UNIT-IV

COMPUTER ARITHMETIC

Introduction:

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we give algorithms.

In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.

And these instructions perform a great activity in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

A processor has an arithmetic processor(as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement. Most computers use the signed magnitude representation for the mantissa.

Addition and Subtraction :

Addition and Subtraction with Signed –Magnitude Data

We designate the magnitude of the two numbers by A and B. Where the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of Table 4.1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to present a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words parentheses should be used for the subtraction algorithm)

UNIT-IV

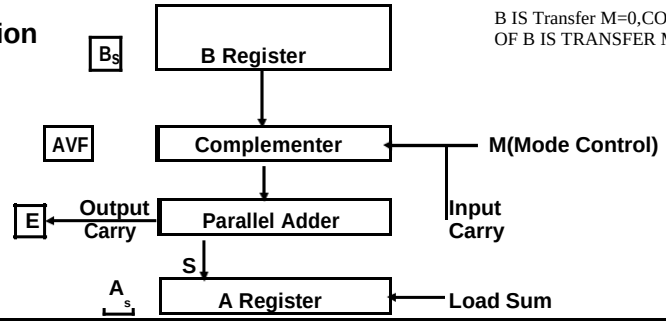
addition and Subtraction of Signed-Magnitude Numbers

SIGNED MAGNITUDEADDITION AND SUBTRACTION

Addition: A + B , A: Augend; B: Addend
Subtraction: A - B: A: Minuend; B: Subtrahend

Operation	Add Magnitude	Subtract Magnitude		
		When A>B	When A<B	When A=B
(+A) + (+B)	+(A + B)			
(+A) + (- B)		+(A - B)	-(B-A)	+(A - B)
(- A) + (+B)		-(A-B)	+(B - A)	+(A - B)
(-A) + (-B)	-(A+B)			
(+A) - (+B)		+(A - B)	-(B-A)	+(A - B)
(+A) - (- B)	+(A + B)			
(- A) - (+B)	-(A+B)			
(-A) - (-B)		-(A-B)	+(B - A)	+(A - B)

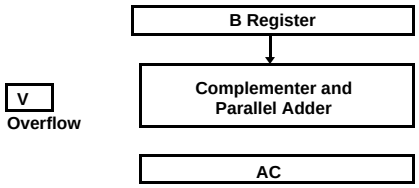
Hardware Implementation



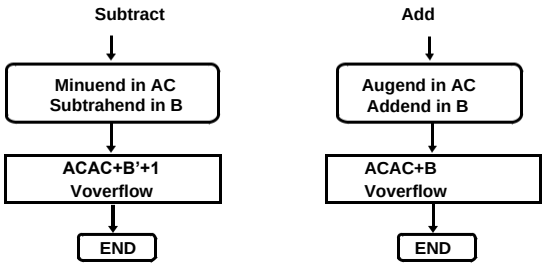
B IS Transfer M=0,COMP OF B IS TRANSFER M=1

SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION

Hardware



Algorithm



Algorithm:

1 The two signs A, and B, are compared by an exclusive-OR gate.

If the output of the gate is 0 the signs are
Identical; If it is 1, the signs are different.

2. For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.

3. The magnitudes are added with a microoperation EA A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.

4. The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.

5. 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.

6. 0 in E indicates that A < B. For this case it is necessary to take the 2's Complement of the value in A. The operation can be done with one microoperation A A' +1.

7. However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.

8. In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.

9. The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.

10. Figure 4.2 shows a block diagram of the hardware for implementing the addition and subtraction operations.

11. It consists of registers A and B and sign flip-flops As and Bs.
12 Subtraction is done by adding A to the 2's complement of B.

13. The output carry is transferred to flip-flop E , where it can be checked to determine the relative magnitudes of two numbers.

