4

input/output operations

Three types of data transfer may be used to receive or transmit information between the PDP-8/E and one or more peripheral I/O devices. Programmed data transfer provides a straight forward means of communicating with relatively slow I/O devices, such as the console terminal. Program interrupt transfers use the interrupt system to service several peripheral devices simultaneously, on an intermittent basis, permitting computational operations to be performed concurrently with the data I/O operations. Data break transfers rely on direct memory access to transfer variable-size blocks of data between high-speed peripherals and the processor with a minimum of program control. This choice of I/O techniques affords PDP-8/E users the means to execute data I/O operations at rates ranging from a few characters per second up to more than 10,000 characters per second, depending only upon the characteristics of a given peripheral device.

PROGRAMMED DATA TRANSFER

Programmed data transfer is the easiest and most common means of performing data I/O. Every input/output transfer (IOT) instruction initiates one programmed transfer which may transmit data or status information either to or from a peripheral device. The amount of information that will be transferred by a single IOT instruction depends upon the particular operation that is coded into the instruction and the complexity of the I/O device interface. In general, programmed data transfers are limited to a maximum of 12 bits of data or 1 bit of status information per IOT instruction; however, there are many exceptions to this rule, and an I/O device may transfer any amount of data in response to a single IOT instruction if its interface circuitry is designed accordingly.

A programmed data transfer begins when the central processor reads an instruction from memory, loads the first three bits of the instruction into the instruction register, and recognizes that the current instruction is an IOT instruction, with an OP-code of 6. If the processor is operating in executive mode (i.e. not in a timesharing environment), it then concludes TS2 by transmitting a control signal to every peripheral device. This control signal instructs each peripheral to accept and decode bits 3-8 of the IOT instruction.

Bits 3-8 of every IOT instruction contain the device selection code that determines the specific I/O device for which the IOT instruction is intended. When an I/O device recognizes a device code as one of its assigned codes, it identifies itself to the processor as either an internal device whose interface control module plugs directly into the OMNIBUS or an external device that is connected to the positive I/O interface. The designated device then accepts and decodes bits 9-11 of the IOT instruction.

Bits 9-11 of every IOT instruction contain the operation specification code that determines the specific operation to be performed. The nature of this operation for any given IOT instruction depends entirely upon the circuitry designed into the I/O device interface; however, the usual practice is to use bit 11 (codes 0-1) to specify operations that test device status, bit 10 (codes 2-3) for operations that set or modify device status, and bit 9 (codes 4-7) for operations involving actual data transfer between the processor and the I/O device.

In order to simplify interface design for external (non-OMNIBUS) I/O devices, operation specification bits are transmitted to these devices as positive-going Input/Output Pulses (IOPs) generated by the KA8-E Positive I/O Interface. IOPs are simply the buffered, low-order 3 bits of the IOT instruction. If bit 11 of an IOT instruction is set, the external device receives an IOP1 pulse. Setting bit 10 generates an IOP2 pulse, while bit 9 generates an IOP4 pulse. If an IOT instruction ends in 0, none of the operation specification bits are set and no IOPs are generated. In any case, the device receives one additional pulse after the last IOP, as a signal to begin the specified I/O operation. IOPs are transmitted serially, so that edge-triggered logic may be used in the device interface. IOP spacing and duration are adjustable over a wide range; however, increasing the spacing or duration of the IOPs results in longer IOT execution time for all external devices.

Peripheral Device Status

Many I/O devices, including almost all serial devices such as the console terminal, maintain only one bit of status information. This is usually the state of a busy/done flip-flop which indicates whether the device is in the process of performing a data transfer or free to commence a new I/O operation. Thus, IOT instructions which set or modify device status often require that no information be transferred other than the operation specification bits of the IOT instruction.

If an information transfer is required, the I/O device must decode the operation specification bits to determine the exact nature of the transfer, gate the content of its device buffer onto the OMNIBUS if necessary, then generate up to three control signals which enable the adder circuits and shift gates of the central processor to perform one of six possible operations:

1. Data may be received from a device, ORed with the content of the AC, and the result loaded into the AC.

2. Data may be received from a device and added to the content of the PC.

3. Data may be received from a device and loaded into the PC.

4. The content of the AC may be sent to a device, and the AC may then be cleared.

5. Data may be received from a device and loaded into the AC.

6. The content of the AC may be sent to a device.

The six operations listed above are the only data transfer operations that may be performed during a programmed data transfer by any I/O device, but not all of these operations are performed by every device. Note that these operations are performed by circuitry in the central processor, controlled by signals generated at the I/O device interface. The maximum of three control signals an I/O device may generate for this purpose provides a total of eight data transfer operations, two of which are redundant.

