Table 8.2 shows a list of typical arithmetic instructions. The increment instruction adds 1 to the value stored in a register or memory word. A unique feature of increment operation when executed in processor register is that a binary number all 1's on incrementing produces a result of all 0's. Similarly in case of decrement instruction a number of all 0's, when decremented produces a number with all 1's.

The four kind of instruction may be available for different types of data. The data type assumed to be in processor registers during the execution of these arithmetic operations is included in The definition of the operation code include the data type that is in processor registers during the execution these arithmetic operations. An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision or double-precision data.

Table 8.2: Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

The instruction "add with carry" performs the addition on two operands plus the value of the carry from the previous computation. Similarly, the "subtract with borrow" instruction subtracts two words and a borrow which may have resulted from a previous subtract operation. The negate instruction forms the 2's complement of a number, effectively reversing the sign of an integer when represented in the signed-2's complement form.

8.5 FLOATING POINT ARITHMETIC OPERATIONS

Let us give you a brief introduction to floating-point arithmetic and floating-point number representation.

A binary floating point number is represented in a normalized form, that is, the number is of the form $\pm 0.$ (significant starting with non-zero bit) $\times 2^{\pm(\text{ExponentValue})}$. Figure 8.14 shows a format of a 32 bit floating point number.

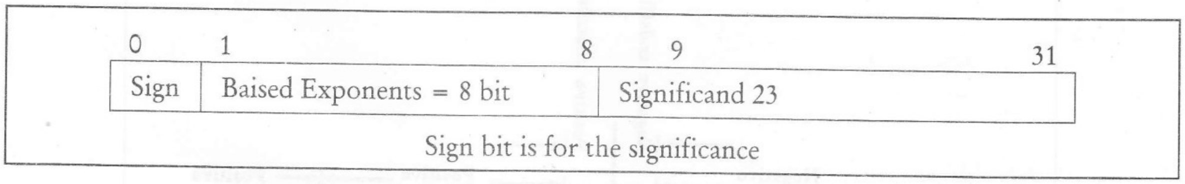


Figure 8.14: Floating Point Nunber Representation

The characteristics of a typical floating point representation of 32 bit in the above figure are:

- Left-most bit is the sign bit of the number
- Mantissa or significand should be in normalized form
- The base of the number is 2
- A value of 128 is added to the exponent. (why?) This is called a bias.

A normal exponent of 8 bit normally can represent exponent values as 0 to 255. However, as we are adding 128 in the biased exponent, thus, the actual exponent values represented will be – 128 to 127.

Now, let us define the range which a normalized mantissa can represent. As for a normalized mantissa the left most bit can not be zero, therefore, it has to be 1. Thus, it is not necessary to store this first bit and it is assumed implicitly for the number. Therefore, a 23 bit mantissa can represent 23 + 1 = 24 bit significand.

Minimum value of the significand:

The implicit first bit as 1 followed by 23 zero's.

0.10000 0000 0000 0000 0000 0000

Decimal equivalent = $1 \times 2^{-1} = 0.5$

Maximum value of the significand:

The implicit first bit 1 followed by 23 one's

0.1111 1111 1111 1111 1111 1111

Decimal equivalent:

Binary:	0.1111	1111	1111	1111	1111	1111		
+	0.0000	0000	0000	0000	0000	0001	=	2^{-24}
	1.0000	0000	0000	0000	0000	0000	=	1

So decimal equivalent of mantissa = $(1 - 2^{-24})$

Therefore, in normalized mantissa and biased exponent form, the format of Figure 8.14 can represent binary floating-point number in the range.

Lowest negative number: Maximum significand and Maximum exponent

$= -(1 - 2^{-24}) \times 2^{127}$

Highest negative number: Minimum significand and Minimum exponent

$= -0.5 \times 2^{-128}$

Lowest positive number : 0.5×10^{-128}

Highest positive number : $(1 - 2^{-24}) \times 10^{127}$

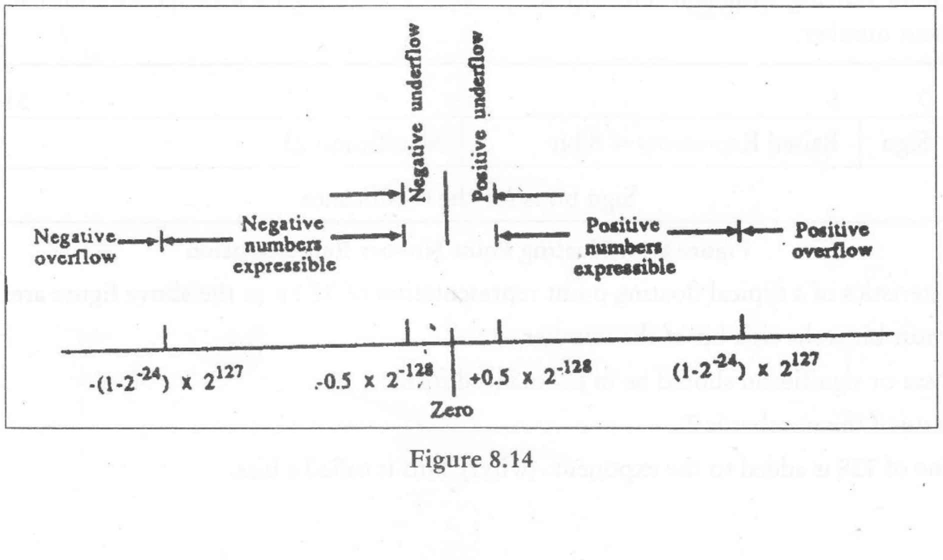


Figure 8.14

In floating point numbers, the basic trade off is between the range of the numbers and accuracy and precision of numbers. If in 32-bit format we increase the exponent bits, the range can be increased, however the accuracy of numbers will go down as significand will become smaller. Let us given an example, which will clarify the term precision. Suppose we have one bit binary significand then we can represent only 0.10 and 0.11 in a normalized form. The values such as 0.101, 0.1011 and so on cannot be represented as a complete numbers. Either they have to be approximated or truncated and will be represented as either 0.10 or 0.11. Thus, it will create an error. The higher the number of bits in significand better will be precision.

In floating point number for increasing both precision and range more number of bits are needed. This can be achieved by using double precision numbers. A double precision format is normally of 64 bits.

Institute of Electrical and Electronics Engineering (IEEE) a society that has created lot of standards regarding various aspects of computer have created IEEE standard 754 for floating-point representation and arithmetic. The basic objective of developing this standard was to facilitate the portability of programs from one to another computer. This standard has resulted in development of some standard numerical capabilities in various micro-processors. This representation is shown in Figure 8.15.

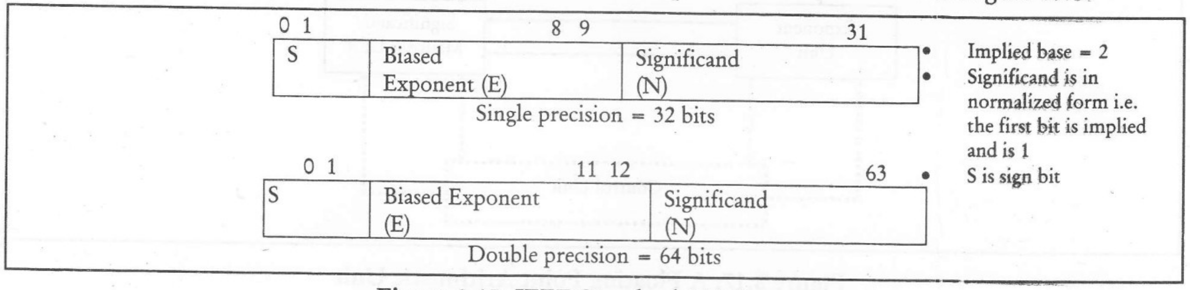

0 1
8 9
31
S
Biased Exponent (E)
Significand (N)
Single precision = 32 bits
0 1
11 12
63
S
Biased Exponent (E)
Significand (N)
Double precision = 64 bits
• Implied base = 2
• Significand is in normalized form i.e. the first bit is implied and is 1
• S is sign bit

Figure 8.15: IEEE Standard 754 Format

Figure 8.16 gives the floating point numbers specified by the IEEE standard 754.

Single Precision Number (32 bits)

Exponent (E)	Significand (N)	Value/Comments
255	Not equal to 0	do not represent a number
255	0	depending on sign bit
$0 < E < 255$	Any	$\pm(1.N)2^{E-127}$ For example, if S is zero that is positive number; N = 101 (rest 20 zeros) and E = 207, then the number is $= +(1.101)\,2^{207--127}$ $= +\mid 1.101 \text{ x} \times 2^{80}$
0		Not equal to 0 $\pm(0.N)2^{-126}$
0	0	0 depending on the sign bit

Double precision Numbers (64 bits)

Exponent (E)	Significand (N)	Value / Comments
2047	Not equal to 0	Do not represent a number
2047	0	$-$ or $+\infty$ depending on sign bit
$0 < E < 2047$	Any	
0	Not equal to 0	
0	0	

Figure 8.16: Values of Floating Point Numbers as per IEEE Standard 754

Please note that IEEE standard 754 specifies plus zero and minus zero and plus infinity and minus infinity. Floating point arithmetic is more sticky than fixed point arithmetic. For floating point addition and subtraction we have to:

- Check where a typical operand is zero
- Align the significant such that both the significands have same exponent
- Add or subtract the significand only and finally
- The significand is normalized again

8.5.1 Floating Point ALU

A floating point ALU is implemented using two loosely coupled fixed point arithmetic circuits. Figure 8.17 shows a simple structure of such a unit.