14. The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.

15. The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

In Signed 2's Complement representation the leftmost bit of a binary number represent the sign bit 0 for positive and 1 for negative .If the sign bit is 1 entire number is represented in 2's complement
Addition of two number in signed 2's complement consist of adding the number with sign bit treated the same as the other bit of the number .A carry out of sign bit position is discarded

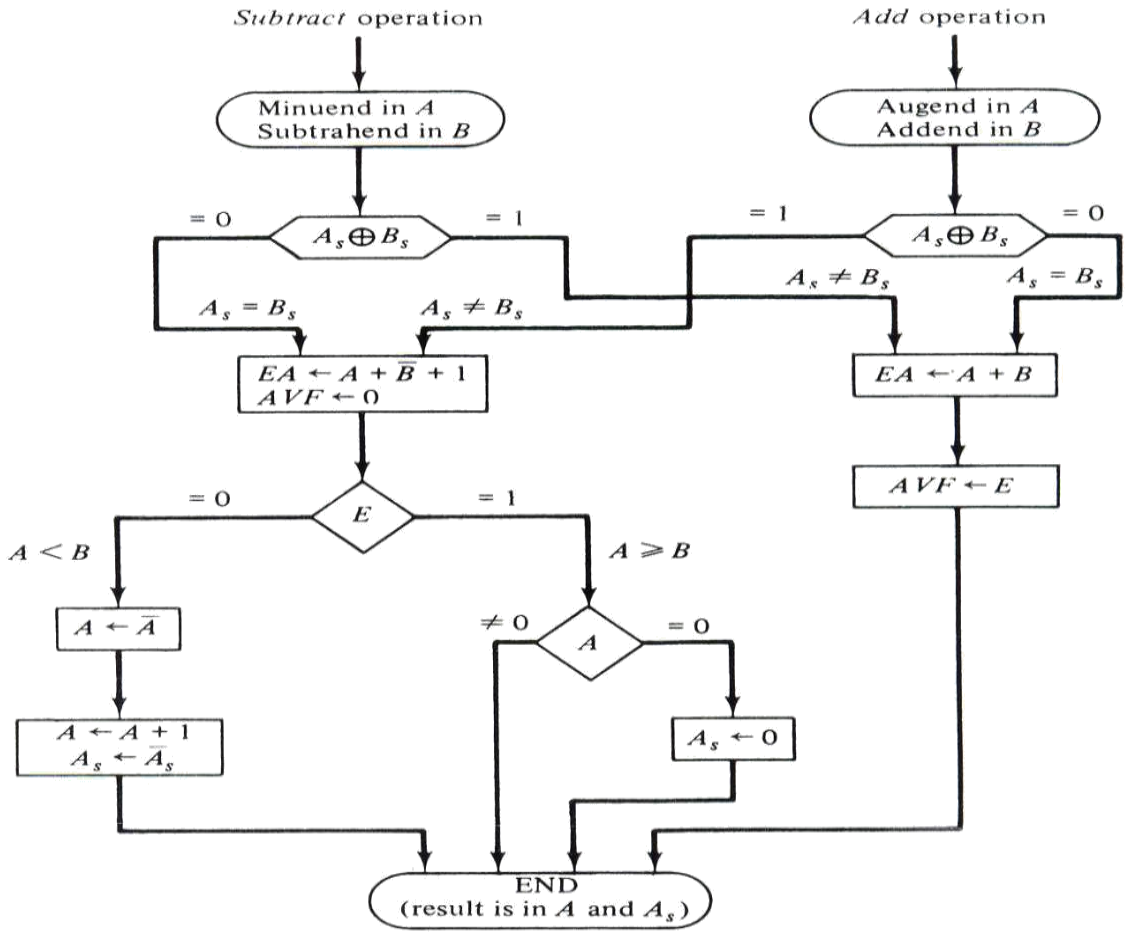
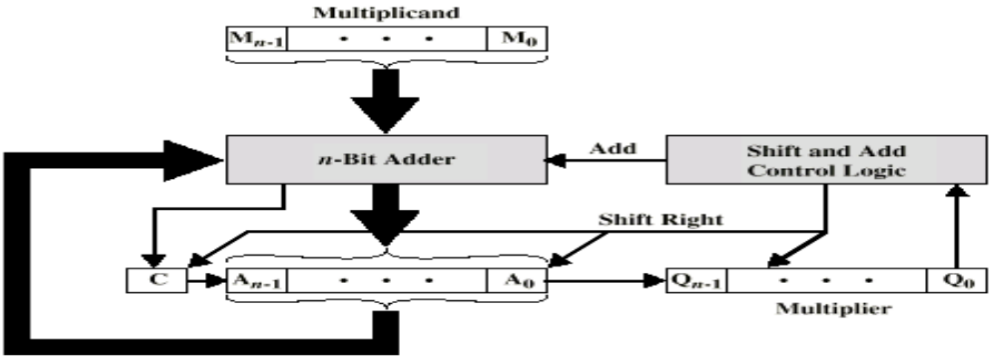


Figure 10-2 Flowchart for add and subtract operations.

