffer not empty
 .  0-6  .   0          unused
 .  0-6  .   1          XOFF currently active
 .  0-6  .   2          port is in RCV mode (half-duplex)
|-|-----|-|-----|
 7 6 5 4 3 2 1 0
```

5.4 APPENDIX

5.4.1 DIBOPT BIT SPECIFICATION

Transmit Mode Bits

Bit 0:	0=fixed length records	1=variable length records
Bit 1:	0=not used	
Bit 2:	0=character mode	1=block mode
Bit 3:	0=no XON/XOFF transmitted	1=XON/XOFF chars enabled
Bit 4:	0=filter chars disable	1=filter chars enabled
Bit 5:	0=half duplex	1=full duplex on transmit
Bit 6:	0=no echoes expected	1=get echo after each char xmt
Bit 7:	0=not used	

Receive Mode Bits

Bit 8:	0=fixed length records	1=variable length records
Bit 9:	0=not used	
Bit 10:	0=character mode	1=block mode
Bit 11:	0=no XON/XOFF checked	1=XON/XOFF chars checked at input
Bit 12:	0=filter chars disable	1=filter chars enabled
Bit 13:	0=half duplex	1=full duplex on receive
Bit 14:	0=no echoes	1=transmit echo of each rcv'd char
Bit 15:	0=not used	

5.4.2 BAUD EQUATES FOR #SER00 TO #SER02

These equates list the baud rates possible for ports #SER00, #SER01, and #SER02. The data is used in FCN CODE 20.

```
B45.5    EQU    $00
B50      EQU    $01
B75      EQU    $02
B110     EQU    $03
B134.5   EQU    $04
B150     EQU    $05
B300     EQU    $06
B600     EQU    $07
B1200    EQU    $08
B1800    EQU    $09
B2000    EQU    $0A
B2400    EQU    $0B
B4800    EQU    $0C
B9600    EQU    $0D
B19200   EQU    $0E
B38400   EQU    $0F
```

5.4.3 BAUD EQUATES FOR #SER03 TO #SER06

These equates list the baud rates possible for ports #SER03, #SER04, #SER05, and #SER06. The data is used in FCN CODE 20.

```
B50       EQU     $00
B75       EQU     $01
B110      EQU     $02
B134.5    EQU     $03
B150      EQU     $04
B300      EQU     $05
B600      EQU     $06
B1200     EQU     $07
B1800     EQU     $08
B2000     EQU     $09
B2400     EQU     $0A
B3600     EQU     $0B
B4800     EQU     $0C
B7200     EQU     $0D
B9600     EQU     $0E
B19200    EQU     $0F
```

5.4.4 FRAME EQUATES

```
STOP1     EQU   $40     1 stop bit
STOP1.5   EQU   $80     1.5 stop bit
STOP2     EQU   $C0     2 stop bits
PAREVEN   EQU   $30     even parity
PARODD    EQU   $10     odd parity
PARNONE   EQU   $01     no parity
DATA5     EQU   $00     5 data bits
DATA6     EQU   $04     6 data bits
DATA7     EQU   $08     7 data bits
DATA8     EQU   $0C     8 data bits
```

5.4.5 CONTROL CHARACTERS

This table shows the default control characters and their placement in the character tables. Separate tables are maintained for the transmitter and the receiver. The XON and XOFF characters are referenced if XON/XOFF is enabled via mode bit #3. Other characters are referenced if mode bit #4 is set. Applications may change or query any character or table via function calls 16-19.

```
IGN1      EQU   $7F     0: delete this character from the stream
IGN2      EQU   $7F     1: delete this character from the stream
XON       EQU   $11     2: xmt/rcv this character for XON
XOFF      EQU   $13     3: xmt/rcv this character for XOFF
RSXB      EQU   $18     4: reset transmit circular buffer
RSRB      EQU   $12     5: reset receive circular buffer
DLXC      EQU   $04     6: delete last character from transmit buffer
DLRC      EQU   $7F     7: delete last character from receive buffer
SEND      EQU   $01     8: enable transmit to send record
                $7F     9: unassigned
TURN      EQU   $14    10: line turn-around character in half-duplex
                $7F    11: unassigned
IGN3      EQU   $7F    12: delete this character from the stream
IGN4      EQU   $7F    13: delete this character from the stream
                $7F    14: unassigned
                $7F    15: unassigned
```

6.0 IEEE-488 INTERFACE DRIVER

6.1 INTRODUCTION

The IEEE-488 Interface Driver acts as a single controller or talker/listener on the IEEE-488 General Purpose Interface Bus.

6.1.1 CHARACTERISTICS AND FEATURES

- Single controller states C1-C4 and C26
- Automatic talker and listener addressing, with extended addressing (TE5 and LE3)
- Service requests (SR1) with parallel (PP1) and serial (C26) polling
- Remote and Local lockout (RL1)
- Device clear (DC1)
- Independent device trigger (DT1)
- Multiple simultaneous listeners only for group execute trigger (GET)
- DMA used for transfers greater than programmable limit.
- Optional timeouts on all transfers
- Optional EOS character generation and recognition
- All GPIB handshaking invisible to high-level caller
- Direct user access to GPIB via FUNCTIONcall sequences

6.2 DEVICE ADDRESSING

The controller device, and other devices on the GPIB, are attached with an address which is used for all subsequent talker and listener addressing. The controller device is typically attached and opened by the system task. Other devices on the bus are attached and opened by application tasks.

The attachment of user devices should be routed through the main controller jump table in order to find the device attach entry point in the driver. At the entry into the driver, data register D1.W is expected to have the device address, with the primary address in the MSB, the extended address (if used) in the LSB. Only the low five bits of each byte are significant, and the range of allowable addresses is $00-$1E. If extended addressing is not used the extension address should be set to value greater than $1E. The printable character '?' can be used to do this.

The system attaches the controller via the call

```
MOVE.B    #'A?',D1      controller address binary 00001
JSR       ATCHCTLR      driver entry point
```

where the address 'A?' creates a primary address of 1 for the controller (extended addressing is not supported for the Computer System itself). The driver entry point is known via the system link map.

A user may attach a device on the bus by the ATCHDEV command. The file must execute the following code

```
MOVE.B    #'BC',D1      device address binary 00010 00011
JSR       ATCHDEV       driver entry point
```

where D1.B is loaded with the particular device address. The code would then proceed with a device OPEN. In this example the bus device has a primary address 2 ('B') and extension 3 ('C').The ATCHDEV label for the JSR is an entry point into the driver; it has nothing to do with the system command.

6.3 REQUEST QUEUING

With multiple devices attached to the GPIB, access is on a first-come first-serve basis; the queueing 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 timeout 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.

6.4 I/O ENTRY POINTS

6.4.1 INIT

The INIT entry point processes the Device Initialization Block (DIB). This is called when SYSIO OPEN is invoked, or may be called explicitly by the user to re-initialize a device via SYSIO INIT.

An example of a DIB for a device which does not have service request capability would be:

```
GPBDIB   EQU      *
         DC.B     '#GPBBC'        device address 23
         DC.B     2               open for input/output
         DC.B     0
         DC.L     0,0
         DC.W     $000F           all options except SRQ
         DC.L     0               no function list
         DC.L     0               byte I/O not supported via SYSIO call
```

The bit signifance of the DIBOPT field is explained further on in the text.

6.4.2 AWRITE, SWRITE

SYSIO calls AWRITE and SWRITE to initiate record output to a device on the bus. A Data Transfer Control (DTC) block must be specified. An example of a valid DTC would be:

```
GPBDTC   EQU      *
         DC.B     1               status byte
         DC.B     0,0,0           termination bytes - not used
```

```
DC.L    BUF     pointer to write buffer
DC.W    16      buffer length- length of record
DC.W    0       buffer pointer, reset by user
DC.L    0       not used
```

6.4.3 AREAD, SREAD

SYSIO calls AREAD and SREAD to initiate input of a record from a device on the GPIB. A DTC must be supplied, as in the case of write-record.

6.4.4 FUNCTION

The FUNCTION entry point is called during a SYSIO OPEN, and may be called explicitly later via SYSIO FUNCTION. This entry point provides a means of supplying device parameters to the driver, as well as performing certain bus operations. The function packet is explained later in this chapter.

6.4.5 DEVICE DETACHMENT

This entry point is called during a SYSIO CLOSE, when a device is being removed from the bus. No control structures are required from the user.

6.5 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, at INIT, bits #5 and #4 are set in DIBOPT. 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 bereturned. The user is responsible for setting the device serial poll response byte via a function packet sequence.

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 transfered to the event block for use by the task.

Task wakeup upon an asynchronous event is not currently implemented. As an interim measure, the task may specify an address pointer to a byte of storage in its own space. The driver will deposit the poll status byte there following a service request. The task must interrogate the contents of the byte in order to detect the service request occurance. This pointer is specified by FCN CODE 31.

6.6 DIB SPECIFICATION

The driver supports six individual mode bits, specified in the DIBOPT field:

Bit#	Value	Description
0	1 =	Enable auto detection of EOS character
1	1 =	Enable auto generation of EOS character
2	1 =	Use 8 bit EOS character (else 7)
3	1 =	Enable timeouts
4	1 =	Device has service request capability
5	1 =	Configure device for parallel poll

6.7 DEVICE FUNCTIONS

The FUNCTION PACKET allows data specification as described in the DEVICE FUNCTIONS section below. However, it also provides the facility to chain individual subroutine calls together during one system I/O request. The FPKT may supply a list of GPIB operations which are executed in sequence by the driver before returning to the applications program.

6.7.1 FUNCTION PACKET SPECIFICATION

The function packet is processed by the driver in a sequential manner. The driver provides access to the GPIB via subroutine calls specified in the FPKT. Access at the subroutine call level allows a convenient means of performing transactions on the GPIB without having to consider hardware programming or handshaking. However, these subroutine calls can be strung together in any order by the applications programmer; the bus sequences are not checked by the driver and are the sole responsibility of the programmer.

Should an error occur in processing the function packet, the value of the FCN CODE executing when the error occured will be placed in the return status.

INVALID Format: DC.W $0001

This code, which is used by the system to select fixed or variable length records, is invalid for the GPIB driver.

DCL Format: DC.W $0002

This code issues an UNLISTEN, then a universal DEVICE CLEAR.

END Format: DC.W $0003

This code requests CS-OS to send EOI on the next data byte transmitted. The driver must be in talker mode.

EOS Format: DC.W $0004

This routine sends the EOS character, as specified in the DIB extension area during OPEN, or later during a function call.

FSH Format: DC.W $0005

This command issues a local FINISH HANDSHAKE command to the NEC D7210 chip.

GET Format: DC.W $0006

This code sends a GROUP EXECUTE TRIGGER command to the GPIB.

GTL Format: DC.W $0007

This code sends a GO TO LOCAL command onto the bus.

GTS Format: DC.W $0008

This issues a local GO TO STANDBY to the NEC D7210 chip.

HSM Format: DC.W $0109
DC.W data word

This command sets the handshake mode on the NEC D7210 chip. Only bits 1 and 0 are significant. The format for these bits is:

%00 Normal handshake mode
%01 RFD holdoff on all data mode
%10 RFD holdoff on END mode
%11 Continuous mode

IFC Format: DC.W $000A

This command issues an INTERFACE CLEAR command onto the GPIB. The duration of the clear command is approximately 200 microseconds.

LLO Format: DC.W $000B

This routine issues a LOCAL LOCKOUT command onto the bus. This will effect all listeners in the REMOTE state.

LOC Format: DC.W $000C

This command resets the REN line on the GPIB.

MLA Format: DC.W $000D

This command sends the driver address onto the bus as a listener device. Extended addressing is supported.

MTA Format: DC.W $000E

This command sends the driver address onto the bus as a talker device. Extended addressing is supported.

OLA Format: DC.W $010F
DC.W device address

This code sends an OTHER LISTENER ADDRESS onto the bus. Extended addressing is supported.

OTA Format: DC.W $0110
DC.W device address

This code sends an OTHER TALKER ADDRESS onto the bus. Extended addressing is supported.

REM Format: DC.W $0011

This command sets the REN line on the GPIB.

SCOM Format: DC.W $0112
DC.W command byte

This code sends a byte command onto the GPIB. The driver must be bus controller. The routine will wait until the driver is ready to accept the command byte for transmission, i.e., it will wait for previous handshaking to be completed.

SDC Format: DC.W $0013

This code sends a SELECTED DEVICE CLEAR onto the GPIB.

SPD Format: DC.W $0014

This will send a SERIAL POLL DISABLE command onto the GPIB.

SPE Format: DC.W $0015

This will send a SERIAL POLL ENABLE command onto the GPIB.

UNL Format: DC.W $0016

This code sends an UNLISTEN command onto the GPIB.

UNT Format: DC.W $0017

This code sends an UNTALK command onto the GPIB.

DBI Format: DC.W $0418
              DC.W space in which driver places data byte

This code reads a single data byte from the GPIB. The bus talker and listeners (including the driver) must be previously configured. A timeout will be started if the option is enabled.

TCE Format: DC.W $0019

This command will TAKE CONTROL on END. The driver will take control of the bus following the next END message.

TCS Format: DC.W $001A

This command will tell the driver to TAKE CONTROL SYNCHRONOUSLY after the data byte transfer.

SPL Format: DC.W $0F1B
              DC.W device address
              DC.W space for device status response
              DC.W device address
              DC.W space for device status response
                .
                .
                .
              DC.W '??'

This command issues a SERIAL POLL to a list of devices. The list must be terminated with an illegal device address (the printable characters '??' do it). The device response is entered into the location in the FPKT list following the device address.

TCS Format: DC.W $001C

This command will tell the driver to TAKE CONTROL ASYNCHRONOUSLY regardless of the current bus state.

TMRDI Format: DC.W $001D

This command disables timeouts.

TMREN Format: DC.W $011E
DC.W timeout period per byte

This command enables timeouts and specifies the timeout interval.

SRQBFS Format: DC.W $021F
DC.L address pointer

This command specifies an address pointer to a location in user space. In the case of a service request by the device, the driver will place the status in this location. This is a temporary means of asynchronous event posting.

EOSC Format: DC.W $0120
DC.W data byte

This command specifies the EOS character for automatic EOS generation and detection.The character must be in the high byte

DAB Format: DC.W $0121
DC.W data byte

This code sends a data byte onto the GPIB. The Computer System must be previously configured as a talker. A timeout, if enabled, will be started.

DMALIM Format: DC.W $0122
DC.W data byte limit

This command sets the limit at which data transfers switch from direct I/O to a DMA operation. If timeouts are enabled while DMA is in progress, the timeout period is 100 ms multiplied by the record length.

6.8 EXAMPLES AND ILLUSTRATIONS

6.8.1 DIB EXAMPLE

This example specifies a device with the primary address $18 (via printable character 'X'). Extended addressing is not used, as indicated by the invalid extension address $1F (via printable character '?').