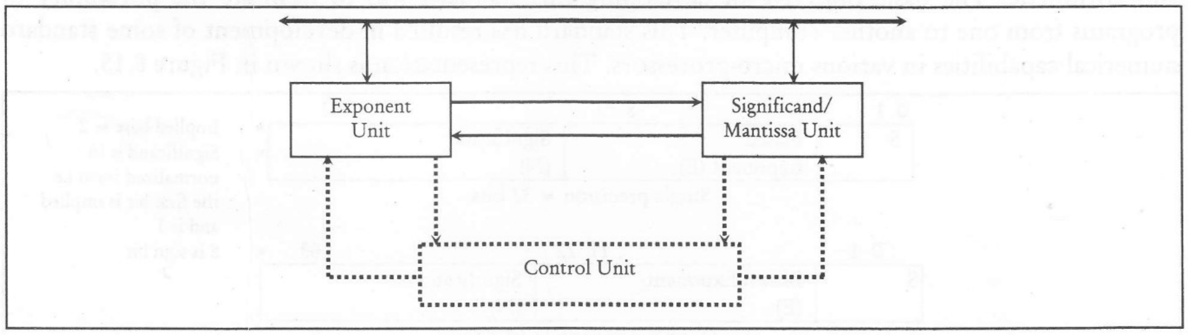


Figure 8.17: A Floating Point Arithmetic Unit

The two units can be termed as exponent unit and mantissa unit. The mantissa unit has to perform all the four arithmetic operations on the mantissa. Therefore, a general purpose fixed-point mantissa unit may be used for this purpose. However, for exponent unit we only need circuits to add, subtract and compare the exponents. Thus, a simple circuit containing these functions will be sufficient. The comparison can be performed by a comparator or by simple subtraction operation.

The implementation details or floating point arithmetic on floating point ALUs can be seen from the further readings.

8.6 DECIMAL ARITHMETIC OPERATIONS

Decimal numbers in BCD are stored in computer registers in groups of tour bits. Each 4-bit group stands for a decimal digit and must be taken as a unit when performing decimal micro-operations.

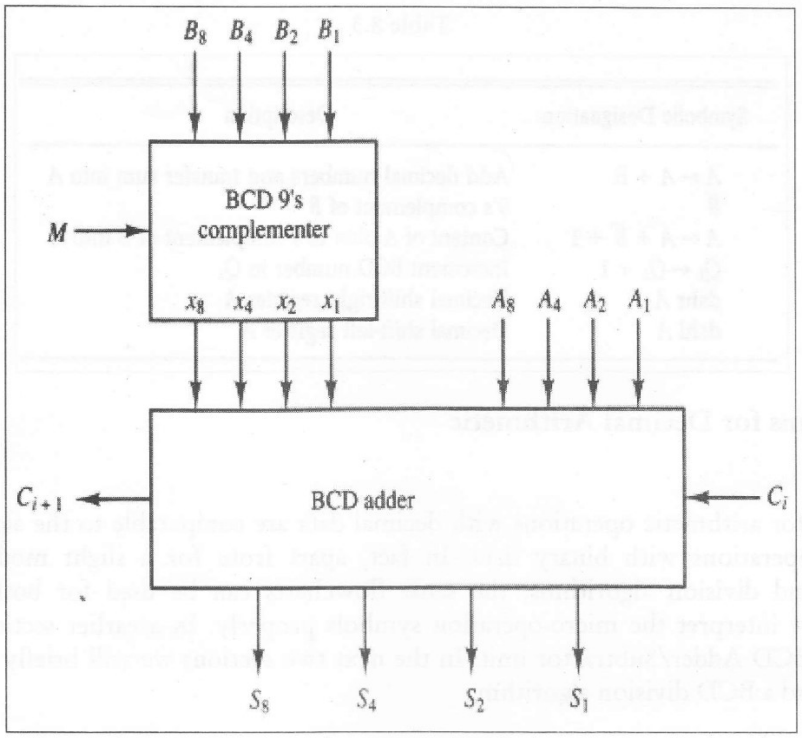


Figure 8.18

8.6.1 Decimal Arithmetic Micro-operations

For expediency, we will use the similar symbols for binary and decimal arithmetic microoperations but give them d different interpretation. As shown in Table 8.3, a bar over the register letter symbol denotes the 9's complement of the decimal number stored in the register Adding 1 to the 9's complement produces the 10's complement. Thus, for decimal numbers, the symbol A←A + B' - 1 denotes a relocate of the decimal sum formed by adding the original content A to the 10's complement of R. The use of identical symbols for the 9's complement and the 1's complement may be puzzling if both types of data are employed in the same system. If this is the case, it may be better to adopt a different symbol for the 9's complement. If only one type of data is being considered, the symbol would apply to the type of data used.

Incrementing or decrementing a register is the similar for binary and decimal except for the number or states that the register is allowed to have. A binary counter goes through 16 states, from 0000 to 1111, when incremented. A decimal counter goes through 10 states from 0000 to 1001 and back to 0000, since 9 is the last count. Similarly, a binary counter sequences from 1111 to 0000 when decremented. A decimal counter goes from 1001 to 0000. A decimal shift right or left is preceded by the letter d to indicate a shift over the four bits that hold the decimal digits.

Table 8.3

Symbolic Designation	Description
$A \leftarrow A + B$	Add decimal numbers and transfer sum into A
$\overline{B}$	9's complement of B
$A \leftarrow A + \overline{B} + 1$	Content of A plus 10's complement of B into A
$Q_L \leftarrow Q_L + 1$	Increment BCD number in Q_L
dshr A	Decimal shift-right register A
dshl A	Decimal shift-left register A

8.6.2 Algorithms for Decimal Arithmetic

Operations

The algorithms for arithmetic operations with decimal data are comparable to the algorithms for the corresponding operations with binary data. In fact, apart from for a slight modification in the multiplication and division algorithms, the same flowcharts can be used for both types of data provided that we interpret the micro-operation symbols properly. In a earlier section we described how to build a BCD Adder/Subtractor unit. In the next two sections we will briefly describe a BCD multiplication and a BCD division algorithm.

Multiplication

The multiplication of fixed-point decimal numbers is similar to binary apart from for the way the partial products are formed. A decimal multiplier has digits that range in value from 0 to 9, whereas a binary multiplier has only 0 and 1 digits. In the binary case, the multiplicand is additional to the partial product if the multiplier bit is 1. In the decimal case, the multiplicand must be multiplied by the digit multiplier and the result added to the partial product. This operation can be proficient by adding the multiplicand to the partial product a number of times equal to the value of the multiplier digit. The registers organization for the decimal multiplication is shown in Figure 8.19. We are assuming here four-digit numbers, with each digit occupying four bits, for a total of 16 bits for each number. There are three registers, A, B, and Q, each having a corresponding sign flip-flop A_s, B_s and Q_s.

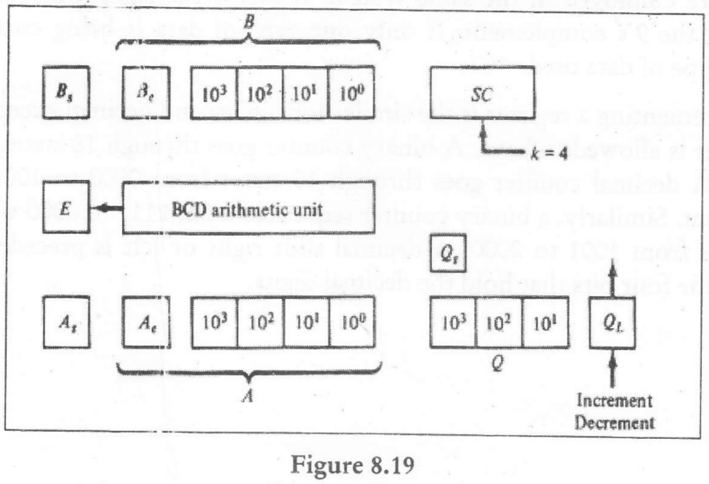


Figure 8.19

Registers A and B have four more bits chosen by A_e and B_e that provide an extension of one more digit to the registers. The BCD arithmetic unit adds the five digits in parallel and places the sum in the five-digit A register. The end-carry goes to flip-flop E. The function of digit A_e is to accommodate an overflow while adding the multiplicand to the partial product during multiplication. The purpose of digit B_e is to form the 9's complement of the divisor when subtracted from the partial remainder throughout the division operation. The least important digit in register Q is denoted by Q_L. This digit can be incremented or decremented. A decimal operand coming from memory consists of 17 bits. One bit (the sign) is transferred to B_s and the scale of the operand is placed in the lower 16 bits of B. Both B_e and A_e are cleared initially. The result of the operation is also 17 bits long and does not use the A_e part of the A register.

Division

Decimal division is comparable to binary division except of course that the quotient digits may have any of the 10 values from 0 to 9. In the reinstate division method, the divisor is subtracted from the dividend or partial remainder as many times as necessary until a negative remainder results. The correct remainder is then restored by adding the divisor. The digit in the quotient reflects the number of subtractions up to but excluding the one that caused the negative difference.

Check Your Progress

Fill in the blanks:

1. A typical CPU needs most of the .... and data processing hardware for implementing non-arithmetic functions.
2. ...., unlike peripheral processors, are tailor made for a particular family of CPUs.
3. A combinational circuit that performs the .... of two bits is called a half-adder.
4. A full–subtractor is a .... circuit that performs a subtration between two bits, taking into account that a 1 may have been borrowed by a lower significant stage.
5. The .... of fixed-point decimal numbers is similar to binary except for the way the partial products are formed.