Multiplication Algorithm:

In the beginning, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs respectively. We compare the signs of both A and Q and set to corresponding sign of the product since a double-length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to the number of bits of the multiplier. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.

Now, the low order bit of the multiplier in Qn is tested. If it is 1, the multiplicand (B) is added to present partial product (A), 0 otherwise. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. When SC = 0 we stops the process.



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

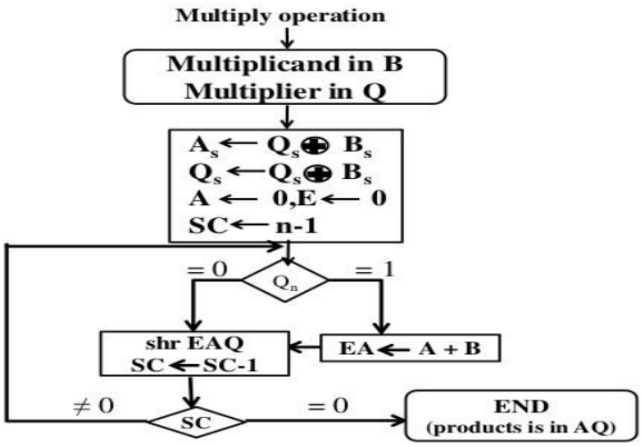


Figure: Flowchart for multiply operation.

Booth's algorithm :

- Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

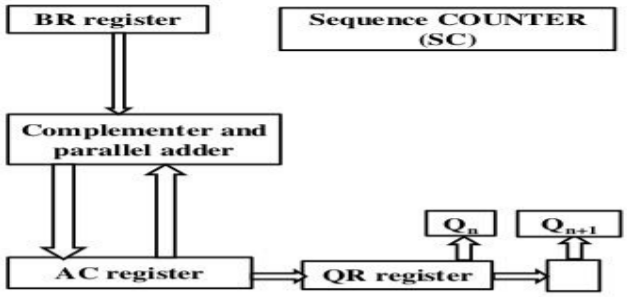
  For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m$. = $2^4 - 2^1$ = 16 – 2 = 14.

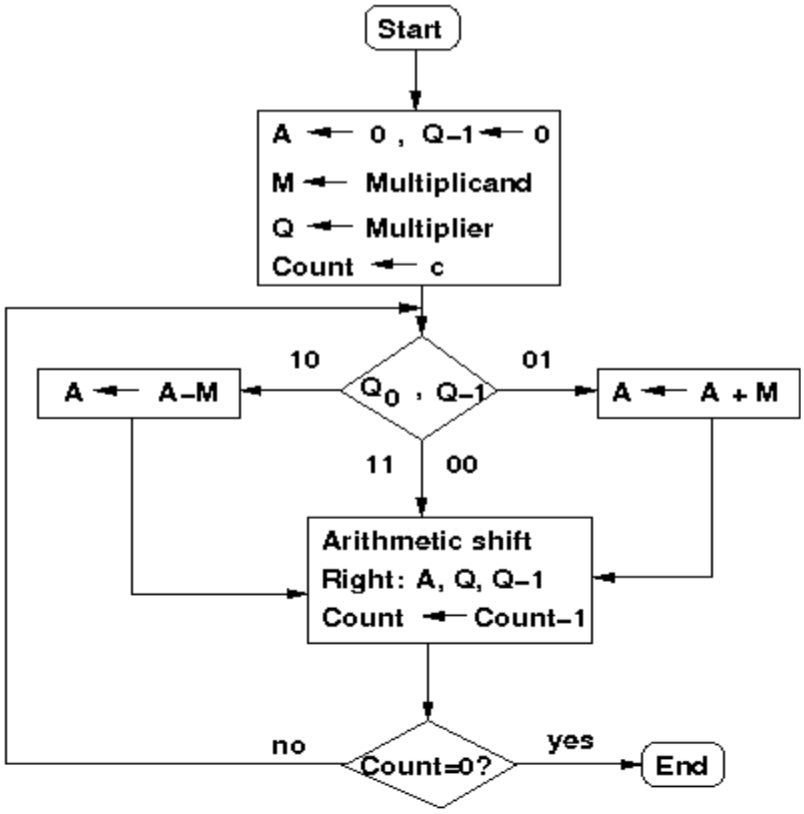
Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.

- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Hardware for Booth Algorithm

- Sign bits are not separated from the rest of the registers
- rename registers A,B, and Q as AC,BR and QR respectively
- Q_n designates the least significant bit of the multiplier in register QR
- Flip-flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier





As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, or left unchanged according to the following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.

3. The partial product does not change when multiplier bit is identical to the previous multiplier bit.

- The algorithm works for positive or negative multipliers in 2’s complement representation.

- This is because a negative multiplier ends with a string of 1’s and the last operation will be a subtraction of the appropriate weight.

- The two bits of the multiplier in Qn and Qn+1 are inspected.

- If the two bits are equal to 10, it means that the first 1 in a string of 1 's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.

- If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.

- When the two bits are equal, the partial product do

Booth's Algorithm for Binary Multiplication Example

Multiply 14 times -5 using 5-bit numbers (10-bit result).

14 in binary: `01110`

-14 in binary: `10010` (so we can add when we need to subtract the multiplicand)

-5 in binary: `11011`

Expected result: -70 in binary: `11101 11010`

Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	`01110`	Initialization	`00000 11011 0`

1	`01110`	`10`: Subtract Multiplicand	`00000+10010=10010`
			
			`10010 11011 0`

		Shift Right Arithmetic	`11001 01101 1`
2	`01110`	`11`: No-op	`11001 01101 1`
		Shift Right Arithmetic	`11100 10110 1`

3	`01110`	`01`: Add Multiplicand	`11100+01110=01010`
			(Carry ignored because adding a positive and negative number cannot overflow.)
			
			`01010 10110 1`

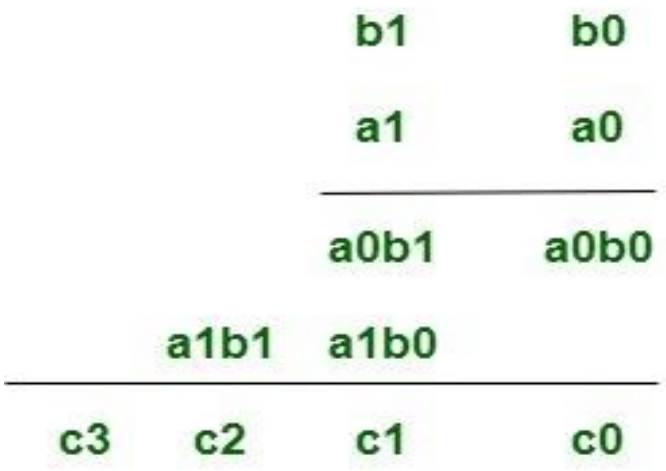
		Shift Right Arithmetic	`00101 01011 0`

4	`01110`	`10`: Subtract Multiplicand	`00101+10010=10111`
			
			`10111 01011 0`

		Shift Right Arithmetic	`11011 10101 1`
5	`01110`	`11`: No-op	`11011 10101 1`
		Shift Right Arithmetic	`11101 11010 1`

Array Multiplier

- An **array multiplier** is a digital combinational circuit used for multiplying two binary numbers by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array.
- Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift micro-operations. The multiplication of two binary numbers can be done with one micro-operation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two numbers since all it takes is the time for the signals to propagate through the gates that form the multiplication array. However, an array multiplier requires a large number of gates, and for this reason it was not economical until the development of integrated circuits.
- For implementation of array multiplier with a combinational circuit, consider the multiplication of two 2-bit numbers as shown in figure. The multiplicand bits are b1 and b0, the multiplier bits are a1 and a0, and the product is

