HEWLETT-PACKARD

Assembly Language I/O Reference Manual

HP-83/85

HEWLETT
PACKARD

Assembly Language
I/O Reference Manual

HP-83/85

November 1982

Reorder Number
00085-90818

Printed In U.S.A.

CONTENTS

Section

Page

I INTRODUCTION

II I/O PROCESSOR COMMANDS AND PROTOCOL

III PERFORMING I/O OPERATIONS

ILLUSTRATIONS

TABLES

INTRODUCTION

1.1 Overview of Low-Level I/O

The purpose of this manual is to document HP-83/85 input/output operations. For example, you may need to speed up a specific data transfer, or do a custom I/O operation. These are problems that cannot be solved with a BASIC program and an I/O ROM. This manual should be used with the Assembler ROM manual (your reference for writing binary programs).

This documentation is arranged in the following sections:

Section I. Introduction--Read this section first. It is an overview of how I/O is used on the HP-83/85.

Section II. I/O Processor Commands and Protocol--Use as a reference for the commands that communicate with the I/O processor on each interface card, and the protocol for communications in this multi-processor system. You will find a flowchart outlining the protocol required to pass a command to the I/O processor. Refer back to this section after you analyze your I/O operation using section 3.

Section III. Performing I/O Operations--Each I/O operation is discussed along with the programming steps required for execution. You will find a detailed discussion of simple input/output operations, burst input/output, interrupt operations, and status and control operations. Documentation on the fastest rates possible to do I/O operations for each interface is included, as well as sample I/O utilities.

Section IV. A Sample Program--This example contains simple, burst, and interrupt transfer routines. It also includes a hardware vector hook interrupt service routine, and other general purpose utilities such as a variable set-up utility.

If you are trying to speed up an I/O operation, you should first make sure the computer is capable of attaining your speed requirements. To do this, compare your requirements with the rates documented in section 3.

A tradeoff of speed is required to gain the power of using BASIC with the I/O ROM. The more general your application, the more speed you will gain using binary code. For example, if you are doing a SIMPLE ENTER from a GPIO interface at select code 4 (ENTER 4 in BASIC), you can write a binary program that sends a SIMPLE INPUT command and handshakes each byte into a string variable until the GPIO device is out of data or the string is full. This program bypasses the extra image checking, data formatting, and other options that the I/O ROM allows, making execution at least three times faster.

Once you have determined the feasibility of your requirements, use the following guidelines:

1. Read section 1.
2. Refer to the appropriate flowchart and sample code in section 3.
3. Use section 2 as a reference for commands and protocol.
4. Design a program using the example in section 4.
5. Code and debug your program.

1.2 Interface Functional Description

The purpose of interface operations is to transfer data between computer memory and some other device or devices. The source or destination of the data may be a data storage area (buffer) in RAM or direct program interaction.

Interface devices use a variety of methods to communicate. Different interfaces are available with selectable options to allow customizing to a specific method of communication.

Between the interface and the computer, every attempt has been made to have only one method of communication. The only difference between interfaces (as seen by the HP-83/85 low level code) is their respective interface select code numbers which do not affect interface type. Even though different interfaces have different capabilities, they all speak the same language. For example, consider the statement:

```
OUTPUT 5 ; "1"
```

If there is a serial interface at select code 5, then the ASCII character "1" will be output serially. If there is a BCD interface at select code 5, then "1" will be output as a BCD digit. In both cases the computer program operations (data values and instructions executed) are identical.

To maintain uniformity there is a microcomputer in each interface. This microcomputer acts as an interpreter able to listen in one language and speak in another.

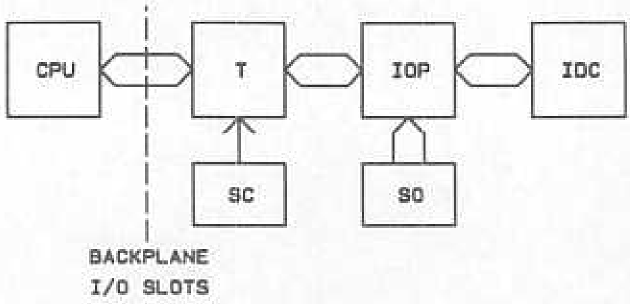

Figure 1-1. I/O Hardware Diagram

The CPU is the central processing unit which executes machine language instructions (either SYSTEM/BASIC or ROM/BINARY). It has unique designations of logic levels, timing, etc. Terminology used in the above diagram and throughout this manual includes:

Input/Output Processor (IOP): An 8Ø49 microcomputer which executes preprogrammed microcode. This code allows the processor to converse through interface dependent circuitry (IDC) according to the selected options (SO).

Translator (T): A two-byte wide channel with HP-83/85 logic levels, timing, and control on one side and 8Ø49 logic levels, timing, and control on the other side.

Select Code (SC): Indicates to the translator where to appear in memory address space.

1.3 Interpreting the Translator Bytes

As an I/O programmer, the only access you have to the I/O processor is through the translator. Each translator appears as two consecutive bytes in memory. The I/O processor also sees two bytes. These are full duplex bytes. Unlike memory bytes, what you read from these two bytes is not what you just wrote there. What you read is what the I/O processor wrote into them (with the exception of two control bits). What you write to these bytes is what the I/O processor receives next time it reads them (with the same exception).

Because of this read and write process, four names are associated with these two bytes. The first byte (lower, even address) is called the calculator control register (CCR) when you write to it and the processor status register (PSR) when you read from it. In general, "calculator" refers to the computer CPU, and "processor" refers to the I/O processor. The second byte (higher, odd address) is called the output buffer (OB) when written to, and input buffer (IB) when read from.

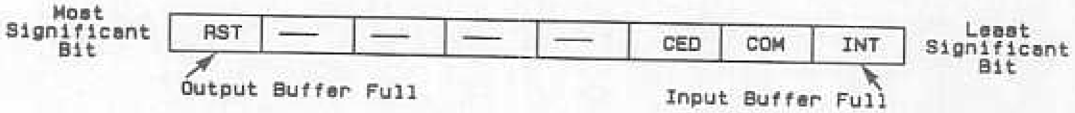

Figure 1-2. Calculator Control Register

RST (reset): When set, the I/O processor initiates the reset operation.

CED (calculator end data): When set, the CED bit indicates to the I/O processor that the computer has declared the current data byte to be the last of the current sequence.

COM (command): This bit tells the I/O processor to interpret the byte it finds in the output buffer. If the COM bit is set then the output buffer contains a protocol command. If the COM bit is clear, then the output buffer contains a byte of data.

INT (interrupt): Setting INT interrupts the I/O processor.

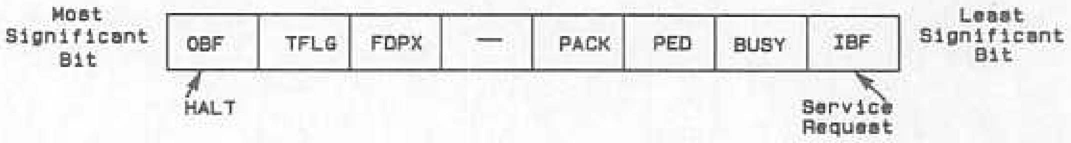

Figure 1-3. Processor Status Register

OBF (output buffer full): Writing a byte to the output buffer sets OBF. It is cleared when the I/O processor reads the output buffer.

TFLG (transfer flag): There are times when more than one byte can be transferred during a single interrupt (for example, with a multi-digit BCD field). When TFLG is set it indicates that the I/O processor has or wants additional bytes.

FDPX (full duplex): When set, FPDX indicates that this I/O processor can do interrupt transfers in both directions concurrently.

PACK (processor acknowledge): This bit is set to confirm that the CPU has interrupted the I/O processor.

PED (processor end data): When set, PED indicates to the computer that the current byte is to be the last of the current input sequence.

BUSY (not idle): When set, BUSY indicates that the I/O processor is occupied.

IBF (input buffer full): IBF is set when the I/O processor writes a byte to the input buffer and is cleared when the CPU reads the input buffer.

The input and output buffers are both eight-bit bytes. The meaning of the bits is entirely situation dependent. For the CCR/PSR, six of the bits are read by the I/O processor as what was written there. What you read is what the I/O processor wrote there. The highest and lowest order bits are control bits. They are read like this:

1. When the I/O processor reads the RST bit, it receives output buffer full.
2. When the I/O processor reads the INT bit, it receives input buffer full.
3. Where you read output buffer full, the I/O processor writes HALT.
4. Where you read input buffer full, the I/O processor writes SRQ (interrupt).

The translators are positioned in address space according to the three select code settings as indicated in table 1-1.

Table 1-1. Translator Addressing

Address	Name	Switches	Select Code
177520 177521	CCR/PSR OB/IB	0 0 0	3
177522 177523	CCR/PSR OB/IB	0 0 1	4
177524 177525	CCR/PSR OB/IB	0 1 0	5
177526 177527	CCR/PSR OB/IB	0 1 1	6
177530 177531	CCR/PSR OB/IB	1 0 0	7
177532 177533	CCR/PSR OB/IB	1 0 1	8
177534 177535	CCR/PSR OB/IB	1 1 0	9
177536 177537	CCR/PSR OB/IB	1 1 1	10

These bytes are accessed using instructions such as LD, ST, PU, and PO, with the exception of multi-byte instructions which do not work as expected. If you read from an address which is not claimed by a translator, you will read all 1's. If you write to such an address your data will be lost.

The mapping of select codes into memory space is used by the I/O ROM and the standard I/O interfaces. A similar set of addresses exists from 177500 to 177517. This additional set has some differences which are:

- The I/O ROM does not handle these locations. If an interrupting translator is in one of these locations, the I/O ROM service routine branches to the hook NEWIRQ.
- A factory mask option, rather than a select code switch, causes a translator to occupy this region.
- The locations 177500 and 177501 (which correspond to switch settings of 0 0 0 or select code 3 in this region) are unavailable for a select code because of the global uses of these addresses.

1.4 RAM Hooks Available to the I/O Programmer

1.4.1 IRQ20 (102470)

In the system reserved area of RAM memory, IRQ20 is a location that is called when an IOP interrupts the CPU. The system code does not use this hook except to initialize it with a RTN instruction at power-on.

Control passes to IRQ20 when the CPU is interrupted by an IOP. This is how the transition looks at the hardware level:

1. The CPU, rather than executing the next instruction in its normal sequence, pushes the address of that next instruction onto the R6 (return) stack. It then notifies the interrupting device (translator) that it's ready.
2. The translator returns the number 20 (octal) as an interrupt vector for the CPU. Other types of interrupting devices return other vectors.
3. The CPU reads location 20 (in system ROM), gets the address of IRQ20 (which is stored there), and commences execution at IRQ20. Thus, when considering the code for IRQ20 and the interrupt service routine which it calls, the following conditions can be assumed from the fact that the code at IRQ20 is executing: there is a return address on the R6 stack; and the interrupting device is an IOP.

In order for the interfaces to function, ROM or binary code must take the hook at IRQ2Ø. The IOPs need interrupt service to complete their power-on reset routines (they must interrupt to report the self-test results). If any one of the three ROMs which use interfaces (I/O, Plotter/Printer, and Mass Storage) is present in the system, this power-on reset protocol will be handled by the ROMs before your binary program is loaded. If none of these three is present, then you must take the IRQ2Ø hook and complete the reset yourself before you can use the interfaces.

You may do simple I/O without taking IRQ2Ø if another ROM is handling power-on reset. Also, if the other ROM is the I/O or Mass Storage ROM, then the service routine hooked in IRQ2Ø by one or the other (I/O if both ROMs are present), will be able to handle your burst termination interrupt and terminate the infinite loop. For all other situations you will want to manage your own interrupt service procedures so you will take the hook at IRQ2Ø.

If you have one of the above ROMs doing the power-on reset, you will miss the opportunity to identify the select codes of all translators present. In this case, load a copy of the byte at 1ØØ667 (octal). This is a system location used by these ROMs as a "select code present" indicator. If the bit is set, the select code is there.

Table 1-2. Select Code Byte Interpretation

MSB	Bit Number	7	6	5	4	3	2	1	Ø	LSB
	Select Code	1Ø	9	8	7	6	5	4	3	

If you need to write an interrupt service routine (ISR), the rest of this section provides an explanation of the code required. Following is the code at the hook.

```
   1Ø247Ø  IRQ2Ø  RTN          (before the hook is first taken)
```

Once it's been taken:

```
   1Ø247Ø  IRQ2Ø  SAD
   1Ø2471         STBD R#,GINTDS
   1Ø2474         JSB =ROMJSB
   1Ø2477  IRQ2Ø+ DEF ISR
   1Ø25Ø1         BYT ROM#
   1Ø25Ø2         STBD R#,GINTEN
   1Ø25Ø5  IRQPAD PAD
   1Ø25Ø6  IRQRTN RTN
```

Taking the hook at IRQ2Ø is accomplished by storing the above instructions at IRQ2Ø (IRQ2Ø+ is a convenience label to allow two multi-byte store operations).

The individual instructions are discussed next. Basically these instructions are the first and the last of your interrupt service routine (ISR). Recall that a proper ISR leaves no trace of its execution as far as the interrupted code is concerned. On the HP-83/85 this means that the ISR must preserve the CPU state, current ROM selection, the CPU registers, and the stacks (R6 and R12).

```
SAD              Saves the CPU state.
STBD R#,GINTDS   Assures that the ISR cannot be interrupted.
JSB =ROMJSB      Calling through ROMJSB preserves the current ROM
                 selection.
DEF ISR          Address of service routine.
BYT ROM#         ROM number of service routine (or Ø for binary
                 program ISR).
STBD R#,GINTEN   Re-enables global interrupts.
PAD              Restores the CPU state.
RTN              Pops the return address off the R6 stack and resumes
                 execution where it was interrupted (except at burst
                 I/O termination where the return address is
                 intentionally altered by the ISR).
```

The code at the hook handles preservation of the CPU state and ROM selection. Preservation of the CPU registers and stack conditions is handled by the ISR code. For the CPU registers this amounts to pushing the contents of registers that might be used onto the R6 stack. For stack conditions this amounts to popping them back off before your ISR returns. There is, however, a possibility for stack overflow which must be addressed by the ISR. To understand the problem, we need a picture of the R6 stack from the moment of interrupt to the time when your ISR checks for this condition.

Start with the address pushed by the CPU when it is interrupted. Next, there is a SAD instruction at IRQ2Ø which pushes the CPU status in three bytes. The jump instruction to ROMJSB saves the return address on the stack. ROMJSB increments this address by three (to step past the DEF ISR and BYT ROM# locations that it uses as the desired JSB target) and then pushes CPU registers RØ and R1, the currently selected ROM number, and its own return address as it jumps (JSB) to the ISR. The first thing the ISR does is to push any CPU registers that it might use.

Second, the ISR checks for stack overflow. The R6 stack at the time of the check looks like this:

```
          2 bytes     Interrupted return address.
          3 bytes     SAD.
          2 bytes     IRQ2Ø return address.
          2 bytes     RØ and R1.    \  ROMJSB puts these
          1 byte      ROM number.   /  3 bytes on.
          2 bytes     ROMJSB return address.
          n bytes     Pushed by ISR.
R6-----------
```

Now go back and look at the last three instructions at the IRQ2Ø hook:

```
   STBD R#,GINTEN
   PAD
   RTN
```

These instructions are executed after the ISR has finished and returned to ROMJSB. At the time of execution the R6 stack appears like this:

```
          2 bytes     Interrupted return address.
          3 bytes     SAD.
R6-----------
```

and after the execution of the PAD instruction:

```
          2 bytes     Interrupted return address.
R6-----------
```

and after execution of the RTN instruction:

```
                -     Empty, the state before the interrupt.
R6-----------
```

The problem arises because interrupts are enabled by the STBD R#, GINTEN, but the stack is not empty until after the execution of RTN. If a fast interface is interrupting, the next interrupt will occur while ISR is still busy. As soon as STBD R#,GINTEN is executed, the interrupt will be recognized and PAD will not be executed.

The address will be pushed as the interrupted return address and the stack will look like this at the stack overflow check:

```
          2 bytes     Real interrupted return address.
          3 bytes     Real SAD data.
          2 bytes     Extra interrupted return address.
trouble----
          3 bytes     Extra SAD data.
          2 bytes
          2 bytes
          1 byte       (not changed)
          2 bytes
          n bytes
 R6-------------
```

If the interrupt should occur after the PAD instruction but before the RTN, the stack appears like this:

```
          2 bytes     Real interrupted return address.
trouble----
          2 bytes     Extra interrupted return address.
          3 bytes
          2 bytes
          2 bytes
          1 byte
          2 bytes
          n bytes
 R6-------------
```

If there are many fast interrupts, the extra bytes will build up until the stack exceeds the allocated size. The ISR knows how many bytes it pushed for CPU register preservation. It takes a copy of R6 and subtracts this number plus 12 (decimal), from the 12 bytes known to be there from the saving done by the CPU, IRQ2Ø, and ROMJSB. This gives it a pointer to the "interrupted return address" on the R6 stack. To distinguish "real" from "extra," the ISR compares this address to the two known addresses (1Ø25Ø5 and 1Ø25Ø6) of the PAD and RTN instructions. The names IRQPAD and IRQRTN will be used to refer to the addresses of these two instructions. If the address found is IRQRTN, the ISR knows that the previous ISR was one instruction short of completion when the current interrupt occurred and that the real address is just in front of the current one.

To fix the problem, the ISR moves the top contents of the R6 stack (the three bytes used by SAD through "n" bytes used by the ISR) down two bytes (eliminating the extra return address to IRQRTN) and decrements R6 by two. Nothing is lost because the return to IRQRTN would simply have executed the RTN instruction there which would have returned to the real address.

If the address found is IRQPAD, the ISR knows that the previous ISR did not execute the PAD instruction, so there are five extra bytes. It moves the stack top contents (starting at the return address for IRQ2Ø through the bytes occupied by ISR) down five bytes and decrements R6 by five. Again, nothing is lost because the extra SAD data was about to be replaced by the real SAD data when the PAD instruction was interrupted.

You can see that every time the ISR is called for an interrupt, it must clean up the stack if the previous ISR did not.

1.4.2 IOSP (1Ø24Ø7)

IOSP is also a location in the RAM system reserved area. The executive loop jumps to IOSP when it gets to the end of a BASIC program line and finds that the service request bit is set (bit 4 in XCOM (R17)) and the I/O interrupt bit is set (bit 1 in the RAM location SVCWRD (1ØØ151)). This hook is the means of implementing end-of-line branches. When an end-of-line branch condition is noticed while executing a BASIC line, the type of the condition is stored in RAM. The bits are set in XCOM and SVCWRD and the code goes on executing the current BASIC program line. At the end of the line, the executive loop branches through IOSP to the end-of-line service routine (EOLSV) whose address was set up in the IOSP hook. EOLSV notes the condition and executes the appropriate GOTO or GOSUB.

Note: Use the rest of this section to create an ISR that performs an end-of-line branch.

To take the hook at IOSP, store these instructions at IOSP (1Ø24Ø7).

```
1Ø24Ø7   IOSP   JSB =ROMJSB
1Ø2412          DEF EOLSV
1Ø2414          BYT ROM#      (Ø for binary programs)
1Ø2415          RTN
```

End-of-line branching requires a GOTO or GOSUB as part of the statement that sets it up. The parsing must be done correctly, so let's discuss a sample statement, "ON SELFTEST <select code> GOTO/GOSUB line#." This statement is to set up an end-of-line branch which will be triggered by the select code's IOP interrupting for a self-test report (after being reset). The parse code will be executed when the keyword ON SELFTEST is scanned.

At first things are fairly normal; the instruction PUBD R43,+R6 is used to save the keyword token. The instruction JSB =NUMVA+ is used to parse the select code and then to pop the keyword token back and push it onto the R12 stack with a 37Ø,ROM# or a 371,Ø.

At this point you must handle the GOTO or GOSUB. NUMVA+ called SCAN before it returned (if you don't have any arguments you must do an explicit call to SCAN in place of NUMVA+) and SCAN left the primary attribute byte of the next token in R47. If R47 contains octal 21Ø, then the next token is a GOTO or GOSUB. If R47 is not 21Ø, you have a syntax error.

Having confirmed the 21Ø in R47, the parse code executes:

```
JSB =ROMJSB
DEF GOTOSU
BYT Ø
GTO ROMRTN
```

GOTOSU is the system routine to parse GOTO/GOSUB and is at DAD 17435. If the syntax is correct, this routine pushes three bytes onto the R12 stack (and thus appends them to the program line being parsed) after the bytes put there by the ON SELFTEST parse code.

The run time code for this keyword token has two tasks. The first is to recover the select code value from R12 and an indicator in RAM. The ISR will then know that a self-test interrupt from this select code is the cause for setting the service request bits (in XCOM and XVCWRD) for an end-of-line branch.

The second task is to set up (but not execute) the GOTO or GOSUB. Taking a look at the compiled BASIC line we see:

Token for Fetch select code	Token for Execute ON SELFTEST	Token for GOTO/GOSUB line#

When the ON SELFTEST run time code is executed, the BASIC program counter (R1Ø) is pointing to the GOTO/GOSUB token which is a random GOTO/GOSUB token parsed by the system routine GOTOSU. The run time code must store the contents of R1Ø somewhere in RAM for future use by the end-of-line service routine. The run time code should also increment R1Ø by three to skip execution of the GOTO/GOSUB when the ON SELFTEST statement is executed.

After execution of the ON SELFTEST statement, the chosen select code interrupts with a self-test report. The ISR, noting that ON SELFTEST is active for that select code, sets an indicator to state that this particular end-of-line branch condition has been met. It then sets the XCOM and SVCWRD bits. Note that the complementary OFF SELFTEST statement needs only to reset the "active" indicator set by the ON SELFTEST statement.

The first thing the end-of-line service routine does is to determine why execution passed to it. Due to the structure of the system's executive loop, the EOLSV routine must interact with the system code in a complex way. You should copy these portions of code from the sample program. The execution of the EOLSV is explained next.

If more than one "on condition" statement is active, the desired statement is selected. When the token from this statement was executed, the GOTO or GOSUB token following it was bypassed, but the R1Ø BASIC program counter pointing to it was saved. The EOLSV routine now recovers that copy of R1Ø. It stores the current R1Ø in a system RAM location (1ØØØ4Ø) called ONFLAG as a return address in case the branch is a GOSUB. It sets CSTAT (R16) to 7 to indicate that a GOTO or GOSUB is taking place as part of the execution of a line. Then it places the recovered pointer into R1Ø and returns to the executive loop which performs the actual branch. The EOLSV routine keeps the request bits in XCOM and SVCWRD set so that IOSP will be called again. When EOLSV is called again, it then decides whether or not it has finished with end-of-line branching and if it has, the request bits will be cleared.

1.4.3 NEWIRQ

This is a hook provided by the I/O ROM in its stolen RAM at IOBASE plus 63Ø octal. When the ISR from the I/O ROM gets to the point of reading the address of the interrupting translator's CCR/PSR and the address turns out to be in the lower block of select codes not recognized by the I/O ROM, the ISR (from the I/O ROM) jumps to NEWIRQ. It has already saved everything (including the CPU registers listed below), performed the stack overflow check, and set up three register pairs: RØ and R24 point to the CCR/PSR and R26 points to the OB/IB of the interrupting translator. When the NEWIRQ routine returns through the hook, the ISR restores everything and returns through IRQ2Ø. The hook should be taken with the following code:

```
    JSB =ROMJSB
    DEF NEWISR
    BYT ROM #
    RTN
```

The CPU registers saved by the ISR are: R2-3; R14-15; R2Ø-27; R3Ø-37; R4Ø-47; and R6Ø-67.

I/O PROCESSOR COMMANDS AND PROTOCOL

2.1 Communications Protocol Between the CPU and the IOP

The way the CPU and the IOP communicate is referred to as IOP protocol. This protocol defines commands and a handshaking system for interfacing at the machine level. The location of the input and output buffers allows transfer of individual bytes between the CPU and the I/O processor. Bytes from the I/O processor to the CPU are always interpreted as data bytes (some of this data is I/O processor status information but there is no indicator bit to flag this; it is a matter of context). Bytes from the CPU to the I/O processor may be either data bytes or command bytes. The I/O processor reads the calculator control register before it reads the output buffer and uses the COM bit to decide if the byte in the output buffer is a data byte or a command byte. If it is a command byte (COM = 1), the I/O processor interprets it according to the protocol command language.