All of the operations associated with a programmed date transfer must be completed by the end of TS3 of the FETCH cycle in which the IOT instruction was read from memory. If a peripheral device requires additional time to complete a data transfer, it may transmit a control signal that disables processor timing during some or all of the operations it is capable of performing. All peripheral devices connected to the positive I/O interface rely on this feature to extend IOT instruction timing for up to 3.4 microseconds, depending upon the nature of the operation being performed.

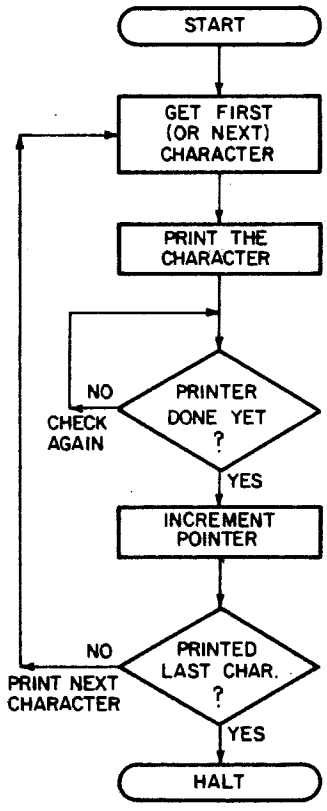
Since IOPS are transmitted serially, the time required to execute an external IOT instruction depends upon how many IOPs must be generated. This, in turn, depends upon the operation specification code of the IOT instruction. Thus, input/output transfer instructions directed to standard external I/O devices will be executed in 2.6 microseconds if the octal code for the IOT ends in 1, 2, or 4 (one operation specification bit set). The IOT instruction will require 3.6 microseconds if its octal code ends in 3, 5, or 6 (two operation specification bits set), 4.6 microseconds if the octal code ends in 7, or 1.2 microseconds if the octal code ends in 0.

Programmed Data Transfer Timing Constraints

Most I/O devices are capable of performing one 12-bit data transfer between the accumulator and the device buffer in 4.6 microseconds or less. Once this transfer is completed, however, the I/O device must dispose of the transferred data, if the operation was an output transfer, or load new data into its buffer, for the next input transfer. In the case of the Teletype terminal, for example, it is possible to read the content of the Teletype data buffer into the AC in 2.6 microseconds, which implies a maximum transfer rate of about 120,000 characters per second. The actual Teletype transfer rate of 10 characters per second reflects the fact that the mechanical mechanism within the keyboard/reader requires at least one tenth of a second to recognize which key on the console has been typed and load the device buffer with the corresponding ASCII code. If manual input is being generated at the keyboard, the maximum transfer rate may fall to 3 or 4 characters per second, corresponding to the highest typing speed of an average typist.

Similar problems arise to limit the maximum rate at which output transfers occur. The Teletype mechanism requires a minimum of about 100 milliseconds to read the device buffer, select a corresponding ASCII character, and print or punch the character. Even relatively high-speed devices such as line printers typically operate at rates of about 2000 characters per second or less, which is much slower than the maximum rate at which the processor may perform programmed data transfers.

This explains why programmed transfers often involve an exchange of status information. If the central processor is executing programmed transfers to transmit information to a device on a continual basis, it is essential that the processor check the status of the device busy/done flip-flop before executing a transfer IOT instruction, and postpone each transfer until after the device has finished all operations required to complete the previous data transfer. If data is to be read from an I/O device, the processor must still monitor the device status in this manner and postpone each transfer until the I/O device has finished assembling the data and loading this information into its device buffer. When programmed data transfer is employed, the processor uses IOT instructions that transfer device status information to monitor (or modify) the state of the device busy/done flip-flop, as well as any other status indicators which may be present in the device.




```
0200    *200
1607    START,  TAD I POINTR
6046            TLS
6041    TEST,   TSF
5202            JMP TEST
2207            ISZ POINTR
5200            JMP START
7402            HLT
7600    POINTR, 7600
        $
```