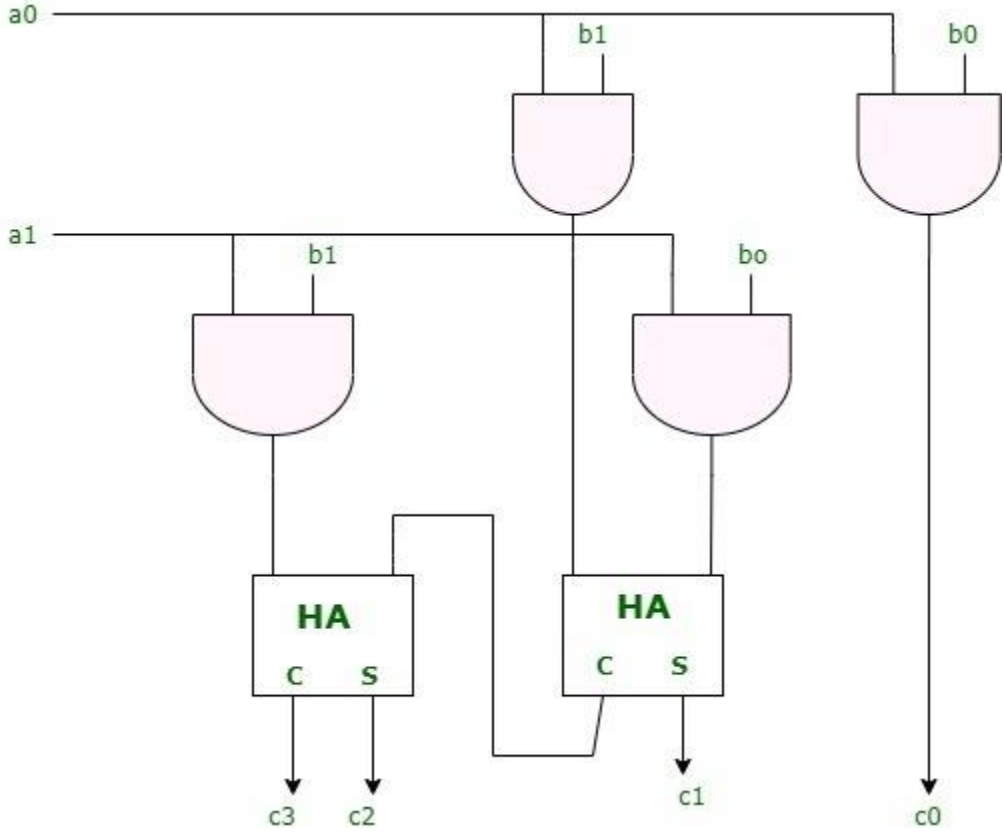


Note that the least significant bit of the Assuming A = a1a0 and B= b1b0, the various bits of the final product term P can be written as:-
1. P(0)= a0b0
2. P(1)=a1b0 + b1a0
3. P(2) = a1b1 + c1 where c1 is the carry generated during the addition for the P(1) term.
4. P(3) = c2 where c2 is the carry generated during the addition for the P(2) term.

For the above multiplication, an array of four AND gates is required to form the various product terms like a0b0 etc. and then an adder array is required to calculate the sums involving the various product terms and carry combinations mentioned in the above equations in order to get the final Product bits.

1. The first partial product is formed by multiplying a0 by b1, b0. The multiplication of two bits such as a0 and b0 produces a 1 if both bits are 1; otherwise, it produces 0. This is identical to an AND operation and can be implemented with an AND gate.
2. The first partial product is formed by means of two AND gates.
3. The second partial product is formed by multiplying a1 by b1b0 and is shifted one position to the left.

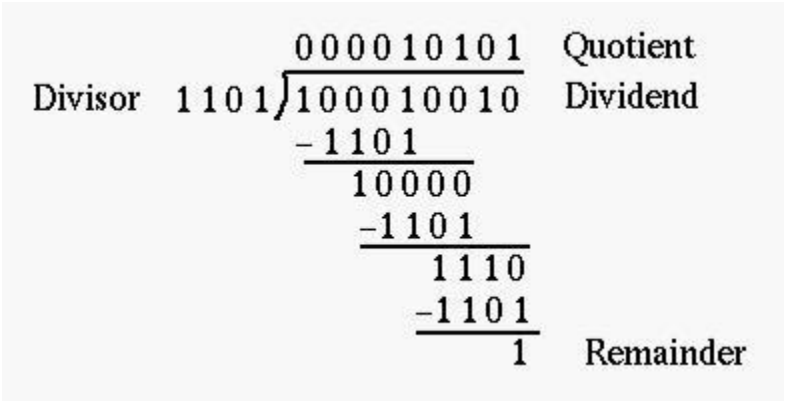
4. The above two partial products are added with two half-adder(HA) circuits. Usually there are more bits in the partial products and it will be necessary to use full-adders to produce the sum.
5. product does not have to go through an adder since it is formed by the output of the first AND gate.



A combinational circuit binary multiplier with more bits can be constructed in similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates is added in parallel with the partial product of the previous level to form a new partial product. The last level produces the product. For j multiplier bits and k multiplicand we need **j*k** AND gates and **(j-1)** k-bit adders to produce a product of **j+k** bits.