Each command byte is an opcode and a field. The opcodes and their field identifiers are shown in table 2-1.

Table 2-1. Command Bytes

	Opcode (4 Bits)	Field (4 Bits)
Most Significant Bit	Ø Ø Ø Ø	Read Status
	Ø Ø Ø 1	Input
	Ø Ø 1 Ø	Burst I/O
	Ø Ø 1 1	Interrupt control
	Ø 1 Ø Ø	Interface control
	Ø 1 Ø 1	(unused)
	Ø 1 1 Ø	(unused)
	Ø 1 1 1	Read auxiliary
	1 Ø Ø -	Write Control
	1 Ø 1 Ø	Output
	1 Ø 1 1	Send
	1 1 Ø Ø	(unused)
	1 1 Ø 1	(unused)
	1 1 1 Ø	Write auxiliary
	1 1 1 1	Extension

Of the 16 possible four-bit opcodes, one disappears because opcodes 1 Ø Ø Ø and 1 Ø Ø 1 are really one opcode with a five-bit field. The four unused numbers and "extension" leave 1Ø opcodes of interest which will be discussed individually. It is conceptually helpful to note that opcodes with the most significant bit set are "wait for data" commands. The first thing the I/O processor has to do after receiving the command is wait for a related data byte from the CPU. The low-numbered opcodes are "immediate execute" commands as they start off by doing something other than wait for the CPU to release a byte.

2.2 Command Protocol Flowcharts

The following flowcharts demonstrate the handshaking system used in IOP protocol.

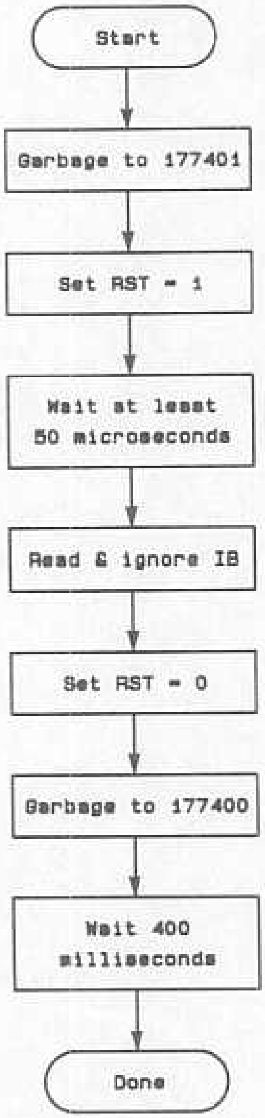

Figure 2-1. Reset One IOP

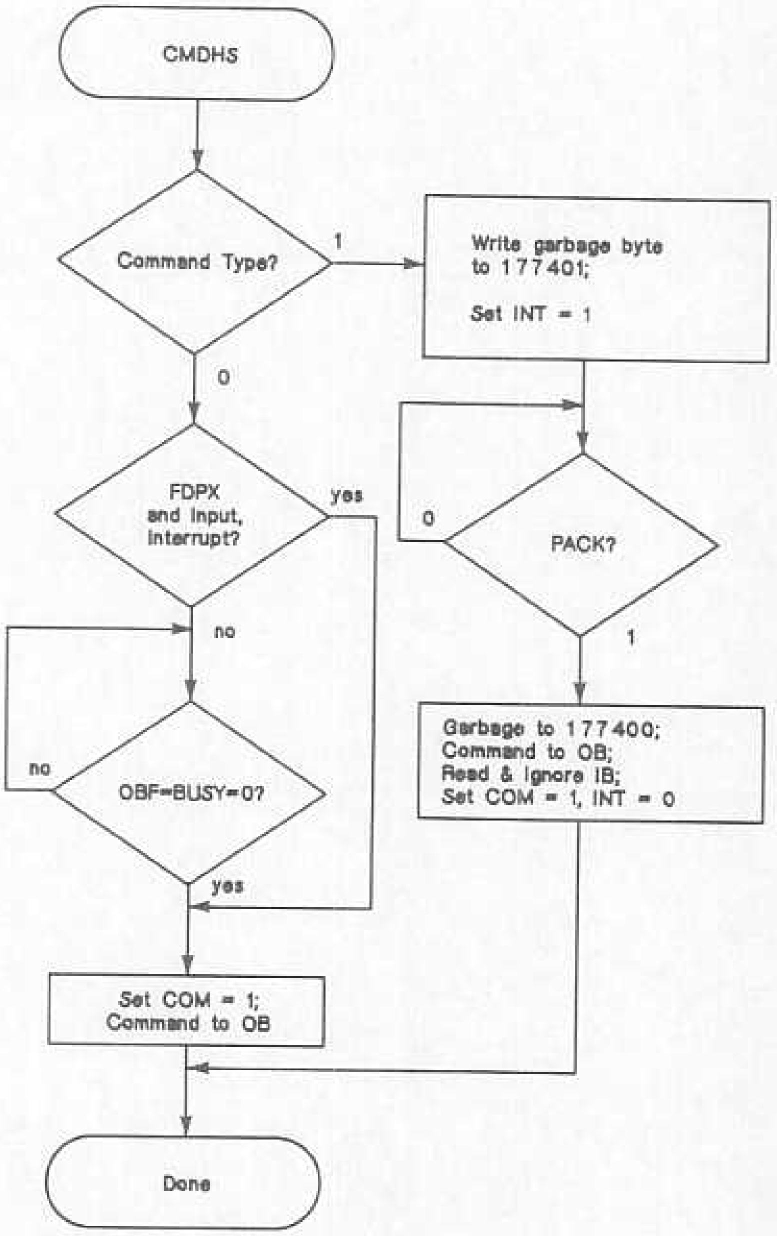

Figure 2-2. Command Handshake

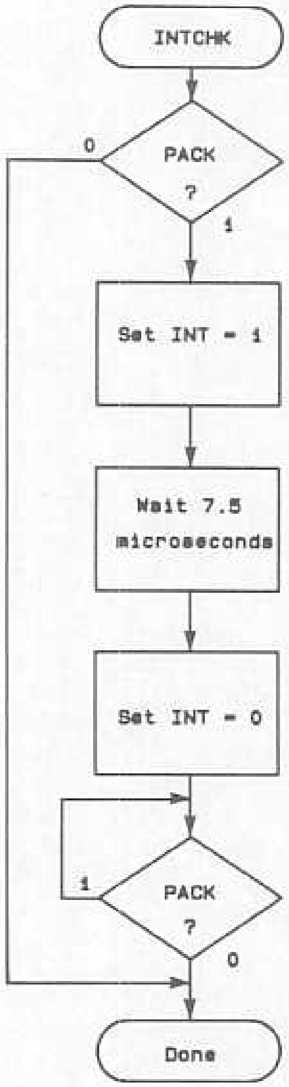

Figure 2-3. Revive An Interrupted IOP

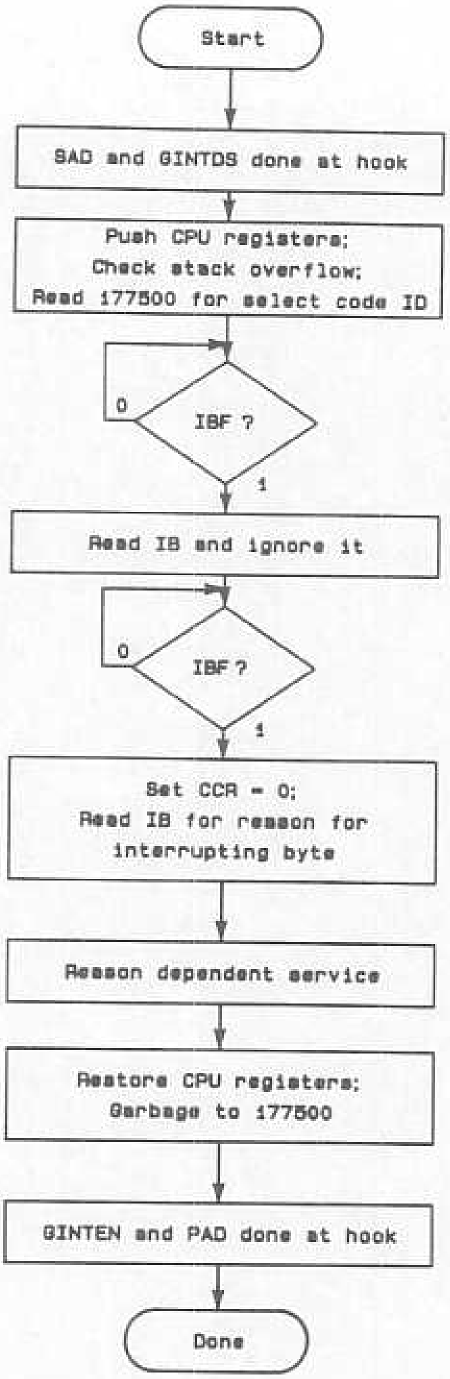

Figure 2-4. Interrupt Service Routine

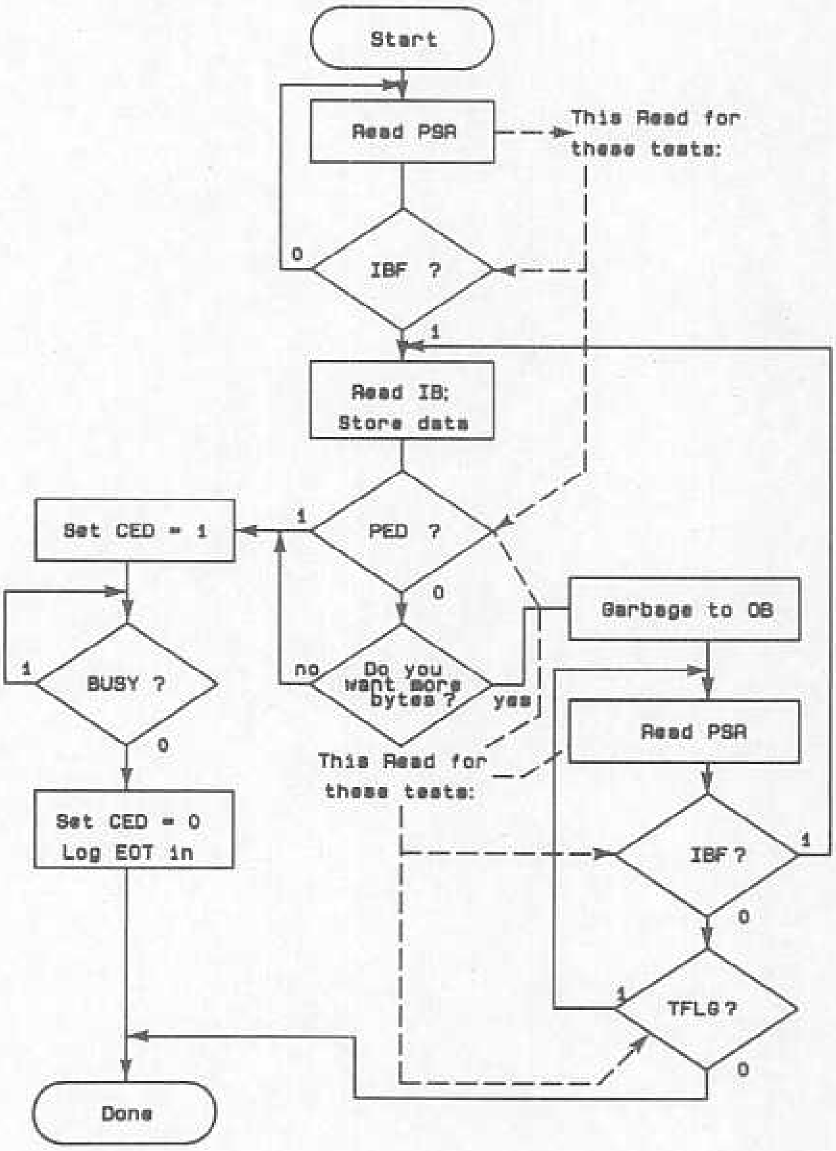

Figure 2-5. Interrupt Output

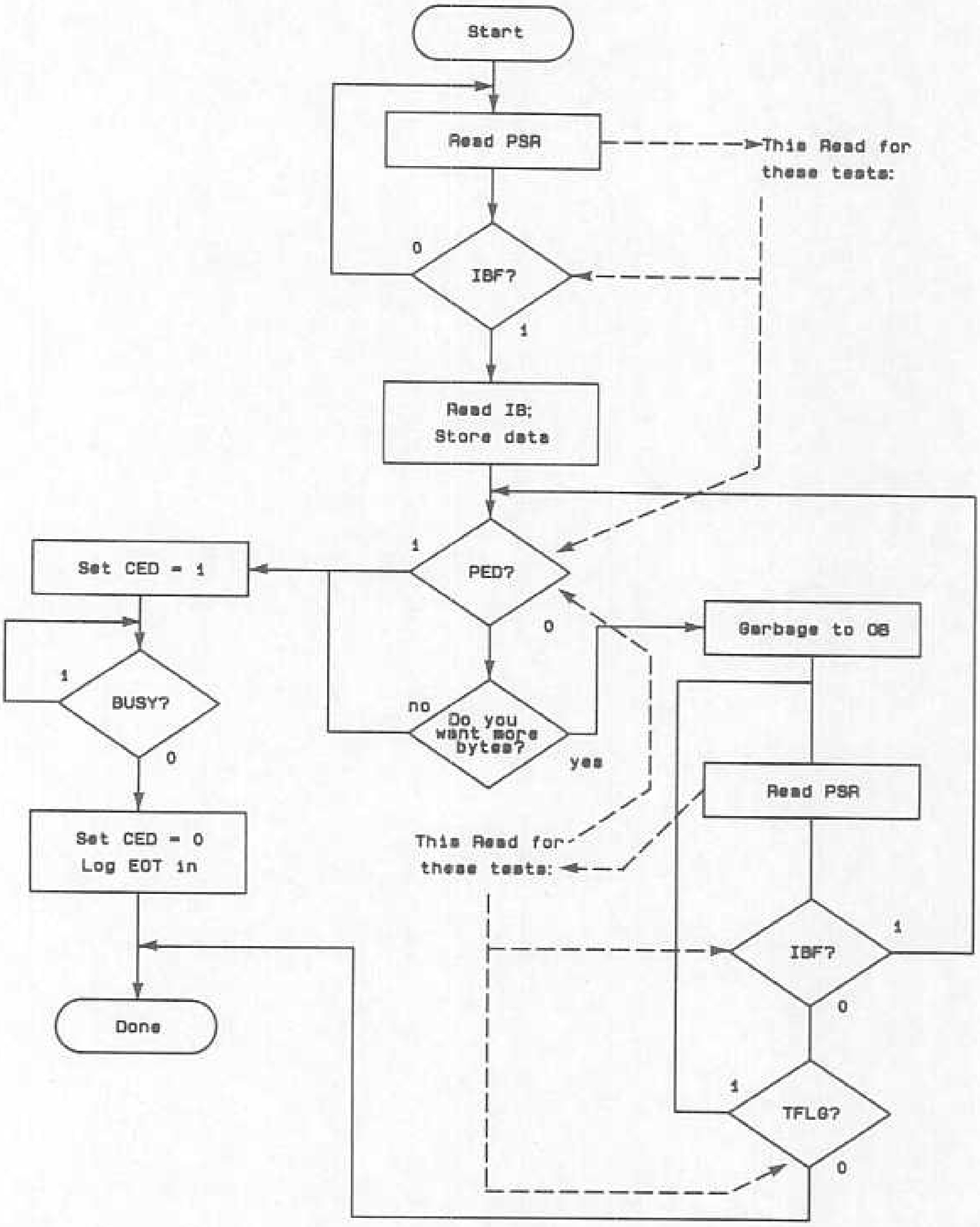

Figure 2-6. Interrupt Input

2.3 I/O Processor Commands

2.3.1 Read Status Ø Ø Ø Ø

The four-bit field is the number of the status register to be read first. Successive reads get consecutive registers. This command implements the STATUS statement.

2.3.2 Input Ø Ø Ø 1

The field is:

MSB	Count Term.	Char. Term.	INTR SIMPLE	IOP Term.	LSB
	3	2	1	Ø	

This opcode is used for both simple input and interrupt input. If bit 1 is "Ø," it's a simple input. If bit 1 is "1," it's an interrupt input. Bit Ø set also indicates that the I/O processor should terminate the input if the interface dependent condition is met (EOI). To terminate an input operation the I/O processor sets the PED bit in the processor status register. If the operation is an interrupt input, bits 3 and 2 may be set to enable two other termination criteria. For bit 3 set the I/O processor will terminate the transfer if the number of bytes transferred is equal to the number stored in control registers 25 and 26 (refer to opcode 1 Ø Ø). For bit 2 set the IOP will terminate the transfer upon receipt of a byte equal to the one stored in control register 27 (refer to opcode 1 Ø Ø). The CPU may terminate the input operation by setting CED = 1.

In fact, the CPU must set CED = 1 if PED = 1. Between the time that the command is received and some termination takes place the I/O processor fetches bytes from the I/O device and sends them to the CPU. For simple input, it does so by putting them into the input buffer because the CPU is waiting to take them out. For an interrupt input, the I/O processor interrupts the CPU with the reason for interrupting being the availability of one or more bytes for transfer in.

2.3.3 Burst I/O Ø Ø 1 Ø

The field is:

MSB	Ø	IOP TERM	EOL OUT	INPUT OUTPUT	LSB
	3	2	1	Ø	

This opcode is for burst I/O, both input and output. Bit Ø indicates input ("1") or output ("Ø"). If it is an output, setting bit 1 will cause the I/O processor's programmed EOL sequence to be sent out at the end of the transfer (otherwise the interface EOI condition will be asserted with the last byte). If the operation is an input, clearing bit 2 allows the I/O processor to terminate the burst if its EOI termination condition is met. The CPU must always give the IOP a byte count (control registers 25 and 26) before a burst operation. After giving the burst command, the CPU enters a very tight infinite loop to transfer data as fast as it can. This burst is always terminated by the I/O processor which interrupts the CPU with the reason for interrupting being burst termination. By tampering with the interrupt service routine's return stack the CPU breaks out of the infinite loop.

2.3.4 Interrupt Control Ø Ø 1 1

The field is Ø. This opcode is a special "no op" command. When protocol commands are passed by interrupting the IOP it sets PACK = 1 and enters an "interrupted" state for the duration of the command execution. The I/O processor will remain in this state (with normal operations suspended) until the CPU declares the command's operation to be complete by strobing the I/O processor's INT bit. For burst operation, the global interrupt disable feature can't be used because the active I/O processor must be able to interrupt the CPU to terminate the burst. Before a burst operation, all resident I/O processors are sent this "no op" and are put into the "interrupted" state by the command handshaking. The interrupt bit of the I/O processor to be used for burst is strobed and that I/O processor "revives" to perform the burst. After the burst, all I/O processors with PACK = 1 are strobed, reviving them (by allowing completion of the "no op" command) to continue with their normal operations.

2.3.5 Interface Control Ø 1 Ø Ø

The field, from Ø to 9, selects one of the 1Ø interface control operations. These are immediate execution with no data involved (except a parallel poll response byte which is placed in the input buffer after that operation).

Table 2-2. Interface Control Fields

Field	Command
Ø	ABORT I/O.
1	Set REN = 1.
2	Set REN = Ø.
3	Set ATN = Ø.
4	Perform Parallel Poll.
5	Send "MY TALK ADDRESS."
6	Send "MY LISTEN ADDRESS."
7	Send EOL character sequence.
8	BREAK I/O.
9	RESUME I/O.

2.3.6 Read Auxiliary Ø 1 1 1

This is a diagnostic not used by the I/O ROM.

2.3.7 Write Control 1 Ø Ø

The field is a five-bit register number. This opcode causes the IOP to wait for data bytes which are to be written into consecutive control registers beginning with the one indicated in the field. The CPU sets the CED bit equal to 1 with the last byte sent. This opcode implements the CONTROL statement. However, there are five control registers (R25 through R29) that the I/O ROM hides from BASIC programmers. These registers are:

R25 - (least significant byte) character count

R26 - (most significant byte) character count

These two bytes contain a 16-bit binary integer which the I/O processor uses to terminate a data transfer by character count.

R27 - Input termination character

This byte is used by the I/O processor as a termination match character when bit 2 is set on an input interrupt command.

R28 - ASSERT byte

The ASSERT operation is performed by writing the byte to be asserted into this control register.

R29 - Service Request Byte

This is the byte that is to be sent to the HP-IB bus if this processor is serially polled.

2.3.8 Output 1 Ø 1 Ø

The field is:

MSB	Ø	Ø	INTR SIMPLE	Ø	LSB
	3	2	1	Ø	

This opcode commands a simple (bit 1 = Ø) or interrupt (bit 1 = 1) output operation. The I/O processor either waits for data bytes to output (simple) or interrupts the CPU with the reason for interrupting being readiness to transmit one or more bytes. In both cases the operation is terminated by the CPU setting CED = 1 in the calculator control register just before the last byte is put into the output buffer.

2.3.9 Send Commands 1 Ø 1 1

The field is Ø. The send commands tell the processor the next bytes should be in command mode as opposed to data mode. This is an HP-IB opcode and causes that interface to handshake the data bytes over the HP-IB bus with ATN = 1 (true).

2.3.1Ø Write Auxiliary 1 1 1 Ø

Like read auxiliary, this opcode is a diagnostic not used by the I/O ROM.

2.4 What Happens When the I/O Processor Interrupts the CPU?

When an I/O processor interrupts the CPU, the first thing the CPU service routine does after identifying the processor is to read the input buffer to let the processor know that the service request is now being handled. The processor then places a byte into the input buffer to tell the CPU why it was interrupted. The recognized values of this byte are:

Table 2-3. IOP Interrrupt Byte

Byte	Interrupt Reason
Ø Ø Ø Ø Ø Ø Ø Ø	Interrupt output.
Ø Ø Ø Ø Ø Ø Ø 1	Burst termination.
Ø Ø Ø Ø Ø Ø 1 Ø	ON INTR condition met.
Ø Ø Ø Ø Ø Ø 1 1	Self-test passed.
Ø Ø Ø Ø Ø 1 Ø Ø	Interrupt input.
Ø Ø Ø Ø Ø 1 1 Ø	Finished EOL sequence.
1 1 1 1 1 Ø 1 1	Self-test failed.
1 1 1 1 1 1 1 1	Invalid I/O operation.
Ø X X X X X 1 1	Interface type dependent error.

Interrupt Output: The IOP is prepared to process one or more output data bytes (during an output transfer by interrupt) and is interrupting to get some data from the CPU.

Burst Termination: The IOP has determined that the current burst operation is finished and the CPU should abandon its infinite loop.

ON INTR Condition Met: Some interrupt condition masked in control register 1 (interrupt mask) has occurred.

Self-Test Passed: The IOP has completed its reset procedure and passed the self-test.

Interrupt Input: Like interrupt output with the IOP indicating it has bytes for the CPU.

Finished EOL Sequence: The IOP has completed sending the EOL character sequence after an interrupt output.

Self-Test Failed: The IOP has completed its reset procedure and has failed the self-test.

Invalid I/O Operation: The IOP cannot execute the protocol commands it has received (for example, a serial interface was sent a parallel poll command).

Interface-Type Dependent Error: The IOP is reporting an interface dependent error. These errors are documented in the I/O ROM manual. Adding 112 (decimal) to bits 6 through 2 in the interrupt byte will give you the error number.

Following is an interpretation of the "reason byte."

Reason byte: X X X X X X 1 1 - "Reports"

These are error reports or the "self-test passed" report. The I/O ROM displays a message for the errors. You need not do anything as far as the IOP is concerned.

Reason byte: Ø Ø Ø Ø Ø 1 1 Ø - "Finished EOL sequence"

Ø Ø Ø Ø Ø Ø 1 Ø - "ON INTR trigger"

These two are also reports in the sense that the IOP is telling the CPU it is finished sending the preprogrammed end-of-line character sequence in the first case and one of the register 1 interrupt mask conditions has been met in the second case. The I/O ROM records an EOL branch indicator for the appropriate ON EOT or ON INTR. You need not do anything.

The above reasons for interrupting don't really obligate you, as the I/O programmer, to do anything besides acknowledge the interrupt in your service routine. Burst termination, interrupt input, and interrupt output do require you to take appropriate action.