Figure 4-1 Program to Print ASCII Characters Using Programmed Data Transfer

Figure 4-1 shows a flowchart and the corresponding assembly language program which may be used to print 128 consecutive ASCII characters on the PDP-8/E console terminal. If the ASCII character codes are stored in page 37 of a 4K memory while the routine is loaded into the first 8 locations of page 2 and started at location 0200, this program will print all 128 characters in about 13 seconds. One character is transferred every 100 milliseconds, almost all of which represents time required for the Teletype terminal to complete the previous data transfer. Thus, the program executes a TLS instruction in 3.6 microseconds, then cycles through the delay loop about 250,000 times until the device status changes and it skips out of the loop to execute the next data transfer.

In summary, programmed data transfer provides an easy, convenient means of performing data I/O with a minimum of hardware and software support. The highest transfer rate that may be realized using programmed I/O is only slightly lower than the maximum rate at which the fastest available I/O device may operate. The major drawback associated with this technique is that the central processor must hang up in a waiting loop while the I/O device completes the last transfer and prepares for the next transfer. It is possible to use some of this waiting time to perform intermediate calculations, but it is rarely convenient to do so. On the other hand, programmed data transfer techniques permit easy hardware implementation and simple, economical interface design. For this reason, almost all I/O devices except bulk storage units rely heavily on programmed data transfer for routine data I/O.

PROGRAM INTERRUPT TRANSFERS

The program interrupt system may be used to initiate programmed data transfers in such a way that the time spent waiting for device flags is greatly reduced or eliminated altogether. It also provides a means of performing concurrent programmed data transfers between the central processor and two or more peripheral devices. This is accomplished by isolating the I/O handling routines from the mainline program and using the interrupt system to ensure that these routines are entered only when an I/O device flag is set, indicating that the device is actually free to perform the next data transfer, or that it requires some sort of intervention from the running program.

Interrupt System Operation

All peripheral device status indicators are ORed onto a special OMNIBUS signal line called the interrupt request line, which is asserted whenever one or more of the device flags is set. The processor interrogates the interrupt enable flip-flop and the interrupt request line during TS4 of every memory cycle in which execution of an instruction was completed. If the interrupt system is enabled and the interrupt request line is asserted, indicating that the interrupt system has been turned on and that one or more device flags were set while the current instruction was being executed, the processor executes a program interrupt.

A program interrupt is simply a conventional JMS instruction to memory location 0000 in field 0 (octal code 4000) which is built and executed by circuits in the processor. Three conditions must be satisfied before a program interrupt may occur: The interrupt system must be

enabled, the interrupt request line must be asserted by a peripheral whose device flag is set, and the processor must be in TS4 of the cycle in which execution of an instruction was completed, implying that the major state generator has just enabled the FETCH major state for the following cycle. If these conditions are met, the processor asserts an OMNIBUS signal called INT IN PROG, which forces a 4 (OP-code for a JMS instruction) into the instruction register, clears the CPMA register and enables the EXECUTE major state.

During the next machine cycle, the processor turns off the interrupt system so that no further interrupts may occur until the current interrupt has been serviced. It then executes the hardware-generated JMS to memory location 00000 which was built during TS4 of the previous cycle. Since the processor has no way of knowing that the JMS instruction is actually a program interrupt, it proceeds as though the EXECUTE cycle had been entered from FETCH and treats the hardware-generated JMS as a normal machine instruction. This causes the address of the instruction that was due to be executed when the program interrupt occurred to be stored in location 0000 of memory field 0. The PC is then loaded with 0001, so that the instruction stored in location 0001 of field 0 is the next instruction to be executed.

Figure 4-2 is a simplified flow chart that shows how the processor interrogates the interrupt system during TS4 of every memory cycle. If all conditions for a program interrupt are met, the processor concludes TS4 by initializing its control circuitry to execute a program interrupt during the following cycle. Once the interrupt has been executed, the previous content of the PC will be stored in location 0000 of field 0, the interrupt system will be turned off, and the PC will point to location 00001, which contains the next instruction to be executed.