8.7 LET US SUM UP

The very first question in this regard is: "What is an arithmetic processor?" and, "What is the need for arithmetic processors". A typical CPU needs most of the control and data processing hardware for implementing non-arithmetic functions. An example of one such arithmetic processor is the AMD 9511/12 one chip floating point processor. The advantage of this processor is that they can be utilized with any CPU, while the disadvantages are that they need explicitly programmed and slow communication links with the CPU. The name of the former stems from the fact that two half-adders can be employed to implement a full-adder. The two adder circuits are the first of the combinational circuits we shall design. A full adder is a combinational circuit that forms the arithmetic sum of three input bits it consists of three inputs and two outputs. Let us denote these input variables by x, y and z and output variables by S for sum and C for carry.

The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most of the computers carry instructions for all four operations. For computers which have only addition and possibly subtraction instructions, the other two operations i.e. multiplication and division must be generated by means of software subroutines.

8.8 KEYWORDS

Coprocessors: Unlike peripheral processors, are tailor made for a particular family of CPUs.

Peripheral Processor: If an arithmetic processor is treated as one of the I/O or peripheral unit then it is termed as peripheral processor.

Half-adder: This circuit needs two binary inputs and two binary outputs.

Full-adder: It is the combinational circuit that forms the arithmetic sum of three input bits it consists of three inputs and two outputs.

Floating point ALU: It is implemented using two loosely coupled fixed point arithmetic circuits.

Multiplexing: It means transmitting a large number of information units over a smaller number of channels or lines.

8.9 QUESTIONS FOR DISCUSSION

1. What is arithmetic processor design?
2. Differentiate between half adder and half subtractor.
3. Explain floating point ALU.
4. Explain the multiplication and division algorithm of decimal arithmetic operations.

Check Your Progress: Model Answers

1. Control
2. Coprocessors
3. Addition
4. Combinational
5. Multiplication

8.10 SUGGESTED READINGS

Sajjan G. Shiva; *Computer Design and Architecture*; Marcel Dekker

Silvia Melitta Mueller, Wolfgang J. Paul; *Computer Architecture*; Springer

Joseph D. Dumas II; *Computer Architecture*; CRC Press

Nicholas P. Carter; *Schaum's Outline of Computer Architecture*; Mc. Graw-Hill Professional

UNIT V

LESSON 9

PERIPHERAL DEVICES

CONTENTS

9.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Explain I/O Interface
- Describe asynchronous data transfer
- Discuss Direct Memory Access (DMA)
- Explain priority interrupts
- Define input/output processor

9.1 INTRODUCTION

A computer process data which is fed into it and gives results in the form you like. You can feed in this data for processing using a number of input mediums available and take the resultant output on various other medias available. In this lesson we will discuss the structure and working of peripheral devices.

9.2 PERIPHERAL DEVICES

A computer must have a system to get information from the outside world and must be able to communicate results to the external world. Programs and data should be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user. The most familiar method of entering information into a computer is using a typewriter like keyboard that allows a person to enter alphanumeric information directly. Every time a key is depressed, the terminal sends a binary coded character to the computer. When input information is transferred to the processor via a keyboard, the processor will be idle most of the time while waiting for the information to arrive. To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium. The information in the disk is then transferred into computer memory at a rapid rate. Results of programs are also transferred into a high-speed storage, which can be transferred later to output device for results.

Devices are said to be connected online that are under the direct control of the computer. These devices are designed to read information into or out of the memory unit when the CPU gives a command. Input or output devices connected to the computer are also called peripherals. Among the most common peripherals are keyboards, display units and printers. Peripherals that provide auxiliary storage for the system are magnetic disks.

Other input and output devices are digital incremental plotters, optical and magnetic character readers, analog-to-digital converters etc. Not all input comes from people, and not all output is intended for people. Computers are used to control various processes in real time, such as machine tooling, assembly line procedures, and chemical and industrial processes. For such applications, a method must be provided for sensing status conditions in the process and sending control signals to the process being controlled.

9.3 INPUT-OUTPUT INTERFACE

Input-output interface gives a method for transferring information between internal memory and I/O devices. Peripherals connected to a computer require special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.

The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore a conversion of signal values may be required
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU.

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other.

I/O Bus and Interface Modules and I/O versus Memory Bus have describe in lesson 2 section 2.7.

9.3.1 Isolated versus Memory-mapped I/O

One common bus may be employed to transfer information between memory or I/O and the CPU. Memory transfer and I/O transfer differs in that they use separate read and write lines. The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines. The I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

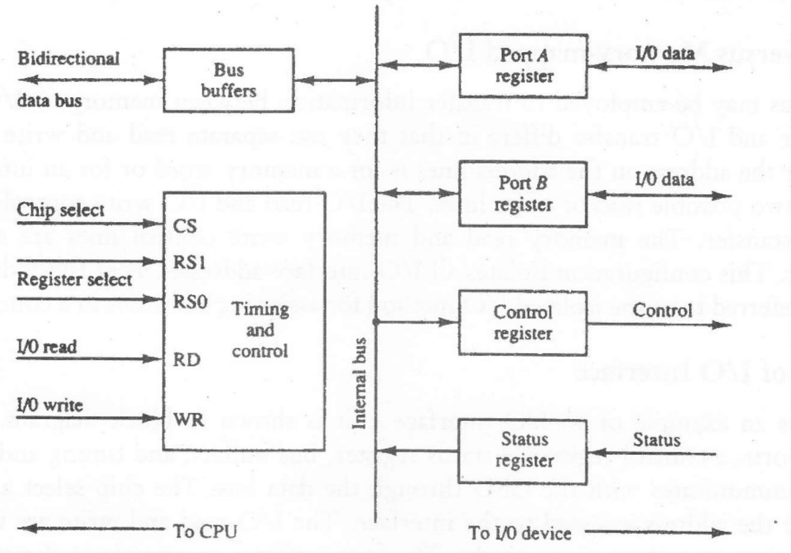
9.3.2 Example of I/O Interface

Figure 9.1 shows an example of an I/O interface unit is shown in block diagram. It has two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The input-output data to and from the device can be transferred into either port A or port B. The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data. A magnetic disk unit is used to transfer data in both directions but not at the same time, so the interface can use bi-directional lines. A command is passed to the I/O device by sending a word to the appropriate interface register. In a system like this, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data are transferred to and from ports A and B registers. Thus the transfer of data, control, or status information is determined from the particular interface register with which the CPU communicates.

The control register gets control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes. For example, port A may be defined as an input port and port B as an output port, A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction. The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer. For example, a status bit may indicate that port-A has received a new data item from the I/O device.

The interface registers uses bi-directional data bus to communicate with the CPU. The address bus selects the interface unit through the chip select and the two register select inputs. A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the Chip Select (CS) input to select the address bus. The two register select-inputs RSl and RSO are usually connected to the two least significant lines of the address bus. Those two inputs select on of the four registers in the interface as specified in the table accompanying the diagram. The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is ended. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

Figure 9.1: Example of I/O Interface Unit

9.4 DATA TRANSFER TECHNIQUES

9.4.1 Asynchronous Data Transfer

The internal operations in a digital system are synchronized using a common pulse generator. Clock pulses are used by all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse. Two units, such as a CPU and an I/O interface, are independent of each other. If the registers in the interface a common clock with the clock registers, the transfer between the two units is said to be synchronous. In most cases, the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. Hence, the two units are said to be asynchronous to each other. This approach is widely used in most computer systems.

Strobe Control

The strobe control method uses a single control line to transfer each time. We can activate the strobe by either the source or the destination unit. Figure 9.2(a) shows a source-initiated transfer.

The data bus is used t carry the binary information from source unit to the destination unit. Usually, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.

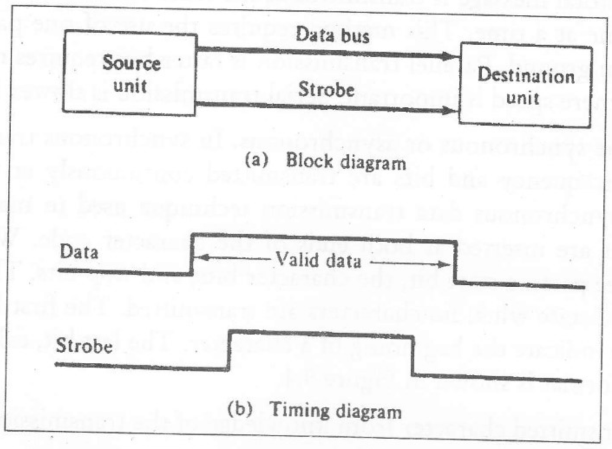


Figure 9.2: Source-initiated Strobe for Data Transfer

It is clear from Figure 9.2(b), that the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse. The information on the data bus and the strobe signal do not change in the active state for a sufficient time period to allow the destination unit to receive the data. Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers. The source removes the data from the bus a brief period after it disables its strobe pulse. Actually, the source does not have to change the information in the data bus. The fact that the strobe signal is disabled indicates that the data bus does not have valid data. New valid data will be available only after the strobe is enabled again.