Division Algorithms

Division of two fixed-point binary numbers in signed magnitude representation is performed with paper and pencil by a process of successive compare, shift and subtract operations. Binary division is much simpler than decimal division because here the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor. The division process is described in Figure

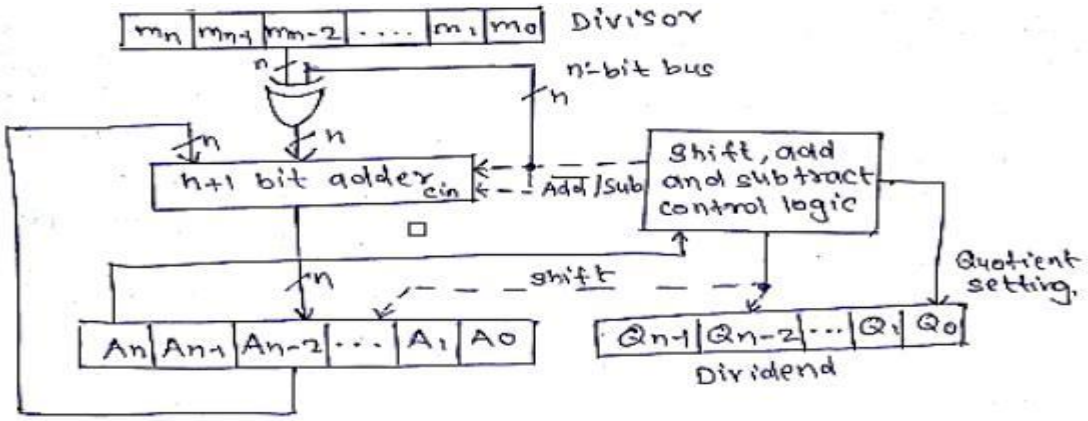


The devisor is compared with the five most significant bits of the dividend. Since the 5-bit number is smaller than B, we again repeat the same process. Now the 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend. Now we shift the divisor once to the right and subtract it from the dividend. The difference is known as a partial remainder because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial remainder. Comparing a partial remainder with the divisor continues the process. If the partial remainder is greater than or equal to the divisor, the quotient bit is equal to 1. The divisor is then shifted right and subtracted from the partial remainder. If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Obviously the result gives both a quotient and a remainder.

Hardware Implementation for Signed-Magnitude Data

In hardware implementation for signed-magnitude data in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, two dividends, or partial remainders, are shifted to the left, thus leaving the two numbers in the required relative position. Subtraction is achieved by adding A to the 2's complement of B. End carry gives the information about the relative magnitudes.

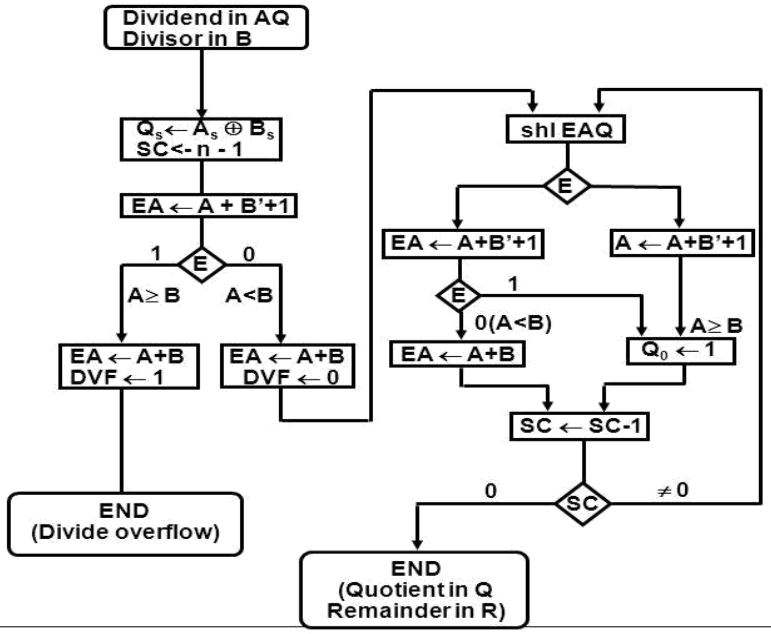
The hardware required is identical to that of multiplication. Register EAQ is now shifted to the left with 0 inserted into Qn and the previous value of E is lost. The example is given in Figure to clear the proposed division process. The divisor is stored in the B register and the double-length dividend is stored in registers A and Q. The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value. E