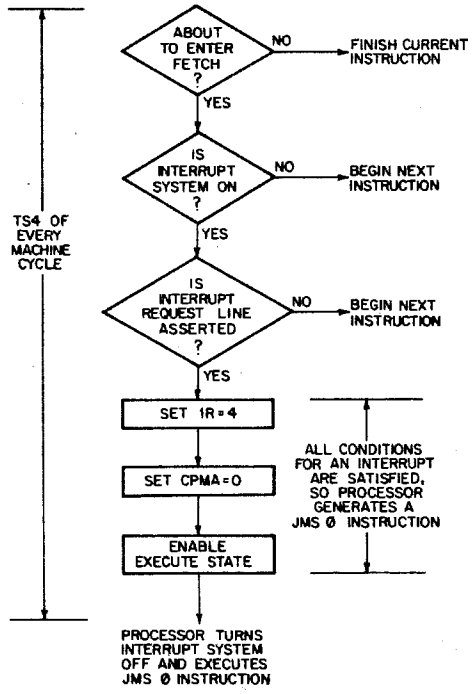


Figure 4-2 Program Interrupt Flow of Events

The interrupt system is a simple piece of hardware that has far-reaching program implications. When an interrupt occurs, mainline execution is suspended and the instruction stored in memory location 00001 is executed next. This instruction is usually a jump to the starting address of an interrupt service routine, which generally performs all of the following operations:

1. Save the content of any registers (AC, MQ, link, etc.) that will be used by the interrupt service routine.
2. Determine which peripheral device caused the program interrupt.
3. Determine why the device caused an interrupt.
4. Correct the condition that resulted in the interrupt.
5. Restore the content of registers that were used by the interrupt service routine.
6. Turn the interrupt system on.
7. Resume mainline execution, usually by executing a JMP I 0 instruction (octal code 5400).

A running program maintains control over the interrupt system by executing the processor IOT instructions listed in Table 4-1. Several of these interrupt IOT instructions are also used in conjunction with the KM8-E Extended Memory and Time Share option, to service program interrupts that originate in extended memory or in a timesharing environment. The user flag, save field register and interrupt inhibit flip-flop are hardware components of the extended memory and time share control, whose functions are described, briefly, in the following paragraphs. The greater than flag (GTF) is contained in the KE8-E Extended Arithmetic Element.

Table 4-1 Interrupt IOT Instructions

MNEMONIC	OCTAL	OPERATION
SKON	**6000**	**Skip if Interrupt System On.** The state of the interrupt enable flip-flop is tested. If this flip-flop is set, indicating that the interrupt system is enabled, the PC is incremented to skip the next sequential instruction and the interrupt system is turned off in the same manner as by an IOF instruction.
ION	**6001**	**Interrupt Turn On.** The next program instruction is executed, then the interrupt system is enabled. Delaying the interrupt enable in this manner gives the interrupt service routine time to resume background program execution (by means of a JMP I 0 instruction) before another program interrupt occurs.

Table 4-1 Interrupt IOT Instructions (continued)

MNEMONIC	OCTAL	OPERATION
IOF	**6002**	**Interrupt Turn Off.** The interrupt system is disabled during TS3, inhibiting further program interrupts, including any interrupt request that might have been flagged during execution of the IOF instruction.
SRQ	**6003**	**Skip on Interrupt Request.** The state of the OMNIBUS interrupt request line is tested. If this line is asserted, indicating that one or more devices are requesting a program interrupt, the PC is incremented to skip the next sequential instruction.
GTF	**6004**	**Get Flags.** The following machine states are read into the indicated bits of the accumulator: AC BIT LOADED WITH CONTENT OR STATE OF: AC0 Link AC1 Greater than flag AC2 Interrupt request line AC3 Interrupt inhibit flip-flop AC4 Interrupt enable flip-flop AC5 User flag AC6-11 Save field register
RTF	**6005**	**Restore Flags.** This instruction is the converse of GTF. AC0 is loaded into the link, AC1 is loaded into the Greater Than Flag, AC5 is loaded into the User Flag, AC6-11 are loaded into the Save Field Register, the interrupt system is enabled in the same manner as by an ION instruction, and the interrupt inhibit flip-flop is set.
SGT	**6006**	**Skip if Greater Than.** If the Greater Than Flag is set, the PC is incremented to skip the next sequential instruction.
CAF	**6007**	**Clear All Flags.** This instruction is logically equivalent to operating the CLEAR switch on the programmer's console. It generates an INITIALIZE pulse on the OMNIBUS and at the external I/O interface. The AC and Link are cleared. The action of INITIALIZE depends upon the design of each peripheral control, but it generally clears all I/O device flags and motion control flip-flops, and sets the interrupt enable flip-flop in each peripheral device. A CAF instruction should not be executed while a device is active. For example, a CAF instruction should not be executed within 100 milliseconds of a TLS instruction.