Figure 9.3 describe a data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by putting the requested binary information on the data bus. The data should be valid and remain in the bus long enough for the destination unit to accept it. We can use the falling edge of the strobe pulse again to trigger a destination register. The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

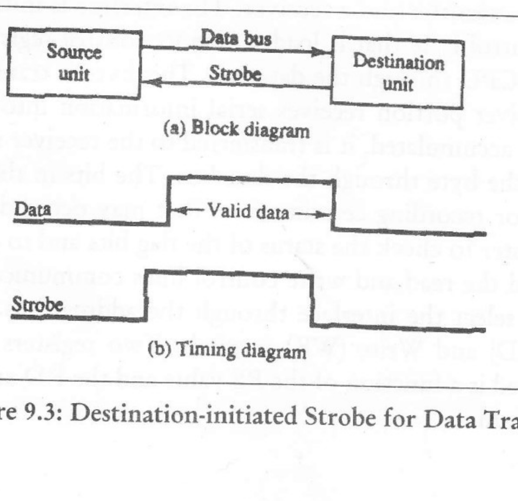


Figure 9.3: Destination-initiated Strobe for Data Transfer

9.4.2 Asynchronous Serial Transfer

We can transfer the data between two units in parallel or serial. In parallel data transmission, each bit has its own path and the total message is transmitted at the same time. In serial data transmission, each bit is sent in sequence one at a time. This method requires the use of one pair of conductors or one conductor and a common ground. Parallel transmission is faster but requires many wires. So we use it for short distances and where speed is important. Serial transmission is slower but is less expensive.

Serial transmission can be synchronous or asynchronous. In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. A serial asynchronous data transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code. With this technique, each character consists of three parts: a start bit, the character bits, and stop bits. The convention is that the transmitter rests at the 12-state when no characters are transmitted. The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit, called the stop bit is always a 1. An example of this format is shown in Figure 9.4.

Receiver can detect a transmitted character from knowledge of the transmission rules:

1. When a character is not being sent, the line is in the state 1.
2. The initiation of a character transmission is detected from the start bit, which is always 0.
3. The character bits always follow the start bit.
4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the I-state for at least one bit time.

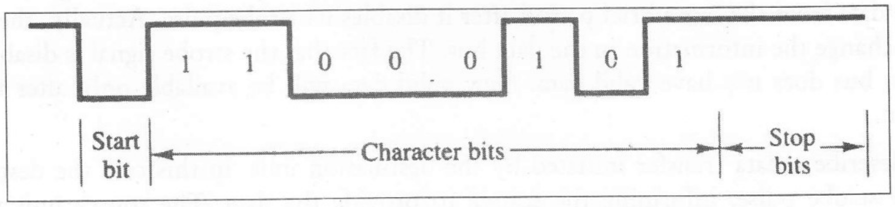
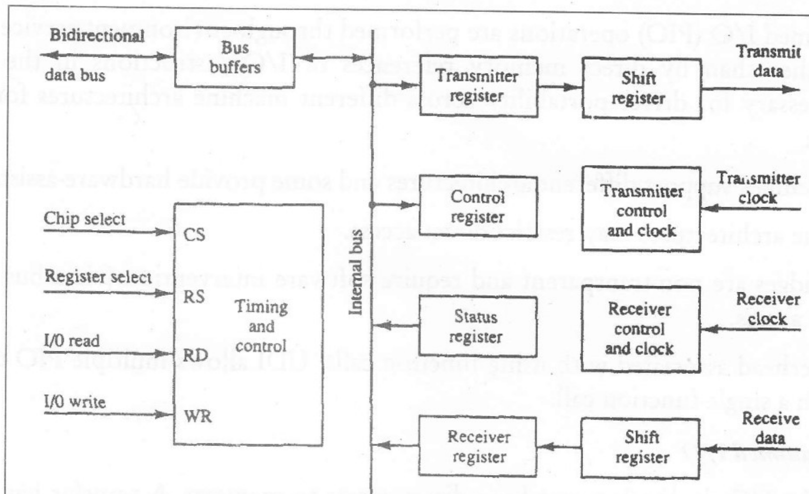


Figure 9.4: Asynchronous Serial Transmission

Asynchronous Communication Interface

Figure 9.5 shows the block diagram of an asynchronous communication interface is shown in Figure 9.5. It acts as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register. The transmitter register accepts a data byte from the CPU through the data bus. This byte is transferred to a shift register for serial transmission. The receiver portion receives serial information into an other shift register, and when a complete data byte is accumulated, it is transferred to the receiver register. The CPU can select the receiver register to read the byte through the data bus. The bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission. The CPU can read the status register to check the status of the flag bits and to determine if any errors have occurred. The chip select and the read and write control lines communicate with the CPU. The chip Select (CS) input is used to select the interface through the address bus. The Register Select (RS) is associated with the Read (RD) and Write (WR) controls. Two registers are write-only and two are read-only. The register selected is a function of the RS value and the RD and WR status, as listed in the table accompanying the diagram.



CS	RS	Operation	Register selected
0	×	×	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Figure 9.5: Block Diagram of a Typical Asynchronous Communication Interface

9.5 MODES OF TRANSFER

For later processing, binary information received from an external device is usually stored in memory. Information transferred from the central computer into an external device initiates in the memory unit. The CPU only executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit. Data transfer to and from peripherals may be done in either of three possible modes:

1. Programmed I/O
2. Interrupt-Initiated I/O
3. Direct Memory Access (DMA)

9.5.1 Programmed I/O (PIO)

Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device. This is contrasted with Direct Memory Access (DMA), which involves transfers initiated by a device to access system memory. On some hardware platforms PIO is handled via normal memory loads and stores ("memory-mapped I/O"); on others it requires special I/O instructions. UDI hides this difference from drivers.

In UDI, Programmed I/O (PIO) operations are performed through environment service calls coded as function calls rather than by direct memory references or I/O instructions in the drivers. This abstraction is necessary for driver portability across different machine architectures for (at least) the following reasons:

1. Different machines support different architectures and some provide hardware-assisted swapping.
2. Some machine architectures may restrict direct access.
3. Some bus bridges are non-transparent and require software intervention (via a bus bridge driver) for each PIO access.

To reduce the overhead associated with using function calls, UDI allows multiple PIO transactions to be performed with a single function call.

Example of Programmed I/O

In this method, the I/O device does not have direct access to memory. A transfer has I/O device to memory needs the execution of several instruction, input CPU, including an input instruction to transfer the data from the device CPU and a store instruction to transfer the data from the CPU to memory instruction may be needed to verify that the data are available from the source and to count the numbers of words transferred.

Figure 9.6 shows data transfer from an I/O device through an interface into the CPU. The device transfers bytes of data one at a time, as they are available. When a byte of data is available, the device places it in the I/O bus and enables its data valid line. The interface accepts the byte into its data register and enables the data accepted line. The interface sets a bit in the status register that we will refer to as a "flag" bit. The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled.

A program is written to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device. This can be done by reading the status register into a CPU register and checking the value of the flag bit. If the flag is equal to 1, the CPU reads the data from the data register. The flag-bit is then cleared to 0 by either the CPU or by the interface, depending on how the interface circuits are designed. Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte. A flowchart of the program is shown in Figure 9.7. It is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each byte needs three instructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step I if not set or to step if set.
3. Read the data register.

A CPU register read each byte and then transferred to memory with a store instruction. I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

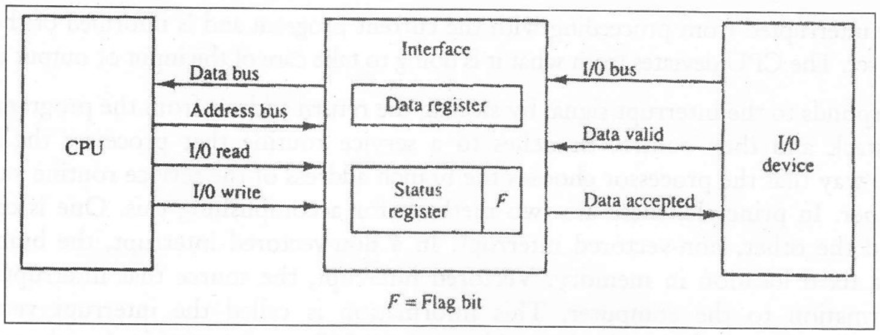


Figure 9.6: Data Transfer from I/O Device to CPU

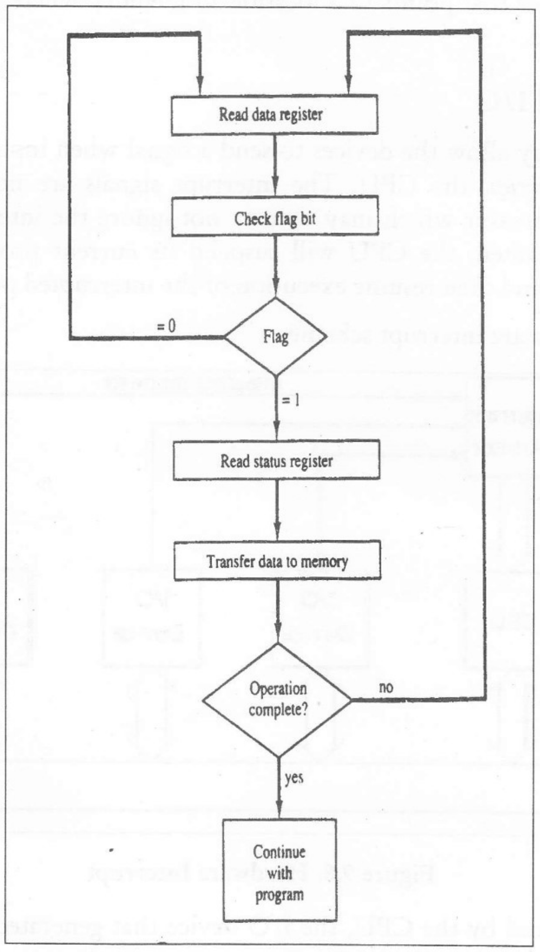


Figure 9.7: Flowchart for CPU Program to Input Data

This method is used in small low-speed computers or in systems that are dedicated to monitor a device continuously. The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient. Another way of constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer.