Hardware Implementation for Signed-Magnitude Data

Algorithm:

FLOWCHART OF DIVIDE OPERATION



Example of Binary Division with Digital Hardware

Divisor B = 10001 $\quad \overline{B} + 1 = 01111$

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
Shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Floating-point Arithmetic operations :

In many high-level programming languages we have a facility for specifying floating-point numbers. The most common way is by a real declaration statement. High level programming languages must have a provision for handling floating-point arithmetic operations. The operations are generally built in the internal hardware. If no hardware is available, the compiler must be designed with a package of floating-point software subroutine. Although the hardware method is more expensive, it is much more efficient than the software method. Therefore, floating- point hardware is included in most computers and is omitted only in very small ones.

Basic Considerations :

There are two part of a floating-point number in a computer - a mantissa m and an exponent e. The two parts represent a number generated from multiplying m times a radix r raised to the value of e. Thus

$$m \times r^e$$

The mantissa may be a fraction or an integer. The position of the radix point and the value of the radix r are not included in the registers. For example, assume a fraction representation and a radix 10. The decimal number 537.25 is represented in a register with m = 53725 and e = 3 and is interpreted to represent the floating-point number

$$.53725 \times 10^3$$

A floating-point number is said to be normalized if the most significant digit of the mantissa in nonzero. So the mantissa contains the maximum possible number of significant digits. We cannot normalize a zero because it does not have a nonzero digit. It is represented in floating-point by all 0's in the mantissa and exponent.

Floating-point representation increases the range of numbers for a given register. Consider a computer with 48-bit words. Since one bit must be reserved for the sign, the range of fixed-point integer numbers will be + (24^7 – 1), which is approximately + 101^4. The 48 bits can be used to represent a floating-point number with 36 bits for the mantissa and 12 bits for the exponent. Assuming fraction representation for the mantissa and taking the two sign bits into consideration, the range of numbers that can be represented is

$$+ (1 - 2^{-35}) \times 2^{2047}$$

This number is derived from a fraction that contains 35 1's, an exponent of 11 bits (excluding its sign), and because 2^{11}–1 = 2047. The largest number that can be accommodated is approximately 10^{615}. The mantissa that can accommodated is 35 bits (excluding the sign) and if considered as an integer it can store a number as large as (2^{35} –1). This is approximately equal to 10^{10}, which is equivalent to a decimal number of 10 digits.

Computers with shorter word lengths use two or more words to represent a floating-point number. An 8-bit microcomputer uses four words to represent one floating-point number. One word of 8 bits are reserved for the exponent and the 24 bits of the other three words are used in the mantissa.

Arithmetic operations with floating-point numbers are more complicated than with fixed-point numbers. Their execution also takes longer time and requires more complex hardware. Adding or subtracting two numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas. We do this alignment by shifting one mantissa while its exponent is adjusted until it becomes equal to the other exponent. Consider the sum of the following floating-point numbers:

$$.5372400 \times 10^{2}$$

$$+ .1580000 \times 10^{-1}$$

Floating-point multiplication and division need not do an alignment of the mantissas. Multiplying the two mantissas and adding the exponents can form the product. Dividing the mantissas and subtracting the exponents perform division.

The operations done with the mantissas are the same as in fixed-point numbers, so the two can share the same registers and circuits. The operations performed with the exponents are compared and incremented (for aligning the mantissas), added and subtracted (for multiplication) and division), and decremented (to normalize the result). We can represent the exponent in any one of the three representations - signed-magnitude, signed 2's complement or signed 1's complement.

Biased exponents have the advantage that they contain only positive numbers. Now it becomes simpler to compare their relative magnitude without bothering about their signs. Another advantage is that the smallest possible biased exponent contains all zeros. The floating-point representation of zero is then a zero mantissa and the smallest possible exponent.

Register Configuration

The register configuration for floating-point operations is shown in figure As a rule, the same registers and adder used for fixed-point arithmetic are used for processing the mantissas. The difference lies in the way the exponents are handled.

The register organization for floating-point operations is shown in Fig. Three registers are there, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part has the same uppercase letter symbols as in fixed-point representation. The exponent part may use corresponding lower-case letter symbol.