Interrupt System Hardware Components

The only means of turning the interrupt system on is by executing an ION instruction, which unconditionally sets the interrupt enable flip-flop on the timing generator module. Once the interrupt enable flip-flop has been set, TP1 of the next FETCH cycle automatically sets the interrupt delay flip-flop, and fully enables the interrupt system. The 1-cycle delay provided by the interrupt delay flip-flop prevents a program interrupt from occurring during TS4 of the cycle in which the ION instruction was executed (or during any DMA cycles that might intervene following execution of the ION instruction). This gives the running program time to execute one additional instruction, usually a JMP I 0 (octal code 5400) to transfer program control to the background routine before another interrupt occurs.

The interrupt system monitors two OMNIBUS signals, F SET L, which is asserted whenever an instruction is in its concluding cycle, and MS IR DIS L, which is asserted during every DMA operation. If an interrupt request occurs during any cycle in which the interrupt delay flip-flop is set (implying that the interrupt enable flip-flop is also set), F SET L is asserted, and MS IR DIS L is not asserted, the interrupt system generates a pulse that is clocked into the interrupt sync flip-flop by OMNIBUS signal INT STROBE H. This sets the interrupt sync flip-flop, which generates signal INT IN PROG H and gates it onto the OMNIBUS.

Signal INT IN PROG H loads the IR register, zeroes the CPMA register and enables the EXECUTE major state during the following cycle. At TP1 of this cycle, the interrupt enable flip-flop is cleared to prevent further interrupts until after the current interrupt has been serviced. This flip-flop is also cleared by signal INITIALIZE H and by the IOF instruction. Clearing the interrupt enable flip-flop automatically clears the interrupt delay and interrupt sync flip-flops.

When extended memory is installed, it becomes necessary to inhibit program interrupts while the processor is loading the instruction field register and transferring control from one memory field to another. The extended memory and time share control contains an interrupt inhibit flip-flop which is set during TS3 of every instruction that modifies the content of the instruction field register (i.e. RTF, RMF and CIF). This flip-flop remains set until the next JMP or JMS instruction is executed. While the interrupt inhibit flip-flop is set it generates signal INT INHIBIT, which is displayed in bit 3 of the programmer's console STATUS indicator register (labelled NO INT), and grounds OMNIBUS signal INT IN PROG H, thereby preventing any device from entering a program interrupt request. This allows the processor ample time to complete any operations that may be required to initialize the machine registers before branching to a different memory field, and then execute the JMP or JMS into extended memory. The interrupt inhibit flip-flop is cleared during TS3 of every JMP or JMS instruction.

The extended memory and time share control also contains two interrupt buffers that preserve memory field and machine status information during an interrupt. When an interrupt occurs, the content of the 1-bit user flag and the 3-bit instruction field register are automatically loaded into bits 0 and 1-3, respectively, of interrupt buffer A, while the

content of the 3-bit data field register is loaded into interrupt buffer B. The two interrupt buffers are often considered as a single, 7-bit register called the save field register. The save field register is never cleared; information is always jam transferred in during execution of a program interrupt, and retained until the next program interrupt occurs or the next RTF instruction is executed. The content of the save field register is displayed in bits 5-11 of the programmer's console STATUS indicator register.