The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from one unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, non-vectored interrupt. In a non-vectored interrupt, the branch address is assigned to a fixed location in memory. Vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector. In some computers the interrupt vector is the first address of the I/O service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

9.5.2 Interrupt-Initiated I/O

In this scheme the CPU may allow the devices to send a signal when input is waiting to be processed. The signal is used to *interrupt* the CPU. The interrupt signals are normally sent directly (using hardware) to the micro-processor which may or may not ignore the interrupt request. In most cases the interrupt request is granted, the CPU will suspend its current program execution, execute an *interrupt handler* program, and then resume execution of the interrupted program.

Figure below shows a hardware interrupt scheme.

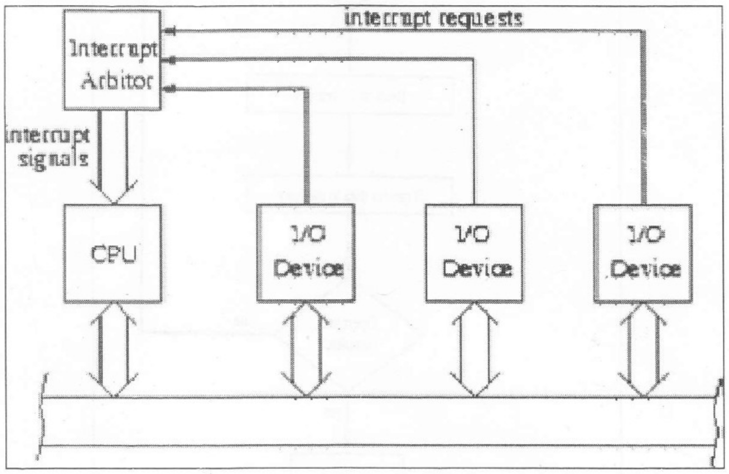


Figure 9.8: Hardware Interrupt

When an interrupt is serviced by the CPU, the I/O device that generated the interrupt (or in general the set of devices) is serviced.

I/O Interrupts

A priority interrupt establishes a priority to decide which condition is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher-priority interrupt levels are assigned to requests, which if delayed or interrupted, could have serious consequences. Devices with high-speed transfers are given high priority, and slow devices receive low priority. When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first. Establishing the priority of simultaneous interrupts can be done by software or hardware. We can use a polling procedure to identify the highest-priority. There is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt. We test the highest-priority source first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on. Thus the initial service routine interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines. The particular service routine reached belongs to the highest-priority device among all devices that interrupted the computer.

Daisy-Chaining Priority

The daisy-chaining method has a serial connection of all devices that request an interrupt. The device with the highest priority is kept in the first position, followed by lower-priority devices and so on. This method of connection is shown in Figure 9.9. The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU. When no interrupts are pending, the interrupt line stays in the high-level state and as a result CPU does not recognize any interrupt. This is equivalent to a negative logic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device I has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.

A device with PI=0 input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked. A device that makes a request for an interrupt and has a I in its Pi input will intercept the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus. The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.

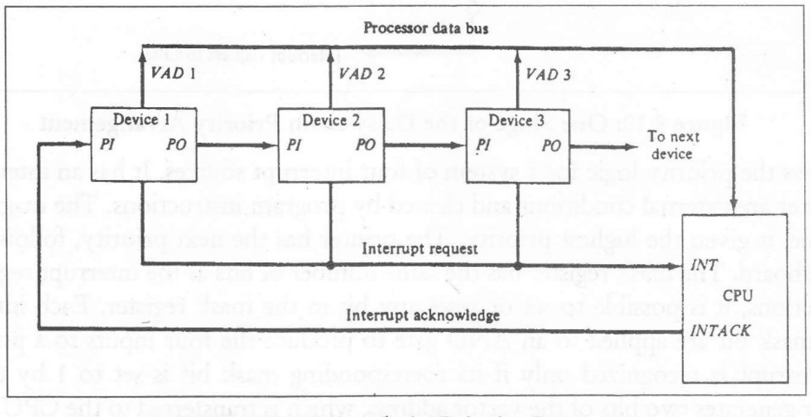
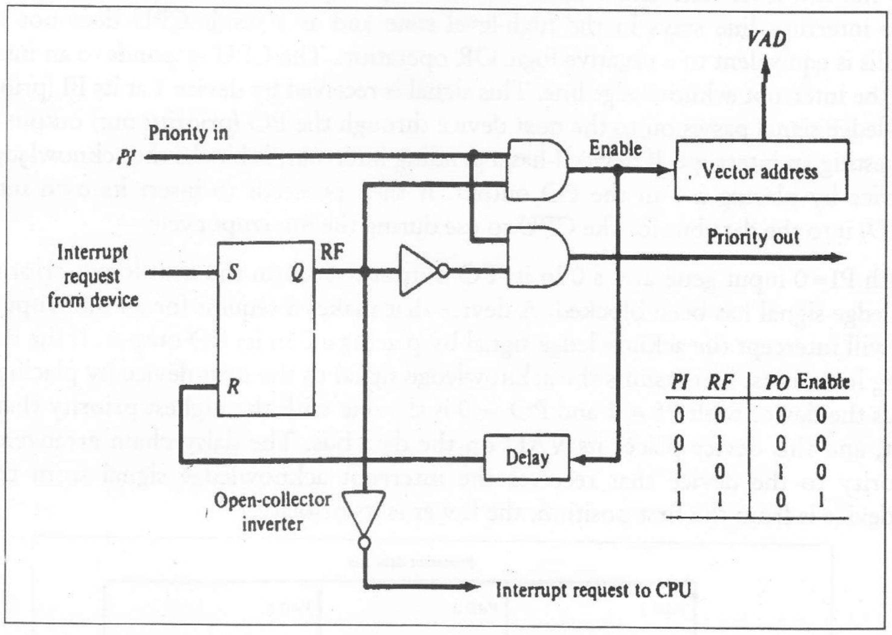


Figure 9.9: Daisy-chain Priority Interrupt

Figure 9.10 shows the internal logic that must be included within each device when connected in the daisy-chaining scheme. The device sets its RF flip-flop when it wants to interrupt the CPU. The output of the RF flip-flop goes through an open-collector inverter (a circuit that provides the wired logic for the common interrupt line). If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF. If PI = 1 and RF= 0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO. The device is active when PI = 1 and RF = 1. This condition places a 0 in PO and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

Priority Interrupt

The method uses a register whose bits are set separately by the interrupt signal from each device. Now we establish the priority according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request. The mask register can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the while a lower-priority device is being serviced.



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Figure 9.10: One Stage of the Daisy-chain Priority Arrangement

Figure 9.10 shows the priority logic for a system of four interrupt sources. It has an interrupt register. The bits of this register are external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.

Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system. The outputs of 1ST AND with IEN provide a common interrupt signal for the CPU. The interrupt acknowledge INTACK signal form the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus. We will now explain the priority encoder circuit and then discuss the interaction between the priority interrupt controller and the CPU.

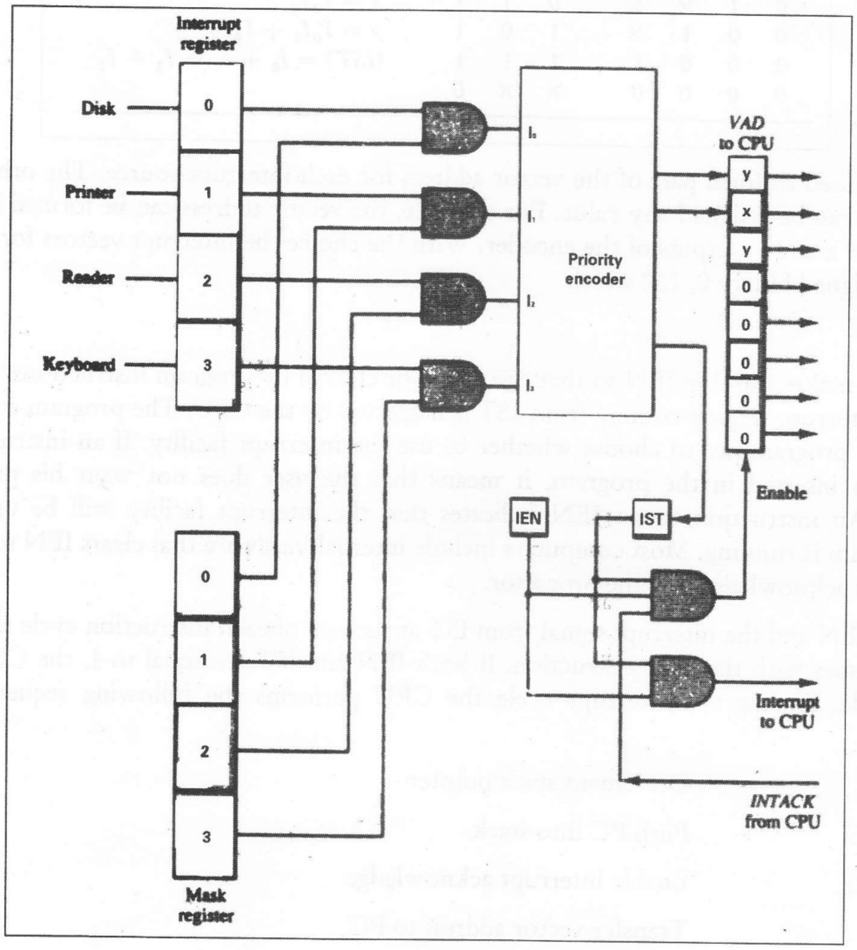


Figure 9.11: Priority Interrupt Hardware

Priority Encoder

This circuit implements the priority function. The logic is that if two or more inputs arrive at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in the Table 9.1. The x's in the table designate don't-care conditions. Input I_0 has the highest priority.