```
DC.B    '#GPBX?'        Primary address $18
DC.B    2               Bi-directional data transfer
DC.B    0               System stuff
DC.L    0,0             More stuff
DC.W    $0010           Device has service request capability
DC.L    0               User functions, none specified
DC.L    0               System stuff
```

6.8.2 FPKT EXAMPLE

This example executes a GET to two devices. In the test, one device was a programmable WAVETEK signal generator, previously configured via a WRITE command, and the other device was an HP programmable digital voltmeter, also previously configured via a WRITE command. The GET initiated waveform generation which was then sampled by the DVM.

Function packet start

```
DC.W    UNL             Unlisten
DC.W    OLA             DVM to listen
DC.W    'X?'            DVM address
DC.W    OLA             signal generator to listen
DC.W    'Y?'            signal generator address
DC.W    GET             Group Execute Trigger; start devices
DC.W    ENDLIST         end of function packet list
```

7.0 USER PARALLEL PORT DRIVER

7.1 OVERVIEW

The IBM Instruments Computer System can be configured with an interrupt-driven parallel port supporting a Centronics-compatible interface. This interface employs an Intel 8255 which has been programmed for strobed output operation. This means that data is transmitted in conjunction with handshake signals. The parallel port generates and accepts these handshake signals off Port C on the 8255. Port A is used for transmitting data, and Port B is used by the keyboard.

7.2 INITIALIZATION SEQUENCE

For a description of how the driver-defined control block fields are initialized during cold start, see Section 7.6. The driver configures the 8255 Port A for output and disables interrupts on <not> Output Buffer Full. The spare I/O lines on Port C are programmed for input to prevent sending the interface spurious signals. The interrupt timeout block is disabled until data is to be transmitted. The Physical Device Block informs the IOMGR that the driver supports (1) nonshareable device attribute, (2) byte I/O, and (3) surrogate task operation. Nonshareable means that only one task at a time can open the parallel port. Byte I/O is a faster way of transferring data without requiring excess IOMGR overhead. Surrogate task means that a separate instruction stream is started up for each I/O operation unblocking the user task.

7.3 INTERRUPT HANDLING

The parallel port driver receives an interrupt when 1) a character is moved out of the parallel port output buffer and 2) when no transfer acknowledge is received from the interface within 5 seconds. In this case the interrupt handler assumes that the device is 'NOT READY', issues an IOMGR DRVRDONE with that error status, and does an RTE.

7.4 DRIVER FUNCTIONS

1. OPEN

   The OPEN function sets the transfer mode to fixed or variable, according to the DIB, processes the function packet if there is one specified in the DIB, and enables interrupts on <not> Output Buffer Full.

2. WRITE

   SYSIO AWRITE or SWRITE transfer a line from the user buffer specified in the DTCB to the parallel port. The format of the DTCB is as follows:

DTCSTA	DS.B 1	User looks here for status on I/O operation.
DTCTBU	DS.B 1	User puts upper bound character for variable-length transfer here.
DTCTBL	DS.B 1	User puts lower bound character for variable-length transfer here.
DTCRS0	DS.B 1	This field is reserved.
DTCBFS	DS.L 1	User puts pointer to data buffer here.
DTCBFL	DS.W 1	User puts count of bytes in data buffer here.
DTCBPT	DS.W 1	User puts byte offset into data buffer here. This field is updated by the driver for every byte transmitted.
DTCREC	DS.L 1	This field is not used by the parallel port.

3. WRITE BYTE

   The byte write logic tests the output buffer full signal on Port C,

transfers the byte from D0.B if <not> OBF is true, and returns with a completion status. Otherwise it returns with a status indicating buffer full on write byte. (OB)

4. READ

   Until bidirectional mode is supported in the hardware, READ and READ BYTE are not supported by this driver, and return with error codes $62 and $66, respectively.

5. FUNCTION

   The only function packet supported is the standard function packet, which sets the transfer mode to fixed or variable.

6. CLOSE

   The close logic resets internal variables and disables interrupts on <not> Output Buffer Full.

7. CANCEL

   The cancel logic resets internal variables and disables interrupts on <not> Output Buffer Full.

8. The detach driver entry point detaches the driver timeout block, closes the interrupt descriptor block and detaches the physical device block whose address is passed in register A6 when the detach driver entry point is entered.

9. The function packet entry point verifies that the function to be performed is SET TRANSFER mode. If not it returns with a $0E error status. If it is, it sets the transfer mode to FIXED or VARIABLE, as specified in the function packet. If the transfer mode is not one of these, it returns with a $21 error status.

7.5 DEVICE INITIALIZATION BLOCK (DIB)

The user will be provided with a DIB generation macro to initialize a skeletal DIB for the parallel port. All he needs to provide is a

parameter specifying whether he wants fixed or variable-length transfers and an optional pointer to a function packet, if desired. The macro syntax is: 'PPUDIB dibname, <FIXED> or <VARIABLE>', fpktname, where 'dibname' is the address of the storage area to contain the DIB. Fpktname is the address of an area containing the SETTRANS function packet. The macro will initialize the DIB device name to '#PPU ', and set the DIBTRN field to indicate which type of transfer is desired and place the address of the function packet into the DIB.

7.6 DRIVER DEFINITIONS OF SYSTEM CONTROL BLOCK FIELDS

1. INTERRUPT DESCRIPTOR BLOCK (IDB)

    a. Hardware interrupt level - 6

    b. Controller ID - 'PIC6'

    c. Device ID - 'USER'

    d. Local interrupt level - 0

    e. Device mask for same hardware interrupt level - 0

2. PHYSICAL DEVICE BLOCK (PDB)

    a. PDB name - '#PPU ' (for Parallel Port User)

    b. PDB attributes - $B0 for nonshareable device, byte I/O, and asynchronous I/O

    c. PDB task control block - 1500 bytes combined for PCB and stack

7.7 ERROR CODES

Error codes are code returned (in hexadecimal) to register &7.B.

Code	Condition
0A	device not ready
0B	buffer full on write byte
0E	invalid code in function packet
0F	invalid transfer mode in DIB
21	invalid value for SETTRANS function packet
62	read not supported
66	read byte not supported
68	tstbyte not supported
6A	attach device not supported
6B	detach device not supported

8.0 DISKETTE DRIVER

8.1 INTRODUCTION

The floppy disk device driver provides both physical-sector and track access as well as a relative-sector access method within the IBM Instruments Computer System scheme of logical I/O.

8.2 DATA MANAGEMENT FUNCTIONS

1. Logical Volume-File Access

   a. Relative Sector Access Method

2. Physical Device Access

   a. Sector I/O

   b. Track formatting

8.3 FEATURES

1. 8" diskettes follow IBM standards for media compatibility, including volume-label recognition.

2. 5.25" diskettes use a volume label field for compatibility with 8" diskette volume recognition.

3. Four diskette drives are supported in any mixture of 5.25" and 8".

4. Designed around direct memory access and interrupts for true asynchronous data transfer.

5. Auto-Mount Feature: Every time a diskette is accessed, a hardware "door-open" status is tested. If the condition is set, the volume label sector of the inserted diskette is read, thus automatically reconfiguring the drive for density, sector size, number of sides per diskette, and volume identifier. The user need not indicate to the operating system that a diskette has been changed.

8.4 DEFINITIONS

1. LOGICAL SECTOR NUMBER

   This is a long word used by an application program to refer to a sector on a diskette without regard for physical characteristics of the diskette such as sectors per track, tracks per diskette, number of sides, and sector origin.

2. LOGICAL SECTOR NUMBER ALGORITHM

   The logical sector number is divided by the number of sectors per track, giving a physical track number (quotient), and a physical sector number (remainder+1). If the diskette is two sided then the physical track number (actually cylinder) is divided by two giving the track number (quotient) and diskette side number (remainder = side 0 or side 1).

8.5 USING RELATIVE SECTOR ACCESS METHOD

1. SYSIO OPEN,lun,DIBname,errorlabel is the call to open a file. End of file 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.

2. SYSIO SREAD,lun,DTCBname,errorlabel is the call to read a relative sector in the above opened file. No matter what sector in the file has been read or written last the requested relative sector within the present file will be read and placed in the users buffer. Currently the buffer should be word-aligned and 252 bytes (a sector's worth of data).

3. SYSIO SWRITE,lun,DTCBname,errorlabel is the call to write a relative sector to the above opened file. Whatever sector specified will be written whether within present file or beyond it. Currently the buffer should be word-aligned and 252 bytes (a sector's worth of data).

4. SYSIO CLOSE,lun,FKPTname,errorlabel is the call to close the above opened file. FKPT name can be one of three choices: 0 - no function packet, end of file information function packet, delete file function packet. See function packets EOFOFS and DELFILE.

   Note: The delete funtion packet will always be honored. If an EOF function packet is specified, and the file was not written to during this open, the function packet will be ignored.

8.6 DISK INFORMATION BLOCK (DIB)

The "DIB" is a control block to be used by the application program at OPEN time.

DIB FORMAT