Reason byte: Ø Ø Ø Ø Ø Ø Ø 1 - "Burst termination"

You must break the CPU out of its infinite loop. Refer to Burst I/O.

Reason byte: Ø Ø Ø Ø Ø Ø Ø Ø - "Interrupt output"

Ø Ø Ø Ø Ø 1 Ø Ø - "Interrupt input"

See figures 2-5 and 2-6 at the beginning of this section for details of these reason bytes.

PERFORMING I/O OPERATIONS

3.1 Introduction

This section defines and illustrates the operation high-level I/O. It is divided into three parts: flowcharts which illustrate the operation of high-level and utility routines, utilities and sample code, and the description of the steps involved in performing an I/O operation.

3.2 I/O Operation Flowcharts

The flowcharts shown assume that the reset and interrupt facilities are set up.

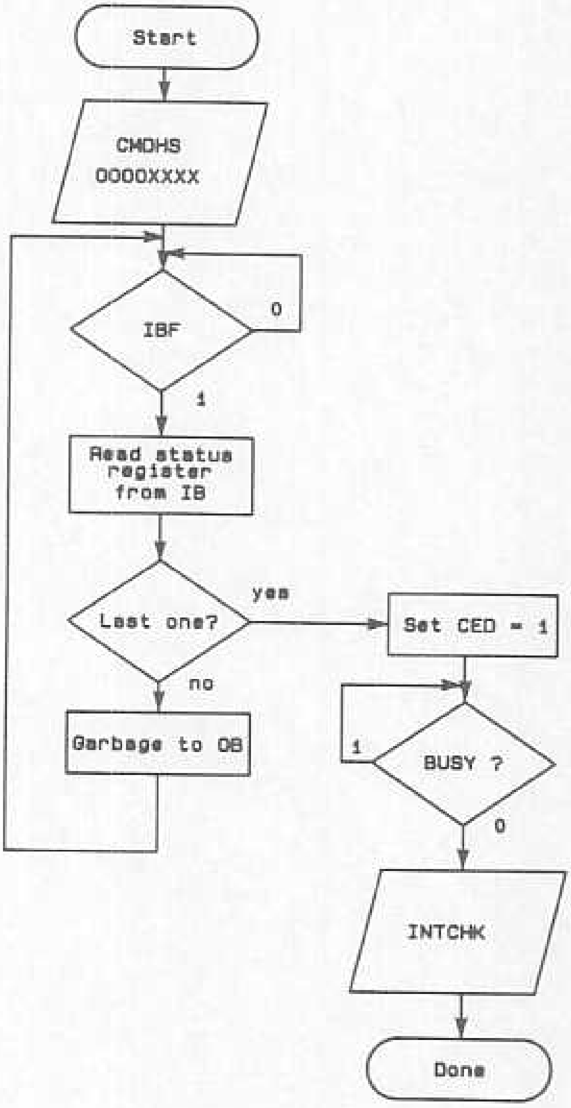

Figure 3-1. Read Status

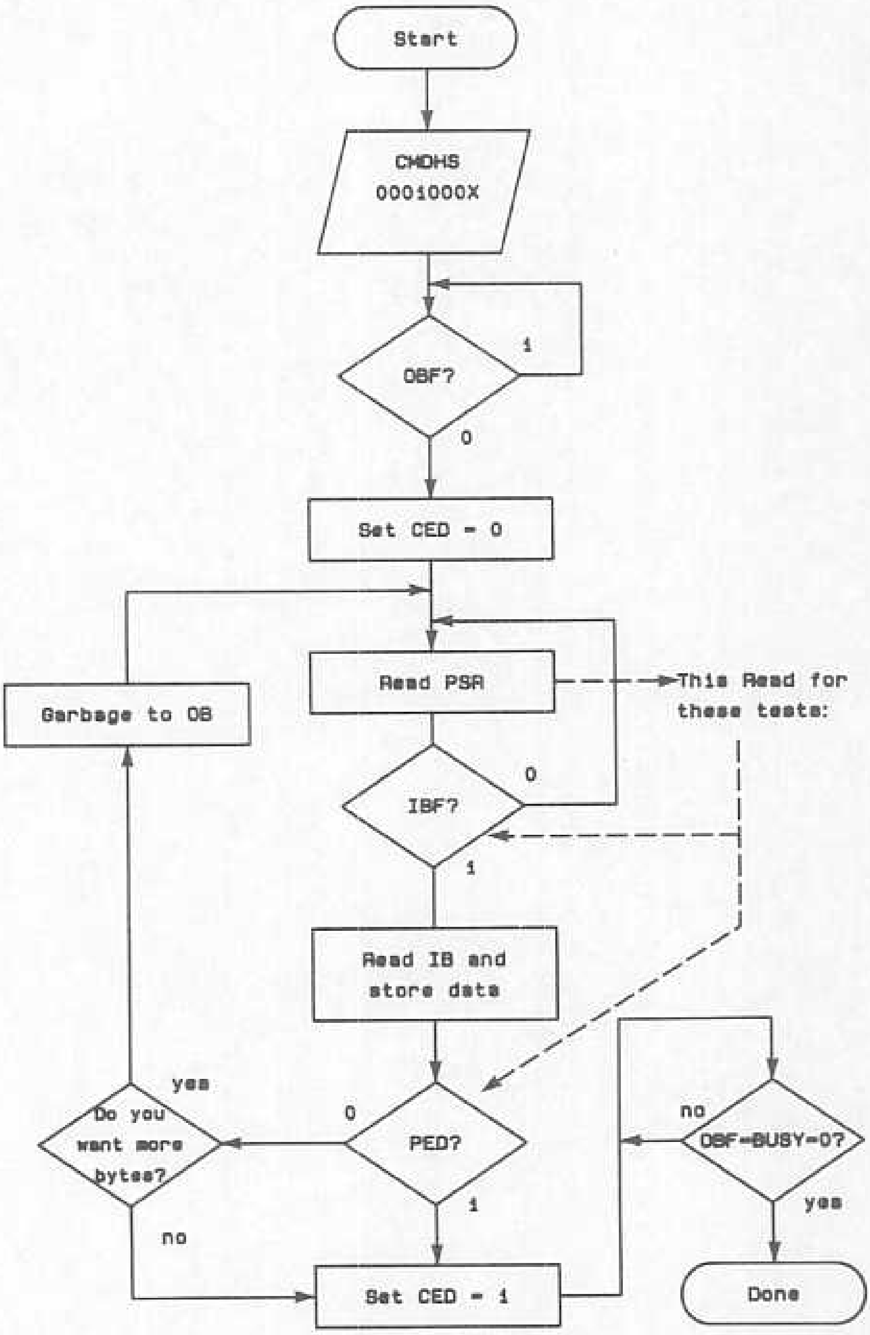

Figure 3-2. Simple Input

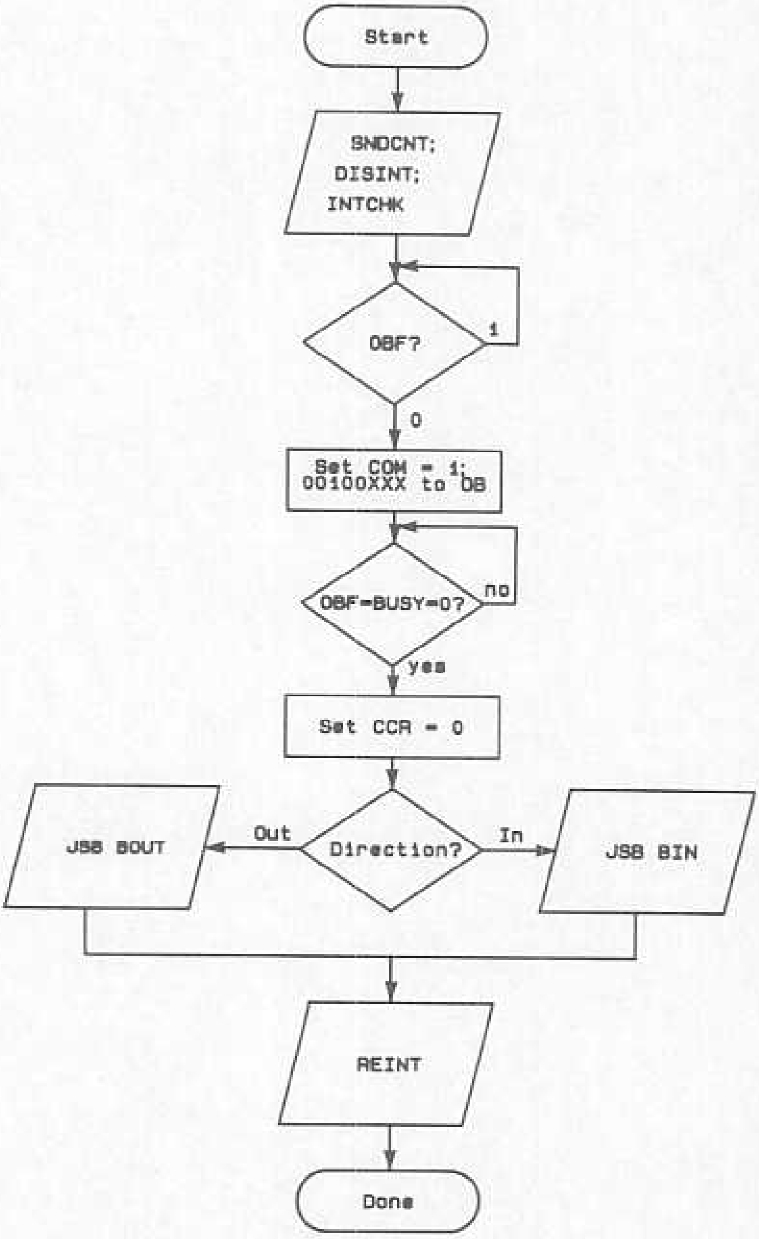

Figure 3-3. Burst I/O

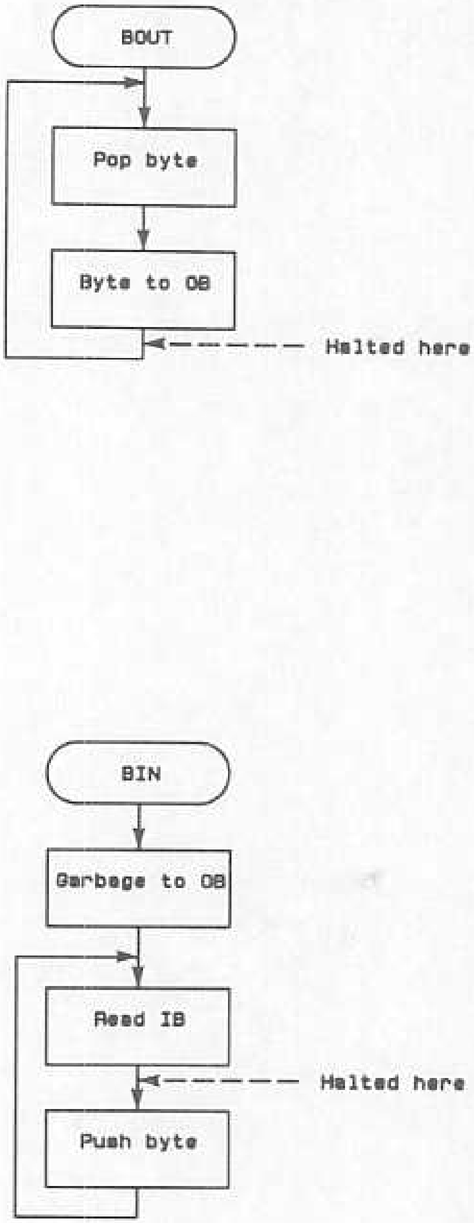

Figure 3-4. Burst Loops

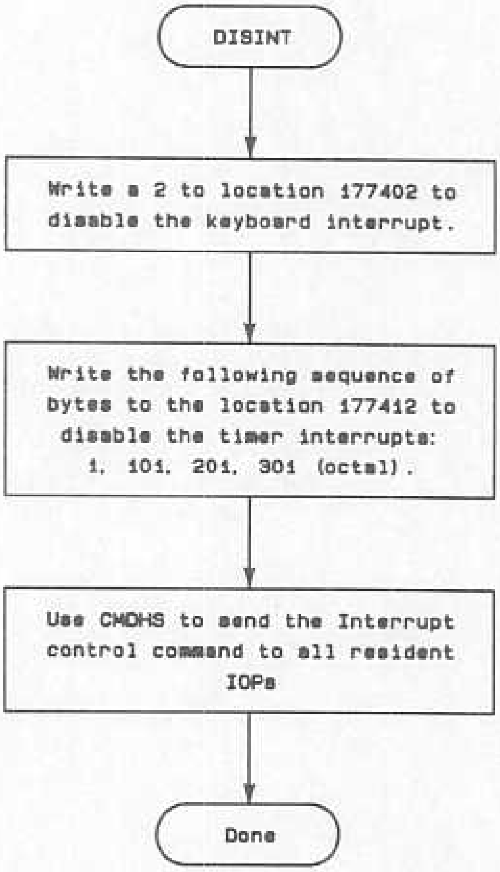
DISINT
Write a 2 to location 177402 to disable the keyboard interrupt.
Write the following sequence of bytes to the location 177412 to disable the timer interrupts: 1, 101, 201, 301 (octal).
Use CMDHS to send the Interrupt control command to all resident IOPs
Done

Figure 3-5. Disable All Interrupts

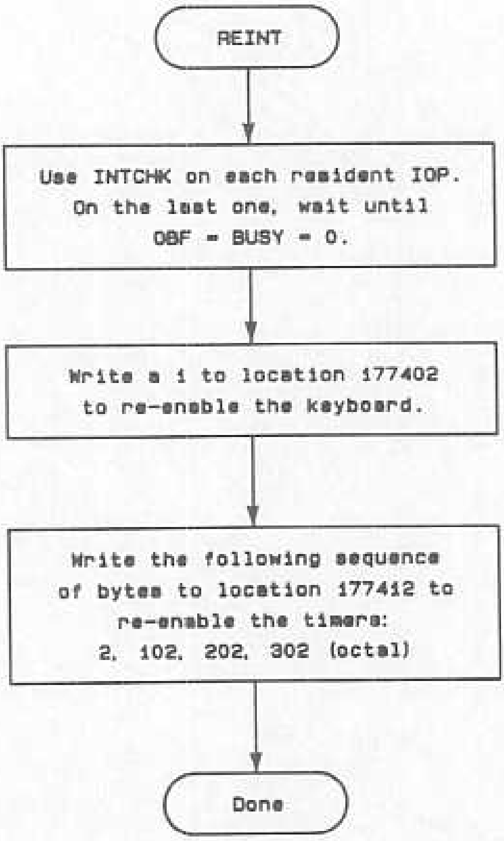
REINT
Use INTCHK on each resident IOP. On the last one, wait until OBF = BUSY = 0.
Write a 1 to location 177402 to re-enable the keyboard.
Write the following sequence of bytes to location 177412 to re-enable the timers: 2, 102, 202, 302 (octal)
Done

Figure 3-6. Re-enable All Interrupts

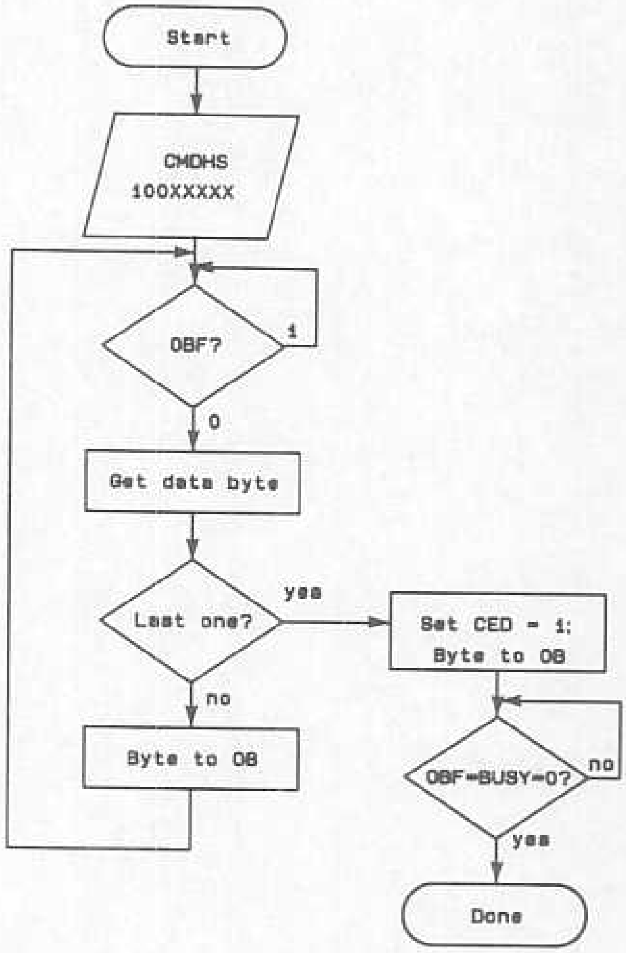

Figure 3-7. Write Control

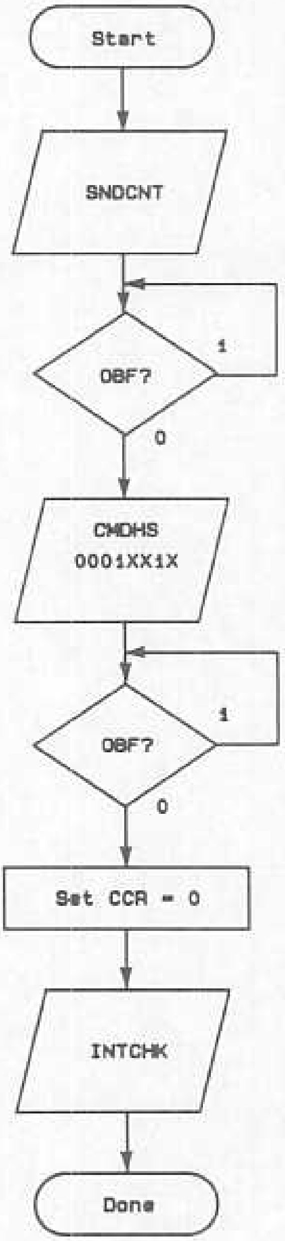

Figure 3-8. Initiate Interrupt Input

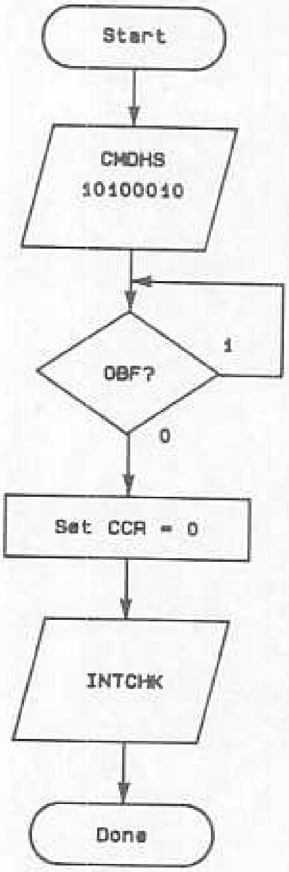

Figure 3-9. Initiate Interrupt Output

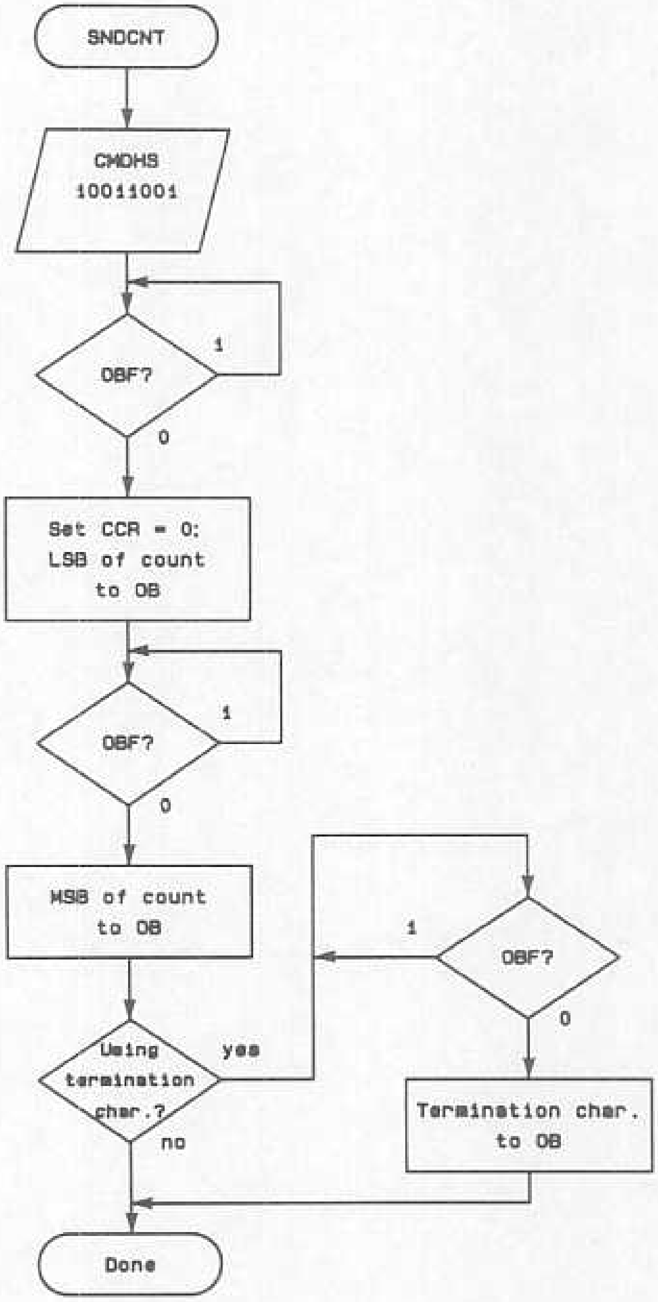

Figure 3-10. Send Byte Count

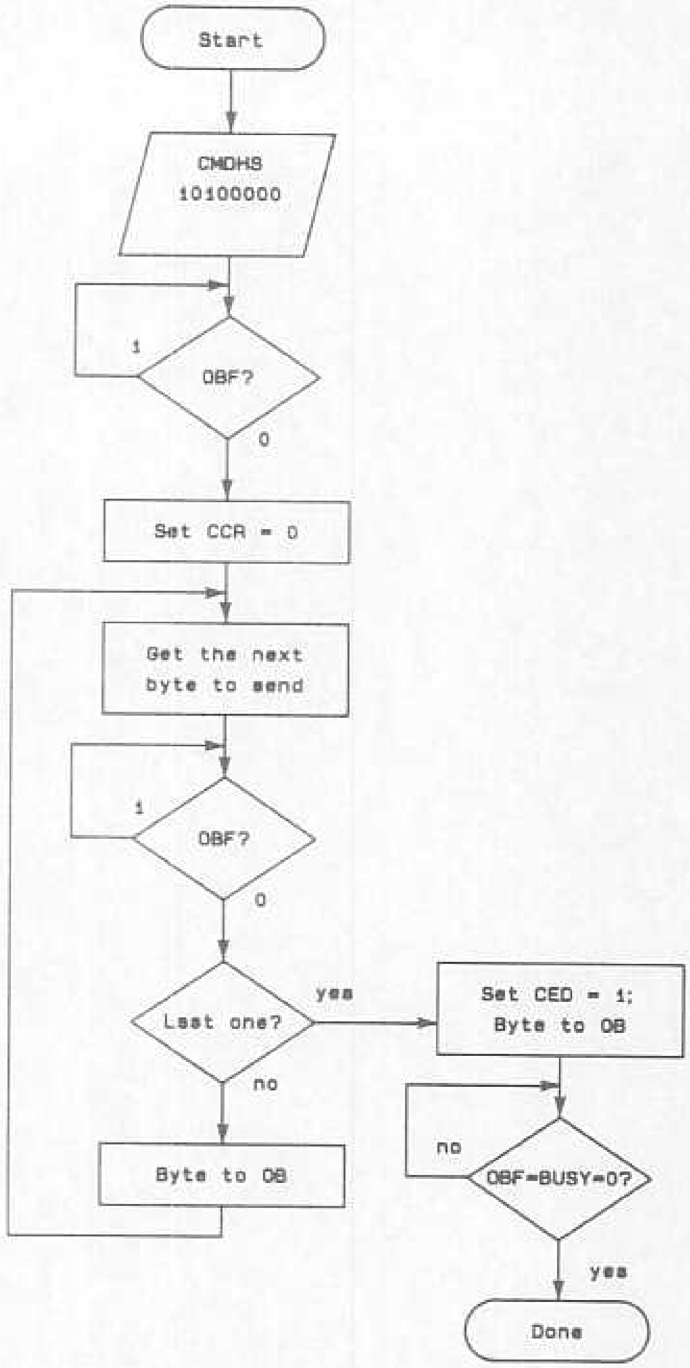

Figure 3-11. Simple Output

3.3 I/O Operations: Utilities and Sample Code

3.3.1 Register Conventions

In the examples of binary code throughout this manual, it is assumed that all data transfers take place between the IOP and the CPU registers. Therefore, the choice of registers, data sources, and sinks is up to you.