If it becomes necessary to service multiple or nested program interrupts, the GTF instruction may be used to read the content of the save field register into the AC so that this value may be stored in memory, along with the content of the AC, the MQ, location 00000, and other registers that might be modified by a program interrupt. It is then possible to re-enable the interrupt system while the current program interrupt is being serviced. When extended memory and timesharing status information is saved in this manner, it may be restored to the save field register by means of an RTF instruction. Further information on techniques for servicing multiple interrupts and program interrupts that originate in extended memory or a timesharing environment appears later in this handbook, as well as in Chapter 6 of *Introduction to Programming 1972.*

```
*0                                      PAL8-V7          PAGE 1

            0000  *0
00000  0201        ROTATE          /FIRST RETURN ADDRESS
00001  2022        ISZ POINTR      /INCREMENT POINTER
00002  7410        SKP             /PRINTED ALL CHARS?
00003  7402        HLT             /YES: HALT
00004  3017        DCA SAVEAC      /NO: SAVE AC
00005  7004        RAL             /GET THE LINK
00006  3020        DCA SAVEL       /AND SAVE IT
00007  1422        TAD I POINTR    /GET NEXT CHAR
00010  6046  PRINT, TLS             /PRINT IT
00011  7300        CLA CLL         /CLEAR AC AND LINK
00012  1020        TAD SAVEL       /GET STORED LINK
00013  7010        RAR             /ROTATE INTO PLACE
00014  1017        TAD SAVEAC      /GET STORED AC
00015  6001        ION             /ENABLE INTERRUPTS
00016  5400        JMP I 0         /RESUME MAINLINE
00017  0001  SAVEAC, 1
00020  0000  SAVEL,  0
00021  0000  TIMER,  0
00022  7600  POINTR, 7600
            0200  *200                    /DUMMY MAINLINE FOLLOWS
00200  5010  START,  JMP PRINT      /PRINT 1ST CHAR
00201  7004  ROTATE, RAL            /ROTATE AC BIT LEFT
00202  2021  DELAY,  ISZ TIMER      /DELAY LOOP;
00203  5202        JMP DELAY       /STALLS ABOUT 20 MSEC
00204  5201        JMP ROTATE
            $
```

Figure 4-3 Simplified Interrupt Service Routine

The program example of Figure 4-3 illustrates a very simple interrupt service routine. This example performs the same operations as the example of Figure 4-1; however, the interrupt driven program executes a dummy background routine that rotates one bit endlessly through the accumulator while the stored characters are being printed on the Teletype. A delay loop is included in the background routine, so that the rotating display moves slowly enough to be visible at the programmer's console AC register. This example requires nearly the same amount of time as the previous example to print all 128 stored characters, even though it is executing another data processing operation concurrently with the data transfer.

Highly developed interrupt service routines written for the PDP-8/E permit the processor to exercise simultaneous control over many peripheral devices while executing a background program that may be completely unaware of the detailed operation of the I/O process. Devices can be serviced on a first come, first served or a round robin basis. Additional techniques permit interrupts from a high priority device to supersede low priority interrupts, so that the high priority device is always serviced in the shortest possible time. At large installations, a software priority interrupt system may be designed to service many different I/O devices on a priority basis. Whichever system is employed, use of the program interrupt system affords a significant increase in I/O processing capability by eliminating the processor waiting time that is often associated with programmed I/O.

DATA BREAK TRANSFERS

Data break, sometimes called direct memory accessor DMA, is the preferred form of data transfer for use with high-speed storage devices such as magnetic disk or DECtape units. Direct memory access is indicated whenever it becomes necessary to transfer data contained in a block of consecutive memory locations out to a high-speed peripheral device, or to read sequential words of data from a high-speed device into a specified memory buffer area.

Data break peripherals are supplied with software subroutines featuring convenient, standardized calling sequences that perform all normal functions of data I/O. Thus, for most applications, users need not be concerned with the detailed operating characteristics of a particular DMA device. The remainder of this chapter describes the general operation of the data break system in terms that apply to all standard DMA devices.

Current Address (CA) Register

A 12-bit current address register is associated with every data break device. At the beginning of a data break transfer, the CA register contains the 12-bit address of the memory location in which the last data break transfer was performed. The content of the CA register is incremented by 1 during a data break transfer, and the incremented value is used as the address of the memory location with which the current transfer will be performed. In this manner, a single I/O operation may transfer up to 4096 words of data between a peripheral device and a series of sequential memory locations.

Word Count (WC) Register

A 12-bit word count register is also associated with every data break device. At the beginning of a data break transfer, the WC register contains the negative (two's complement) of the number of 12-bit words that remain to be transferred. The content of the WC register is incremented by 1 during every data break transfer. If this value becomes zero, word count overflow has occurred indicating that the word currently being transferred is the last word in the data block. Word count overflow generates a control signal which clears the I/O device enabling circuits and inhibits further data transfers.