```
DIB       DS.W  0
DIBVOL    DS.B  6   Volume or device name
DIBDTD    DS.B  1   Device transmit direction
DIBTRN    DS.B  1   Transfer mode. 0 = fixed 1 = variable
DIBFOR    DS.B  8   "FOR" task number
DIBOPT    DS.W  1   Device configuration options (see below)
DIBFCN    DS.L  1   Configuration function packet pointer
DIBBIO    DS.L  1   For byte I/O into manager
---------------------- DIB EXTENSION -----------------------
DIBNMS    DS.W  1   Return area (number of sectors in file)
DIBOFS    DS.W  1   Return area (EOF offset in last sector of old file)
DIBCAT    DS.B  8   File catalog (not currently used, set to zero)
DIBNAM    DS.B  8   File name
DIBEXT    DS.B  3   File name extension
DIBTYP    DS.B  1   File type (see below)
DIBACS    DS.W  1   File access attributes (see below)
DIBEND    DS.B  0   End of DIB
```

DIB EXTENSION

This area is used by the device driver as an extension of the DIB to hold file-related information. The fields are set by the caller prior to OPEN,

with the exception of DIBNMS and DIBOFS fields, which are information returned to the caller.

DIBOPT DEFINITION

This defines to the driver at OPEN time the kinds of things that the application expects to be doing. This will be the key as to which "access method" is going to be used (i.e. physical, logical, etc).

```
Bit  15:  0 = Ignore bits 0-14     1 = Use bits 0-14
Bit  14:  0 = Return error status  1 = Always return good status
                                        (see note 1)
Bit  13:  Reserved
Bit  12:  Reserved
Bit  11:  0 = Old file             1 = Create new file
Bit  10:  Reserved
Bit 9&8:  0 0 = Relative Sector Access
          0 1 = Contiguous Relative Sector Access (bit 8 on)
                (used only for DIR.DIR)
Bit 0-7:  Reserved
```

Note 1: Used only by system at coldstart.

DIBTYP DEFINITION

Byte value:

```
$00 = Binary file
 01 = Binary file with transfer address
 02 = Unused
 03 = Text file
 04 = Unused
 05 = Contiguous file (Used only by DIR.DIR)
```

DIBACS DEFINITION

```
Bit 3-7:  Unused
Bit   2:  0 = Writes are accepted, 1 = File is read-only
Bit   1:  0 = File can be renamed, 1 = File cannot be renamed
Bit   0:  0 = File can be deleted, 1 = File cannot be deleted
```

EXAMPLE: Setting value $07 indicates file is not deletable, not renamable, and read-only.

8.7 FUNCTION PACKETS

The general organization of floppy disk function packets is a variable-length list of CODE DATA words followed by a terminating code of zero to delimit end-of-list. The list can include stand-alone codes with no data, codes with a single word of data, codes with a long word of data, or multiple data words.

1. GETPDB - Retrieve Physical Device Block

   CODE is $0002. DATA is a long word containing a buffer address. This command transfers the current physical device block (PDB) to the buffer specified in the function packet.

2. WTRACK - Write Track

   CODE is $0003. DATA is a word containing a track number, followed by a long word containing a buffer address. This function is normally used for formatting a diskette. It causes a track of data to be sent to the disk controller with a write track-command. The track data must be contained in the buffer specified in the function packet.

3. WSECTOR - Write Sector

   CODE is $0004. DATA is a long word following the code word which contains the logical sector number, followed by a long word containing a buffer address. This function transfers a sector from diskette to the buffer specified in the function packet.

4. DELFILE - Delete File

   CODE IS $0005. There is no data value. The function packet need only be the word containing the function packet code.

5. SETORG - Set Sector Origin

   CODE is $0006. DATA is a word following the code word and must contain 0 (origin cylinder 0) or 1 (origin cylinder 1). This function can be used to change the sector origin, which is used in translating the logical sector number into physical information such as side number, track number, and sector number. It would be issued by programs attempting to access diskettes that are not recorded in the standard formats.

6. SETSPD - Set Sectors Per Diskette

   CODE is $0007. DATA is a long word following the code word and must contain the new number of sectors per diskette. This function can be used to set the number of sectors per diskette in the PDB. It would be issued by programs attempting to use diskettes that are not recorded in the standard formats.

7. SETBYT - Set Bytes per Sector

   CODE is $0008. DATA is a word following the code word and must contain the number of bytes per sector. This function can be used to set the sector size in the PDB. It would be issued by programs attempting to access diskettes that are not recorded in the standard formats.

8. SETSPT - Set Sectors Per Track

   CODE is $0009. DATA is a word following the code word and must contain the number of sectors per track. This function sets the number of sectors per track in the PDB. It would be issued by programs attempting to access diskettes that are not recorded in the standard formats.

9. SETSIDES - Set Number of Sides

   CODE is $000A. DATA is a word following the code word and must contain either 0 (1 side) or 1 (2 sides). This function sets the number of sides in the PDB. The variable is used in translating logical sector number into physical device information. It would be issued by programs attempting to access diskettes that are not recorded in the standard formats.

10. SETDEN - Set Density

    CODE is $000B. DATA is a word following the code word and must be either 0 (single density) or a 1 (double density). This function sets the density parameter in the PDB. It would be issued by programs attempting to access diskettes that are not recorded in the standard formats.

11. SETBPT - Set bytes per track

    CODE is $000C. DATA is a word following the code word. This function sets the number of bytes written by the WTRACK function when formatting a diskette.

12. SETTPD - Set tracks per disk

    CODE is $000D. DATA is word following the code word. This function sets the logical number of tracks per diskette used by the WTRACK function when formatting a diskette.

13. EOFOFS - Store EOF Offset in File Directory

    CODE is $000E. DATA is a word following the code word. A word of zeroes follows the data word to indicate the end of the function packet. This function accompanies SYSIO CLOSE to store the EOF offset in the file's last sector into the directory. It is recommended that the data value be the number of good data bytes in the file's last sector.

14. SETVOL - Set Volume Name

    Code is $000F. DATA is six bytes following the code word. This function writes the given volume identifier into the diskette volume sector, and also sets it in the alternate name field of the PDB.

15. REINIT - Reinitialize PDB

    CODE is $0010. There is no DATA value. This function reads the volume label of the diskette and updates the appropriate fields in the PDB. It also invalidates the sector cache and the free space sector value in FRETAB.

8.8 DEVICE CONTROL BLOCK EXTENSION (INTERNAL SYSTEM CONTROL BLOCK)

The device control block (DCB) should be used to hold information that pertains to a sequence of commands to the device driver through a logical unit number. This list includes all fields from the FCB used in earlier drivers. Some fields (relative sector number and number of sectors) have been made long words. A second level of qualification to the filename has been allowed for.

DCB EXTENSION FORMAT

```
--------- DCB EXTENSION AREA (18 BYTES) ----------------
DCBRES    DS.B  6   Reserved
DCBVOL    DS.B  6   Volume or Device Name
DCBOPT    DS.W  1   Copy of DIBOPT
```

```
DCBDCBX   DS.L  1   Address of DCB Extension (DCBX)
DCBEND    DS.B  0   End of DCB

--------- DCBX  Control Block -------------------------
DCBXBUF   DS.B  256     Access method buffer
DCBXCAT   DS.B    8     Catalog name (2nd level qualifier to filename)
DCBXNAM   DS.B    8     Filename
DCBXPER   DS.B    1     Period in filename
DCBXEXT   DS.B    3     Filename extension
DCBXDLM   DS.B    1     Filename delimiter
DCBXTYP   DS.B    1     File type (see DIBTYP)
DCBXACS   DS.B    1     File access attributes (see DIBACS)
DCBXCWD   DS.B    1     00 = Close do not write the directory  FF = write
DCBXFTS   DS.L    1     First sector in file
DCBXLTS   DS.L    1     Last sector in file
DCBXNMS   DS.W    1     Number of sectors in file
DCBXFWD   DS.L    1     Forward link (of chain file)
DCBXBAK   DS.L    1     Backward link (of chain file)
DCBXCSN   DS.L    1     Sector # of current sector in DCBXBUF
DCBXCLR   DS.L    1     Logical record (relative sector) of
                        current sector in DCBXBUF
DCBXPTR   DS.L    1     Pointer within DCBXBUF space
DCBXLFWD  DS.L    1     Forward link for old last sector
DCBXOFS   DS.W    1     EOF offset into last sector of file
DCBXRES   DS.B    1     Reserved for future
DCBXEND   DS.B    1     End of DCBX
```

8.9 PHYSICAL DEVICE BLOCK EXTENSION (INTERNAL SYSTEM CONTROL BLOCK)

```
PDBFR1    DS.L  1   Reserved
PDBSPD    DS.L  1   Sectors Per Diskette
PDBBYT    DS.W  1   Bytes Per Sector
PDBBPT    DS.W  1   Bytes per track
PDBSPT    DS.W  1   Sectors per track
PDBTPD    DS.W  1   Tracks per diskette
PDBTRK    DS.B  1   Current track
PDBSEL    DS.B  1   Drive-select byte
PDBFLG    DS.B  1   Diskette attribute flags
PDBLEV    DS.B  1   File structure format
PDBEND    DS.B  0   End of PDB
```

PDBFLG DEFINITION (This byte used to remember drive attributes)

```
Bit  7:  Unused
     6:  Unused
     5:  Configure drive --- 0 = no, 1 = yes (see note 1)
     4:  Seek track -------- 0 = verify, 1 = no verify
     3:  Discharged -------- 0 = no, 1 = yes
     2:  Sector origin ----- 0 = cylinder 0, 1 = cylinder 1
     1:  Number of sides --- 0 = single, 1 = double
     0:  Current side ------ 0 = side 0, 1 = side 1
```

Note 1: Each drive must be configured for either an 8" or a 5.25" diskette drive. This can only be determined when there is a drive ready status, which may or may not occur at coldstart. Once the drive type is determined, the configure bit is cleared.

PDBSEL DEFINITION (This byte is sent to hardware select register)

```
Bit  7 = Mini-floppy motor      0 = On,      1 = Off
     6 = Clock divider          0 = No div,  1 = Divide by 2
     5 = Density select         0 = Double,  1 = Single
     4 = Side select            0 = Side 0,  1 = Side 1
     3 = Drive #FD03            Only one of the
     2 = Drive #FD02            drive select bits
     1 = Drive #FD01            will be on in each
     0 = Drive #FD00            PDBSEL field
```

8.10 HARDWARE REGISTER SPECIFICATIONS

The following registers are specified by Western Digital:

FDSTCMND - Western Digital Status Command register
FDTRACK - WD Track register - holds track of currently selected drive
FDSECTOR - WD Sector register
FDDATA - WD Data register -used for seeks and read write

The following registers are implemented on the planar board specifically for the Computer System. A definition of each register is included here.

FDIRQDRQ - Computer System Floppy disk status register