The sample code is written for a binary program. If you are going to write ROM-based code, refer to the Assembler ROM manual for a discussion of the changes you should make to convert from the binary program format. To make things easier to understand, a few register conventions are adopted here:

R22,23 Base address (BINTAB or stolen RAM pointer, if you are writing ROM code).

I/O Addresses:

R24,25 Pointer to the IOP calculator control register/ processor status register port.

R26,27 Pointer to the IOP output buffer/input buffer port.

For String Enter or Output:

R3Ø,31 String length.
R32,33 String pointer.

Other:

R35 Command byte for SEND CMD operations (send a bus protocol command, that is, unlisten).
R36 Command byte for I/O protocol commands.
R37 Scratch for flag tests, etc.

3.3.2 Interrupting Versus Noninterrupting IOP Commands

In general, the IOPs should be dealt with in discrete, mutually exclusive operations. One exception to this is when doing interrupt I/O with a full duplex interface or with two or more different interfaces. There are some protocol commands that must be able to operate at once, even if the interface is busy at the time.

They are passed to the IOP by a handshake method which interrupts the IOP from whatever it is currently doing. Because they are either noninterfering (read, status) or benevolently interfering (abort, resume), these interrupting commands are given the privilege of bypassing the normal wait when the IOP is busy.

The best way to handle the different command passing procedures is to simply have two command handshaking routines (one for the interrupting commands and another for direct commands), and two operation termination routines (refer to the DIRCMD, INTCMD, INTCHK, and O=B=Ø utilities). If you are doing interrupt I/O with a full duplex interface, the interrupting commands will allow you to do those operations that will work. In all other cases, the difference between the two types of commands is one of handshaking method, because the CPU is presumably only doing one thing at a time.

We will discuss the I/O operations as individual events with a beginning, a middle, and an end. The handshaking difference shows up in the beginning of an operation as the method of passing the command, and also at the end as the method used to terminate the operation. (Wait until the IOP has finished the operation in the case of direct commands and make sure that the IOP is "uninterrupted" and returned to its presumably interrupted task in the case of the interrupt-type commands). In the following discussions, it is assumed that all operations are discrete (interrupt I/O is discussed separately) and the only task interrupted by an interrupt-type command is the task of idling while waiting for a command.

3.4 Definition of an I/O Operation

An operation is one complete interaction with an IOP. For instance, an input operation includes the configuration and addressing (if needed) as well as the transfer of data. A protocol command is an order to an IOP to execute some part of an operation. A typical data transfer operation will involve a number of protocol commands.

An interrupt-type protocol command involves three stages of execution:

1. The IOP is interrupted by the CPU.
2. The command is given and executed by the IOP (using the CPU if necessary).
3. The IOP then returns to its previous task.

The utility routine INTCMD will interrupt the IOP and pass the protocol command to it. The utility routine INTCHK will uninterrupt the IOP. A direct command is passed to the IOP by the utility routine DIRCMD. This routine will wait until the IOP is not busy and then handshake the command. The IOP does not need to be uninterrupted after a direct command but it is common to wait until the IOP returns to idle after the command by calling the utility routine O-B-Ø.

For purposes of discussion the operations will be grouped into these categories:

- Status and Control.
- Simple Input/Output.
- Primary Addressing and HP-IB Interface Message.
- Miscellaneous Utilities.
- Burst Input/Output.
- Interrupt Operations.

For a discussion of the protocol commands as an instruction set and explanation of bits, ports, and addresses refer to section 2.

3.4.1 Command Handshaking Utility Subroutines

There are a few common waits and handshakes involved in many I/O operations and they are presented here as subroutines which you can include in the binary code you write. These are examples that make the code that follows easier to understand.

These utilities follow the register conventions outlined at the beginning of this section.

Wait until the input buffer is full:

```
   IBF=1   LDBD R37,R24       !READ THE PSR
           JEV IBF=1          !JIF IBF - Ø
           RTN
```

Wait until the output buffer is empty:

```
   OBF=Ø   LDBD R37,R24       !READ THE PSR
           JNG  OBF=Ø         !JIF OBF = 1
           RTN
```

Send the command byte in R36 to the IOP by direct method:

```
    DIRCMD  JSB  X22,0=B=0      !WAIT FOR OBF=BUSY=0
            LDB  R37,=2         !SET CMD BIT TO 1
            STBD R37,R24        ! IN CCR
            STBD R36,R26        !SEND THE COMMAND BYTE TO OB
            JSB  X22,OBF=0      !WAIT TILL THE IOP HAS IT
            CLB  R37            !CLEAR THE CMD BIT
            STBD R37, R24       ! IN CCR
            RTN
```

Wait until the output buffer is empty and the IOP is not busy. This will terminate direct command operations.

```
    O=B=0   LDBD R37,R24        !READ THE PSR
            ANM  R37,=202       !MASK OFF OBF AND BUSY
            JNZ  O=B=0          !JIF THEY'RE NOT BOTH 0
            RTN
```

Send the command in R36 to the IOP by interrupting it:

```
    INTCMD  STBD R37,=GINTDS    !DISABLE ALL INTERRUPTS
            LDB  R37,=1         !SET THE INT BIT TO 1
            STBD R37,R24        ! IN THE CCR

    INTCM1  LDBD R37,R24        !WAIT UNTIL THE IOP SEES IT
            ANM  R37,=10        ! AND ACKNOWLEDGES
            JZR  INTCM1         ! (JIF PACK = 0)
            STBD R37,=GINTEN    !OTHER INTERRUPTS OK NOW
            STBD R36,R26        !STUFF THE COMMAND INTO OB
            LDBD R37,R26        !BE SURE THE IB IS EMPTY
            LDB  R37,=2         !SET CMD BIT & CLEAR INT BIT
            STBD R37,R24        ! IN THE CCR TO START IOP
            JSB  X22,OBF=0      !WAIT TILL IOP GETS COMMAND
            CLB  R37            !CLEAR THE CMD BIT
            STBD R37,R24        ! IN THE CCR
            RTN
```

Check to see if the IOP is busy. If it is, the interrupt bit (INT in the CCR) must be strobed. The test for PACK=1 allows this routine to be called for an IOP which wasn't interrupted in the first place. Use this code to terminate interrupt-type command operations:

```
INTCHK  LDBD R37,R24          !READ THE PSR
        ANM  R37,=10          !MASK OFF THE PACK BIT
        JNZ  INTCH1           !IF PACK = 1
INTRTN  RTN                   !ELSE, IT'S DONE ALREADY
INTCH1  LDB  R37,=1           !STROBE THE INT BIT
        STBD R37,R24          ! IN THE CCR
        JSB  X22, INTRTN      ! (waste some time)
        CLB  R37              ! RESET THE BIT TO 0
        STBD R37,R24
INTCH2  LDBD R37,R24          !NOW WAIT UNTIL PACK = 0
        ANM  R37,=10
        JNZ  INTCH2
        RTN
```

Send the byte in register R35 as a bus command (that is, the equivalent to SEND <s.c.>; CMD <R35>):

```
SNDCMD LDB  R36,=260       !PROTOCOL COMMAND FOR SEND
       JSB  X22,DIRCMD     ! GOES TO THE IOP
       LDB  R37,=4         !ONLY ONE BYTE, SO SET CED
       STBD R37,R24        ! IN THE CCR
       STBD R35,R26        !THE BYTE (DIRCMD DID OBF=0)
       JSB  X22,O=B=0      ! WAIT UNTIL THE IOP IS DONE
       RTN
```

3.4.2 Status and Control Operations

These operations correspond to the STATUS and CONTROL keywords in the I/O ROM and are implemented in assembler code through the I/O protocol commands "Read Status" (opcode = 0000) and "Write Control" (opcode = 1000 or 1001). There are some control registers available to the assembly language I/O programmer which are not directly accessible through the I/O ROM. They will be discussed separately at the end of this section.

The status operation is an interrupt-type command because it might be needed while an interface is busy with an interrupt input or output transfer. Control operations are considered direct commands because they should not be executed while a transfer is in progress. (They change the configuration of the interface.)

Table 3-1. Execution Times (milliseconds)

	HP-IB	Serial	BCD	GP-IO
Assembler	0.9/0.15	0.85/0.3	0.7/0.18	0.65/0.15
BASIC	9/3	11/3	9/3	9/3

These times are given as:

```
<time to do one byte's worth>
-----------------------------
 <time for each extra byte>
```

3.4.3 Status and Control Utility Subroutines

These examples assume that the CPU registers are already set up as shown under Register Conventions. The string length in register pair R30,31 is the number of bytes to be read in the status operation and written in the control operation. In addition, the starting status and control register number is in CPU register R34 and is assumed to be valid.

```
STATUS      LDB  R36,=0          !Status opcode = 0
            ORB  R36,R34         !Starting register is field
            JSB  X22, INTCMD     !Tell the IOP to do status
STAT10      JSB  X22,IBF=1       !Wait till IOP gets a byte
            LDBD R37,R26         !Read the status byte
            PUBD R37,+R32        !Store it
            DCM  R30             !Was that the last one?
            JZR  STAT20          ! (JIF yes - last one)
            STBD R#,R26          ! else ask for another one
            JMP  STAT10          ! and go get it
STAT20      LDB  R37,=4          !We're done so set CED
            STBD R37,R24         ! in the CCR
            JSB  X22,0=B=0       !Wait for IOP to finish up
            JSB  X22,INTCHK      !Uninterrupt the IOP
            RTN

CONTRL      LDB  R36,=200        !Control (opcode = 100)
            ORB  R36,R34         ! field = starting reg. #
            JSB  X22,DIRCMD      !Tell IOP to do control write
CONT10      JSB  X22,OBF=0       !Wait till IOP is ready for
            POBD R36,+R32        ! this next byte.
            DCM  R30             !Is this the last one?
            JZR  CONT20          ! (JIF yes - last one)
            STBD R36,R26         ! otherwise just send the byte
            JMP  CONT10          ! and go for the next one
CONT20      LDB  R37,=4          !On last one set CED
            STBD R37,R24         ! in the CCR
            STBD R36,R26         ! and then send the byte.
            JSB  X22,O=B=0       !Wait till the IOP is done
            RTN
```

The control and status operations differ in various interfaces only in the registers which are implemented by a particular interface. Legal operations correspond to those that are legal to use with the I/O ROM statements: STATUS and CONTROL.

3.4.4 Special Control Operations Not Available With BASIC

There are five control registers in each interface which are not visible to the BASIC programmer. These are control registers 25 through 29. The five registers implement the following four functions:

1. Transfer Count. Before each burst transfer or interrupt input transfer, the IOP must receive a byte count. This is the count that terminates a burst transfer and, among other possible conditions, an interrupt input transfer. This count is specified by writing it to control registers 25 (least significant byte) and 26 (most significant byte).

2. Delimiter Character. Interrupt input transfers can also be terminated by the receipt of a particular byte value. This value is specified by writing it to control register 27. This corresponds to the keyword "DELIM" in the I/O ROM.

3. Assert Byte. The ASSERT operation is performed by writing the byte to be asserted to control register 28. The difference between this operation and a write to control register 2 (they both put the byte into control register 2) is that ASSERT is implemented as an interrupt-type command. Thus, the operation can take place even while the interface is busy.

4. Service Request. The REQUEST operation is performed by writing the response byte to control register 29. This sets up a service request on the HP-IB interface, sends a break over the serial interface and is an error for the BCD and GPIO interfaces.

These special control write operations are distinguished from the normal control write operations by the command handshaking method used. For the Assert and Request operations the handshaking is always interrupt-type. For the writing of byte count and input termination match byte, the handshaking is interrupt-type if the interface is full duplex (FDPX bit in PSR is equal to 1). Otherwise the handshaking is direct-type. These control registers can be accessed using the sample CONTROL code above by making the following two substitutions when interrupt-type handshake is needed: replace DIRCMD with INTCMD and replace O=B=Ø with INTCHK. Because the byte counts are known, the simplified versions are presented below. The assert, response, or termination byte is assumed to be in R34. The count is assumed to be in register pair R34,35.

Send byte count to a full duplex interface:

```
ICOUNT     LDB  R36,=231      !Protocol = write control 25
           JSB  X22,INTCMD    !(full duplex)
           STBD R34,R26       !Send least significant byte
           JSB  X22,OBF=Ø     !Wait till IOP gets first one
           LDB  R37,=4        !This is last, so set CED
           STBD R37,R24       ! in the CCR
           STBD R35,R26       !Send most significant byte
           JSB  X22, INTCHK   !Wait till IOP is done
           RTN
```

Send count to a nonfull duplex interface:

```
DCOUNT     LDB  R36,=231      !Protocol = write control 25
           JSB  X22,DIRCMD    ! (NOT full duplex)
           STBD R34,R26       ! first byte (DIRCMD did OBF)
           JSB  X22,OBF=Ø     !Wait for IOP to get first
           LDB  R37,=4        !Second is last so set CED
           STBD R37,R24       ! in the CCR
           STBD R35,R26       !Second (most significant)
           JSB  X22,O=B=Ø     !Wait till IOP is done
           RTN
```

Send delimiter character to a full duplex interface:

```
ITERM      LDB  R36,=233      !Protocol = write control 27
           JSB  X22,INTCMD    ! (full duplex)
           LDB  R37,=4        !First is last, so set CED
           STBD R37,R24       ! in the CCR
           STBD R34,R26       !Send the byte
           JSB  X22,INTCHK    !Uninterrupt the IOP
           RTN
```

Send delimiter character to a nonfull duplex interface:

```
DTERM      LDB  R36,=233      !Protocol = write control 27
           JSB  X22, DIRCMD   ! (NOT full duplex)
           LDBD R37,=4        !First is last, so set CED
           STBD R37,R4        ! in the CCR
           STBD R34,R26       !Send the byte
           JSB  X22,O=B=Ø     !Wait till IOP is done
           RTN
```

Note that the count and termination character can be specified in one operation by sending the three bytes in order.

Assert the byte in R34 (any kind of interface):

```
ASSERT     LDB  R36,=234     !Protocol = write control 28
           JSB  X22,INTCMD   !This is ALWAYS interrupting
           LDB  R37,=4       !Only one byte so set CED
           STBD R37,R24      ! in the CCR
           STBD R34,R26      !Send the assert byte
           JSB  X22,INTCHK   !Wait till IOP is done
           RTN
```

Request service or break:

```
RQUEST     LDB  R36,=235     !Protocol = write command 29
           JSB  X22,INTCMD   !This is ALWAYS interrupting
           LDB  R37,=4       !Only one byte so set CED
           STBD R37,R24      ! in the CCR
           STBD R34,R26      !Send the response byte
           JSB  X22,INTCHK   !Wait till IOP is done
           RTN
```

3.4.5 Simple Input/Output

These operations perform programmed I/O where the handshaking of transferred data is handled directly by the CPU in the normal execution flow of its binary program. They correspond to but need not be limited to the I/O ROM keywords ENTER and OUTPUT. Multiple concurrent I/O transfers can be performed with these simple operations so long as no interface needs to operate in a full duplex mode. A binary program which manages enough of an interrupt service routine to reset the interfaces at power-on could run four concurrent I/O operations on four interfaces. By polling the input interfaces for input buffer full and the output interfaces for output buffer empty, the CPU can control the data transfers entirely.

The operations described here assume that the interface involved has been configured and addressed as needed (send bus command operations are discussed later). It is assumed that the number of bytes to be transferred is indicated by the string length in R3Ø,31 and the data source or sink is pointed to by the contents of R32,33. Where data comes from and where it goes is, of course, up to the I/O programmer. The transfers to and from all interfaces always involve one or more bytes.

The BASIC language I/O programmer has a large variety of data types and structures that can be specified in ENTER and OUTPUT statements. The keyword code in the I/O ROM translates these data types and structures into a string of bytes before it outputs them to an interface (using identical translation procedure regardless of interface type). They are translated into a string of bytes after they are input for an ENTER operation (again the interface type makes no difference). When coding I/O operations directly in assembly language you must manage your own data formats. The content of the byte or bytes transferred depends upon your application and the type of interface.

In general, the HP-IB interface isn't affected by the content of a given data byte; neither is the serial interface (except for some control codes). The BCD interface requires a restricted set of ASCII symbols in a particular format depending on the configuration of the ports. The GPIO expects only to handle even numbers if you are using one of the 16-bit ports.

Output

For outputs, an interface must be configured (refer to Control), addressed (refer to Send), given the output protocol command, and given a byte or a series of bytes to output with the CED (calculator end data) bit in the CCR set to 1 just before transfer of the last byte. The interface will send its end-of-line character sequence (as specified in control registers 16 through 23) and then go into its nonbusy state (recall that the BCD interface does not have an end-of-line character sequence).

The configuration and addressing need not be repeated before each output operation if you know that it has already been done. The output protocol command must be used before a data byte is output to an interface if the CED bit was set for the previous data byte output to the same interface. You may omit the setting of the CED bit and the sending of the next output protocol command if you keep track of whether or not the interface is busy (the I/O ROM does this to allow the OUTPUT USING # option).

Input

Inputs are similar to outputs in configuration, addressing, and command sequence. There are some added complications involved in the termination of the transfer. The CPU may terminate an input at any time by setting the CED bit in the CCR, similar to the OUTPUT termination.

The IOP may also decide to terminate an input operation by setting the PED bit (processor end data) in the PSR. Whether or not this occurs depends upon the particular interface and the option bits included in the input protocol command. The three option bits are: bit 3 (count), bit 2 (character), and bit Ø (EOI). Bits 7 through 4 are the command opcode (ØØØ1). Bit 1 specifies whether the input is a simple input (bit 1 = Ø) or an interrupt input (bit 1 = 1). We will examine these options for each interface.

HP-IB Input

The HP-IB interface allows you to use any of the three options. If you specify termination by count, you must provide the count (by writing to control registers 25 and 26) before beginning the input operation. The same holds true for the character termination option (you must provide the termination match character by writing it into control register 27). The EOI condition on the HP-IB interface is taken to be the receipt of a data byte (device dependent message) with the END message (EOI) true.

Serial Input

The serial interface also allows you to use any of the three options, but this interface will use the EOI condition whether or not you specify it! For this interface, the EOI condition is an incoming character that matches one of the termination characters specified in the control registers 12, 13, 14, or 15 along with the enabling bits in control register 11. Note that these four termination match characters are in addition to the one that you may or may not have specified as the termination character in special control register 27.

BCD Input

The BCD interface does not use or allow any of these input termination operations. The BCD interface will only accept the protocol command 21 (octal) as the simple input command.

GPIO Input

The GPIO interface allows the count and character termination options and completely ignores the EOI bit. If you are operating in 16-bit mode, the count termination option may be used but the character termination option may not be used.

Besides these specified input termination conditions, the BCD interface will set the PED bit when it has exhausted the bytes needed to transfer the data defined by its current primary address and port configuration. The other interfaces will just keep on sending bytes until the CED bit is set, or one of the enabled and allowed conditions is met.

It is the responsibility of the CPU to recognize the assertion of the PED bit, set CED in response, and send a new input protocol command before asking for additional input bytes.

Execution Speeds for Simple Enter and Output

Execution speeds for simple enter and output operations depend upon external events as well as I/O protocol execution so they will be discussed rather than simply listed. The BASIC execution times for equivalent operations depend heavily on formatting options and will be mentioned but not discussed in detail.

HP-IB Interface

The HP-IB interface requires Ø.3 milliseconds to process an input protocol command and Ø.25 milliseconds to output each data byte (device dependent message). It will process an output protocol command in Ø.4 milliseconds and send each data byte in Ø.16 milliseconds. This means that about 4,ØØØ bytes per second can be input and about 6,ØØØ bytes per second can be output. These times assume that any devices on the HP-IB interface bus are fast enough to keep up with the interface at these speeds.

Serial Interface

The serial interface requires about Ø.5 milliseconds to process either an input or output protocol command. Due to the timing methodology used and the interface, the execution time for the protocol command is lost in the baud rate, FIFO (first in, first out) operations, and external device response. Because 96ØØ baud is the maximum data transfer rate available on the serial interface, this will limit the speeds at which bytes can be transferred to about 96Ø bytes per second (assuming 1Ø bits per character). The CPU has no trouble keeping up with this speed. If you are operating under conditions that guarantee you will be inputting bytes that are already in the FIFO buffer then you can expect to get them out in approximately Ø.3 milliseconds each.

BCD Interface

The BCD interface requires Ø.3 milliseconds to process an input protocol command and Ø.25 milliseconds for each byte actually input. The time for the output protocol command is Ø.6 milliseconds and Ø.25 milliseconds per data byte. Remember that each data byte corresponds to one port digit (in the BCD interface) and that the number of bytes transferred depends upon signs, exponents, and punctuation, as well as the number of port digits involved. The BCD interface will always include a line feed character at the end of each reading where it sets the PED bit.

If we assume that the externally connected device is as fast as the interface, then we can get some "transfers per second" figures. If we just use one digit, we can output one data byte but must input two data bytes (digit and line feed; recall that there is a sign character if you're using the mantissa instead of the function digit as assumed here) so we can expect to get about 1,1ØØ transfers out per second and about 1,2ØØ transfers in per second.

If we look at a large format number with eight mantissa digits and an exponent (which is always three digits as far as the BCD interface is concerned) then we need to output 14 digits and input 15 digits. The result will be about 250 tranfers per second in either direction.

GPIO Interface

The protocol command handshake and the transfer of a 16-bit number (two data bytes) takes one millisecond in either direction using the GPIO interface. For eight-bit format transfers, the command handshaking takes 0.4 milliseconds and each data byte transferred also takes 0.4 milliseconds in either direction. This translates into 1000 transfers per second for 16-bit data and 2500 transfers per second for eight-bit data.

BASIC

The comparable times in BASIC depend upon the overhead required for the IMAGE specifiers. In general, there will be approximately 20 milliseconds for the interpreter and statement set-up. There will also be at least 50 microseconds per character transferred depending upon what kind of transformations are being done to the data. While you can escape this overhead by doing your I/O operations directly from assembler code, you must do something in the way of sourcing and/or sinking data. This will take some time in addition to the time for the CPU-IOP transfer.

3.4.6 Simple Input and Simple Output Utility Subroutines

The examples below assume the correct number of bytes to be transferred is in register pair R3Ø,31 and the source/sink pointer is in register pair R32,33. For input, it is assumed that a valid bit mask for the termination options is available in register R34. It is also assumed that the interface involved has already been addressed as needed and pointers to its CCR/PSR and OB/IB are in register pairs R24,25 and R26,27.

```
INPUT      LDB  R36,=2Ø        !Protocol = simple input
           ORB  R36,R34        ! or in the options
           JSB  X22,DIRCMD     !Tell IOP to do input
INloop     JSB  X22,IBF=1      !Wait till there's a byte
           LDBD R36,R26        ! get it from IOP
           PUBD R36,+R32       ! and sink it
           ANM  R37,=4         !Was PED set?
           JNZ  INPend         ! JIF yes - IOP says "Stop!"
           DCM  R3Ø            !Is sink satisfied?
           JZR  INPend         ! JIF yes - CPU says "Stop"
           STBD R#,R26         ! Request another
           JMP  INlop          ! byte and go to get it
INPend     LDB  R37,=4         !Set CED to declare/confirm
           STBD R37,R24        ! that the operation is over
           JSB  X22,O=B=Ø      !Wait till IOP is all done
           RTN

OUTPUT     LDB  R36,=24Ø       !Protocol = simple output
           JSB  X22,DIRCMD     !Tell IOP to do output
OUTlop     POBD R36,+R32       !Get the next data byte
           JSB  X22,OBF=Ø      !Wait till IOP is ready
           DCM  R3Ø            !Is this the last byte?
           JZR  OUTend         ! JIF yes - time to stop
           STBD R36,R26        ! else byte to IOP and
           JMP  OUTloop        ! go for the next one
OUTend     LDB  R37,=4         !Set CED
           STBD R37,R24
           STBD R36,R26        !Last data byte to IOP
           JSB  X22,O=B=Ø      !Wait till IOP is all done
           RTN
```

Note that any end-of-line character sequence which is set up for the interface will be sent at the end of the output operation by the interface as its response to the setting of the CED bit.