Table 9.1: Priority Encoder Truth Table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I'_0 I'_1$
0	0	1	×	1	0	1	$y = I'_0 I_1 + I'_0 I'_2$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

The output is used to form part of the vector address for each interrupt source. The other bits of the vector address can be assigned any value. For example, the vector address can be formed by appending six zeros to the x and y outputs of the encoder. With the choice the interrupt vectors for the four I/O devices are assigned binary 0, 1, 2 and 3.

Interrupt Cycle

The interrupt makes flip-flop IEN so that can be set or cleared by program instructions. When IEN is cleared, the interrupt request coming from 1ST is neglected by the CPU. The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted. An instruction to set IEN indicates that the interrupt facility will be used while the current program is running. Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor.

CPU checks IEN and the interrupt signal from IST at the end of each instruction cycle the. If either 0, control continues with the next instruction. If both IEN and IST are equal to 1, the CPU goes to an interrupt cycle. During the interrupt cycle the CPU performs the following sequence of micro-operations:

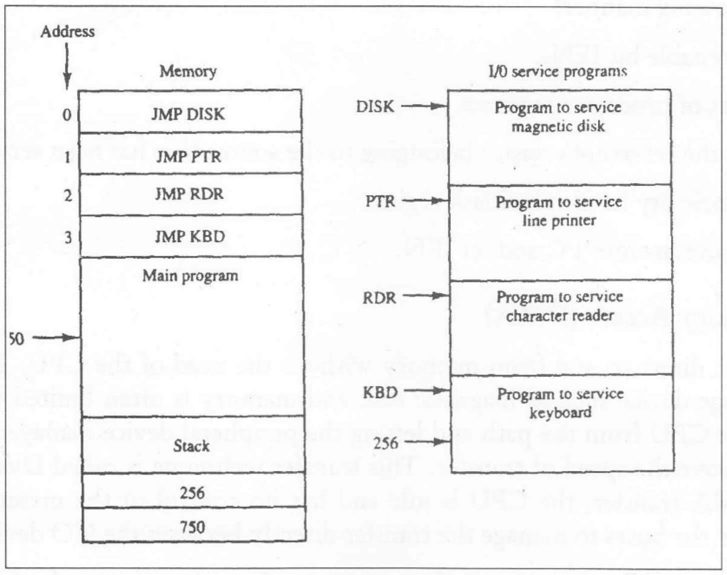
$SP \leftarrow SP - 1$	Decrement stack pointer
$M[SP] \leftarrow PC$	Push PC into stack
$INTACK \leftarrow 1$	Enable interrupt acknowledge
$PC \leftarrow VAD$	Transfer vector address to PC
$IEN \leftarrow 0$	Disable further interrupts

Go to fetch next instruction

The return address is pushed from PC into the stack. It then acknowledges the interrupt by enabling the INTACK line. The priority interrupt unit responds by placing a unique interrupt vector into the CPU data bus. The CPU transfers the vector address into PC and clears IEN prior to going to the next fetch phase. The instruction read from memory during the next fetch phase will be the one located at the vector address.

Software Routines

A priority interrupt system uses both hardware and software techniques. Now we discuss the software routines for this. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers. Figure 9.12 shows the programs that must reside in memory for handling the interrupt system. Each device has its own service program that can be read through a jump (JMP) instruction stored at the assigned vector address. The symbolic name of each routine represents the starting address of the service program. The stack shown in the diagram is used for storing the return address after each interruption.



Address
Memory
0
JMP DISK
1
JMP PTR
2
JMP RDR
3
JMP KBD
Main program
50
Stack
256
750
I/0 service programs
DISK
Program to service magnetic disk
PTR
Program to service line printer
RDR
Program to service character reader
KBD
Program to service keyboard
256

Figure 9.12: Programs Stored in Memory for Servicing Interrupts

Now we take an example to illustrate it. Let the keyboard sets interrupt bit while the CPU is executing the instruction in location 749 of main program. At the end of the instruction cycle, the computer goes to interrupt cycle. It stores the return address 750 in the stack and then accepts the vector address 00000011 from the bus and transfers it to PC. The instruction in location 3 is executed next, resulting in transfer of control to the KBD routine. Now suppose that the disk sets its interrupt bit when the CPU is executing the instruction at address 255 in the KBD program. Address 256 is pushed into the stack and control is transferred to the DISK service program. The last instruction in each routine is a return from interrupt instruction. When the disk service program is completed, the return instruction pops the stack and places 256 into PC. This returns control to the KBD routine to continue servicing the keyboard. At the end of the KBD program, the last instruction pops the stack and returns control to the main program at address 750. Thus, a higher-prior device can interrupt a lower-priority device. It is assumed that the time spent in servicing the high-priority interrupt is short compared to the transfer rate of the low-priority device so that no loss of information takes place.

Initial and Final Operations

We should remember that the interrupt enable IEN is cleared at the end of an interrupt cycle. This flip-flop must be set again to enable higher-priority interrupt requests, but not before lower-priority interrupts are disabled. The initial sequence of each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear lower-level mask register bits.
2. Clear interrupt status bit 1ST.
3. Save contents of processor registers.
4. Set interrupt enable bit IEN.
5. Proceed with service routine.

The final sequence in each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

1. Clear interrupt enable bit IEN.
2. Restore contents of processor registers.
3. Clear the bit in the interrupt register belonging to the source that has been serviced.
4. Set lower-level priority bits in the mask register.
5. Restore return address into PC and set IEN.

9.5.3 Direct Memory Access (DMA)

We can transfer data direct to and from memory without the need of the CPU. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manager the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals. Figure 9.13 shows two control signals in the CPU that facilitate the DMA transfer. The Bus-Request-input (BR) is used by the DMA controller to request the CPU to relinquish control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance. The CPU activates the Bus-Grant-output (BG) to inform the external DMA that the buses are in the high-impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminate the transfer, it disables the bus request line. The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.

DMA communicates directly with the memory, when it takes control of the bus system. We can transfer in several ways. In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. This mode of transfer is needed for fast devices such as magnetic disks where data transmission cannot be stopped or slowed down until an entire block is transferred. An alternative technique called cycle stealing allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

DMA Controller

The DMA controller requires the usual circuits of an interface to communicate with the CPU and an I/O device. It also needs an address register, and count register, and a set of address lines. The address register and address line are used for direct communication with the memory. The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of the DMA.

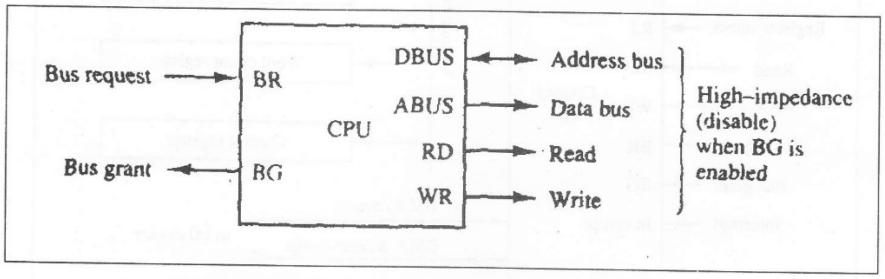


Figure 9.13: CPU Bus Signals for DMA Transfer

Figure 9.13 shows a DMA controller. The unit can communicate with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs. The RD (read) and WR (write) inputs are bi-directional. When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG = 1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control. The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.

The DMA controller consists of three registers namely an address register, a word count register, and a control register. The address register contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory. The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero. The control register specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers. Thus the CPU can read from or write into the DMA registers under program control via the data bus.

CPU first initializes the DMA. Then the DMA starts and continues to transfer data between memory and peripheral unit until an ending block is transferred. The initialization process is essentially a program consisting of I/O instructions that include the address for selecting particular DMA registers. The CPU initializes the DMA by sending the following information through the data bus:

1. The starting address of the memory block where data are available (for read) or where data are to be stored (for write)
2. The word count, which is the number of words in the memory block
3. Control to specify the mode of transfer such as read or write
4. A control to start the DMA transfer

The starting address is stored in the address register. The word count is stored in the word count register, and the control information in the control register

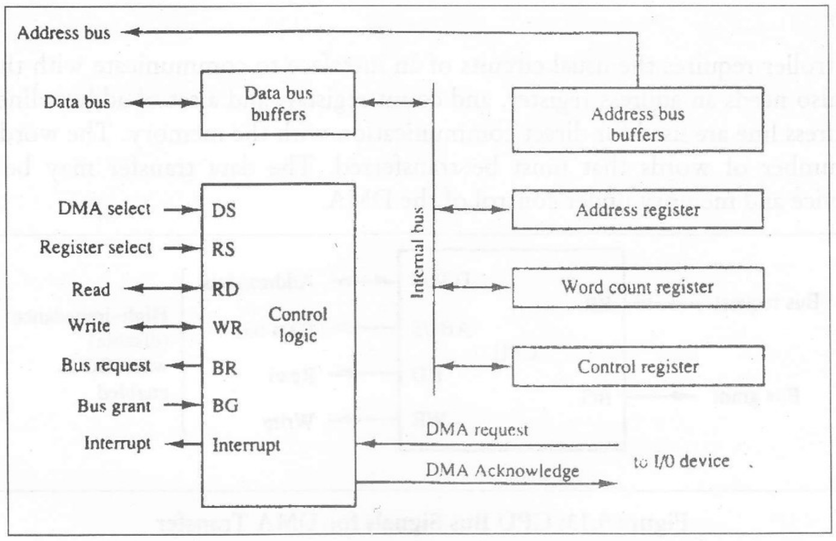


Figure 9.14: Block Diagram of DMA Controller

DMA Transfer

Figure 9.15 shows the position of the DMA controller among the other components in a computer system. The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

The DMA controller activates the BR line, when the peripheral device sends a DMA request, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing the DMA that its buses are disabled. The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device. Note that the RD and WR lines in the DMA controller are bi-directional. The direction of transfer depends on the status of the BG line. When BG =0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG = 1, the RD and WR are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device gets a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.