```
Bit 7: 1 = Data request, transfer data byte, also wired to DMA.
    6: Unused, set to 0 by hardware
    5: Unused, set to 0 by hardware
```

4: Unused, set to 0 by hardware
3: 0 = Single Sided 8", 1 = Double Sided 8"
2: 0 = No disk change 5.25", 1 = disk change 5.25" (see notes 1,2)
1: 0 = 8" disk change, 1 = no disk change (see note 1)
0: 1 = Interrupt request (see note 3)

Note 1: These bits are latched in the hardware and can be read only when the specific drive has been selected. Deselecting the drive resets the bit.
Note 2: The 5.25" diskette change bit is also used as a ready not ready status indication for the 5.25" drive.
Note 3: This is also tied to an interrupt input line which will allow the floppy device driver to WAIT on the DMA-oriented I O without a spin loop.

FDSELECT - Floppy Disk command register

Bit 7: 0 = 5.25" motor on, 1 = 5.25" motor off (see note 4)
6: 0 = No clock divider, 1 = divide clock by 2. Used with mini-floppy
5: 0 = Double density, 1 = Single density
4: 0 = Side 0, 1 = Side 1
3: 1 = Select drive 3
2: 1 = Select drive 2
1: 1 = Select drive 1
0: 1 = Select drive 0

Note 4: All 5.25" drive motors turn on and off with the selection of this bit.

8.11 DISKETTE FORMATS TO BE SUPPORTED

- 5 1/4-inch Double Sided/Double Density
  40 Tracks per side (all available for data).
  16 Sectors per track.
  256 Bytes per sector.
  Total of 1280 sectors available, or 327,680 bytes.
  User data storage is 1257 sectors at 252 bytes/sector, or 316,764 bytes.

- 5 1/4-inch Single Sided/Double Density
  40 Tracks per side (all available for data).
  16 Sectors per track.
  256 Bytes per sector.

Total of 640 sectors available, or 163,840 bytes.
User data storage is 617 sectors at 252 bytes/sector,
or 155,484 bytes.

- 8-inch" Double Sided/Double Density IBM 2D Format
  Cylinder 0 reserved as index cylinder per IBM Diskette Standards.
  Cylinders 1 through 74 available for data.
    26 Sectors per track.
    256 Bytes per sector.
    Total of 3848 Sectors available, or 985,088 bytes.
    User data storage is 3825 sectors at 252 bytes/sector,
    or 963,900 bytes.
  Cylinders 75 and 76 reserved.

- 8-inch Single Sided/Single Density IBM Diskette 1
  Cylinder 0 reserved as index cylinder per IBM Diskette Standards.
  Cylinder 1 through 74 available for data.
    15 Sectors per track.
    256 Bytes per sector.
    Total of 1110 Sectors available, or 284,160 bytes.
    User data storage is 1087 sectors at 252 bytes/sector
    or 273,924 bytes.
  Tracks 75 and 76 reserved.

- 8-inch Double Sided/Single Density IBM Diskette 2
  Cylinder 0 reserved as index cylinder per IBM Diskette Standards.
  Cylinder 1 through 74 available for data.
    15 Sectors per track.
    256 Bytes per sector.
    Total of 2220 Sectors available, or 568,320 bytes.
    User data storage is 2197 sectors at 252 bytes/sector,
    or 553,644 bytes.
  Tracks 75 and 76 reserved.

VOLUME LABELS

- 5 1/4-inch Diskette have volume label 5-10information in logical sector 0 in the data area of the diskette. Bytes 5-10 are the volume identifier, byte 11 is the surface indicator ("M" = 2D, "N" = 1D), and byte 12 is the sector size ("1" = 256). There is no index cylinder defined.

- 8-inch diskettes have volume label information on the index cylinder recorded in ASCII per standard. Remaining areas of the index cylinder are initialized per IBM Diskette standard.

8.12 ERROR CODES

1. See Exception Codes (Section 1.2.3) for common error codes.

2. Disk Driver

    $0010 Track seek or restore error
    $0011 Logical sector or track number too big
    $0012 Volume name changed
    $0013 No storage for control blocks
    $0014 DMA timeout, no interrupt received (sector not found)
    $0015 Illegal file access method
    $0016 I/O buffer boundary error
    $0017 Diskette format not recognized
    $0018 Diskette write protected
    $0019 Sector buffer too small
    $001A Invalid data transfer direction
    $001B CRC error

3. Functions

    $0023 Write Track error
    $0024 Write Sector error
    $002F Set Volume Identifier error
    $0030 Reinitialize PDB error

4. Disk Access Methods

    $0081 Delete not supported for file type
    $0082 File not found
    $0083 Directory full
    $0084 Invalid filename
    $0085 Read beyond end of file
    $0086 Buffer size incorrect
    $0088 Disk full
    $0089 Writing to read-only file
    $008A Wrong file type for access
    $008B Invalid sector specified
    $008C Disk file protected from deletion

9.0 SENSOR I/O DRIVER

9.1 INTRODUCTION

This software package contains drivers for interfacing application programs to hardware devices on the Sensor I/O board. This software package contains device drivers for interfacing application programs to hardware devices on the Sensor I/O board. The devices supported by this package are:

1) 4 - A/D Converter Channels
2) 8 - Switch Input Channels
3) 8 - LED Output Channels
4) 2 - Timer/Counters
5) 32- bit Parallel I/O

The package will initially be configured for a single Sensor I/O board. However, the drivers can be easily configured to multiple board systems.

The driver package is divided into 5 independent drivers, one for each of the basic types on the Sensor I/O board. The drivers interface to the operating system via the Interrupt Manager, the Timer Manager, and the I/O Manager.

Device protection is provided by the requirement that the device be "opened" by any process that accesses the device. The drivers are written to allow more than one process to "open" the same device only if the "shared" option is used during the "open." If the shared option is not used, then only one open is allowed for the device at a time. A "close" command is used to release a process from the device.

9.2 A/D CONVERTER CHANNELS

The A/D converter device driver allows an application program to collect data from A/D converter channels. Each channel can be configured to desired characteristics at open time, or reconfigured with a function packet.

Auto-Range/Fixed-Gain. In the autorange mode, the converter self-adjusts its gain for an optimum scale to make the reading. In the fixed-gain mode, the converter gain can be fixed at 1, 8, 64, or 256.

Attenuation. With this parameter the full scale voltage can be changed from ±1 V to ± 10 V. (A hardware wiring jumper must be installed to make this attenuation function operative).

Sample Rate. The hardware A/D conversion rate has a maximum sample rate of 30 samples per second. The rate is specified using a period in microseconds.

Averaging. If a sample rate of less than 30 samples per second is wanted, then the averaging mode may be selected. If averaging is selected, then the data returned to the application program is the average of the data converted during the sample period.

Shared access. If shared access to an A/D channel is desired, then this option can be selected. This allows more than one process to have the channel open at the same time. The collection request of a process will be completely serviced before another request from this or another process is started. This option is specified only during an open operation.

Alternate Channels. The alternate channels option allows the collection of data from more than one physical A/D converter. The option specifies which physical A/D converters to read in each sample period. Note that all channels must be on the same board.

9.2.1 A/D TRANSFER REQUESTS

The A/D device driver supports synchronous and asynchronous read requests. The request specifies the buffer address and buffer length. The buffer length is given in bytes. The buffer address must be word aligned. The data is returned in a scaled 32-bit integer format. The relation between returned value and actual voltage is:

actual voltage (V) = returned value/1048576

The specified channels are sampled and stored in the buffer in sequential order.

If an overrange condition is detected, the result returned is the largest signed number.

9.2.2 A/D DEVICE INITIALIZATION BLOCK

The A/D DIB is designated as ADIB. The format for the ADIB is:

```
ADIB    DS.W    0
DIBVOL  DS.B    6               Device name
DIBDTD  DS.B    1               Device transmit direction
DIBFOR  DS.B    1               "FOR" task Number
DIBOPT  DS.W    1               Device configuration options
DIBFCN  DS.L    1               Function packet address
DIBBIO  DS.L    1               For byte I/O
```

9.2.3 DEVICE NAMES

The A/D converter channels are assigned the following names:

#ADC00	A/D converter 1
#ADC01	A/D converter 2
#ADC02	A/D converter 3
#ADC03	A/D converter 4

If a second sensor I/O board is installed, the converters would be named #ADC10 etc.

The DIBDTD field is 0 for output, 1 for input, and 2 for bidirectional I/O.

The DIBOPT field has the following bits defined.

bit 8	Shared-access option
bit 9	Attenuation on
bits 10-15	Not assigned

The DIBFCN field specifies the address of a function packet if desired.

9.2.4 A/D FUNCTION PACKETS

The A/D function packet is used to set or change one or more of the operating characteristics of an A/D channel. The format of the packet is:

```
FCNPKT          Function Name
                Function Value
                Function Name
                Function Value
                    .
                    .
                    .
                Zero word as Function name for end of list
```

Valid function name - Function value combinations are:

Function Name	Function Value	Parameter Value
ADGAIN =	1	0 = AUTO, 1, 8, 32, 128
SAMPRATE =	2	Number of micro-seconds (32 bits)
AVERAGE =	3	OFF = 0, ON = 1
ALTCHN =	4	Value specified as a list of words. Each word specifies a channel. Range of channels is 0-4 The list is ended with a value of -1

9.3 SWITCH INPUTS

The switch input device driver allows an application program to read the state of one of the switch inputs. Each individual switch is viewed as a separate device. A switch value when read always goes to the off state. There is one option that can be specified when a switch is opened.

Shared Access. If shared access to a switch input is desired, then this option can be selected. This will allow more than one process to have a switch open at the same time.

Switch Transfer Requests. The switch input device driver supports byte, synchronous, and asynchronous read requests. In each case, only one value is returned.

In byte read requests, the current state of the selected switch is returned as a value of 0 for open, and 1 for closed.

For synchronous and asynchronous read requests the driver will wait for the next time that the switch is pressed. At that time it puts the value 1 in the buffer and the request is terminated.

9.3.1 SWITCH INPUT DEVICE INITIALIZATION BLOCK (SWIB)

The switch input DIB is designated as SWIB. The format of the SWIB is :

SWIB	DS.W	0	
DIBVOL	DS.B	6	Device name
DIBDTD	DS.B	1	Device transmit direction
DIBFOR	DS.B	1	"FOR" task number
DIBOPT	DS.W	1	Device configuration options
DIBFCN	DS.L	1	Function packet pointer
DIBBIO	DS.L	1	For byte I/O

9.3.2 DEVICE NAMES

The switch inputs are assigned the following names:

#SW00	Switch input 0
#SW01	Switch input 1
#SW02	Switch input 2
.	.
.	.
.	.
#SW07	Switch input 7

Switch inputs in multiple board systems are similarly named with the first numerical digit reflecting the board number.

The only valid value for the DIBDTD is 1, designating input.

There is one bit defined in the DIBOPT field:

bit 8	Shared-access option

The DIBFCN field is not used.

9.4 LED OUTPUTS

The LED output device driver allows applicatoion programs to output to a selected LED. Each individual LED is considered a separate device. There is one option that can be specified when an LED channel is opened.

9.4.1 SHARED ACCESS

If shared access to an individual LED channel is desired, each process must use the shared access option when they open the channel.

9.4.2 LED TRANSFER REQUESTS

The LED device driver uses only the "Byte Output" form of transfer. All other types of transfer requests will be rejected with an error. A zero byte output to the LED channel will turn the LED off. A non-zero value will turn the LED on.

9.4.3 LED OUTPUTS DEVICE INITIALIZATION BLOCK (LEDIB)

The LED output DIB is designated as LEDIB. The format of the LEDIB is:

```
LEDIB           DS.W  0
DIBVOL          DS.B  6       Device name
DIBDID          DS.B  1       Device transmit direction
DIBFOR          DS.B  1       "For" task number
DIBOPT          DS.W  1       Device configuration options
DIBFCN          DS.L  1       Function packet address
DIBBIO          DS.L  1       For byte I/O
```

9.4.4 DEVICE NAMES

The LED outputs are assigned the following names:

```
    #LED00          LED     Output 0
    #LED01          LED     Output 1
    #LED01          LED     Output 2
    #LED07          LED     Output 7
```

LED outputs in multiple board systems are named similarly with the first numerical digit reflecting the board number.

The only valid value for DIBDTD is 0 for output.
One bit is defined in the DIBOPT field.

bit 8 Shared-access option

The DIBFCN field is not used.

9.5 TIMER/COUNTERS

The timer/counter device driver allows an application programs to interface to two of the 16-bit counters of the 8253 CTC. The two counters can be opened as two 16 bit counters or as one 32-bit counter. Options available during open are:

9.5.1 32-BIT COUNTER

If the 32-bit counter option is used during an open, the 16 bit counter specified during the open is the least significant 16 bits, and the other counter is the most significant 16 bits. Requests to this device act on both counters when this option is selected.

9.5.2 SHARED ACCESS

If shared access is desired to a 16- or 32-bit counter, then this option must be used by all processes when they open the counter. Note, if the 32-bit counter option is used during an open, then all shared opens must also use that option.

9.5.3 SUPPORTED I/O FUNCTIONS

I. Function Request

The function request allows the counter/timer to be configured to desired characteristics. The parameters that can be specified are:

Initial Count. The initial count is a 32-bit number. Only 16 bits are used if in the 16 bit counter mode.

Counter Mode. This specifies the mode of the counter or counters (if in the 32-bit mode). It is always specified by a word. The low byte specifies the mode for the least significant 16 bits of the counter. The high byte specifies the most significant bits if in the 32-bit mode.

Start. If the function request is to load the counter and start it at this time, the start option can be specified. This is useful for waveform generation and event counting.

II. Delay.

The synchronous and asynchronous write requests are used to implement a delay function. When the request is issued, the counter is loaded with the current mode and count. The write is then finished when the interrupt is received.

III. Read Counter.

The synchronous and asynchronous read requests are used to read the current count value in the counter. A 32-bit result is returned if the 32-bit counter mode is selected. If the count has overflowed since the last access, a status indicating this fact is also returned.

9.5.4 TIMER/COUNTER DEVICE INITIALIZATION BLOCK (CTIB)

The Timer DIB is designated CTIB. The format for the CTIB is:

```
CTIB            DS.W  0
DIBVOL          DS.B  6       Device name
DIBDTD          DS.B  1       Device transmit direction
DIBFOR          DS.B  1       "FOR" task number
DIBOPT          DS.B  1       Device configuration options
DIBFCN          DS.L  1       Function packet address
DIBBIO          DS.L  1       For byte I/O
```

9.5.5 DEVICE NAMES

The timer/counters are assigned the following names.

#CT000	Counter #1 of 8253
#CTC01	Counter #2 of 8253

If additional sensor I/O boards are installed in the system, then the names of those counters have the first numerical digit as the board number.

The DIBDTD has the value 0 for output, 1 for input, and 2 for bidirectional I/O.

The DIBOPT field has the following bits defined.

bit 8	Shared access option
9	32-bit counter option

The DIBFCN specifies the address of an optional function packet.

9.5.6 TIMER/COUNTER FUNCTION PACKET

The timer/counter function packetis used to set or change one or more of the operating characteristics of the timer/counter channel. The format of the packet is:

```
FCNPKT          parameter name
                parameter value
                parameter name
                      .
                      .
                      .
```

zero for parameter name to end list

Valid parameter name - parameter value combinations are:

Parameter Name	Parameter Value
Initial Count = 1	32-bit value
Counter Mode = 2	0, 1, 2, 3, 4, 5 See 8253 documentation
Start = 3	1 = load, 0 = no load

9.5.7 PARALLEL I/O CHANNELS

The parallel I/O device driver enables application programs to interface to the 32 bits of parallel I/O. The channels can be opened in units of bits, bytes, or words. The options that can be specified during open are:

9.5.8 SIZE

The size of the channel to be opened can be a bit, byte, or words. If a bit is desired, then the bit number must also be specified.

9.5.9 STROBE

If this option is selected, the channel operates in the strobed mode of operation. This mode is not legal for bit I/O.

9.5.10 SHARED ACCESS

If shared access is desired with a parallel I/O channel, the shared-access option must be used by all processes when they open that channel.

9.5.11 TRANSFER DIRECTION

A parallel I/O channel can be opened for input, output, or bidirectional input/output. The bidirectional input/output mode is legal only on the port A bytes of the 8255's in byte or word mode.

9.5.12 RESOURCE ALLOCATION

The 32 bits of parallel I/O can be viewed as analgous to 32 "sectors" on a disk. Opening a byte of parallel I/O would be equivalent to opening a file containing 8 sectors. Files can be of word, byte, or bit size. The following rules determine whether a file can be opened or not.

1. The file cannot overlap an already open file.
   1a) An exception to (1) is that shared access to the same in 10 file is allowed if all opens to that file are for shared access.

2. Strobed mode is not legal on bit files.

3. Bidirectional strobed mode is available only on byte or word files on the 8255 port A's.

9.5.13 TRANSFERS

The parallel I/O device driver supports byte and buffered requests. Byte requests are used in the nonstrobed mode for bit and byte "files". On input, bit files return a "zero" byte or a "one" byte reflecting the state of the bit. On output, a zero byte results in the bit being cleared. A non-zero value results in the bit being set.

Buffered I/O is used with channels in the strobed mode of operation. The request specifies the buffer start address and the buffer length in bytes. In the bidirectional strobed mode, separate read and write requests can be

serviced simultaneously.

9.5.14 PARALLEL I/O DEVICE INITIALIZATION BLOCK

The parallel I/O DIB is designated as PPIB. The format for the PPIB is:

```
PPIB        DS.W    0
DIBVOL      DS.B    6       Device name
DIBDTD      DS.B    1       Device transmit direction
DIBFOR      DS.B    1       "FOR" task number
DIBOPT      DS.W    1       Device configuration options
DIBFCN      DS.L    1       Function packet address
DIBBIO      DS.L    1       For byte I/O
```

9.5.15 DEVICE NAMES

The parallel I/O channels of a single sensor I/O board are divided into 4 physical devices.

```
#PPA00      Port A, high byte
#PPB00      Port B, high byte
#PPA01      Port A, low byte
#PPB01      Port B, low byte


bit 8       Shared-access option
  9-10      Size:  0 = Byte, 1 = bit, 2 = word, 3 = undefined
```

11-13	Bit number for bit I/O
14	Non-strobed I/O option
15	Not assigned

The DIBFCN field is not used.

Note: If bits 8-15 of DIBOPT are zero, then the port is opened as a strobed byte.

9.6 ERROR CODES

The following error codes have been defined and may be returned by this driver package.

Error Code	Meaning
$10	Device Locked
$11	Device Already open
$12	A/D overrange
$13	CTC counter overflow
$14	Illegal open mode

10.0 CRT GRAPHICS DRIVER

10.1 INTRODUCTION

The CRT graphics driver is intended to provide the user with a means of performing graphics primitive operations in 2 dimensions via the I/O manager. Access to the graphics routines is primarily through the I/O manager function call. The user gains access to the CRT by opening a graphics window, which is treated as an independent logical unit. A graphics window is similar to an alpha window in that it is a rectangular area of the screen in which the graphics primitives will be displayed. Once a window has been opened the user is free to perform any one of the supported functions, including:

SETCOP	Move the current operating point
SETVEC	Draw a line
SETCHR	Draw a character string
BLKFIL	Fill a rectangular area
CLRWIN	Clear the window
FRAME	Draw a box around the window
SETMAP	Define a 2-D mapping

When the user has completed his display, the window may be closed by issuing a "close" call to the I/O manager against the logical unit number of the window.

10.1.1 COORDINATE SYSTEMS

There are three coordinate systems that the graphics driver interacts with.

User coordinates - are the coordinates in which the user specifies information to the graphics driver.

Device coordinates - are the integer coordinates from 0..767 in the X dimension and 0-479 in the Y dimension that correspond to pixel

location on the screen relative to the lower left corner.

Memory coordinates - refer to the final address and bit number in either of two pages of graphics memory.

10.1.2 THE GRAPHICS WINDOW

The graphics window is a rectangular region in either of the two pages of graphics memory coresponding to an area on the CRT on which graphics primitives are to be displayed. The window is defined in terms of device coordinates at open time.

Note: primitives occuring outside the window boundaries will not be displayed.

10.1.3 COORDINATE MAPPING

Coordinates are supplied to the graphics driver in user coordinates which must be mapped into device coordinates. Three integer 2-D mappings are supported at present.

1	User coordinates =	Device coordinates
2	User coordinates =	Device coordinates - Window offset vector
3	User coordinates =	Device coordinates with scale and offset

In the third mapping the user must supply the coordinates that correspond to the graphics window.

10.1.4 PIXEL CONTROL

Graphics primitives will cause pixels (bits in graphics memory) to be filled according to the graphics control word and the fill word. The fill word is a means by which the user can generate fill patterns and dotted lines etc. As each pixel is encountered the fill word is rotated one bit to the right. The least significant bit determines whether any action will be performed on the pixel. If the bit is set, the pixel will be set, reset, or exclusive-ORed according to the pixel control bits in the

graphics control word. If the least significant bit of the fill word is not set then no action will take place.

10.1.5 CURRENT OPERATION POINT (COP)

The current operation point (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 a SETVEC, SETCHR, and SETCOP, while others simply use the COP as a reference point.

10.2 GRAPHICS DRIVER CONTROL BLOCKS

10.2.1 THE GRAPHICS DEVICE INFORMATION BLOCK (DIB)

The particular information necessary to complete the graphics device information block is as follows:

```
Device name (DIBVOL) =          '#GR    '
Transmit direction (DIBDTD) = $00
Options (DIBOPT) =              $XX       not used
Initial function packet         $00000000 no packet or
          (DIBFCN):             $NNNNNNNN address of packet
```

The open option word is a bit significant word that can be used to select graphics control parameters at open time

```
Bits 1,0   Pixel control options

           00 = set pixels
           01 = reset pixels
           10 = exclusive or pixels
           11 = not used-error

Bit 2      coordinate interpretation select

           0 = absolute coordinates
           1 = relative coordinates

Bits 4,3   map mode select

           00 = screen coordinates-mode 0
```

01 = screen coordinates+offset-mode 1
10 = scale+offset-mode 2
11 = not used-error

Bits 6,5 character orientation

00 = left to right
01 = bottom to top
10 = right to left
11 = top to bottom

Bit 7 page select

0 = page 0
1 = page 1

Bits 14-8 ignored

Bit 15 option word enable

0 = ignore bits 14-0
1 = decode option bits

For example: The following is a valid DIB for the graphics driver.