Also note that outputting a series of data bytes is identical in procedure to writing a series of control registers except for the value of the protocol command passed.

3.4.7 Primary Addressing and HP-IB Interface Messages

Primary addressing is an operation that is implied by the use of three or four digits in the device selector in the I/O ROM. The second two of these digits are the primary address portion of the selector (the first one or two digits are the interface select code). For the serial interface, primary addressing has no meaning and causes an error. For the HP-IB interface, the primary address is the HP-IB bus address of the intended data source or destination device. For the BCD and GPIO interfaces, the primary address is the means of choosing among the various partial field options for the BCD and port formats for the GPIO.

The areas of configuration and addressing tend to overlap a bit; a couple of the following operations are accomplished by writing to control registers. What is being established is: the direction of the data flow, and its source or destination as far as the interface is concerned. Because the GPIO and BCD interfaces handle primary addressing in a simple way, we'll look at them first, and then go through the HP-IB interface in some detail.

BCD and GPIO Addressing

Setting the proper direction of data flow for the GPIO and BCD interfaces is a matter of enabling any needed outputs (CONTROL and/or switch settings) and being sure that the BCD digits have been properly assigned. The operation of interest is the passing of the primary address to the interface. This is done by pretending that the interface is an HP-IB interface and sending a "Talk Address" or "Listen Address" interface message.

Where the HP-IB interface will actually send the specified interface message, the BCD and GPIO interfaces set their primary address to the address specified by the "Talk Address" or "Listen Address" message. These two interfaces don't distinguish between "Talk" and "Listen," they just take the address. A "Listen Address" message byte is octal 40 plus the primary address and a "Talk Address" message byte is octal 100 plus the primary address. The example here will select primary address 03 (channel A mantissa and exponent for BCD and eight-bit input B and output D for GPIO).

```
ADDRS3     LDB  R35,=43         !Listen Address 3
           JSB  X22,SNDCMD      !HP-IB interface message util
           RTN or further code.
```

HP-IB Addressing

The HP-IB interface has a lot of addressing requirements. First, there is the distinction between a protocol command and an HP-IB interface message. A protocol command is a byte from the CPU to the IOP that tells the IOP to do something. An HP-IB interface message is a byte from the CPU to be sourced on the HP-IB bus by the IOP with the ATN message true and is supposedly destined for the interface functions in the devices on the HP-IB bus.

The direction of data transfer is declared by sending the HP-IB interface one of the two protocol commands: "Send My Talk Address" (opcode: Ø1ØØ, field: Ø1Ø1, that is, Interface control - 5) or "Send My Listen Address" (opcode:Ø1ØØ, field: Ø11Ø, that is, Interface control - 6). The HP-IB interface gets the HP-IB bus address, puts together the appropriate "Listen Address" or "Talk Address" interface message and sources it onto the HP-IB bus in order that all other devices know (these operations are illegal unless the HP-IB is the active controller currently) whether to source or sink data.

Note that we have sent a protocol command only and the interface has sent an interface message independently. We could have sent the same interface message by first sending the protocol command (opcode and field: 26Ø octal) and then writing the HP-IB command (octal Ø77) to the output buffer.

The HP-IB interface is configured for output if the TA (talker active) state is true and configured for input if the LA (listener active) state is true. The interface maintains these states in accordance with the HP-IB protocol and the interface messages on the bus. This is true whether or not the interface is the active controller.

In addition to setting up the HP-IB interface for the ensuing data transfer, the source or sink(s) on the HP-IB bus need to know whether to talk or listen. This is done by forming the appropriate address command byte (listen: 4Ø octal + address, talk: 1ØØ octal + address) and sending it out as an HP-IB interface message. It is prudent to send the "Unlisten" interface message before a transfer to be sure that any devices left addressed to listen are unlistened. These addressing and unaddressing operations are all done by putting together the appropriate interface message byte and sending it to the IOP using a "Send" protocol command.

3.4.8 Primary Addressing and Interface Message Subroutines

If you're about to output to one or more devices, you should: send "Unlisten" (interface message), send "My Talk Address" (protocol command whose execution includes the sourcing of an interface message), and then send a listen address interface message to each device which is supposed to receive the data that is about to be output. If you're about to input from a device on the bus you should: send "Unlisten" (interface message), send "My Listen Address" (protocol command whose execution includes the sourcing of an interface message) and then send talk address interface message to the source device (interface message). If you want other devices to listen also, just send their listen addresses (interface message) any time after the "Unlisten."

Some examples:

1. Set-up to output to device 5.

```
LDB  R35,=77            !Send Unlisten
JSB  X22,SNDCMD         ! (Interface message)
LDB  R36,=1Ø5           !Send My Talk Address
JSB  X22,DIRCMD         ! (Protocol command)
LDB  R35,=45            !Send Listen Address 5
JSB  X22,SNDCMD         ! (Interface message)
.... now ready to execute code at "OUTPUT."
```

2. Set-up to input from device 5.

```
LDB  R35,=77            !Send Unlisten
JSB  X22,SNDCMD         ! (Interface message)
LDB  R36,=1Ø6           !Send My Listen Address
JSB  X22,DIRCMD         ! (Protocol command)
LDB  R35,=1Ø5           !Send Talk Address 5
JSB  X22,SNDCMD         ! (Interface message)
....now ready to execute code at "INPUT."
```

3. Set up to input from device 5 and have devices 6 and 7 also listen to the data from device 5.

```
LDB  R35,=77           !Send Unlisten
JSB  X22,SNDCMD        ! (Interface message)
LDB  R36,=1Ø6          !Send My Listen Address
JSB  X22,DIRCMD        ! (Protocol command)
LDB  R35,=1Ø5          !Send Talk Address 5
JSB  X22,SNDCMD        ! (Interface message)
LDB  R35,=46           !Send Listen Address 6
JSB  X22,SNDCMD        ! (Interface message)
LDB  R35,=47           !Send Listen Address 7
JSB  X22, SNDCMD       ! (Interface message)
.... now ready to execute code at "INPUT."
```

Table 3-2. Execution Times

Primary address to BCD:	Ø.53 milliseconds.
Primary address to GPIO:	Ø.58 milliseconds.
Interface message to HP-IB:	Ø.7 milliseconds for the first byte and Ø.35 milliseconds for each additional byte.
Send "My Talk Address" and "My Listen" Address:	Ø.45 milliseconds.

Interface messages in general are another consideration when using the HP-IB interface (they also have some limited uses with the GPIO interface as listed in the I/O ROM manual under the SEND statement). In addition to device addresses, there are interface messages which select secondary addresses and perform assorted operations (trigger, clear, poll). These interface messages are sent from the CPU to the IOP in exactly the same way as data bytes for output except that the protocol command used is octal 26Ø (send) instead of octal 24Ø (output data). The utility routine SNDCMD that is used in primary addressing and other places is a special case of the send operation where there is only one byte to be sent as an interface message.

The following example will trigger the device at HP-IB bus address 11. Note that all three messages involved (Unlisten, Listen Address 11, Group Execute Trigger) are interface messages and the HP-IB interface doesn't need to be configured for input or output because there won't be any actual input or output of data.

```
TRGR11      LDM  R65,=77,53,1Ø  !The three messages
            LDB  RØ,=65         !Pointer to the messages
TRIG1p      LDB  R35,R*         !Get next message
            JSB  X22,SNDCMD     !Send it
            ICB  RØ             !Point to next one
            CMB  RØ,=7Ø         !Done them all?
            JNZ  TRIG1p         ! JIF no - not yet
            RTN                 ! otherwise, done
```

This same operation could have been done by supplying the three interface messages as a three-byte string to a routine which is identical to the one labeled "OUTPUT" except that where the protocol command simple output (octal 24Ø) is used in "OUTPUT" you would use the send protocol command (octal 26Ø) instead.

3.4.9 Miscellaneous I/O Utilities

There are four protocol commands which perform utility functions on almost all of the interfaces: Abort, Halt, Resume, and Send End-of-Line Character Sequence (the GPIO interface won't accept Resume and the BCD interface won't accept Resume or Send EOL). There are five additional protocol commands which perform utility functions on the HP-IB interface only. Two of these, (Send "My Talk Address" and Send "My Listen Address") were discussed in Primary Addressing. The other three, (Set REN True, Set REN False and Parallel Poll) will be discussed here.

These protocol commands are sent to the IOP using one of the command handshaking utility routines (there is no transfer of data with the exception of the parallel poll operation which returns a response byte) and the action is done when the command handshake routine has returned. Some of these are interrupt-type commands (Abort, Halt, Resume, and the two REN operations) and the others are direct-type commands. The difference shows up in the protocol command handshake used and the method of waiting for the IOP to say it's finished with the operation.

The specific operations performed by the four general utility protocol commands are those explained in the I/O ROM manual under ABORTIO, HALT, RESUME, and SEND;data...EOL. For this last one, it is the "EOL" that is performed by the utility protocol command. The "SEND;data..." part is at your option on the assembly language level and would have been done according to the "OUTPUT" operation just described.

```
ABORT      LDB  R36,=100        !Protocol - Abort
           JSB  X22,INTCMD      ! (interrupting type)
           JSB  X22, INTCHK     !Uninterrupt the IOP
           RTN

HALT       LDB  R36,=110        !Protocol = Halt
           JSB  X22,INTCMD      ! (interrupting type)
           JSB  X22,INTCHK      !Uninterrupt the IOP
           RTN

RESUME     LDB  R36,=111        !Protocol = Resume
           JSB  X22,INTCMD      ! (interrupting type)
           JSB  X22,INTCHK      !Uninterrupt the IOP
           RTN

SNDEOL     LDB  R36,=107        !Protocol = Send EOL sequence
           JSB  X22,DIRCMD      ! (direct type command)
           JSB  X22,O=B=0       !Wait till IOP is finished
           RTN
```

The following examples show usage of the three HP-IB utilities that were not discussed in Primary Addressing. The first two allow the I/O programmer to set the "Remote Enable" interface single line message (REN line) true or false.

```
REMOTE     LDB  R36,=101        !Protocol = set REN True
           JSB  X22,INTCMD      ! (interrupting type)
           JSB  X22,INTCHK      !Uninterrupt the IOP
           RTN

LOCAL      LDB  R36,=102        !Protocol = set REN False
           JSB  X22,INTCMD      ! (interrupting type)
           JSB  X22,INTCHK      !Uninterrupt the IOP
           RTN
```

This last utility performs a parallel poll operation on the HP-IB interface bus. It is assumed that all required parallel poll configuring operations have been done using the send operation to handshake the appropriate interface messages. The response byte which came in from the parallel poll operation (ATN and EOI both set true and then the data lines read) will be returned by this example in R34.

```
PPOLL     LDB  R36,=104       !Protocol = parallel poll
          JSB  X22,DIRCMD     ! (direct type)
          JSB  X22,IBF=1      !Wait till response is ready
          LDBD R34,R26        !Get the response byte
          JSB  X22,O=B=0      !Wait till IOP is done
          RTN
```

Table 3-3. Execution Times for HP-IB Interface Operations

Set REN true or false	0.66 milliseconds
Parallel poll	0.43 milliseconds

Execution times for sending the EOL sequence are shorter than those for outputting data bytes because the IOP already has the bytes to send. For the serial interface, this doesn't matter because the baud rate determines the speed. The BCD interface doesn't have such a sequence. For HP-IB and GPIO figure 0.7 milliseconds to handshake the command and 0.05 (HP-IB) or 0.15 (GPIO) milliseconds per character sent.

3.4.10 Burst Input/Output

Burst I/O is the fastest method available for data transfer and is also the most restricted in terms of handshakes and formats. There is considerable overhead required to set up and terminate a burst transfer so it shouldn't be used for very short strings of data.

Burst I/O corresponds to the TRANSFER FHS statement in the I/O ROM. Recall that the serial interface does not support this kind of transfer and that severe restrictions apply in the cases of the BCD and GPIO interfaces.

Prior to a burst transfer you must configure and address the appropriate interface as discussed under Primary Addressing. Termination of a burst transfer is discussed with interrupt service routines and will be handled by the I/O ROM or the Mass Storage ROM (if either one is present). We will assume for now that a ROM is handling the termination.

It is the IOP and not the CPU that decides when the time for transfer termination has arrived; the I/O programmer, therefore, must be sure to let the IOP know the proper criteria for it. The two possible criteria are: the specified number of bytes has been transferred, and the interface is an HP-IB. The transfer must be an input and a device dependent message has been accepted with the EOI line true (that is, it was an end message). The count is sent to the IOP before the burst using the technique discussed in Special Control Operations Not Available With BASIC. The EOI option is selected by a bit in the protocol command which will be discussed next.

The protocol command that tells the IOP to execute a burst transfer is a direct command. The opcode is ØØ1Ø and the field is Ø, bit 2, bit 1, and bit Ø.

Bit 2 - Set for an input burst if you don't want the EOI condition to terminate the burst. This is a disable bit (unlike all the others). If you want the transfer to be able to terminate on receipt of an EOI as well as upon exhaustion of the byte count, leave this bit clear (Ø). Bit 2 has no meaning for output bursts.

Bit 1 - Set for an output burst if you want the interface to send the end-of-line character sequence after it has finished the burst output. If bit 1 is set, and the interface is an HP-IB interface, and it has the number 128 decimal in control register 16 (EOI enabled, character count = Ø) then the END message will be sent true along with the last character in the burst (that is, it will be sent as an END message rather than a data byte message).

If the interface sends an EOL character sequence, it will do so after it has interrupted the CPU to break it out of its burst loop. If you leave this bit clear and the interface is an HP-IB, it will send the last byte as an END message and will not send the EOL character sequence regardless of the contents of control register 16. The BCD interface either has (bit is set) or does not have (bit is clear) an EOL sequence.

Bit Ø - This bit indicates the direction of the burst. If the bit is set to 1, then an input transfer will be done. If the bit is clear (Ø) then an output transfer will be done.

The four valid versions of the burst protocol command in terms of each interface are:

Ø44 (octal): Output with no EOL character sequence at the end. The HP-IB interface will send the last byte as an END message. The BCD and GPIO interfaces will simply terminate the transfer.

Ø46 (octal): Output using the EOL character sequence at the end of the transfer. The HP-IB interface will send the EOL sequence if it is one or more characters long. If it is zero characters the EOI enable bit is set (control register 16 = 128 decimal) then the last byte will be sent as an END message as though this were protocol command 44. If control register 16 contains a Ø, the HP-IB interface will not do anything at the end of the burst except terminate it. The BCD interface doesn't have an EOL sequence. The GPIO interface will send its EOL character sequence after it has terminated the transfer.

Ø41 (octal): Input with termination upon receipt of an END message or exhaustion of the byte count on the HP-IB interface. The BCD and GPIO interfaces regard this protocol command as an error.

Ø45 (octal): Input with termination on exhaustion of byte count only. This is the normal protocol for burst input. All three interfaces will simply transfer as many bytes as were requested and then terminate the transfer.

Once the interface is properly configured, addressed, and supplied with the appropriate protocol command the CPU must prepare for burst operation and then enter a burst loop by a jump (JSB) to it. When the transfer is finished the CPU will return to the code following this jump instruction.

3.4.11 Burst-In and Burst-Out Utility Subroutines

Assume the same CPU register conventions as in previous examples: R30,31 is the byte count, R32,33 is the buffer pointer and R22 through R27 contain the base address, CCR/PSR address, and the OB/IB address. Assume also that R34 contains your choice of burst commands as described previously.

```
BURSTT     JSB  X22,SCOUNT      !Send the byte count
           JSB  X22,DISINT      !Disable all interrupters
           JSB  X22,OBF=Ø       !Wait till the OB is empty
           LDB  R37,=2          !Set the CMD bit
           STBD R37,R24         ! in the CCR
           STBD R34,R26         !Write the burst command
           JSB  X22,O=B=Ø       ! then wait for not busy
           CLB  R37
           STBD R37,R24         !Set the CCR to Ø
           STMD R26,=TEMP2      !Prepare OB index address
           JSB  X22,BOUTSB      !Go do the burst (magic RTN)
           JSB  X22,REINT       !Undo the DISINT above
           JSB  X22,O=B=Ø       !Wait till IOP is done
           RTN

BOUTSB     DRP  37              !Data bytes go through here
           ARP  32              ! with this stack pointer
BOUTLP     POBD R#,+R#          !Get next byte to send
           STBI R#,=TEMP2       !Send it (this halts the CPU)
           JMP  BOUTLP          !Repeat apparently forever!

BURSTN     JSB  X22,SCOUNT      !Give byte count to IOP
           JSB  X22,DISINT      !Disable all interrupters
           JSB  X22,OBF=Ø       !Wait till OB empty
           LDB  R37,=2          !Set the CMD bit
           STBD R37,R24         !in the CCR
           STBD R34,R26         !Write the burst command
           JSB  X22,O=B=Ø       ! then wait for not busy
           CLB  R37
           STBD R37,R24         !Set the CCR to Ø
           STMD R26,=TEMP2      !Prepare OB index address
           JSB  X22,BINSUB      !Go do the burst (magic RTN)
           JSB  X22,REINT       !Turn interrupters back on
           JSB  X22,O=B=Ø       !Wait till IOP is done
           RTN

BINSUB     DRP  37              !Data bytes pass here
           ARP  32              ! using this stack pointer
           STBI R#,=TEMP2       !Signal to start up the IOP
BINLOP     LDBI R#,=TEMP2       !Read a byte (halt til IBF)
           PUBD R#,+R#          !Put it into the buffer
           JMP  BINLOP          !Repeat apparently forever!
```

Here are the utility routines called from the burst examples. SCOUNT sends the byte count to the IOP. DISINT disables all interrupting devices so the burst operation will not be interrupted (global interrupt disable is not available for this because the IOP must interrupt the CPU in order to terminate the burst). REINT re-enables all interrupting devices to restore normal operation after a burst is finished.

```
SCOUNT     LDB  R36,=231        !Protocol = write control 25
           JSB  X22,INTCMD      ! (interrupting type)
           STBD R3Ø,R26         !Least significant byte
           JSB  X22,OBF=Ø       !Wait till IOP has it
           STBD R31,R26         !Most significant byte
           JSB  X22,OBF=Ø       !Wait till IOP has it
           RTN

DISINT     LDB  R37,=2          !Disable keyboard interrupts
           STBD R37,=KEYDIS     ! (DAD 1774Ø2)
           LDB  R37,=1          !Disable the first timer
           STBD R37,=TIMIS      ! (DAD 177412)
           LDB  R37,=1Ø1        !second timer
           STBD R37,=TIMDIS
           LDB  R37,=2Ø1        !third timer
           STBD R37,=TIMDIS
           LDB  R37,=3Ø1        !last timer
           STBD R37,=TIMDIS
           |    |    |
           |    |    |
```

Note: the next operation (disabling the IOPs from interrupting) must be done to each IOP which is present (including the one which is going to do the burst; it's part of the command handshaking for burst). If you know which IOPs are present, they can be disabled individually. In this example, we assume that the I/O ROM is present (or the Mass Storage ROM or the Plotter/Printer ROM) and that the system RAM variable byte which we'll call SCLOG (DAD 1ØØ667) has a bit set for each select code (IOP) present. Do not just do all eight possible select codes (because of the lack of handshaking from nonresident select codes). This note applies also to the REINT operation that follows.

```
           |    |    |
           |    |    |
           PUMD R24,+R6         !Save pointers to the
           PUMD R26,+R6         ! bursting IOP
           LDM  R24,=12Ø,377    !Start at select code 3
           LDM  R26,=121,377
           LDB  R2Ø,=1Ø         !There are 8 select codes
           LDBD R21,=SCLOG      !Get the presence indicator
DISTØ1)    |    |    |
           |    |    |
DISTØ1     TSB  R21            !Is this one here?
           JEV  DISTØ2         !JIF No - not this one
           LDB  R36,=6Ø        !Protocol = Interrupt control
           JSB  X22,INTCMD     ! (interrupting type)
DISTØ2     LRB  R21            !Set up for next select code
           ADM  R24,=2,Ø
           ADM  R26,=2,Ø
           DCB  R2Ø            !Have we tried all 8 select codes?
           JNZ  DISTØ1         !JIF No - try the next
           POMD R26,-R6        ! else restore pointers to the
           POMD R24,-R6        ! IOP we're going to use
           JSB  X22,INTCHK     ! and uninterrupt it
           RTN

REINT      PUMD R24,+R6        !Save pointers to the IOP
           PUMD R26,+R6        ! we've bursted with
           LDB  R2Ø,=1Ø        !There are 8 select codes
           LDBD R21,=SCLOG     !This tells which ones to do
           LDM  R24,=12Ø,377   ! start with select code 3
           LDM  R26,=121,377
RENTØ1     TSB  R21            !Is this select code present?
           JEV  RENTØ2         !JIF No - not this one
           JSB  X22, INTCHK    ! else uninterrupt it
REINØ2     LRB  R21            !Set up for next select code
           ADM  R24,=2,Ø
           ADM  R26,=2,Ø

           DCB  R2Ø            !Have we done all 8 of them?
           JNZ  REINØ1         !JIF No - go for the next one
           POMD R26,-R6        !Restore pointers to IOP we've
           POMD R24,-R6        ! just done burst with
           LDB  R37,=1         !Re-enable the keyboard
           STBD R37,=KEYDIS
           LDB  R37,=2         !And the timers
           STBD R37,=TIMDIS
           LDB  R37,=1Ø2
           STBD R37,=TIMDIS
           LDB  R37,=2Ø2
           STBD R37,=TIMDIS
           LDB  R37,=3Ø2
           STBD R37,=TIMDIS
           RTN
```

3.4.12 Burst Command Protocol

Before proceeding to burst termination, let's take a look at the protocol command handshaking that's used with burst I/O because it's both direct and interrupting.

As already mentioned, each interrupting device must be disabled prior to a burst operation and re-enabled afterward. The keyboard and the timer will be disabled and re-enabled according to the examples. The procedure for handling IOPs is explained next.

To prevent an IOP from interrupting the CPU, the CPU sends the protocol command Interrupt Control (opcode ØØ11, field is Ø, which is equal to 6Ø octal). There is an interrupting command handshake procedure sent to each IOP currently on the bus (could be up to four, one for each I/O backplane slot currently in use for burst I/O). The IOP interprets this command as a no operation command and goes through all the motions of accepting an interrupting protocol command from the CPU but does nothing about I/O in response.

During this time, the IOPs cannot interrupt the CPU; therefore, the IOP interrupts are said to be disabled, and the burst transfer will be protected. The last thing the DISINT procedure does is execute the INTCHK procedure on the current IOP. The other IOPs will remain in the interrupted state throughout the burst transfer.

The IOPs can continue their previous operations after the CPU has executed the INTCHK operation. This is the normal termination to an interrupting protocol command.

3.4.13 Burst Execution Speed

Data transfer rates are approximately 25K bytes per second for the HP-IB interface and 2ØK bytes per second for the BCD and GPIO interfaces.

3.4.14 Interrupting Operations

We have seen interrupt-type protocol commands that involve the IOP being interrupted by the CPU. There is a set of operations which involve the CPU being interrupted by the IOP and this is referred to as "Interrupting Operations." Because these operations are involved in more than just input and output data transfers, they will be discussed from the standpoint of the reason the IOP is interrupting. These discussions will include the action to be taken by the CPU interrupt service routine. Following this will be a discussion on the general requirements for this service routine and how and when you can let some of the enhancement ROMs work for you.

The shell of the CPU interrupt service routine will find out which IOP interrupted and why. The reasons for interrupting are each discussed here. The binary number shown with the name of each reason is the byte that explains why the IOP interrupted the CPU.

Interrupt Output Ready (ØØØØ ØØØØ)

This is the IOP interrupting to get the next byte during a transfer out by interrupt. The IOP is ready to output another byte. The procedure is to find the next byte that should be output to that select code and write it to the OB.