The DMA increments its address-register and decrements its word count-register, for each word that is transferred. If the word count does not reach zero, the DMA checks the request line coming from the peripheral. For high-speed device, the line will be active as soon as the previous transfer completed. A second transfer is then initiated, and the process continues until the entire block is transferred. If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disables the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

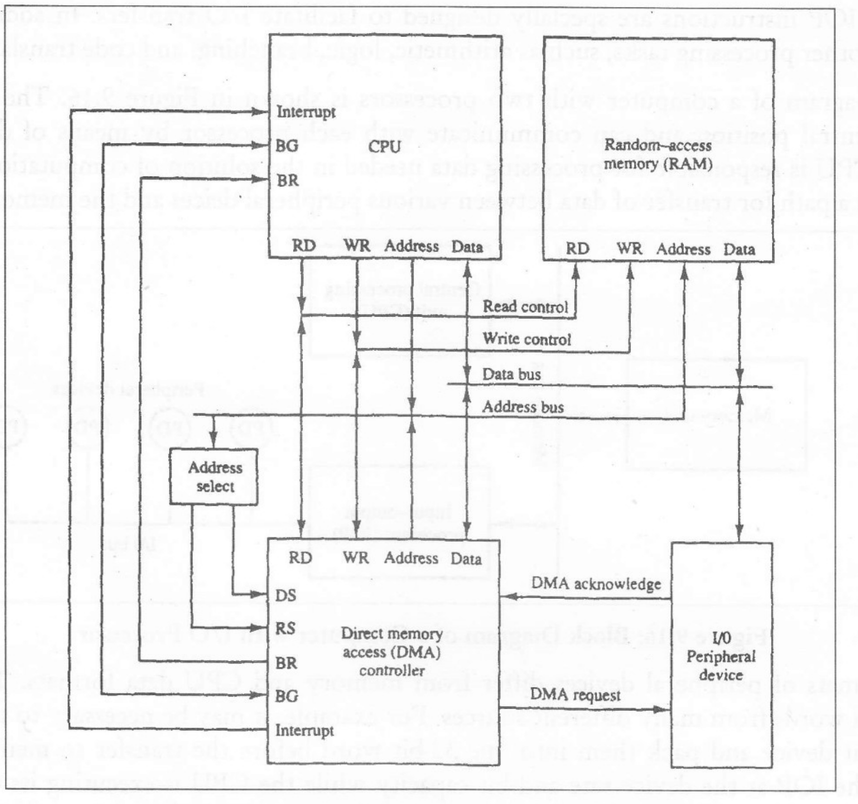


Figure 9.15: DMA Transfer in a Computer System

If the word count register reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt. When the CPU responds to the interrupt, it reads the content of the word count register. The zero value of this register indicates that all the words were transferred successfully. The CPU can read this register at any time to check the number of words already transferred.

DMA transfer is very useful. It is used for fast transfer of information between magnetic disks and memory. It is also useful for updating the display of the terminal is kept in memory, which can be updated under program control.

9.6 INPUT-OUTPUT PROCESSOR (IOP)

A computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices. An Input-Output Processor (IOP) may be classified as a processor with direct memory access capability that communicates with I/O devices. In this configuration, the computer system can be divided into a memory unit, and a number of processors comprised of the CPU and one or more IOPs. Each IOP takes care of input and output tasks, relieving the CPU from the housekeeping chores involved in I/O transfers.

The IOP is similar to a CPU except that it is designed to handle the details of I/O processing. Unlike the DMA controller that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions. IOP instructions are specially designed to facilitate I/O transfers. In addition, the IOP can perform other processing tasks, such as arithmetic, logic, branching, and code translation.

The block diagram of a computer with two processors is shown in Figure 9.16. The memory unit occupies a central position and can communicate with each processor by means of direct memory access. The CPU is responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transfer of data between various peripheral deices and the memory unit.

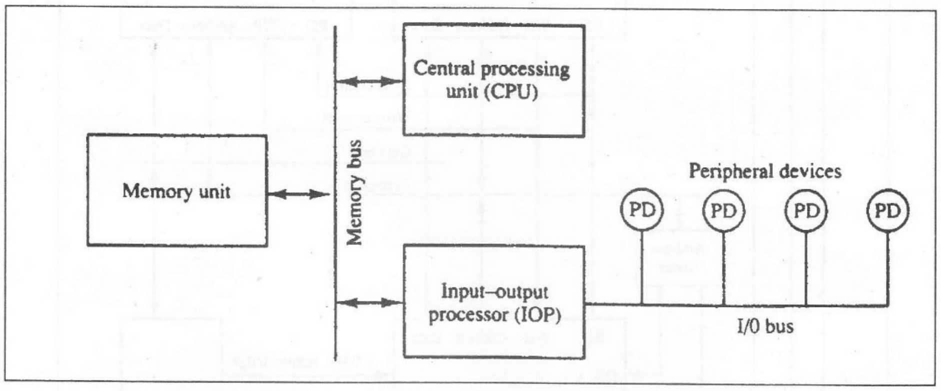


Figure 9.16: Block Diagram of a Computer with I/O Processor

The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources. For example, it may be necessary to take four bytes from an input device and pack them into one 32-bit word before the transfer to memory. Data are gathered in the IOP at the device rate and bit capacity while the CPU is executing its own program. After the input data are assembled into a memory word, they are transferred from IOP directly into memory by "stealing" one memory cycle from the CPU. Similarly, an output word transferred from memory to the IOP is directed from the IOP to the output device at the device rate and bit capacity.

The communication between the IOP and the devices attached to it is similar to the program control method of transfer. The way by which the CPU and IOP communicate depends on the level of sophistication included in the system. In most computer systems, the CPU is the master while the IOP is a slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are execute in the IOP. CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities. The IOP, in turn, typically asks for CPU attention by means of an interrupt. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program. When an I/O operation is desired, the CPU informs the IOP where to find the I/O program and then leaves the transfer details to the IOP.

9.6.1 CPU-IOP Communication

There are many form of the communication between CPU and IOP. These are depending on the particular computer considered. In most cases the memory unit acts as a message center where each processor leaves information for the other. To appreciate the operation of a typical IOP, we will illustrate by a specific example the method by which the CPU and IOP communicate. This is a simplified example that omits many operating details in order to provide an overview of basic concepts.

The sequence of operations may be carried out as shown in the flowchart of Figure 9.17. The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer. The CPU refers to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer. The memory address received with this instruction tells the IOP where to find its program.

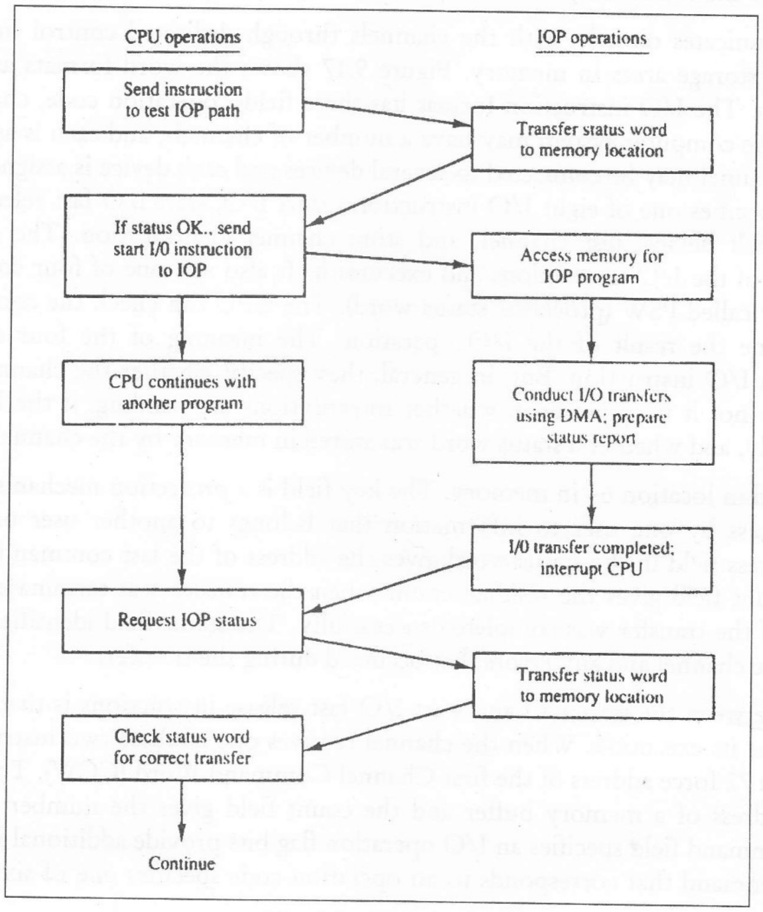

CPU operations
IOP operations
Send instruction to test IOP path
Transfer status word to memory location
If status OK., send start I/0 instruction to IOP
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; prepare status report
I/0 transfer completed; interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer
Continue

Figure 9.17: CPU-IOP Communication

The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer. When the IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The IOP responds by placing the contents of its status report into a specified memory location.