```
GRDIB     EQU       *
          DC.B      '#GR  '              DIBVOL
          DC.B      0                    DIBVOL
          DC.B      0                    DIBDTD
          DC.B      0                    DIBRS1
          DC.W      0                    DIBOPT
          DC.L      GRFPKT00             DIBFCN
          DS.L      0                    DIBBIO

GRFPKT00  EQU       *          GRAPHICS FUNCTION PACKET
*         ETC...                (SEE BELOW)
```

10.2.2 THE GRAPHICS FUNCTION PACKET

The graphics function packet is the principal means by which information is relayed to the driver. The function packet consists of one or more individual function information cells. Each cell specifies a complete function call to one of the functions described below. The cell begins with a one word number corresponding to the desired function followed by the required information fields. All fields in the function cell must be specified as shown in the function descriptions below. Completely missing

fields will cause termination of function packet processing and generate an error message. Function number '0' terminates the packet.

Note: All function packets are an integral number of words in length

10.3 GRAPHICS FUNCTIONS

10.3.1 END OF PACKET -- ENDFPKT

ENDLIST terminates function packet processing.

MNEMONIC: ENDLIST	FUNCTION NUMBER: 0
ARGUMENTS:	NONE

10.3.2 SET WINDOW BOUNDARIES -- SETWIN

SETWIN redefines the window boundaries in device coordinates. At open time the window is given the default values of 0,0,767,479, or a full screen window. SETWIN is used to modify these values.
Note that the following must be true:

```
0 <= XD1 < XD2 <= 767
0 <= YD1 < YD2 <= 479
```

MNEMONIC: SETWIN FUNCTION NUMBER: 1

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
XD1	Word	0	Left window boundary	0..767
YD1	Word	0	Bottom window boundary	0..479
XD2	Word	767	Right window boundary	0..767
XD2	Word	479	Top window boundary	0..479

10.3.3 INQUIRE WINDOW BOUNDARIES -- INQWIN

INQWIN returns the values of the window boundaries in device coordinate. The arguments are identical to SETWIN and are returned in the function packet.

MNEMONIC: INQWIN			FUNCTION NUMBER: 2	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
XD1	Word	0	Left window boundary	0..767
YD1	Word	0	Bottom window boundary	0..479
XD2	Word	767	Right window boundary	0..767
YD2	Word	479	Top window boundary	0..479

10.3.4 SET GRAPHICS PAGE-SETPAGE

SETPAGE selects the page of graphics refresh RAM to be written to.

MNEMONIC: SETPAGE			FUNCTION NUMBER: 3	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Graphics page	0,1

10.3.5 INQUIRE GRAPHICS PAGE -- INQPAGE

INQPAGE returns the graphics page number in the function packet.

MNEMONIC: INQPAGE			FUNCTION NUMBER: 4	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Graphics page	0,1

10.3.6 SET PIXEL CONTROL MODE -- SETPXCTL

SETPXCTL defines the logic by which the graphics primitives will set/reset pixels in graphics memory. Three modes are available: set, reset and exclusive OR.

MNEMONIC: SETPXCTL FUNCTION NUMBER: 5

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	1	Pixel control mode	0-2
			Reset pixels	0
			Set pixels	1
			XOR pixels	2

10.3.7 INQUIRE PIXEL CONTROL MODE -- INQPXCTL

INQPXCTL returns the current pixel control mode in the function packet.

MNEMONIC: INQPXCTL FUNCTION NUMBER: 6

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	1	Pixel control mode	0-2

10.3.8 SET MAPPING MODE -- SETMPMOD

SETMPMOD selects the desired coordinate mapping mode. Three modes are supported:

Mode 0 Device coordinates = user coordinates
(1-to-1 mapping)

IE XD = XW
YD = YW

Mode 1 Device coordinates = user coordinates with window offset vector

IE XD = XW-XD1
YD = YW-YD1

Mode 2 Device coordinates = user coordinates with window offset and scale

IE XD = (XW-XW1)*(XD2-XD1)/(XW2-XW1)+XD1
YD = (YW-YW1)*(YD2-YD1)/(YW2-YW1)+YD1

Where: XD1, YD1, XD2, YD2 are the device-coordinate window boundaries

XW1, YW1, XW2, YW2 are the user-coordinate window boundaries

(XD,YD) defines a point in device coordinates

(XW,YW) defines a point in user coordinates

Nondefault values for the device coordinate window boundaries may be supplied with SETWIN
device-window default boundaries = (0,0,767,479)

Nondefault values for the user-coordinate window boundaries may be supplied with SETMAP
user-window default boundaries = (0,0,767,479)

MNEMONIC: SETMPMOD FUNCTION NUMBER: 7

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Coordinate mapping mode	0-2
			Mode 0	0
			Mode 1	1
			Mode 2	2

10.3.9 INQUIRE COORDINATE MAPPING MODE -- INQMPMOD

INQMPMOD returns the current coordinate mapping mode in the function packet.

MNEMONIC: INQMPMOD FUNCTION NUMBER: 8

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Coordinate mapping mode	0-2

10.3.10 SET COORDINATE INTERPRETATION MODE -- SETCORIM

SETCORIM allows the user to select either absolute or relative coordinate interpretation. Absolute coordinates are interpreted literally. Relative coordinates are interpreted as offsets relative to the current operating point.

MENMONIC: SETCORIM FUNCTION NUMBER: 9

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Coordinate interpretation mode	0/1
			Absolute	0
			Relative	1

10.3.11 INQUIRE COORDINATE INTERPRETATION MODE -- INQCORIM

INQCORIM returns the current coordinate interpretation mode in the function packet.

MNEMONIC: INQCORIM FUNCTION NUMBER: 10

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Coordinate interpretation mode	0/1

10.3.12 SET CHARACTER ORIENTATION MODE -- SETCHOR

SETCHOR allows the user to select the orientation of text strings on the page. Four modes are available: left to right, bottom to top, right to left and top to bottom. In each of these modes the characters are rotated to match the text orientation.

MNEMONIC: SETCHOR FUNCTION NUMBER: 11

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Character orientation	0-3
			Left to right	0
			Bottom to top	1
			Right to left	2
			Top to bottom	3

10.3.13 INQUIRE CHARACTER ORIENTATION MODE -- INQCHOR

INQCHOR returns the current value of the chracter orientation mode in the function packet.

MNEMONIC: INQCHOR FUNCTION NUMBER: 12

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	0	Character orientation	0-3

10.3.14 SET CHARACTER MAGNIFICATION-SETMAG

SETMAG allows the user to select an integer magnification value for character field dimensions times the character magnification.

MNEMONIC: SETMAG			FUNCTION NUMBER: 13	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	1	Character magnification	>=1

10.3.15 INQUIRE CHARACTER MAGNIFICATION-INQMAG

INQMAG returns the character magnification factor in the function packet.

MNEMONIC: INQMAG			FUNCTION NUMBER: 14	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	1	Character magnification	>=1

10.3.16 SET CHARACTER FONT -- SETFONT

SETFONT allows the user to select a non-default font. A pointer to a font control block is passed with the SETFONT function call. The font control block consists of the following:

```
*         FONT CONTROL BLOCK

FNTCB     DC.W      FNTCOL      NUMBER OF COLUMNS IN FONT (1-32)
          DC.W      FNTLIN      NUMBER OF LINES IN FONT (1-32)
          DC.L      FNTPTR      POINTER TO FONT TABLE
```

The purpose of the font control block is to convey physical information about the font to the driver. The font is assumed to be organized as follows:

1. Characters are arranged in the font table in numerical order starting at 0-256.

2. Characters are entered in the font table line at a time from the top down.

3. All lines must be entered.

4. Each line entry is a byte, word or long word -- whichever is the smallest (i.e., a 9 column character would use a word length line).

MNEMONIC: SETFONT			FUNCTION NUMBER: 15	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	L.Word	INTERNAL	Pointer to control block	0-$FFFFFF

10.3.17 INQUIRE CHARACTER FONT DIMENSIONS-INQFONT

INQFONT returns the physical font dimensions in the function packet.

MNEMONIC: INQFONT			FUNCTION NUMBER: 16	
ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	8	Font width (columns)	1-32
2	Word	14	Font height (lines)	1-32

10.3.18 SET CHARACTER FIELD -- SETCHFLD

SETCHFLD selects the field in which the physical font is to be set. The font field is a means by which the font can be extended or clipped by a few columns or lines. For instance, the default font is 8 by 14 but is displayed in a 9 by 16 field. A separate field definition allows the user to minimize the size of the font table by eliminating the need for borders. The character is positioned in the upper right corner of the

font field. If the font field exceeds the dimensions of the font, the gap will be filled with null bits.

MNEMONIC: SETCHFLD FUNCTION NUMBER: 17

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	9	Character field width	1-32
2	Word	16	Character field height	1-32

10.3.19 INQUIRE CHARACTER FIELD -- INQCHFLD

INQCHFLD returns the character field dimensions in the function packet.

MNEMONIC: INQCHFLD FUNCTION NUMBER: 20

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
1	Word	9	Character field width	1-32
2	Word	16	Character field width	1-32

10.3.20 SET FILL WORD -- SETFILWD

SETFILWD sets the value of the graphics fill word. The graphics fill word is used by functions such as SETVEC, BLKFIL, CIRCLE etc. to generate patterns. The fill word acts as a mask through which pixels are acted on according to the pixel control bits in the graphics control word (see above).

MNEMONIC:SETFILWD FUNCTION NUMBER: 19

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
FILWD	Word	$FFFF	Fill word	0000..$FFFF

10.3.21 INQUIRE FILL WORD -- INQFILWD

INQUIRE fill word returns the value of the fill word in the function packet.

MNEMONIC: INQFILWD FUNCTION NUMBER: 20

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
FILWD	Word	$FFFF	Fill Word	000.$FFFF

10.3.22 SET MAPPING COORDINATES -- SETMAP

SETMAP sets the user coordinate window boundaries necessary to perform mapping option 2 described in SETMODWD.

MNEMONIC: SETMAP FUNCTION NUMBER: 21

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
XW1	Word	0	Left window boundary (user coordinates)	+/-32767
YW1	Word	0	Bottom window boundary (user coordinates)	+/-32767
XW2	Word	767	Right window boundary (user coordinates)	+/-32767
YW2	Word	479	Top window boundary (user coordinates)	+/-32767

10.3.23 INQUIRE MAPPING COORDINATES -- INQMAP

INQMAP returns the user coordinate map in the function packet.

MNEMONIC: INQMAP FUNCTION NUMBER: 22

ARGUMENT	LENGTH	DEFAULT	DEFINITION	RANGE
XW1	Word	0	Left window boundary (user coordinates)	+/-32767
YW1	Word	0	Bottom window boundry User coordinates)	+/-32767
XW2	Word	767	Right window boundary (user coordinates)	+/-32767
YW2	Word	479	Top window boundry (user coordinates)	+/-32767

10.3.24 SET CURRENT OPERATING POINT -- SETCOP

SETCOP defines the current operating point. The current operating point is the last point referenced during a primitive operation. The current operating point may be considered analogous to the position of the pen carriage on a plotter.

Note 1: The COP is initialized to 0.0
Note 2: The COP may exceed the window boundaries.

MNEMONIC: SETCOP FUNCTION NUMBER: 23

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
XCOPW	Word	None	COP X value	User coordinates
YCOPW	Word	None	COP Y value	User coordinates

10.3.25 INQUIRE CURRENT OPERATING POINT -- INQCOP

INQCOP returns the current operating point in the function packet.

MNEMONIC: INQCOP FUNCTION NUMBER: 24

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
XCOPW	Word	None	COP X value	User Coordinates

YCOPW	Word	None	COP Y Value	User Coordinates

10.3.26 SET A PIXEL - SETPIXEL

10.3.27 INQUIRE A PIXEL -- INQPIXEL

10.3.28 DRAW A VECTOR -- SETVEC

SETVEC draws a vector from the current operating point to a specified point. 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.

MNEMONIC: SETVEC FUNCTION NUMBER: 27

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
XW	Word	None	Vector end point	User coordinates
YW	Word	None	Vector end point	User coordinates

Note: Following SETVEC the COP is repositioned to XW,YW

10.3.29 FILL A RECTANGULAR AREA -- BLKFIL

BLKFIL fills a rectangular area defined by the current operating point and a point in user coordinates. If the fill area exceeds the window boundary, it will be clipped and only the region within the window will be filled.

MNEMONIC: BLKFIL FUNCTION NUMBER: 28

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
XW	Word	None	Fill X Boundary	User coordinates
YW	Word	None	Fill Y Boundary	User coordinates

10.3.30 DRAW A CHARACTER STRING -- SETCHR

SETCHR draws a character string from the current operating point in a direction specified in the graphics control word. If the string exceeds the window boundaries it will be clipped. The string is defined as a one word byte count followed by that number of bytes.

MNEMONIC: SETCHR		FUNCTION NUMBER: 29	
ARGUMENT LENGTH	DEFAULT	DESCRIPTION	RANGE
STRNGPTR Long	None	Pointer to string	0...$FFFFFF

Note: Following SETCHR the COP is repositioned to the lower left corner of the character position following the last character printed.

10.3.31 CLEAR THE WINDOW -- CLRWIN

CLRWIN clears the window by using the inverse of the fill word.

CLRWIN repositions the COP to 0,0 in device coordinates.

MNEMONIC: CLRWIN		FUNCTION NUMBER: 30	
ARGUMENT LENGTH	DEFAULT	DESCRIPTION	RANGE
None			

10.3.32 DRAW A BOX AROUND THE WINDOW -- FRAME

FRAME draws a box around the window. FRAME does not affect the position of the cup.

MNEMONIC: FRAME FUNCTION NUMBER: 31

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
None				

10.3.33 DRAW AN ELIPSE -- SETELIPS

SETELIPS draws an elipse centered at the COP. The X and Y radii of the elipse are to the distance betweent he COP X and Y values and the X and Y values supplied as arguments to the function call. The maximum X or Y radius is 256 pixels!

MNEMONIC: CIRCLE FUNCTION NUMBER: 32

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
XW	Word	None	Circumferential point	User coordinates
YW	Word	None	Circumferential point	User coordinates

NOTE: SETELIPS does not effect the COP.

10.3.34 SCROLL WINDOW CONTENTS WINDOW -- SCRLWIN

SCRLWIN scrolls the contents of the window by distance between the COP and a point passed as arguments.

MNEMONICS: SCRLWIN FUNCTION NUMBER: 33

ARGUMENT	LENGTH	DEFAULT	DESCRIPTION	RANGE
1	Word	None	X coordinate	+/-32767
2	Word	None	Y coordinate	+/-32767

10.4 ERROR MESSAGES

Error messages from the graphics driver are returned in D7.B and follow the conventions for device drivers under the I/O manager.

No error = 0

Unsupported entry = 6X

Open error = 11 no available windows
12 parameter error in DIB

Function error = $20 + FUNCTION NUMBER

11.0 SYSTEM SERVICES

11.1 ISSUING SYSTEM CALLS (SC)

All system calls are performed by a system call (TRAP #0) instruction followed by a routine number word. CS-OS automatically vectors the call to the appropriate address. The SC handler saves the contents of all the registers not explicitly changed by the specific system function.

Since each CS-OS routine call is done in the same way, TRAP #0 with a routine number, they can be made macros and used like new instructions. For example, CS-OS has a routine to print a message from an open file. It would be called as follows:

```
TRAP #0   call CS-OS
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. CS-OS was written with the express purpose of providing a list of such useful "extended instructions."

Using the system call mechanism, up to 65536 different system calls are possible. Programs that use TRAP #0 instructions must be modified to avoid the first 69 vectors, or CS-OS routines will be called with unpredictable results.

11.2 SC ROUTINE INDEX

The following table lists the CS-OS system routines by their number. See the following section for a description of each routine.

```
1       reserved
2       reserved
```

3	reserved	
4	reserved	
5	OPEND	open directory
6	reserved	
7	reserved	
8	GETDR	get an FIB from directory
9	PUTDR	update an FIB in directory
10	reserved	
11	CHAIN	chain in a new program
12	reserved	
13	WARMST	warm restart of system
14	LOADB	load a binary-format file
15	reserved	
16	NXTOK	parse a token from input line
17	GTCMD	get a command line
18	PRTMSG	print a message
19	unused	
20	FMTS	format a filename string
21	CMWC	string compare with wildcards
22	SECSIZ	access DSKTAB entry for drive
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	unused	
32	unused	
33	unused	
34	unused	
35	unused	
36	unused	
37	unused	
38	unused	
39	unused	
40	unused	
41	unused	
42	unused	
43	unused	
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	SETCRTCR	set CRT control register
54	INQCRTCP	inquire CRT control register
55	unused	
56	unused	
57	unused	
58	AFRETAB	acquire the free-table resource
59	RFRETAB	release the free-table resource
60	reserved	
61	reserved	
62	reserved	
63	reserved	
64	ACQDIR	acquire a disk directory
65	RELDIR	release a disk directory
66	CLOSED	close a disk directory
67	unused	
68	to be documented	
69	to be documented	

The remainder of this chapter describes the TRAP #0 services handled by the system call handler.

11.3 COMMAND-PARSING ROUTINES

16 NXTOK

This routine breaks up a command line into "tokens." A token is a substring of the command line which is treated as a unit. CS-OS 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 CS-OS. This includes the period (.), comma (,), colon (:), dollar sign ($), equals sign (=), semicolon (;), and the arithmetic symbols +,-, and / .

CARRIAGE RETURN: The ASCII carriage return character (OD hex).

ERROR: A token not falling into one of the above classes.

NXTOK uses system common for its parameters. Scanning 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 (WCRD)    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
```

CUCHAR is returned pointing one character beyond the end of the present token. If the token is a number (RC=03), then 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
```

11.4 FILENAME FORMATTING

20 FMTS

This routine formats a filename from the input form which may vary in length to the fixed internal form. It also handles the expansion of widecard characters. The calling sequence puts the "from" address into

register A0 and the "to" address into register A1. A byte count is passed in register D0. The from-address is typically the start of a token in the command line. The to-address is typically the FCBNAM field of an FCB. The byte count is the total length of the name; the sum of the length of the three tokens (name, . , ext) which comprise it. FMTS expands the wildcard character "*" into a string of "?" of the proper length. FMTS returns a condition byte in register D0 as follows:

D0.B=00 unambiguous name
D0.B=01 ambiguous name (wildcards found)
D0.B=02 bad name (error)

Example of the use of FMTS:

```
CMDLIN DC.B   'ABC?.*'        length=6 characters

       LEA FCB+FCBNAM,A1      point to FCB name field
       LEA CMDLIN,A0          point to command line
       MOVEQ #6,D0            set length
       FMTS                   format name
```

at this point, D0.b=01 and the name field of the FCB contains

```
       ABC?      .??? where " " indicates a space
```

21 CMWC

This routine compares strings skipping over the wildcard character "?" which matches any character, including a space, when it occurs in the "from" string. Register A0 points to the "from" string, while register A1 points to the "to" 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.

11.5 DIRECTORY-HANDLING ROUTINES

5 OPEND

This routine accesses the directory space on a particular disk and returns a pointer to the first FIB on the disk. It is called with register A6 pointing to an FCB which has the drive number set up in FCBDRV and 'DSK' or 'FDS' in FCBGDT. The FCBDBA must point to a buffer large enough for one disk sector. The status (FCBSTA) is returned as follows:

00=good
01=end of directory found
>1=error condition value

If the status is good, the buffer (FCBDBA) contains the first sector of the directory from the indicated disk and FCBIND is initialized to the start of the first FIB. It is up to the user to check that the FIB is not a deleted file. This is done by looking for a space (20 hex) in the first byte of FIBNAM. Hence, if the index register points to an FCB which has FCBGDT, FCBDRV, and FCBDBA properly set, the following code will check for a valid FIB entry.

```
        OPEND                   open directory
        TST.B FCBSTA(A6)        good status?
        BNE ERROR               no, error!
*
        MOVEA.L FCBIND(A6),A1   point to FIB
        CMPI.B #' ',(A1)        is first char. blank?
        BEQ NOGOOD              if so, not valid
```

Note: GCBSTA=01 indicates a totally empty disk.

8 GETDR

This routine gets subsequent directory entries from a disk after OPEND has been used. Each call to GETDR will move the pointer FCBIND to the next FIB in the sector buffer. GETDR automatically reads new directory sectors as necessary until the end of the directory is encountered. The calling sequence for GETDR is the returned in FCBSTA.

9 PUTDR

This routine is used to put a new FIB into a disk directory. It assumes that OPEND and GETDR have been used to find a spot for the new FIB where it will overlay either a deleted FIB or the next unused FIB on the disk. It assumes that the necessary file specification has been placed into the FCB (FCBNAM, FCBTYP, FCBACS, FCBFTS, FCBLTS, and FCBNMS) and register A6 is pointed to the FCB. PUTDR will copy the FIB entries from the FCB to the disk directory location pointed to by FCBIND. Status information is returned in FCBSTA.

66 CLOSED

This routine closes and releases the directory previously opened with an OPEND SC. If the directory is not closed, further access to that directory is blocked. CLOSED is called with register A6 pointing to the same FCB that was used for the OPEND SC.

11.6 INITIALIZATION AND WARMSTART

13 WARMST

This routine returns control to CS-OS from a running program. WARMST will close all open files on the FCB chain, clear the free-space entries in system common, and prompt for a new command. WARMST only affects the SYSTEM task, all other tasks will be unaffected. (Unless those tasks are working on open files. Programs called from the SYSTEM task can use a simple RTS (return-from-subroutine) instruction to return to the system. This will not affect any open files.) WARMST does not change any device assignments, SET operations, or the TIME values. It also does not affect any device monitors.

11.7 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:

BIT0	PAGE SELECT	0=PAGE 0; 1=PAGE 1
BIT1	NOT USED	
BIT2	INVERSE VIDEO	0=NORMAL; 1=INVERSE
BIT3	VIDEO BLANKING	0=BLANK; 1=NORMAL
BIT4-7	NOT USED	

55 INQCRTCR

INQCRTCR returns the one-byte value of the CRT control register in D0.B (See above).

11.8 UTILITY SYSTEM CALLS

17 GTCMD

This routine accepts a command line from the console or and 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. If the user desires to back up to the start of the line, set CUCHAR=DESCRA in system common.

18 PRTMSG

This routine prints a string on the console device. Register A6 is pointed to the start of the string. If the string terminates with a carriage return, a new string is issued. If the string terminates with 04 hex, no linefeed is issued.

22 SECSIZ

This routine returns in register D0 the size of the sector in bytes.

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 the "SET" variables.

11.9 PROGRAM CHAINING

11 CHAIN

This routine loads a new program file into memory and starts executing it. It uses LOADB to bring in the new file. CHAIN is called with register A6 pointing to an FCB with the desired FCBDRV, FCBNAM, etc. CHAIN moves the data from the user FCB into a system space so that the new file may overlay the user FCB memory. If there was some error, CHAIN will issue an error message and return to the system for a new command. If the file to be CHAINed had no transfer address, this will be flagged as an error. If there was no error, the new file will begin execution at its transfer address.

14 LOADB

This routine loads a binary-format file into memory. The file type (FIBTYP) must be 00 or 01. If it is not, LOADB will issue an error message and return without changing memory. LOADB expects register A6 to point to an FCB with FCBGDT='DSK' or 'FDS', FCBDRV, FCBNAM set to the desired file specification, FCBDBA = address of a sector-long buffer, and FCBDBL =

length of sector. If an error condition is encountered while reading in the new file, LOADB will close the file and return to the system. If the file had a transfer address, it will be stored in the location VALUE in system common. If there was no transfer address, VALUE will be zero.