Beware of multiple byte operations and termination of the transfer. If the interface is BCD or GPIO doing 16-bit format, the TFLG bit in the PSR (bit 6) will be set and you will be expected to transfer bytes until TFLG is clear. (The interfaces interrupt for each handshake operation and will accept as many bytes per interrupt as are needed to set up the next output. The number of bytes is determined by the interface.)

Termination of the transfer is done by the CPU when it decides that the transfer is completed (that is, it doesn't want to be interrupted for a "next" output operation). This is done by setting the CED bit in the CCR before writing the last byte to the output buffer just as is done in simple output. The difference is in the response of the IOP. It will execute an end-of-line character sequence (if it has one) and will then interrupt the CPU one last time to verify that the end-of-line character sequence has been sent (even if no characters were transmitted).

```
INTOUT                        !Procedure pointer and count for this transfer
                              ! R3Ø,31 <=count, R32,33<= pointer

INTTØ1      DCM R3Ø           !Is this the last byte?
            JNZ INTTØ2        !JIF No - not last byte yet
            LDB R37,=4        ! otherwise, set the CED bit
            STBD R37,R24      ! in the CCR
INTTØ2      POBD R37,+R32     !Get the next byte to send
            STBD R37,R26      ! and give it to the IOP

                              !Update the pointer and count now in case
                              !   TFLG is Ø

            JSB  X22,OBF=Ø    !Wait till the IOP takes it
            LDBD R37,R24      !Does the IOP want another?
            ANM  R37,=1ØØ     ! (that is, is TFLG set?)
            JNZ  INTTØ1       !JIF Yes - get another one
            JMP  common end of Interrupt Service Routine
```

Burst Termination (ØØØØ ØØØ1)

This is the IOP interrupting to terminate the burst operation (in this case the next instruction in the burst loop). It is on the R6 stack along with other entries put there as part of the interrupt service routine. This section of the interrupt service routine performs burst termination as described next.

The location of the return address (the "interrupted address," that of the burst loop) is a known distance down the R6 stack. Servicing this interrupt amounts to finding that address and replacing it with a special address in system ROM. This address is that of a RTN instruction in a system routine.

When the common code at the end of the interrupt service routine has cleaned up and returns (to what would normally have been the next instruction), CPU control passes to the special system address and executes an additional RTN instruction. This additional RTN is what does the return operation from the burst loop to the code following the jump (JSB) into the burst loop.

```
    BRSTRM        LDM  R3Ø,=31Ø,Ø      !The special address
                  SBM  R6,=DSTNCE      !Point to return address
                  STMD R3Ø,R6          !Replace the address
                  ADM  R6,=DSTNCE      !Restore R6 to original
                  JMP  Common ISR end code
```

Register 1 Condition Met Interrupt (ØØØØ ØØ1Ø)

When you have written an interrupt mask to control register 1 of an interface and the masked condition is met, the interface will interrupt the CPU with this condition. What you do about it is pretty much application dependent. For example, we'll assume that you at least want to read status register 1 to see what the condition was and clear the occurrence.

```
    REGST1         LDB  R36,=1         !Protocol = Read Status 1
                   JSB  X22,INTCMD     ! (interrupting type)
                   JSB  X22,IBF=1      !Wait for IOP to get it
                   LDBD R36,R26        !Read it
                   LDB  R37,=4         !Set CED
                   STBD R37,R24        ! to say that's all
                   JSB  X22,O=B=Ø      !Wait till IOP's got it
                   JSB  X22,INTCHK     !Uninterrupt the IOP

                                       !Take whatever action is appropriate
                                       !to flag the occurrence of this
                                       !interrupt.

                   JMP Common End of ISR code
```

Reset Finished ... Self-Test Passed (ØØØØ ØØ11)

The IOP interrupts the CPU after it has completed the reset operation (RESET statement, power-on initialization, or you did the reset in assembly code). This interrupt occurs when the self-test is successfully passed. If it is at power-on, then the select code should be logged in (refer to SCLOG in the discussion of burst--disabling all interrupters), otherwise you don't need to do anything here. Whether or not it is at power-on is something you must flag in the RAM area if you are going to handle this procedure.

```
RESTOK      LDBD R37,X22,PWRON?   !Is this power on time?
            JZR  RSTrtn           ! JIF no - not power on
            LDBD R24,=SCLOG       ! otherwise, log it in
            LDB  R35,=1           !Tentative select code 3
RSTKlp      CMM  R24,=12Ø,377     !Right select code?
            JZR  RSTmch           !JIF Yes - this one
            SBM  R24,=2,Ø         ! else bump to next
            LLB  R35              ! select code
            JMP  RSTKlp           !  try again
RSTmch      ORB  R34,R35          !Set this bit into log
            STBD R34,=SCLOG       ! byte and put it back.
RSTrtn      RTN
```

Reset Not Finished ... Self-Test Failed (xxxx xx11)

This is how the IOP notifies the CPU of an error condition. There are three main types. If the reason for interrupting is 1111 1Ø11, then the IOP is reporting self-test failure in response to a reset operation (just like RESTOK except it flunked the test). If the byte is 1111 1111, then the IOP is reporting an "Invalid I/O Operation" error (in BASIC that's "Error 111: I/O OPER"). The only other kind of error reason byte is ØØxx xx11 and this presents an interface-type dependent error. The corresponding error number reported by BASIC is obtained by adding 112 decimal to "xxxx," giving an error number between 113 and 122 decimal. What you do in response to any one of these error conditions is application dependent. It is recommended that you set an appropriate flag in your RAM area and have your binary routines check such a flag at those times when such an error might occur. If you abort directly out of the interrupt service routine, you'd better be careful handling the stack pointers.

Interrupting With Available Input Data (ØØØØ Ø1ØØ)

This is much like interrupting when ready for output. In addition to checking if the CPU has had enough input, you must also check to see if the IOP has decided that the transfer should terminate (refer to the option bits discussed under Interrupt Input). Multi-byte transfers are possible here just as they were for interrupt output.

```
INTIN       !Get count and pointer for this transfer
            ! R3Ø,31 <= count, R32,33 <= pointer

            JSB  X22,IBF=1      !Wait for first data byte
INTlop      LDBD R36,R26        !Get this data byte
            PUBD R36,+R32       !Put it into the buffer
            DCM  R3Ø            !End of buffer?
            JZR  INTend         !JIF Yes - no more wanted
            ANM  R37,=4         !IOP wants to stop?
            JNZ  INTend         !JIF Yes (R37 from IBF=1)
            STBD R#,R26         !Ask for another byte
INTwat      LDBD R37,R24        !Read PSR
            JOD  INTlop         !JIF IBF (got another)
            ANM  R37,=1ØØ       !Is TFLG set?
            JNZ  INTwat         !JIF Yes - worth waiting
            JMP  INTdne         ! otherwise done for now
INTend      LDB  R37,=4         !Set CED to end transfer
            STBD R37,R24
INTbsy      LDBD R37,R24        !Wait till BUSY = Ø
            ANM  R37,=2         ! (forget OBF)
            JNZ  INTbsy
            STBD R37,R24        !Clear CED

            !Log the fact that the transfer is finished

INTdne      JMP  Common ISR End Code
```

End-of-Line Character Sequence Has Been Sent (ØØØØ Ø11Ø)

This interrupt occurs when, after an interrupt output transfer operation, the IOP has finished sending an end-of-line character sequence according to control registers 16 through 23. The BCD interface does it immediately because it has no such sequence. The only thing you need to do is to log the fact that the transfer operation is now complete. The IOP will interrupt with this reason whether or not any characters were actually sent.

Interrupt Service

The above examples of code must be in the shell of an interrupt service routine to get ready for the specific response code to be executed, and then to clean up after such execution. It is the purpose of the shell to insure that processing the interrupt (which can occur between any two consecutive assembly language instructions) does not in any way leave alterations in the machine state.

In addition to saving and restoring the machine state, it is necessary to check for the R6 return address overflow condition. The shell gets the select code (actually, the CCR address) of the interrupting IOP and the reason for the interrupt. The code can branch appropriately to process the interrupt.

All of these code sections end by branching to a common end segment which restores registers and returns CPU execution to its normal flow.

For this example we will save CPU registers 2Ø through 47. The code shown here does not include the hook at IRQ2Ø which vectors the interrupt occurrence. That will be discussed next.

```
ISR      PUMD  R4Ø,+R6      !Save registers
         LDM   R4Ø,R3Ø
         PUMD  R4Ø,+R6
         LDM   R4Ø,R2Ø
         PUMD  R4Ø,+R6
         LDM   R2Ø,R6       !Test for overflow
         SBM   R2Ø,=44,Ø    ! point to return address
                            ! (this number is decimal 12
                            ! plus the number of saved
                            ! registers on the R6 stack)
         LDM   R3Ø,=5,Ø     ! assume IRQPAD
         LDMD  R46,R2Ø
         CMM   R46,=IRQPAD
         JZR   MOVSTK       ! JIF IRQPAD was interrupted
         CMM   R46,=IRQRTN
         JNZ   STACK ok
         LDM   R3Ø,=2,Ø     ! (IRQRTN was interrupted)
MOVSTK   LDM   R32,R2Ø
         ADM   R32,R3Ø
         LDB   R37,=4       ! this moves 32 of 36
MVSTK1   POMD  R4Ø,+R32
         PUMD  R4Ø,+R2Ø
         DCB   R37
         JNZ   MVSTK1
         POMD  R44,+R32     !  this does the last 4
         PUMD  R44,+R2Ø
         SBM   R6,R3Ø       !  adjust R6
STCKok   CLM   R26          !Get interrupting select code
         DCM   R26          ! addresses for CCR & OB
         LDBD  R26,=INTRSC  ! (DAD 1775ØØ)
         STMD  R26,R24      !R24 <= pointer to CCR/PSR
         ICM   R26          !R26 <= pointer to OB/IB
         LDMD  R22,=BINTAB  !Get our base address
         JSB   X22,IBF=1    !Wait for dummy byte
         LDBD  R37,R6       !Acknowledge interrupt
         JSB   X22,IBF=1    !Wait for reason byte
         CLB   R34
```

```
          STBD  R34,R24      !Clear the CCR
          LDBD  R34,R26      !Read the reason for interrupt
                             ! branch as appropriate to the
                             ! individual routines
```

Common End Code for the Interrupt Service Routine

```
  [LABEL] POMD R4Ø,-R6       !Restore registers
          STMD R4Ø,R2Ø
          POMD R4Ø,-R6
          STMD R4Ø,R3Ø
          POMD R4Ø,-R6
          STBD R#,=INTRSC    !IOPs need this.
          RTN
```

INTRSC (DAD 1775ØØ) is a special translator address. When it is read in response to an IOP interrupt, the interrupting translator provides the least significant byte of its own CCR/PSR address.

Taking the Interrupt Vector

The hardware of the CPU, the IOP, and the translator will make sure that when an IOP needs to interrupt for service, it will eventually get the chance to do so. When it does, the CPU will first save a return address on the R6 stack, then branch to a special RAM location called IRQ2Ø. If you are going to have an interrupt service routine, the only way you can get control of the interrupts is to take the hook at IRQ2Ø.

What we're going to discuss is when to take the hook and how to return control to the system. If the I/O ROM, the Mass Storage ROM, or the Plotter/Printer ROM (or any combination of these) is plugged in, then the hook will have been taken immediately. Any one of these ROMs will handle power-on, errors, and resetting. The I/O and Mass Storage ROMs will also handle burst termination for you. You might want to do some interrupt operations in your own interrupt service routine and have a ROM handle others. You can do this if you follow a few precautions:

- Before you take the hook by storing your vector in it, read the old vector and store it in your RAM area.

- If the first byte of the old vector was a RTN instruction, then the hook hasn't been taken since power-on.

- Disable interrupts globally before you change the code at IRQ2Ø whether taking it over or giving it back. This is critical code. Remember to re-enable them after you've made the change.

Interrupt Input and Output Operations

We have already discussed the interrupt service routine and it is this routine's response segments for input and output that do most of the work for an interrupt input or output operation. It is perhaps misleading to call it a procedure. It is more like a process.

The CPU initiates this process (as described next), the IOP and the CPU cooperate during the process, and some occurrence terminates the process. The cooperation and termination are what the interrupt service routine does. Next is a discussion of the preparations required for the CPU to initiate the process.

Preparation for an interrupt input or output includes everything you must manage for the simple case such as configuration and addressing. You must also arrange a byte counter and pointer in your RAM area so that the interrupt service routine, knowing which select code is interrupting, and knowing the direction of the transfer, knows where to find them. You must be sure that the process will be legal: the interface is either not busy, or it is full duplex and not busy in this direction. If you are using any of the input termination options, you must be sure that the necessary preparations have been made (count to control registers 25 and 26, and/or Delim character to control register 27).

The protocol command handshake types for interrupt input and output are:

- If the interface is not full duplex, both commands are direct because the interface cannot begin a new transfer while it is still busy.
- If the interface is full duplex, the output command is the interrupt-type. The input command is uniquely handled by skipping the usual test for OBF=BUSY=Ø before the command is written to the OB in a manner similar to that used for direct commands.

The interrupt output protocol command has no options. It is 242 octal and the transfer terminates when the CPU sets the CED bit. The corresponding input command has three bits which specify optional conditions. The IOP declares the transfer finished by setting the PED bit. These correspond to the conditions explained under simple input. In the syntax of the TRANSFER; INTR statement: Bit Ø set = EOI, Bit 2 set = DELIM, and Bit 3 set = COUNT.

Once the appropriate protocol command for interrupt input or output has been properly passed to the IOP the process has been initiated. The CPU can continue and let the IOP and the interrupt service routine take over. In the following examples it is assumed that all of the above-mentioned preparation has taken place and, for the input examples, the termination option bits are in R34.

```
Full Duplex Input   LDB  R36,=22       !Protocol = input, intr
                    ORB  R36,R34       !Fold in the options
                    JSB  X22,OBF=Ø     !Mustn't crash the OB
                    LDB  R37,=2        !Set CMD bit
                    STBD R37,R24       ! in the CCR
                    STBD R36,R26       !Write the command
                    JSB  X22,OBF=Ø     !Wait till IOP has it
                    CLB  R37           !Clear CCR
                    STBD R37,R24
                    RTN                !No wait for BUSY=Ø


Full Duplex Output  LDB  R36,=242      !Protocol = output, intr
                    JSB  X22,INTCMD    ! (interrupting type)
                    JSB  X22,INTCHK    !Uninterrupt the IOP
                    RTN                !No wait for Busy=Ø

Not Full Dup In     LDB  R36,=22       !Protocol = input, intr
                    ORB  R36,R34       !Fold in the options
                    JSB  X22,DIRCMD    ! (direct type)
                    RTN                !No wait for BUSY=Ø

Not Full Dup Out    LDB  R36,=242      !Protocol - output,intr
                    JSB  X22,DIRCMD    ! (direct type)
                    RTN                !No wait for BUSY=Ø
```

Data transfer rates for interrupt input and output from assembler code will not be much faster than those for BASIC language operation because of the large overhead for the interrupt service routine. For higher speed concurrent I/O, you should consider a polling operation over simple input and output processes. To do this, the CPU must poll each card to see if it is ready for data transfer.

3.4.15 Simulation of I/O ROM Statements

In this section, those statements provided by the I/O ROM will be analyzed in terms of the protocol commands they use. These statements can be very useful for simulating the operation of your binary code.

ABORTIO: The protocol command "Abort" (Ø1ØØ ØØØØ) is sent to the interface.

ASSERT: The given byte is written into control register 28.

CLEAR: This statement has two forms: one with primary addressing and one without. If there is no primary address then the HP-IB interface message "DCL" or "Device Clear" is sent. If there is a primary address (or a batch of them), an addressing routine will be done. This will send the "Unlisten" interface message, execute the "Send My Talk Address" protocol command, send a "Listen Address" interface message for each primary address given, and then send the HP-IB interface message "SCD" or "Selected Device Clear."

CONTROL: This statement performs the control operation as described. The I/O ROM will not allow you to access control registers 25 through 29 with this statement.

ENABLE INTR: Identical to CONTROL to Register 1.

ENTER: This statement performs addressing if a primary address is given and then sends the simple input protocol command and inputs bytes until its argument list is satisfied. The addressing done consists of sending the "Unlisten" interface message, executing the "Send My Listen Address" protocol command, and sending the appropriate "Talk Address" interface message.

HALT: This statement sends the "Halt" (Ø1ØØ 1ØØØ) protocol command.

LOCAL: This statement may or may not have a primary address. If it does, then the addressing sequence of send the "Unlisten" interface message, execute the "Send My Talk Address" protocol command, send a "Listen Address" interface message to each primary address given, and then sends the HP-IB "Go to Local" interface message. If no primary address is given then the protocol command "Set REN to false" (Ø1ØØ ØØ1Ø) is executed.

LOCAL LOCKOUT: This statement sends the HP-IB "Local Lockout" interface message.

OUTPUT: This statement will do the addressing routine if there is a primary address (or addresses) provided, and will then handshake the simple output protocol command (1Ø1Ø ØØØØ) and output however many bytes it takes to satisfy the argument list. The addressing routine consists of sending the "Unlisten" interface message, executing the "Send My Talk Address" protocol command, and sending the appropriate "Listen Address" interface message(s).

PASS CONTROL: If a primary address is indicated, this statement will send the corresponding "Talk Address" interface message. With or without the talk address it will then send the HP-IB "Take Control" interface message and then exit the controller active state.

PPOLL: This function executes the Parallel Poll protocol command (Ø1ØØ Ø1ØØ) and returns the response byte.

REMOTE: If a primary address (or addresses) is provided this statement will send an addressing sequence and then execute the Set REN True (Ø1ØØ ØØØ1) protocol command. If no primary address is given, only the protocol command will be executed. The addressing sequence is: send the "Unlisten" interface message, execute the "Send My Talk Address" protocol command then send a "Listen Address" interface message for each primary address given.

REQUEST: The given "response byte" is written to control register 29.

RESET: The RESET bit in the CCR for the given select code is strobed to initiate the reset for the IOP. The CPU enters a wait loop for about 4ØØ milliseconds to give the IOP time to complete its reset operation and interrupt with the self-test results. The interrupt service routine logs an error if the reason for interrupting is the one that indicates the self-test failed.

RESUME: This statement executes the "Resume" protocol command (Ø1ØØ 1ØØ1).

SEND: This is a very useful statement for simulation purposes. These are its field options:

CMD: All expressions following this keyword are converted into byte strings and are sent as interface messages (that is, they are sent to the IOP using the "Send" protocol command (Ø1ØØ ØØØØ)).

DATA: All expressions following this keyword are converted into byte strings and are sent as data bytes (that is, they are sent to the IOP using the simple output protocol command (1Ø1Ø ØØØØ)).

TALK: The expression following is reduced to five bits and added to octal 1ØØ to form a "Talk Address" interface message. This result is sent to the IOP using the "Send" protocol command (1Ø11 ØØØØ).

LISTEN: The expressions following this keyword are reduced to five bits and added to octal 4Ø to form a "Listen Address" interface message which is sent to the IOP using the "Send" protocol command (Ø1ØØ ØØØØ).

SCG: The expressions following this keyword are reduced to five bits and are added to octal 14Ø to form a Secondary Address interface message which is sent to the IOP using the "Send" protocol command (1Ø11 ØØØØ).

UNL: Octal 77 ("Unlisten") is sent to the IOP using the "Send" protocol command (1Ø11 ØØØØ).

UNT: Octal 137 ("Untalk") is sent to the IOP using the "Send" protocol command (1Ø11 ØØØØ).

MLA: This keyword executes the "Send My Listen Address" protocol command (Ø1ØØ Ø11Ø).

MTA: This keyword executes the "Send My Talk Address" protocol command (Ø1ØØ Ø1Ø1).

SPOLL: This function executes an addressing routine if a primary address is provided and then performs the following sequence:

- Octal 3Ø ("Serial Poll Enable") is sent to the IOP using the "Send" protocol command.
- The simple input protocol command (ØØØ1 ØØØ1) is sent and one byte is input.
- Two bytes (octal 31: "Serial Poll Disable" and then octal 137: "Untalk") are then sent to the IOP using the "Send" protocol command.

The addressing sequence used for a primary address is:

- Send the "Unlisten" interface message (that is, send it to the IOP using the "Send" protocol command).
- Execute the "Send My Listen Address" protocol command.
- Send the appropriate "Talk Address" interface message.

This function statement returns the single byte which was input.

STATUS: This statement executes the read status protocol operation as discussed above.

TRANSFER: This statement executes interrupt I/O and burst I/O according to the INTR or FHS keyword. Three bytes are written to control registers 25 through 27 to set the count and delimiter options. If no count is specified, the length of the buffer string is sent as the count. The delimiter byte is always written (as a coding abbreviation), but it only has a defined value if it has been specified for an interrupt input and it won't be used unless that specification is made (the enable bit must be set in the protocol command). The EOI keyword causes the corresponding bit to be set in the protocol command. After the protocol command has been assembled, it is sent to the IOP and the CPU either enters a burst loop or returns from starting an interrupt process.

TRIGGER: This statement performs an addressing operation if a primary address (addresses) is specified, and then sends the HP-IB "Group Execute Trigger" interface message. The addressing operation consists of sending the "Unlisten" interface message, executing the "Send My Talk Address" protocol command and then sending a "Listen Address" interface message for each primary address indicated.

Notice that the addressing operation invoked by a primary address is the same in all the above statements and functions in the I/O ROM. To summarize:

- For inputs: send Unlisten, My Listen Address, and the talk address given as the primary address.
- For outputs: send Unlisten, My Talk Address, and a listen address for each primary address given.

3.4.16 Timing Methodology

The I/O protocol command set was timed with the clock in the HP-85 using a BASIC program to repetitively call a binary program which executed the command. In order to obtain usable timing data for the I/O programmer who wishes to do I/O in assembly language, the times reported (except for the ones given as BASIC language comparisons) are the times required for execution and/or data transfer between the IOP and the CPU registers. All times having to do with fetching and storing BASIC variables and calling the binary code through BASIC statement execution have been subtracted out as described next.

The time required for calling the binary code from BASIC was removed by repeating the timing loop twice. A flag variable was set to Ø before the timing loop was entered the first time and was tested for Ø after the loop finished. If the flag was found to be Ø, it was given some other value (depending on the operation) and the loop was re-entered by a GOTO command. When the test after the loop found the flag to be not Ø, the first time was subtracted from the second time and the difference divided by the number of repetitions. The resulting time was taken as the execution time of the operation.

The binary code accepted the flag as one of several parameters. After fetching all parameters into CPU registers and preparing all items which are assumed to be done prior to executing an I/O operation (such as loading the contents of BINTAB into a pair of CPU registers), the binary code tested the flag for Ø. If it was Ø, the binary returned immediately (this was the first pass through timing the loop).

If it was not Ø, the binary performed whatever operation had been coded into it. The flag was set to the number of bytes to be transferred. If appropriate, the binary code then transferred that many bytes into or out of the CPU registers. Usually, none of the data bytes transfered in such cases came from or returned to the BASIC code variables. If bytes were input, they were simply ignored. If bytes were to be output, the binary simply sent Ø's for control operations and string output.

For formatted areas (sending to the BCD interface and needing '+' and 'E' for instance) a string variable was used. The pointers were set up in the CPU registers regardless of the value of the flag so the time needed for this was subtracted out.

The only remaining variation in timing between the Ø-flag run and the "not" Ø-flag run was the value of the flag. Because a system routine used by the binary was argument dependent, the difference showed up in the loop time differences. A special binary program that called the system routine ONEB, used with a series of parameters from Ø to 1ØØ (the largest flag value used) gave a set of execution time differences than that with the parameter Ø. These were subtracted from the measured times.

The timing loop went through 1,ØØØ repetitions of each binary call. Because the accuracy of the BASIC timer and resolution are on the order of a millisecond, we start with an upper limit on the accuracy of the timing data of a microsecond. Several influencing factors lower this accuracy to some unknown, lesser accuracy. The coincidence of handshaking signal assertions and tests is probably able to account for ten or so microseconds of jiggle, but this should average out over a thousand operations.

The main error is the argument dependent execution times of number conversion routines used by the BASIC code in taking the system time. The results tended to be repeatable to within a few microseconds per operation. However, with the resolution of the timers being on the order of the times being measured, no statistics on the variability of the individual operation times could be taken. The times are given in milliseconds as one or two digit numbers. They should be taken as good to ten percent or so if it really makes a difference in your application. You should test and time as appropriate to your needs.