The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program. The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

Channel

In the IBM 370, the I/O processor computer is known as a channel. A typical computer system configuration includes a number of channels with each channel attached to one or more I/O devices. There are three types of channels: multiplexer, selector, and block-multiplexer. The multiplexer channel can be connected to a number of slow- and medium-speed devices and is capable of operating with a number of I/O devices simultaneously. The selector channel is designed to handle one I/O operation at a time and is normally used to control one high-speed device.

The CPU communicates directly with the channels through dedicated control lines and indirectly through reserved storage areas in memory. Figure 9.17 shows the word formats associated with the channel operation. The I/O instruction format has three fields: operation code, channel address, and device address. The computer system may have a number of channels, and each is assigned an address. Similarly, each channel may be connected to several devices and each device is assigned an address. The operation code specifies one of eight I/O instructions: start I/O, start I/O fast release, test I/O, clear I/O, halt I/O, halt device, test channel, and store channel identification. The addressed channel responds to each of the I/O instructions and executes it. It also sets one of four condition codes in a processor register called PSW (processor status word). The CPU can check the condition code in the PSW to determine the result of the I/O operation. The meaning of the four condition codes is different for each I/O instruction. But, in general, they specify whether the channel or the device is busy, whether or not it is operational, whether interruptions are pending, if the I/O operation had started successfully, and whether a status word was stored in memory by the channel.

It is always stored in location 64 in memory. The key field is a protection mechanism used to prevent unauthorized access by one user to information that belongs to another user or to the operating system. The address field in the status word gives the address of the last command word used by the channel. The count field gives the residual count when the transfer was terminated. The count field will show zero if the transfer was completed successfully. The status field identifies the conditions in the device and the channel and any errors that occurred during the transfer.

The difference between the start I/O and start I/O fast release instructions is that the latter requires less CPU time for its execution. When the channel receives one of these two instructions, it refers to memory location 72 force address of the first Channel Command Word (CCW). The data address field specifies first address of a memory buffer and the count field gives the number of involved in the transfer. The command field specifies an I/O operation flag bits provide additional information for the channel. The command that corresponds to an operation code specifies one of six basic types of I/O operations:

1. ***Write:*** Transfer data from memory to 10 devices.
2. ***Read:*** Transfer data from I/O device to memory.
3. ***Read backwards:*** Read magnetic tape with tape moving backward
4. ***Control:*** Used to initiate an operation not involving transfer of data, such as rewinding of tape or positioning a disk-access mechanism.
5. ***Sense:*** Informs the channel to transfer its channel status were memory location 64.
6. ***Transfer in channel:*** Used instead of a jump instruction. Here a word in missing address field specifies the address of the next command word to be executed by the channel.

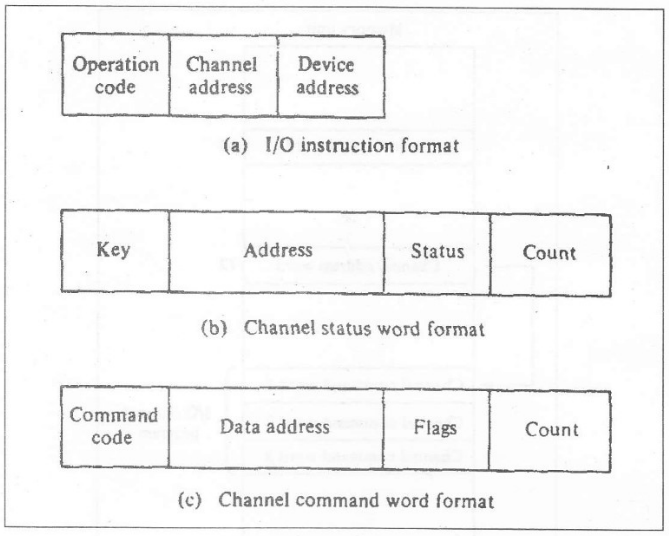


Figure 9.18: IBM 370 I/O Related Word Formats

An example of a channel program is shown in Table 9.2. It consists of three command words. The first causes a byte transfer into a magnetic tape from memory starting at address 4000. The next two command-words perform a similar function with a different portion of memory and byte count. The six flags in each control word specify certain interrelations between count. The first flag is set to 1 in the first command word to specify "data chaining." It results in combining the 60 bytes from the first command word with the word with the 20 bytes of its successor into one record of 80 bytes. The 80 bytes are written on tape without any separation or gaps ever though two memory sections were used. The second flag is set to 1 in the second command word to specify "command chaining." It informs the channel that the next command word will use the same I/O device, in this case, the tape, the channel informs the tape unit to start inserting a record gap on the tape and proceeds to read the next command word from memory. The 40 bytes at the third command word are then written on tape as a separate record. When all the flags are equal to zero, it signifies the end of I/O operations for the particular I/O device.

Table 9.2: BM-370 Channel Program Example

Command	Address	Flags	Count
Write tape	4000	100000	60
Write tape	6000	010000	20
Write tape	3000	000000	40

A memory map showing all pertinent information for I/O processing is illustrated in Figure 9.18. The operation begins when the CPU program encounters a start I/O instruction. The IOP then goes to memory location 72 to obtain a channel address word. This word contains the starting address of the I/O channel program. The channel then proceeds to execute the program specified by the channel command words. The channel constructs a status word during the transfer and stores it in location 64. Upon interruption, the CPU can refer the memory location 64 for the status word.

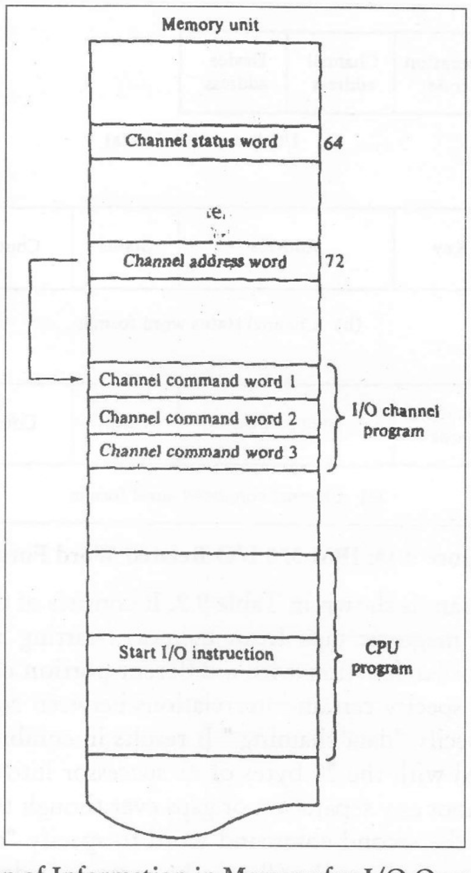


Figure 9.19: Location of Information in Memory for I/O Operation in the IBM 70

Check Your Progress

State whether the following statements are true or false:

1. Input-output interface gives a method for transferring information between internal memory and I/O devices.
2. The I/O bus is made of data lines, address lines and control lines.
3. To communicate with I/O, the processor does not have to communicate with the memory unit.
4. During DMA transfer, the CPU is idle and has no control of the memory buses.

9.7 LET US SUM UP

Programs and data should be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user. I/O devices are designed to read information into or out of the memory unit when the CPU gives a command. I/O interface gives a method for transferring information between internal memory and I/O devices.

The interpretation of the command depends on the peripheral that the processor is addressing. The four types of commands that an inface may receive include control command, status command, data output command and data input command. Various data transfer techniques include Asynchronous data transfer, where the internal timing in each unit is independent from the other, strobe control which used a single control line to transfer each time, handshaking method which used control line in the same direction as that of data flow. Two separate lines are used for different directions.

An input-output processor has direct memory access capability that communicates with I/O devices. The communication between the IOP and the devices attached to it is similar to the program control method of transfer. The priority encoder circuit implements the priority function. The I/O processor computer is also known as a channel in the IBM 370. Each channel is attached to one or more I/O devices. The three types of channels include multiplexer, selector and block multiplexer.

9.8 KEYWORDS

Input Devices: Computer peripherals used to enter data into the computer.

Output Devices: Computer peripherals used do get output from the computer.

Bus Interface: communication link between the processor and several peripherals.

Interrupt: An instruction or command which halts the execution of a program.

Direct Memory Access (DMA): Transferring Data directly to an from memory without the need of CPU.

Input-Output Processor (IOP): An external processor that communicates directly with all I/O devices and has direct memory access capabilities.

Channel: The computer with an I/O processor.

9.9 QUESTIONS FOR DISCUSSION

1. What is the difference between isolated I/O and memory mapped I/O? What are the advantages and disadvantages of each?
2. Give at least six status conditions for the setting of individual bits in the status register of an asynchronous communication interface. Give an example of I/O interface unit.
3. What programming steps are required to check when a source interrupts the computer while it is still being serviced by a previous interrupt request from the same source?
4. Why does DMA have priority over the CPU when both request a memory transfer?
5. What is the advantage of using interrupt - initiated data transfer over transfer under program control without an interrupt?
6. What are the different modes of data transfer to and from peripherals?
7. Discuss the need of Direct Memory Access.

Check Your Progress: Model Answers

1. True	2. True	3. False	4. True

9.10 SUGGESTED READINGS

John R. Borer, *Microprocessors in process control*, Springer

Sajjan G. Shiva; *Computer Design and Architecture*; Marcel Dekker

Silvia Melitta Mueller, Wolfgang J. Paul; *Computer Architecture*; Springer