11.10 TIME OPERATIONS

23 GETTIM

This system call retrieves the time-of-day and date information from the system and returns it in registers D1.L and D2.L. Each register uses the low-order three bytes only, with the high-order byte undefined. Within each register, the time values are formatted as BCD-encoded bytes (except for the day-of-year, which is 2 bytes). Register D1.L contains the hours:minutes:seconds, while register D2.L contains the day-of-year/year. Hence, if the current time is 02:45:10 0355/80, then the registers will contain:

D1.L = xx024510 D2.L = xx035580

24 SETTIM

This system call takes the current values in registers D1.L and D2.L and uses them to update the current time-of-day and date in the system. The register contents should be formatted as shown in GETTIM above. SETTIM does no error checking for nonsense values.

11.11 MULTITASK SYSTEM CALLS

There are 6 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:

00 = good
01 = no available PCBs (the task did not start)
FF = duplicates an existing task name (did not start)

The value in D1.B is valid only if the status is 00. Otherwise, the task did not actually 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 running task). This PCB must be either running or on the ready-queue. It cannot hold any resources (PCBACQ nonzero). GIVPCB returns a status byte in D0.B. The values returned by GIVPCB are:

00 = good
01 = PCB-number invalid
02 = PCB not running or on ready-queue
03 = PCBACQ nonzero

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 in PCB (see CUCHAR description)

If the PCB-number specified is invalid, GSTAT returns task status (D5.B) as zero. (PCB not allocated.)

28 SETPRI

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.

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. 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.

11.12 SYSTEM MONITORS

There are a number of resident monitors that the system uses that are available to the user. They protect the free-space table and free-space sector, the CON and LPT devices for long-term use and the disk directories.

58 AFRETAB

This monitor acquires the free-space table and the free-space sector. It is called with the device number in D0.B. If the resource is available it acquires it and returns, otherwise the calling process is placed on the FRETABQ(drvnum).

59 RFRETAB

This monitor releases the free-space table and the free-space sector. It is called with the drive number in D0.B. It releases the resource and signals on the FRETABQ(drvnum).

64 ACQDIR

This monitor acquires the directory of a particular drive to prevent improper processing of the directory. It is called with an FCB with FCBDRV containing the drive number. If the directory is available ACQDIR acquires it and returns, otherwise the calling process is placed on the DIRQUE(drvnum).

65 RELDIR

This monitor releases the Directory of a particular drive and signals on the DIRQUE(drvnum). It is called with an FCB with FCBDRV containing the drive number.

11.12.1 EXAMPLE OF A SIMPLE MONITOR

The following example illustrates how simple an CS-OS monitor can be. The monitor has three entry points: INITIT initializes the state variable and queue, GETIT acquires the resource controlled by the monitor, and GIVEIT releases the resource. It is vital that INITIT be called before any task tries to use the resource. Failure to initialize the queue and state variable will produce unpredictable system response. Once initialized, the monitor entry points GETIT and GIVEIT control access to the resource. A task requiring the resource calls GETIT. Upon return from GETIT, the task has exclusive use of the resource. No other task can interfere with the use of the resource until this task calls GIVEIT which releases the resource. It is important that tasks release resources they no longer need.

Note the use of the four monitor TRAPS in building up monitors. ENTERMON and EXITMON enclose the GETIT and GIVEIT code. INITIT need not be enclosed, since initialization is done before any task needs the resource

controlled by this monitor. The critical parts of a monitor are its state variable and queue header. There could be more than one acquire or release entry, or multiple state variables and queue headers could exist.

```
*
* EXAMPLE OF AN CS-OS MONITOR
*
STATE   DS.W 1                  STATE VARIABLE
HEAD    DS.L 1                  QUEUE HEADER
*
* INITIALIZATION ENTRY POINT
*
INITIT CLR.W STATE              INITIALIZE VARIABLES
       CLR.L HEAD
       RTS
*
*
* ACQUIRE-RESOURCE ENTRY POINT
*
GETIT  ENTERMON                 START CRITICAL SECTION
       TST.W STATE              RESOURCE IN-USE?
       BEQ.S GOTIT              NO
*
       LEA.L HEAD,A6            YES, CALLER MUST WAIT ON QUEUE
             WAIT
*
GOTIT  MOVE.W$FF,STATE          SET IN-USE
       EXITMON                  END CRITICAL SECTION
       RTS
*
*
* RELEASE-RESOURCE ENTRY POINT
*
GIVEIT ENTERMON                 ENTER CRITICAL SECTION
       CLR.W STATE              NOT IN-USE
       LEA.L HEAD,A6
       SIGNAL                   GET NEXT TASK ON QUEUE
       EXITMON                  END CRITICAL SECTION
       RTS
```

11.12.2 EXAMPLE OF A MORE COMPLEX MONITOR

The following example illustrates the use of CS-OS monitor constructs to control a circular-buffer pool. Here there is one state variable FULL,

but there are two queues VACANT and OCCUPIED. The VACANT queue holds tasks waiting to put bytes into the buffer, OCCUPIED holds tasks wishing to remove bytes from the buffer pool. The resource consists of a buffer CIRCLE, and head-pointer TOP, and a bottom-pointer BOTTOM. The entry GET places the byte passed in D0.B into the circular buffer, while entry GIVE returns the next byte from CIRCLE in register D0.B. In this example, GET is the acquire-resource entry and GIVE is the release-resource entry.

```
* MONITOR TO CONTROL CIRCULAR BUFFER POOL
*
CIRCLE   DS.B 80                 CIRCULAR STORAGE BUFFER
TOP      DS.L 1                  HEAD-POINTER TO CIRCLE
BOTTOM   DS.L 1                  TAIL-POINTER TO CIRCLE
FULL     DS.W 1                  STATE VARIABLE
VACANT   DS.L 1                  QUEUE HEADERS
OCCUPIED DS.L 1
*
* INITIALIZATION ENTRY
*
INITBUF  CLR.W FULL              INITIALIZE VARIABLES
         CLR.L VACANT
         CLR.L OCCUPIED
         LEA.L CIRCLE,A6
         MOVE.L A6,TOP           SET POINTERS
         MOVE.L A6,BOTTOM        CIRCLE IS EMPTY
         RTS
*
*
* ACQUIRE-RESOURCE ENTRY POINT
*
GET      ENTERMON                START CRITICAL SECTION
         MOVE.W FULL,D1          GET STATE VARIABLE
         CMPI.W #80,D1           BUFFER FULL?
         BNE.S NOWAIT            NO
*
         LEA.L VACANT,A6         YES, MUST WAIT
         WAIT
*
NOWAIT   MOVEA.L BOTTOM,A1       GET BOTTOM POINTER
         MOVE.B D0,(A1)+         STORE BYTE INTO BUFFER
         CMPA.L #CIRCLE+80,A1    WRAP AROUND?
         BLT.S NOWRAP            NO
*
         LEA.L CIRCLE,A1         YES
*
NOWRAP   MOVE.L A1,BOTTOM        NEW BOTTOM POINTER
         ADDQ.W #1,FULL          BUMP COUNT
         LEA.L OCCUPIED,A6
         SIGNAL                  SIGNAL THAT DATA IS AVAILABLE
```

```
        EXITMON                 END CRITICAL SECTION
        RTS
*
*
* RELEASE-RESOURCE ENTRY POINT
*
GIVE    ENTERMON                START CRITICAL SECTION
        TST.W FULL              ANY DATA IN CIRCLE?
        BNE.S HAVDAT            YES
*
        LEA.L OCCUPIED,A6       NO, MUST WAIT
        WAIT
*
HAVDAT  MOVEA.L TOP,A1          GET POINTER
        MOVE.B (A1)+,D0         GET DATA BYTE
        CMPA.L #CIRCLE+80,A1    WRAP AROUND?
        BLT.S NOWRAP2           NO
*
        LEA.L CIRCLE,A1         YES
*
NOWRAP2 MOVE.L A1,TOP           NEW POINTER
        SUBQ.W #1,FULL          ONE FEWER BYTE IN CIRCLE
        LEA.L VACANT,A6
        SIGNAL                  SIGNAL BUFFER AVAILABLE
        EXITMON                 END CRITICAL SECTION
        RTS
```

Note that this straightforward monitor completely solves the producer/consumer problem. The four monitor primitives ENTERMON, EXITMON, WAIT, and SIGNAL are sufficient to solve any task synchronization problem in CS-OS. They are also capable of providing inter-task communication, although the mailbox mechanism described in the next example makes task-to-task communication simpler.

A.0 APPENDIX: AUXILLARY SOFTWARE

Several auxillary files have been created to aid the user in preparing Computer System Programs. These files can be broken down into the following categories:

1. MACRO and EQUATE files to be merged in with user source code at assembly time using the include option.

   These files may be distinguished by their names which will terminate with a ".INC" for example:

   TOOLKIT.INC - a collection of system calls.

2. Test programs and utilities which are in source or load module form or both.

   These files may be distinguished by their names which will terminate with a ".SRC", ".SMP" or a ".BIN" for SOURCE SAMPLE or OBJECT MODULES respectively. For example:

   GRTEST00.SMP - a sample graphics program

   MENU.BIN - a menu program

The following is a discussion of the auxillary files for the Computer System.

A.1 TOOLKIT.INC -- A COLLECTION OF SYSTEM CALL MACROS

The TRAP #0 system calls described earlier in the documentatio may be accessed using this MACRO collection. This collection contains MACROS for individual system calls as well as a general purpose system call MACRO to perform all TRAP # calls.

Note: When performing the TRAP #0 system calls described earlier, precede the call with an "@". For example @NXTOK.

A.2 IOMCLBXX.INC-A COLLECTION OF MACROS TO PERFORM INPUT/OUTPUT CALLS

The MACROS in this collection are for use in passing information to the I/O manager. DEVICE INFORMATION BLOCK (DIB) equates are included.

The SYSTO MACRO is used to perform the OPEN CLOSE READ, WRITE, AREAD, AWRITE and cancel operations to device drivers with the following syntax:

SYSIO <OPCO>, <#LUN>, <CBPTR>, <ERRP>

Where <OPCO> is one of the following:

Open close read write aread awrite function

<#LUN> in a logical unit number

DEVICE INFORMATION BLOCK (DIB),

FUNCTION PACKET (FPKT) or

DATA TRANSFER CONTROL BLOCK (DTCB)

<ERRPTR> is an error branch pointer

The SYSIOB MACRO is used to perform byte I/O operations with device drivers via the I/O manager with the following syntax:

SYSIOB <OPCO>, <DIBBIO>

Where <OPCO> is one of the following:

BREAD WBRIT OR BTEST

<DIBBIO> is a long word table offset supplied in the DIBBIO field by the I/O manager at open time.

The DIBGEN MACRO is used to aid users in the preparatio of the device information block to open a device.

DIBGEN DIBVOL,DIBDTD,DIBTRN,DIBOPT,DIBFCN

See DOCUMENTATION on DEVICE INFORMATION BLOCK (DIB) for explanation of the above fields.

A.3 MNEMONIC INCLUDE FILES

Several MNEMONIC equate tables have been creating function packets for the vari ous device drivers--these include--

FDMCLBXX.INC	floppy disk driver equates
GRMCLBXX.INC	graphics driver equates
RSMCLBXX.INC	RS23 driver equates
PRMCLBXX.INC	printer driver equates

A.4 SAMPLE APPLICATION PROGRAMS

Several sample application progrms have been prepared to illustrate the use of the various drivers via the I/O manager. These include:

GRTESTXX.SMP	A sample graphics program
PRTESTXX.SMP	A sample printer program

GC22–9200

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 usefullness 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

Reader's Comment Form

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

IBM Instruments, Inc.
P.O. Box 332
Danbury, Ct. 06810