For operations that don't involve any actual data transfer, the times are listed. If data transfer is involved, there is a base time given which is what is required for a single-byte transfer and an incremental time which should be added to the base time for each additional byte transferred.

BASIC timing loop:

```
 970 !Previous code has set C to the select code and
 980 ! has set S to the protocol command selected.
 990 !
1000 F=0                    !Flag is initially set to 0
1010 TO=TIME                !Time of start through loop
1020 FOR I=1 TO 1000
1030 BINARY F,C,S           !Call the binary
1040 NEXT I
1050 T1=TIME                !Time at end of loop
1060 IF F THEN 1100         !Branch if second time through
1070 T2=T1-TO               ! otherwise T2 gets the first time
1080 F=1                    !Flag now says second time through
1090 GOTO 1010
1100 T3=T1-0                ! second time to T3
1110 !

1120 PRINT "Execution time:"; (T3-T2)/1000;" seconds"
```

The Binary:

```
1000 !Above code set up and put the flag into R20
1010        TSB       R20
1020        JNZ       SECOND
1030        RTN                   !Just return first time
1040 SECOND !Ensuing code performs operation S at
1050        ! select code C for second time through loop.
```

Section
IV

SAMPLE CODE

4.1 Introduction

The binary program in this section illustrates several I/O operations. The register conventions used are:

R2Ø,21	Scratch
R22,23	BINTAB
R24,25	CCR/PSR address
R26,27	OB/IB address
R3Ø,31	Character count
R32,33	Buffer pointer
R34	Active? (Boolean Ø=false)
R35	EOL request (Boolean Ø=false)
R36,37	GOTO/GOSUB pointer

The code assumes that the interface of interest is at select code 1Ø. Additional interfaces (notably HP-IB at select code 7) are allowed. All I/O ROM keyword usage should occur either before the binary is loaded or between a RELINQUISH statement and an UNRELINQUISH statement.

4.2 Keywords

The sample program keywords are:

RELINQUISH
Returns the IRQ2Ø and IOSP hooks to the ROM that had them before SELFIO was loaded.

UNRELINQUISH
Takes IRQ2Ø and IOSP back. If alternated with RELINQUISH, interrupt control can be passed back and forth between the I/O ROM and the binary program.

GIVES (string expression)
Sends the string using simple output protocol.

TAKES (string expression)
Enters the string using simple input protocol. It won't terminate until the dimensioned size of the string has been filled. Line feeds are not specially recognized.

GIVEI (string expression) AFTERWARD GOTO/GOSUB Line#
Sends the string using interrupt output protocol. Equivalent to "ON EOT 1Ø GOTO/GOSUB Line# TRANSFER (string expression) INTR."

TAKEI (string reference) AFTERWARD GOTO/GOSUB Line#
Enters the string using interrupt input protocol. Equivalent to "ON EOT 1Ø GOTO/GOSUB Line# TRANSFER 1Ø TO (string reference) INTR."

ON BREAK GOTO/GOSUB Line#
Sets up an end-of-line branch that takes place upon receipt of a BREAK character (for serial interface; for HP-IB, read "ON IFC"; for GPIO, read "ON ST1"; for BCD, read "ON FUNCTION B MSB"). Equivalent to "ON INTR 1Ø GOTO/GOSUB Line# ENABLE INTR 1Ø; 128."

ACKNOWLEDGE BREAK
This is the first statement which should be executed after the ON BREAK end-of-line branch has taken place. It is equivalent to "STATUS 1Ø,1 ; Z9" where Z9 is ignored.

GIVEB (string expression)
Sends the string using burst output protocol (note that the BCD and GPIO interfaces have format restrictions for burst I/O and the serial interface does not support burst I/O).

TAKEB (string reference)
Enters the string using burst input protocol. Fills the dimensioned size for the string.

SHOVE (register number) , (data)
Does CONTROL 1Ø, (register number) ; (data) for times when the I/O ROM isn't available.

```
10          LST
20 !******************************EXAMPLE I/O BINARY PROGRAM
30          GLO GLOBAL
40 !         NAM SELFIO
50          DEF RUNTAB
60          DEF ASCTAB
70          DEF PARTAB
80          DEF ERRTAB
90          DEF INIT.
100 PARTAB   BYT 0,0
110          DEF RELIN-             !1
120          DEF UNREL-             !2
130          DEF GIVES-             !3
140          DEF TAKES-             !4
150          DEF GIVEI-             !5
160          DEF TAKEI-             !6
170          DEF ONBRK-             !7
180          DEF GIVEB-             !10
190          DEF TAKEB-             !11
200          DEF ACBR-              !12
210          DEF SHOV-
220 RUNTAB   BYT 0,0
230          DEF RELIN.             !1
240          DEF UNREL.             !2
250          DEF GIVES.             !3
260          DEF TAKES.             !4
270          DEF GIVEI.             !5
280          DEF TAKEI.             !6
290          DEF ONBRK.             !7
300          DEF GIVEB.             !10
310          DEF TAKEB.             !11
320          DEF ACBR.              !12
330          DEF SHOV.              !13
340          DEF DUMMY              !14
350          DEF DUMMY              !15
360          DEF DUMMY              !16
370          DEF DUMMY              !17
380          DEF AFTER.             !20
390          BYT 377,377
400 DUMMY    RTN
410 ASCTAB   ASP "RELINQUISH"       !1
420          ASP "UNRELINQUISH"     !2
430          ASP "GIVES"            !3
440          ASP "TAKES"            !4
450          ASP "GIVEI"            !5
460          ASP "TAKEI"            !6
470          ASP "ON BREAK"         !7
480          ASP "GIVEB"            !10
490          ASP "TAKEB"            !11
500          ASP "ACKNOWLEDGE BREAK" !12
510          ASP "SHOVE"            !13
520          BYT 215                !14
```

```
530           BYT 215                  !15
540           BYT 215                  !16
550           BYT 215                  !17
560           ASP "AFTERWARD"          !20
570           BYT 377
580 ERRTAB    BYT 200,200,200,200,200,200,200,200,200
590           ASP "Interface missing"
600           ASP "Interface unwell"
610           ASP "Interface dependent error"
620           ASP "Invalid operation"
630           BYT 377
640 !***************************INITIALIZATION
650 INIT.     BIN
660           LDMD R22,=BINTAB
670           LDBD R20,=ROMFL
680           CMB R20,=3               !LOADBIN?
690           JZR INIT..               !JIF YES
700           CMB R20,=4               !RUN,INIT ?
710           JNZ INIT.+               !JIF NO
720           LDM R20,=INIT.1          !ELSE CLEAR POINTERS
730           ADM R20,R22
740           DCM R20
750           STM R20,R4               !(GTO INIT.1)
760 INIT.+    RTN
770 INIT..    LDM R26,=IRQ20
780           LDBD R20,R26             !SAVE FOR LATER
790           LDM R24,R26              !SAVE OTHER ROM'S IRQ20 HOOK
800           LDM R30,=IOSAVE
810           ADM R30,R22
820           POMD R40,+R24
830           PUMD R40,+R30
840           POMD R40,+R24
850           PUMD R40,+R30
860           STBD R#,=GINTDS          !TAKE HOOKS
870           LDM R71,=232             !(SAD)
880           STBD R#,=GINTDS          !->R72
890           JSB =ROMJSB              !->R75
900           PUMD R71,+R26            !HALF OF IRQ20
910           LDM R70,=ISR             !RELATIVE ADDRESS
920           BYT 0                    !->R72
930           STBD R#,=GINTEN          !->R73
940           PAD                      !->R76
950           RTN                      !->R77
960           LDM R24,R70
970           ADM R24,R22
980           STM R24,R70              !ABSOLUTE ADDRESS
990           PUMD R70,+R26            !REST OF IRQ20
1000           LDM R71,=316            !(JSB)
1010           DEF ROMJSB              !->R72
1020           DEF EOLSV               !->74
1030           BYT 0                   !->R76
1040           RTN                     !->R77
```

```
1050              LDM R24,R74             !RELATIVE
1060              ADM R24,R22
1070              STM R24,R74             !ABSOLUTE
1080              STMD R71,=IOSP
1090              STBD R#,=GINTEN         !DONE HOOKS
1100              CMB R20,=236            !DID I TAKE IT FIRST?
1110              JNZ INIT.1              !JIF NO
1120              CLB R20
1130              STBD R20,X22,STEST?     !ASSUME NO INTERFACE
1140              LDB R31,=4
1150              JSB =CNTRTN             !WAIT 50 MS
1160              STBD R#,=INTRSC         !CLEAR ALL RST BITS
1170              LDB R31,=31
1180              JSB =CNTRTN             !WAIT 400 MS
1190              LDBD R20,X22,STEST?
1200              JNZ INIT.0
1210              LDB R20,=366            !"INTERFACE MISSING"
1220 INIT.E       STBD R20,X22,ERROR#
1230              JSB =ERROR+
1240 ERROR#       BSZ 1
1250 INIT.0       CMB R20,=3
1260              JZR INIT.1              !JIF SELFTEST OK
1270              LDB R20,=365            !"INTERFACE UNWELL"
1280              JMP INIT.E
1290 INIT.1       LDM R24,=136,377        !INITIALIZE RAM
1300              STMD R24,X22,CCRADR
1310              ICM R24
1320              STMD R24,X22,OBADR
1330              LDM R24,=ICOUNT
1340              ADM R24,R22
1350              CLM R70
1360              PUMD R70,+R24
1370              PUMD R70,+R24
1380              PUMD R70,+R24
1390              LDBD R21,=SCLOG
1400              ANM R21,=200            !GOT AN INTERFACE?
1410              JZR INIT.E
1420 INIT.R       RTN
1430 !****************************IR020 INTERRUPT SERVICE ROUTINE
1440 ISR          BIN
1450              PUMD R2,+R6             !SAVE 26 CPU REGISTERS
1460              PUMD R40,+R6            ! MAKING N+12 =
1470              LDM R40,R30             ! 46 OCTAL
1480 N+12         EQU 46
1490              PUMD R40,+R6
1500              LDM R40,R20
1510              PUMD R40,+R6
1520              CMM R6,=R6LIM2          !CHECK STACK OVERFLOW
1530              JNC ISR6OK
1540              LDB R37,=17
1550              JSB =SYSERR
1560 ISR6OK       LDM R20,R6              !CHECK FAST INTERRUPTS
```

```
1570           SBM R20,=N+12          !FIND RETURN ADDRESS
1580           LDM R30,=5,0           !ASSUME IRQPAD
1590           LDMD R46,R20           !GET RETURN ADDRESS
1600           CMM R46,=IRQPAD
1610           JZR MOVSTK             !JIF IRQPAD
1620           CMM R46,=IRQRTN
1630           JNZ R6GOOD             !JIF NOT IRQRTN
1640           LDM R30,=2,0           !ELSE ADJUST DISTANCE
1650 MOVSTK    LDM R32,R20
1660           ADM R32,R30
1670           LDB R37,=5             !40 DEC BYTES WILL MOVE
1680 MOV_LP    POMD R40,+R32          !MOVE THEM
1690           PUMD R40,+R20
1700           DCB R37
1710           JNZ MOV_LP
1720           SBM R6,R30             !ADJUST STACK POINTER
1730 R6GOOD    LDMD R22,=BINTAB
1740           CLM R26                !GET INTERRUPTING CCR ADDRESS
1750           DCM R26
1760           LDBD R26,=INTRSC
1770           STM R26,R24            !R24->CCR/PSR
1780           ICM R26                !R26->OB/IB
1790 ISR1      LDBD R20,R24
1800           JEV ISR1               !AWAIT IBF=1
1810           LDBD R20,R26           !SAY HELLO TO IOP
1820 ISR2      LDBD R20,R24
1830           JEV ISR2               !AGAIN IBF=1
1840           CLB R20
1850           STBD R20,R24           !ASSURE THAT CED=0
1860           LDBD R20,R26           !GET INTERRUPT REASON
1870           JZR ISRIO              !INTERRUPT OUT
1880           CMB R20,=4
1890           JZR ISRII              !INTERRUPT IN
1900           CMB R20,=1
1910           JZR ISRBT              !BURST TERMINATION
1920           CMB R20,=6
1930           JZR ISREOL             !EOL SEQUENCE SENT
1940           CMB R20,=377
1950           JZR ISRIIO             !INVALID I/O
1960           CMB R20,=373
1970           JZR ISRSTE             !SELF TEST ERROR
1980           CMB R20,=3
1990           JZR ISRSTO             !SELF TEST OK
2000           CMB R20,=2
2010           JZR ISRR1I             !REGISTER 1 INTERRUPT
2020 ISRIDE    JSB =ERROR
2030           BYT 364                !"INTERFACE DEPENDENT ERROR"
2040           JMP ISRDON
2050 !***************************SELF TEST ERROR
2060 ISRSTE    LDMD R30,X22,CCRADR    !MY INTERFACE?
2070           CMM R24,R30
2080           JZR ISRSTO             !JIF YES
```

```
2090            JMP ISRIDE
2100 !***************************SELF TEST OK
2110 ISRSTO     LDMD R30,X22,CCRADR      !MY SELECT CODE?
2120            CMM R24,R30
2130            JNZ ISRSTP               !JIF NO
2140            STBD R20,X22,STEST?
2150 ISRSTP     LDB R21,R24              !LOG IN THIS SELECT CODE
2160            LRB R21
2170            ANM R21,=7
2180            LDB R20,=1
2190 ISRLOG     DCB R21
2200            JCY ISRLO+
2210            LLB R20
2220            JMP ISRLOG
2230 ISRLO+     LDBD R21,=SCLOG
2240            ORB R21,R20
2250            STBD R21,=SCLOG
2260 ISRDON     JMP ISRRT1
2270 ISRIIO     JSB =ERROR
2280            BYT 363                  !"INVALID OPERATION"
2290            JMP ISRDON
2300 ISRIO      JMP ISRIO.
2310 ISRII      JMP ISRII.
2320 ISRBT      JMP ISRBT.
2330 !***************************EOL SEQUENCE SENT
2340 ISREOL     LDBD R20,X22,DACTV?      !BRANCH SET UP?
2350            JZR ISRDON               !JIF NO
2360            LDM R20,=0,377
2370            STMD R20,X22,DACTV?      !FLAG EOL
2380 LOGEOL     JSB X22,SETEOL           !REQUEST EOL SERVICE
2390 LOGEOR     JMP ISRDON
2400 !***************************REGISTER ONE INTERRUPT
2410 ISRR1I     LDBD R20,X22,CACTV?      !BRANCH SET UP?
2420            JZR ISRDON               !JIF NO
2430            LDM R20,=0,377
2440            STMD R20,X22,CACTV?      !FLAG EOL
2450            JMP LOGEOL
2460 !***************************BURST TERMINATION
2470 ISRBT.     LDM R20,R6               !GET STACK
2480            SBM R20,=N+12            !FIND RETURN ADDRESS
2490            LDM R24,=SYSRTN          !GET RTN TO RTN
2500            STMD R24,R20             !DECK THE STACK
2510 ISRRT1     JMP ISRRT2
2520 ISRIO.     JMP ISRIO+
2530 !***************************TRANSFER IN
2540 ISRII.     LDMD R40,X22,ICOUNT      !GET POINTERS
2550            STM R40,R30
2560 ISRII1     LDBI R20,X22,CCRADR      !READ PSR
2570            JEV ISRII1               !AWAIT IBF=1
2580 ISRII2     LDBI R21,X22,OBADR       !GET DATA
2590            PUBD R21,+R32            !STORE IT
2600            DCM R30                  !DROP COUNT
```

```
2610           LRB R20
2620           LRB R20
2630           JOD ISRII4             !JIF PED=1
2640           TSM R30
2650           JZR ISRII4             !JIF COUNT USED UP
2660           STBI R#,X22,OBADR      !(REQUEST MORE)
2670 ISRII3    LDBI R20,X22,CCRADR    !READ PSR
2680           JOD ISRII2             !NEXT ONE'S READY
2690           LLB R20
2700           JNG ISRII3             !JIF TFLG=1
2710           JMP ISRII6             !ELSE QUIT FOR NOW
2720 ISRII4    LDB R20,=4
2730           STBI R20,X22,CCRADR    !SET CED=1
2740 ISRII5    LDBI R20,X22,CCRADR
2750           LRB R20
2760           JOD ISRII5             !AWAIT BUSY=0
2770           CLB R20
2780           STBI R20,X22,CCRADR    !SET CED=0
2790           CLM R34                !"TRANSFER NOT ACTIVE"
2800           DCB R35                !"NEED EOL SERVICE"
2810           JSB X22,SETEOL         !REQUEST EOL SERVICE
2820 ISRII6    LDM R40,R30            !PUT AWAY POINTERS
2830           STMD R40,X22,ICOUNT
2840 ISRRT2    JMP ISRRTN
2850 !***************************TRANSFER OUT
2860 ISRIO+    LDMD R40,X22,OCOUNT    !FETCH POINTERS
2870           STM R40,R30
2880 ISRIO1    POBD R20,+R32          !GET NEXT BYTE
2890           DCM R30                !DROP COUNT
2900           JCY ISRIO2             !JIF NOT LAST
2910           LDB R21,=4             !ELSE SET CED=1
2920           STBI R21,X22,CCRADR
2930 ISRIO2    STBI R20,X22,OBADR     !WRITE DATA BYTE
2940 ISRIO3    LDBI R20,X22,CCRADR    !READ PSR
2950           JNG ISRIO3             !AWAIT OBF=0
2960           LLB R20
2970           JNG ISRIO1             !MORE IF TFLG=1
2980           LDM R40,R30            !ELSE PUT AWAY POINTERS
2990           STMD R40,X22,OCOUNT
3000 ISRRTN    POMD R40,-R6           !RESTORE CPU REGISTERS
3010           STM R40,R20
3020           POMD R40,-R6
3030           STM R40,R30
3040           POMD R40,-R6
3050           POMD R2,-R6
3060           STBD R2,=INTRSC        !REVIVE TC'S
3070           RTN
3080 !***************************END-OF-LINE BRANCH SERVICE ROUTINE
3090 EOLSV     BIN
3100           STBD R#,=GINTDS
3110           LDMD R22,=BINTAB
3120           CMB R16,=7             !ONE AT A TIME!
```

```
3130            JZR EOLSV3
3140            CMB R16,=2
3150            JZR EOLSV1                 !JIF RUNNING
3160 MAGIC      LDM R20,R6                 !SYSTEM REQUIRED
3170            SBM R20,=11,0              !  INTERACTION
3180            LDM R46,=CLKHIT
3190            ANM R32,=376,377
3200            TSB R32
3210            JNZ CNDHIT
3220            LDM R46,=CHREDT
3230            LDBD R45,=SVCWRD
3240            JOD CNDHIT
3250            LDM R46,=XCBIT3
3260 CNDHIT     STMD R46,R20
3270            LDB R37,R17
3280            LLB R37
3290            LLB R36
3300            LLB R37
3310            LLB R37
3320            DRP R32
3330            STBD R#,=GINTEN
3340            RTN
3350 EOLSV1     CMMD R10,=PCR
3360            JNZ MAGIC                  !JIF NOT AT END OF BASIC LINE
3370            LDBD R75,X22,SRVEOL
3380            JZR EOLSV4                 !JIF NONE PENDING
3390            CLB R75                    !ELSE, UNPEND IT
3400            STBD R75,X22,SRVEOL
3410 EOLSV2     LDB R37,R17
3420            ANM R37,=10
3430            JZR EOLSV3                 !JIF NOT TRACE MODE
3440            JSB =ROMJSB
3450            DEF TRA?
3460            BYT 0
3470 EOLSV3     JMP EOLSV5
3480 EOLSV4     JSB X22,EOLWHO             !ANY REQUESTS?
3490            JZR EOLSV2                 !JIF NO
3500            CLB R75                    !CLEAR THE REQUEST
3510            PUBD R75,+R76
3520            DCB R75
3530            STBD R75,X22,SRVEOL        !SET PENDING
3540            POMD R56,+R76              !GET STORED R10
3550            JSB =SETTR1                !  (FOR TRACING)
3560            STMD R10,=DNFLAG           !SAVE CURRENT BASIC PC
3570            LDM R10,R56                !DO BASIC BRANCH
3580            LDB R16,=7                 !ALERT SYSTEM
3590            JMP EOLSV6
3600 EOLSV5     JSB X22,EOLWHO
3610            JNZ EOLSV6                 !JIF MORE TAKERS
3620            LDB R32,=375               !ELSE CLEAR BITS
3630            JSB =CLRBIT
3640            JMP EOLSV7
```

```
3650 EOLSV6   LDB R65,=2              !ASSURE ANOTHER IOSP CALL
3660          LDBD R64,=SVCWRD
3670          ORB R64,R65
3680          STBD R64,=SVCWRD
3690          LDB R64,=20
3700          ORB R17,R64
3710 EOLSV7   LDBD R20,=IPHERE        !DON'T RE-ENABLE INTERRUPTS
3720          JNZ EOLSV8              ! IF IPBIN IS HERE
3730          STBD R#,=GINTEN
3740 EOLSV8   RTN
3750 EOLWHO   LDM R76,=IEOLFL
3760          ADM R76,R22
3770          LDBD R75,R76
3780          JNZ EOLHIM              !JIF EOT IN
3790          LDM R76,=OEOLFL
3800          ADM R76,R22
3810          LDBD R75,R76
3820          JNZ EOLHIM              !JIF EOT OUT
3830          LDM R76,=CEOLFL
3840          ADM R76,R22
3850          LDBD R75,R76            !BREAK?
3860 EOLHIM   RTN
3870          LNK EXAMPLES2
```

```
10 !******************************PARSE CODE
20 RELIN-   BSZ 0
30 UNREL-   BSZ 0
40 ACBR-    LDB R77,R43               !SIMPLE STATEMENTS
50          LDB R75,=371
60          PUMD R75,+R12
70          JSB =SCAN
80          RTN
90 GIVES-   BSZ 0
100 GIVEB-   PUBD R43,+R6
110          JSB =STREX+              !STRING EXPRESSION
120          JEZ PARERR
130 PARPU    POBD R77,-R6
140          LDB R75,=371
150          PUMD R75,+R12
160          RTN
170 TAKES-   BSZ 0
180 TAKEB-   PUBD R43,+R6
190          JSB =SCAN
200          JSB =STRREF              !STRING REFERENCE
210          JEN PARPU
220 PARERR   POBD R43,-R6
230          JSB =ERROR+
240          BYT 92D
250 GIVEI-   PUBD R43,+R6
260          JSB =STREX+
270          JEZ PARERR
280          JMP AFTER-
290 TAKEI-   PUBD R43,+R6
300          JSB =SCAN
310          JSB =STRREF
320          JEZ PARERR
330 AFTER-   CMB R14,=371
340          JNZ PARERR
350          CMB R43,=20              !DEMAND "AFTERWARD"
360          JNZ PARERR
370          POBD R77,-R6
380          LDB R75,=371
390          PUMD R75,+R12            !PUSH "GIVEI" OR "TAKEI"
400 ONBRK-   PUBD R43,+R6             !SAVE "ONBREAK" OR "AFTERWARD"
410          JSB =SCAN
420          CMB R47,=210             !DEMAND GOTO/GOSUB
430          JNZ PARERR
440          POBD R77,-R6
450          LDB R75,=371
460          PUMD R75,+R12            !PUSH "ONBREAK" OR "AFTERWARD"
470          JSB =GOTOSU              !SYSTEM HANDLES GOTO/GOSUB
480          RTN
490 SHOV-    PUBD R43,+R6
500          JSB =GET2N               !GET REGISTER NUMBER & DATA
510          JEN SHOV-1
520 SHOV-E   POBD R43,-R6
530          JSB =ERROR+
540         BYT 89D
550 SHOV-1  POBD R43,-R12                ! (FOR GET2N)
560         POBD R57,-R6
570         LDB R55,=371
580         PUMD R55,+R12
590         RTN
600 !
610 !
620 !
630 !
640 !
650 !
660 !
670 !
680         BYT 0,241                   ! TO DECOMPILE "AFTERWARD"
690 AFTER.  RTN
700 !****************************RELINQUISH
710         BYT 0,241
720 RELIN.  BIN
730         LDMD R22,=BINTAB
740         LDM R20,=IOSAVE             !GET ORIGIONAL HOOK
750         ADM R20,R22
760         POMD R40,+R20
770         CMB R40,=236
780         JNZ RELIN1                  !JIF THERE WAS ONE
790         JSB =ERROR+
800         BYT 362                     !"NO ROM!"
810 RELIN1  LDM R24,=IRQ20
820         PUMD R40,+R24
830         POMD R40,+R20
840         PUMD R40,+R24
850         POMD R45,+R20               ! IOSP, ALSO
860         STMD R45,=ESHOOK
870         RTN
880 !****************************UNRELINQUISH
890         BYT 0,241
900 UNREL.  BIN
910         LDMD R22,=BINTAB
920         LDM R20,=IOSAVE             !RE-SAVE OTHER HOOK
930         ADM R20,R22
940         LDM R24,=IRQ20
950         POMD R40,+R24
960         PUMD R40,+R20
970         POMD R40,+R24
980         PUMD R40,+R20
990         LDMD R45,=ESHOOK
1000         PUMD R45,+R20
1010         DCM R24
1020         LDM R40,=ISR               !REMAKE MINE
1030         BYT 0
1040         STBD R#,=GINTEN
1050         PAD
```

```
1060          RTN
1070          LDM R20,R40
1080          ADM R20,R22
1090          STM R20,R40             !ABSOLUTE ADDRESS
1100          PUMD R40,-R24
1110          LDM R41,=232            !(SAD, ETC.)
1120          STBD R#,=GINTDS
1130          JSB =ROMJSB
1140          PUMD R41,-R24
1150          LDM R20,=EOLSV          !RE-TAKE IOSP
1160          ADM R20,R22
1170          STM R20,R45
1180          CLB R47
1190          STMD R45,=ESHOOK
1200          RTN
1210 !***************************GIVE SIMPLE
1220          BYT 0,241
1230 GIVES.   BIN
1240          LDMD R22,=BINTAB
1250          POMD R32,-R12           !ADDRESS
1260          POMD R30,-R12           !LENGTH
1270          JZR GIVESR
1280          LDMD R24,X22,CCRADR
1290          LDMD R26,X22,OBADR
1300          LDB R20,=240            !"OUTPUT, SIMPLE"
1310          JSB X22,CMDHS
1320 GIVES1   LDBD R21,R24
1330          JNG GIVES1              !AWAIT OBF=0
1340          CLB R21
1350          STBD R21,R24            !CLEAR CCR
1360 GIVES2   POBD R20,+R32           !GET NEXT BYTE
1370 GIVES3   LDBD R21,R24
1380          JNG GIVES3              !AWAIT OBF=0
1390          DCM R30
1400          JZR GIVES4              !JIF LAST BYTE
1410          STBD R20,R26            !WRITE OB
1420          JMP GIVES2
1430 GIVES4   LDB R21,=4
1440          STBD R21,R24            !CED<-1
1450          STBD R20,R26            !WRITE LAST BYTE
1460 GIVES5   LDBD R21,R24
1470          ANM R21,=202
1480          JNZ GIVES5              !AWAIT OBF=BUSY=0
1490 GIVESR   RTN
1500 !***************************TAKE SIMPLE
1510          BYT 0,241
1520 TAKES.   BIN
1530          LDMD R22,=BINTAB
1540          JSB X22,SETSTR          !ARRANGE DATA SINK
1550          TSM R30
1560          JZR TAKESR              !JIF NO BYTES
1570          LDMD R24,X22,CCRADR
```

```
1580            LDMD R26,X22,OBADR
1590            LDB R20,=20              !"INPUT, SIMPLE"
1600            JSB X22,CMDHS
1610 TAKES0     LDBD R20,R24
1620            JNG TAKES0               !AWAIT OBF=0
1630            CLB R20
1640            STBD R20,R24             !CED<-0
1650 TAKES1     LDBD R21,R24
1660            JEV TAKES1               !AWAIT IBF=1
1670            LDBD R20,R26             !READ DATA
1680            PUBD R20,+R32            !STORE IT
1690            ANM R21,=4
1700            JNZ TAKES2               !JIF PED=1
1710            DCM R30                  !DROP COUNT
1720            JZR TAKES2               !JIF NO MORE ROOM
1730            STBD R30,R26             !REQUEST MORE
1740            JMP TAKES1
1750 TAKES2     LDB R20,=4
1760            STBD R20,R24             !CED<-1
1770 TAKES3     LDBD R21,R24
1780            ANM R21,=202
1790            JNZ TAKES3               !AWAIT OBF=BUSY=0
1800 TAKESR     RTN
1810 !**************************GIVE INTERRUPT
1820            BYT 0,241
1830 GIVEI.     BIN
1840            LDMD R22,=BINTAB
1850 GIVEI1     LDBD R20,X22,OACTV?      !1 AT A TIME
1860            JNZ GIVEI1
1870            LDMD R24,X22,CCRADR
1880            LDMD R26,X22,OBADR
1890            POMD R32,-R12            !ADDRESS
1900            POMD R30,-R12            !LENGTH
1910            JZR GIVEIR
1920            LDM R34,=377,0           !ACTIVE BUT NO EOT
1930            ADM R10,=3,0             !STEP PAST "AFTERWARD"
1940            STM R10,R36              !POINTER TO GOTO/GOSUB
1950            ADM R10,=3,0             !STEP PAST GOTO/GOSUB
1960            LDM R40,R30              !STORE POINTERS
1970            STMD R40,X22,OCOUNT
1980            LDB R20,=242             !"OUTPUT, INTERRUPT"
1990            JSB X22,CMDHS
2000 GIVEI2     LDBD R20,R24
2010            JNG GIVEI2               !AWAIT OBF=0
2020            CLB R20
2030            STBD R20,R24             !CCR<-0
2040            JSB X22,INTCHK           !NORMALIZE IOP
2050 GIVEIR     RTN
2060 !**************************TAKE INTERRUPT
2070            BYT 0,241
2080 TAKEI.     BIN
2090            LDMD R22,=BINTAB
```

```
2100 TAKEI1   LDBD R20,X22,IACTV?    !ONE AT A TIME
2110          JNZ TAKEI1
2120          LDMD R24,X22,CCRADR
2130          LDMD R26,X22,OBADR
2140          JSB X22,SETSTR         !ARRANGE DATA SINK
2150          TSM R30                !(LENGTH)
2160          JZR TAKEIR
2170          LDM R34,=377,0         !ACTIVE BUT NO EOT
2180          ADM R10,=3,0           !STEP PAST "AFTERWARD"
2190          STM R10,R36            !POINTER TO GOTO/GOSUB
2200          ADM R10,=3,0           !STEP PAST GOTO/GOSUB
2210          LDM R40,R30            !STORE POINTERS
2220          STMD R40,X22,ICOUNT
2230          JSB X22,SNDCNT         !BYTE COUNT TO IOP
2240          LDB R20,=22            !"INPUT, INTERRUPT"
2250          JSB X22,CMDHS
2260 TAKEI2   LDBD R20,R24
2270          JNG TAKEI2             !AWAIT OBF=0
2280          CLB R20
2290          STBD R20,R24           !CCR<-0
2300          JSB X22,INTCHK         !REVIVE CARD
2310 TAKEIR   RTN
2320 !**************************GIVE BURST
2330          BYT 0,241
2340 GIVEB.   BIN
2350          LDMD R22,=BINTAB
2360          LDMD R24,X22,CCRADR
2370          LDMD R26,X22,OBADR
2380 GIVEB1   LDBD R20,X22,OACTV?    !ONE AT A TIME
2390          JNZ GIVEB1

2400          POMD R32,-R12          !ADDRESS
2410          POMD R30,-R12          !LENGTH
2420          JZR GIVEBR
2430          JSB X22,SNDCNT         !BYTE COUNT TO IOP
2440          JSB X22,INTOFF         !DISABLE ALL INTERRUPTERS
2450          JSB X22,INTCHK         !RE-ENABLE MY IOP
2460 GIVEB2   LDBD R20,R24
2470          JNG GIVEB2             !AWAIT OBF=0
2480          LDB R20,=2
2490          STBD R20,R24           !COM<-1
2500          LDB R20,=42            !"OUTPUT, BURST"
2510          STBD R20,R26
2520 GIVEB3   LDBD R21,R24
2530          ANM R21,=202
2540          JNZ GIVEB3             !AWAIT OBF=BUSY=0
2550          STBD R21,R24           !CCR<-0
2560          LDMD R66,X22,OBADR
2570          STMD R66,=TEMP2        !INDIRECT ADDRESS
2580          DRP R40
2590          JSB X22,BOUTLP         !GO TO TIGHT LOOP
2600          JSB X22,INTON          !ALL DONE!
2610 GIVEBR   RTN
```

```
2620 !**************************TAKE BURST
2630            BYT 0,241
2640 TAKEB.     BIN
2650            LDMD R22,=BINTAB
2660            LDMD R24,X22,CCRADR
2670            LDMD R26,X22,OBADR
2680 TAKEB1     LDBD R20,X22,IACTV?     !1 AT A TIME
2690            JNZ TAKEB1
2700            JSB X22,SETSTR          !ARRANGE DATA SINK
2710            TSM R30                 !(LENGTH)
2720            JZR TAKEBR
2730            JSB X22,SNDCNT          !BYTE COUNT TO IOP
2740            JSB X22,INTOFF          !DISABLE ALL INTERRUPTERS
2750            JSB X22,INTCHK          !REVIVE MY IOP
2760 TAKEB2     LDBD R20,R24
2770            JNG TAKEB2              !AWAIT OBF=0
2780            LDB R20,=2
2790            STBD R20,R24            !COM<-1
2800            LDB R20,=45             !"INPUT. BURST"
2810            STBD R20,R26
2820 TAKEB3     LDBD R21,R24
2830            ANM R21,=202
2840            JNZ TAKEB3              !AWAIT OBF=BUSY=0
2850            STBD R21,R24            !CCR<-0
2860            LDMD R66,X22,OBADR      !INDIRECT POINTER
2870            STMD R66,=TEMP2
2880            DRP R40
2890            JSB X22,BINLOP          !TIGHT LOOP
2900            JSB X22,INTON           !ALL DONE
2910 TAKEBR     RTN
2920 !**************************TIGHT LOOPS
2930 !
2940 !
2950 !
2960 !
2970 !
2980 !
2990 !
3000 !
3010 !
3020 BOUTLP     ARP R32                 ! (JSB X22 EATS ARP)
3030 BOUTL      POBD R#,+R#             !NEXT BYTE FROM BUFFER
3040            STBI R#,=TEMP2          !ON TO IOP
3050 !******HALTED HERE**********
3060            JMP BOUTL
3070 !
3080 !
3090 !
3100 !
3110 !
3120 BINLOP     ARP R32                 ! (JSB X22 EATS ARP)
3130            STBI R#,=TEMP2          !TRIGGER TO OB
```

```
3140 BINLP+   LDBI R#,=TEMP2          !NEXT BYTE FROM IOP
3150 !******HALTED HERE*********
3160          PUBD R#,+R#             !ON TO BUFFER
3170          JMP BINLP+
3180 !**************************INTERRUPTERS OFF
3190 INTOFF   LDB R20,=2              !KEYBOARD
3200          STBD R20,=KEYDIS
3210 KEYDIS   DAD 177402
3220          LDB R20,=1
3230          STBD R20,=OTHRDS        !TIMERS
3240 OTHRDS   DAD 177412
3250          LDB R20,=101
3260          STBD R20,=OTHRDS
3270          LDB R20,=201
3280          STBD R20,=OTHRDS
3290          LDB R20,=301
3300          STBD R20,=OTHRDS
3310          LDB R20,=10             !EACH SELECT CODE
3320          LDBD R21,=SCLOG
3330          LDM R24,=120,377        !SC 3 CCR
3340          LDM R26,=121,377        !SC 3 OB
3350 INTOF1   TSB R21
3360          JEV INTOF2              !NOT RESIDENT
3370          PUMD R20,+R6
3380          LDB R20,=60             !"TURN OFF!"
3390          JSB X22,CMDHS
3400          POMD R20,-R6
3410 INTOF2   LRB R21                 !NEXT SELECT CODE
3420          ADM R24,=2,0
3430          ADM R26,=2,0
3440          DCB R20
3450          JNZ INTOF1
3460          LDMD R24,X22,CCRADR
3470          LDMD R26,X22,OBADR
3480          RTN
3490 !**************************INTERRUPTERS ON
3500 INTON    LDB R20,=10             !EACH SELECT CODE
3510          LDBD R21,=SCLOG
3520          LDM R24,=120,377        !SC 3 CCR
3530          LDM R26,=121,377        !SC 3 OB
3540 INTON1   TSB R21
3550          JEV INTON2              !NOT RESIDENT
3560          PUMD R20,+R6
3570          JSB X22,INTCHK          !REVIVE IOP
3580          LDM R74,R24             !FOR OBF,BUSY CHECK LATER
3590          POMD R20,-R6
3600 INTON2   LRB R21                 !NEXT SELECT CODE
3610          ADM R24,=2,0
3620          ADM R26,=2,0
3630          DCB R20
3640          JNZ INTON1
3650          STM R74,R24
```

```
3660 INTON3   LDBD R21,R24
3670          ANM R21,=202
3680          JNZ INTON3              !AWAIT OBF=BUSY=0
3690          LDB R20,=1              !KEYBOARD
3700          STBD R20,=KEYDIS
3710          LDB R20,=2              !TIMERS
3720          STBD R20,=OTHRDS
3730          LDB R20,=102
3740          STBD R20,=OTHRDS
3750          LDB R20,=202
3760          STBD R20,=OTHRDS
3770          LDB R20,=302
3780          STBD R20,=OTHRDS
3790          LDMD R24,X22,CCRADR
3800          LDMD R26,X22,OBADR
3810          RTN
3820 !**************************REVIVE INTERRUPTED IOP
3830 INTCHK   LDBD R21,R24
3840          ANM R21,=10
3850          JNZ INTCH1
3860          RTN                     !RTN IF PACK=0
3870 INTCH1   LDB R20,=1
3880          STBD R20,R24            !INT<-1
3890          JSB X22,DUMMY           !WASTE TIME
3900          CLB R20
3910          STBD R20,R24            !INT<-0
3920 INTCH2   LDBD R21,R24
3930          ANM R21,=10
3940          JNZ INTCH2
3950          RTN
3960 !**************************COMMAND TO IOP ROUTINE
3970 CMDHS    TSB R20
3980          JLZ INTHS               !JIF STATUS
3990          LDBD R21,R24
4000          ANM R21,=40
4010          JZR CMDHS0              !JIF NOT FULL DUPLEX IOP
4020          CMB R20,=22
4030          JZR CMDHS2              !JIF XFER IN
4040          CMB R20,=242
4050          JZR INTHS               !JIF XFER OUT
4060          CMB R20,=231
4070          JZR INTHS               !JIF WRITE TERMS
4080 CMDHS0   CMB R20,=235
4090          JZR INTHS               !JIF SERVICE REQUEST
4100          CMB R20,=234
4110          JZR INTHS               !JIF ASSERT
4120          CMB R20,=111
4130          JZR INTHS               !JIF RESUME
4140          CMB R20,=110
4150          JZR INTHS               !JIF HALT
4160          CMB R20,=103
4170          JCY CMDHS1              !JIF > SET REN=0
```

```
4180            CMB R20,=60
4190            JCY INTHS                   !JIF > DISABLE INTERRUPTS
4200 CMDHS1     LDBD R21,R24
4210            ANM R21,=202
4220            JNZ CMDHS1                  !AWAIT OBF=BUSY=0
4230 CMDHS2     LDB R21,=2                  !COM<-1
4240            STBD R21,R24
4250            STBD R20,R26                !OB<-COMMAND
4260            RTN
4270 INTHS      LDB R21,=1
4280            STBD R21,=GINTDS
4290            STBD R21,R24                !INT<-1
4300 INTHS1     LDBD R21,R24
4310            ANM R21,=10
4320            JZR INTHS1                  !AWAIT PACK=1
4330            STBD R21,=GINTEN
4340            STBD R20,R26                !WRITE COMMAND
4350            LDBD R20,R26                !READ IB
4360            LDB R20,=2
4370            STBD R20,R24                !COM<-1, INT<-0
4380            RTN
4390 !***************************SET UP DATA BUFFER
4400 SETSTR     POMD R66,-R12               !ADDRESS
4410            POMD R66,-R12               !LENGTH
4420            POMD R66,-R12               !BASE ADDRESS
4430            JSB =FETSVA                 !GET TRUE POINTER
4440            POMD R64,+R34               !TOT & MAX LEN'S
4450            PUMD R66,+R34               !ACT <- MAX
4460            STM R66,R30                 !R30 <- BYTE COUNT
4470            LDM R32,R34                 !R32<-BUFFER ADDRESS
4480            RTN
4490 !***************************SEND BYTE COUNT TO IOP
4500 SNDCNT     LDB R20,=231                !"WRITE TERMS"
4510            JSB X22,CMDHS
4520 SNDCN1     LDBD R21,R24
4530            JNG SNDCN1                  !AWAIT OBF=0
4540            CLB R20
4550            STBD R20,R24                !CED<-0
4560            STBD R30,R26                !SEND COUNT LSB
4570 SNDCN2     LDBD R20,R24
4580            JNG SNDCN2                  !AWAIT OBF=0
4590            STBD R31,R26                !SEND COUNT MSB
4600 SNDCN3     LDBD R20,R24
4610            JNG SNDCN3                  !AWAIT OBF=0
4620            RTN
4630 !***************************ON BREAK GOTO/GOSUB
4640            BYT 0,241
4650 ONBRK.     BIN
4660            LDMD R22,=BINTAB
4670            LDMD R24,X22,CCRADR
4680            LDMD R26,X22,OBADR
4690            STM R10,R46                 !POINTER TO GOTO/GOSUB
```

```
4700            ADM R10,=3,0                !STEP PAST GOTO/GOSUB
4710            LDB R44,=377                !CONDITION ACTIVE
4720            LDB R45,=0                  !ARMED, BUT NO TRIGGER YET
4730            STMD R44,X22,CACTV?         !STORE THE SETUP
4740            LDB R20,=201                !"WRITE CONTROL REG #1"
4750            JSB X22,CMDHS
4760 ONBRK1     LDBD R21,R24
4770            JNG ONBRK1                  !AWAIT OBF=0
4780            LDB R21,=4
4790            STBD R21,R24                !CED<-1
4800            LDB R21,=200                !BREAK INTERRUPT
4810            STBD R21,R26                !   ENABLE MASK
4820 ONBRK2     LDBD R21,R24
4830            ANM R21,=202
4840            JNZ ONBRK2                  !AWAIT OBF=BUSY=0
4850            JSB X22,INTCHK              !NORMALIZE IOP
4860            RTN
4870 !*************************TELL SYSTEM TO CALL IOSP HOOK
4880 SETEOL     STBD R#,=GINTDS             !REQUEST END OF LINE BRANCH
4890            LDB R20,=2
4900            LDBD R21,=SVCWRD
4910            ORB R21,R20
4920            STBD R21,=SVCWRD
4930            LDB R21,=20
4940            ORB R17,R21
4950            STBD R#,=GINTEN
4960            RTN
4970 !*************************ACKNOWLEDGE BREAK
4980            BYT 0,241
4990 ACBR.      BIN
5000            LDMD R22,=BINTAB
5010            LDMD R24,X22,CCRADR
5020            LDMD R26,X22,OBADR
5030            LDB R20,=1                  !"STATUS (REG 1)"
5040            JSB X22,CMDHS
5050 ACBR.1     LDBD R21,R24
5060            JNG ACBR.1                  !AWAIT OBF=0
5070            CLB R21
5080            STBD R21,R24                !CCR<-0
5090 ACBR.2     LDBD R21,R24
5100            JEV ACBR.2                  !AWAIT IBF=1
5110            LDBD R21,R26                !READ REG 1
5120            LDB R21,=4
5130            STBD R21,R24                !CED<-1
5140 ACBR.3     LDBD R21,R24
5150            ANM R21,=202
5160            JNZ ACBR.3                  !AWAIT OBF=BUSY=0
5170            JSB X22,INTCHK              !REVIVE IOP
5180            RTN
5190 !************************* SHOVE
5200            BYT 0,241
5210 SHOV.      JSB =TWOB                   !R56<-REG#; R46<-DATA
```

```
5220              BIN
5230              LDMD R22,=BINTAB
5240              LDMD R24,X22,CCRADR
5250              LDMD R26,X22,OBADR
5260              LDB R20,=200                 !"WRITE CONTROL"
5270              LDB R21,R56                  !REGISTER #
5280              ANM R21,=37
5290              ORB R20,R21
5300              JSB X22,CMDHS
5310 SHDV.1       LDBD R21,R24
5320              JNG SHDV.1                   !AWAIT OBF = 0
5330              LDB R21,=4
5340              STBD R21,R24                 !CED<-1
5350              STBD R46,R26                 !OB<-DATA
5360 SHDV.2       LDBD R21,R24
5370              ANM R21,=202
5380              JNZ SHDV.2                   !AWAIT OBF=BUSY=0
5390              RTN
5400              LNK EXAMPLES3
```

```
10 !******************************BINARY PROGRAM'S DATA AREA
20 IOSAVE    BSZ 23                                 !OTHER ROM'S HOOK STORAGE
30 CCRADR    BSZ 2                                  !MY IOP'S CCR/PSR ADDRESS
40 OBADR     BSZ 2                                  !MY IOP'S OB/IB ADDRESS
50 ICOUNT    BSZ 2                                  !INPUT CHARACTER COUNTER
60 IPOINT    BSZ 2                                  !"  BUFFER POINTER
70 IACTV?    BSZ 1                                  !"  TRANSFER ACTIVE BOOLEAN
80 IEOLFL    BSZ 1                                  !"  EOL BRANCH REQUEST BOOLEAN
90 IEOL10    BSZ 2                                  !"  GOTO/GOSUB POINTER
100 OCOUNT    BSZ 2                                 !OUTPUT CHARACTER COUNT
110 OPOINT    BSZ 2                                 !"  BUFFER POINTER
120 OACTV?    BSZ 1                                 !"  TRANSFER ACTIVE BOOLEAN
130 OEOLFL    BSZ 1                                 !"  EOL BRANCH REQUEST BOOLEAN
140 OEOL10    BSZ 2                                 !"  GOTO/GOSUB POINTER
150 CACTV?    BSZ 1                                 !ON BREAK ACTIVE BOOLEAN
160 CEOLFL    BSZ 1                                 !"  EOL BRANCH REQUEST BOOLEAN
170 CEOL10    BSZ 2                                 !"  GOTO/GOSUB POINTER
180 STEST?    BSZ 1                                 !TEMPORARY SELF TEST RESULT STORAGE
190 SRVEOL    BSZ 1                                 !EOL BRANCH IN PROGRESS BOOLEAN
200 ELBORM    BSZ 12                                !MAKE INITIALIZATION EASY
210 !****************************SYSTEM ADDRESSES NOT IN "GLOBAL"
220 CNTRTN    DAD 36002                             !WAITS [R31] *16.67 MILLISECONDS
230 SCLOG     DAD 100667                            !BIT LOG OF RESIDENT SELECT CODES
240 R6LIM2    DAD 101720                            !STACK OVERFLOW FENCE
250 SYSERR    DAD 76713                             !SYSTEM ERROR LOGGING ROUTINE
260 IRQPAD    DAD 102505                            !ADDRESS FOR STACK OVERFLOW TEST
270 IRQRTN    DAD 102506                            !   " SAME "
280 INTRSC    DAD 177500                            !I/O ADDRESS FOR TC CONTROL, ETC.
290 SYSRTN    DAD 310                               !ADDRESS OF A RTN INSTRUCTION
300 CLKHIT    DAD 507                               !SYSTEM EOL SERVICE ENTRY
310 CHREDT    DAD 364                               !   " SAME "
320 XCBIT3    DAD 274                               !   " SAME "
330 TRA?      DAD 1523                              !SYSTEM TRACE ENTRY
340 SETTR1    DAD 2323                              !   " SAME "
350 CLRBIT    DAD 546                               !SYSTEM UNLOGS I/O'S EOL REQUEST
360 IPHERE    DAD 101042                            !BOOLEAN => IPBIN BINARY IS LOADE
370 GOTOSU    DAD 17435                             !SYSTEM GOTO/GOSUB PARSE ROUTINE
380 TEMP2     DAD 101120                            !OB/IB ADDRESS FOR BURST INDEXED
390 ESHOOK    DAD 102412                            !ROUTINE ADDRESS IN IOSP HOOK
400           FIN
```

INDEX