FLOATING POINT ARITHMETIC OPERATIONS

$F = m \times r^e$
where m: Mantissa
r: Radix
e: Exponent

Registers for Floating Point Arithmetic

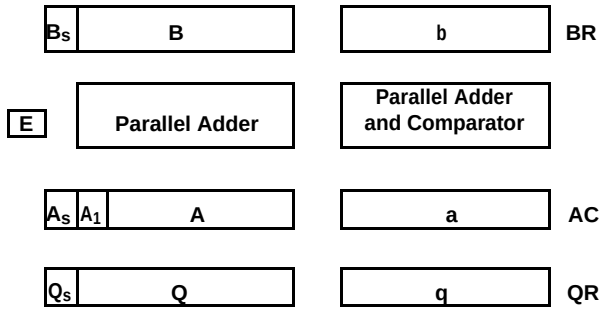


Figure: Registers for Floating Point arithmetic operations

Assuming that each floating-point number has a mantissa in signed-magnitude representation and a biased exponent. Thus the AC has a mantissa whose sign is in As, and a magnitude that is in A. The diagram shows the most significant bit of A, labeled by A1. The bit in his position must be a 1 to normalize the number. Note that the symbol AC represents the entire register, that is, the concatenation of As, A and a.

In the similar way, register BR is subdivided into Bs, B, and b and QR into Qs, Q and q. A parallel-adder adds the two mantissas and loads the sum into A and the carry into E. A separate parallel adder can be used for the exponents. The exponents do not have a district sign bit because they are biased but are represented as a biased positive quantity. It is assumed that the floating-point number are so large that the chance of an exponent overflow is very remote and so the exponent overflow will be neglected. The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.

The number in the mantissa will be taken as a fraction, so they binary point is assumed to reside to the left of the magnitude part. Integer representation for floating point causes certain scaling problems during multiplication and division. To avoid these problems, we adopt a fraction representation.

The numbers in the registers should initially be normalized. After each arithmetic operation, the result will be normalized. Thus all floating-point operands are always normalized.

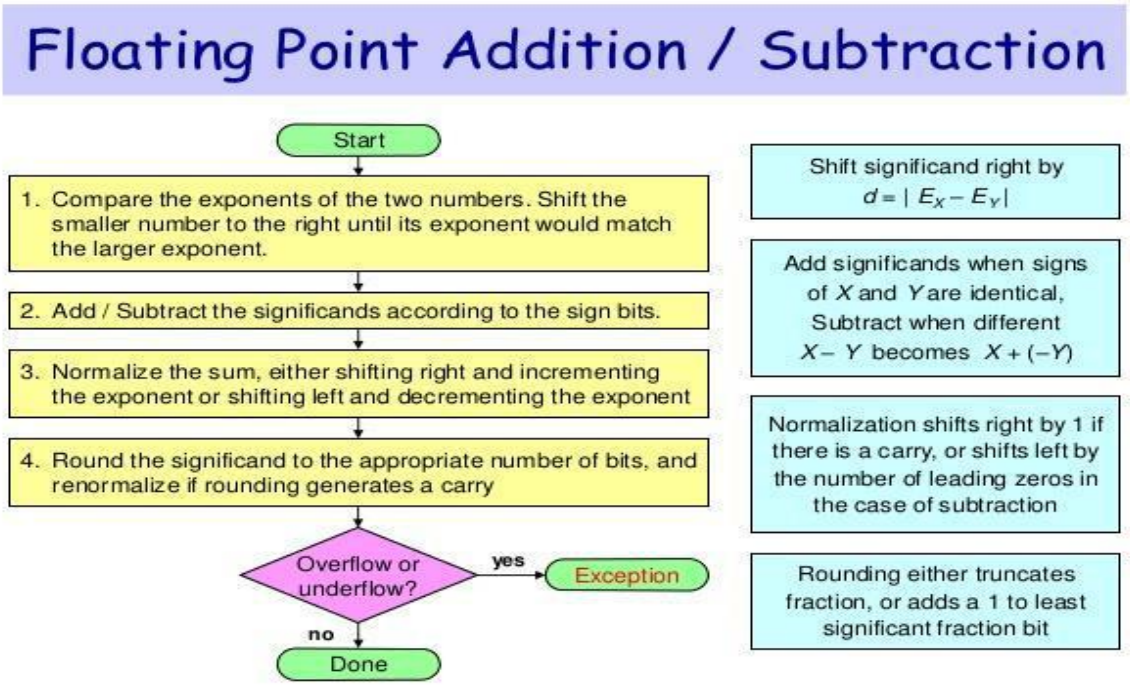
Addition and Subtraction of Floating Point Numbers

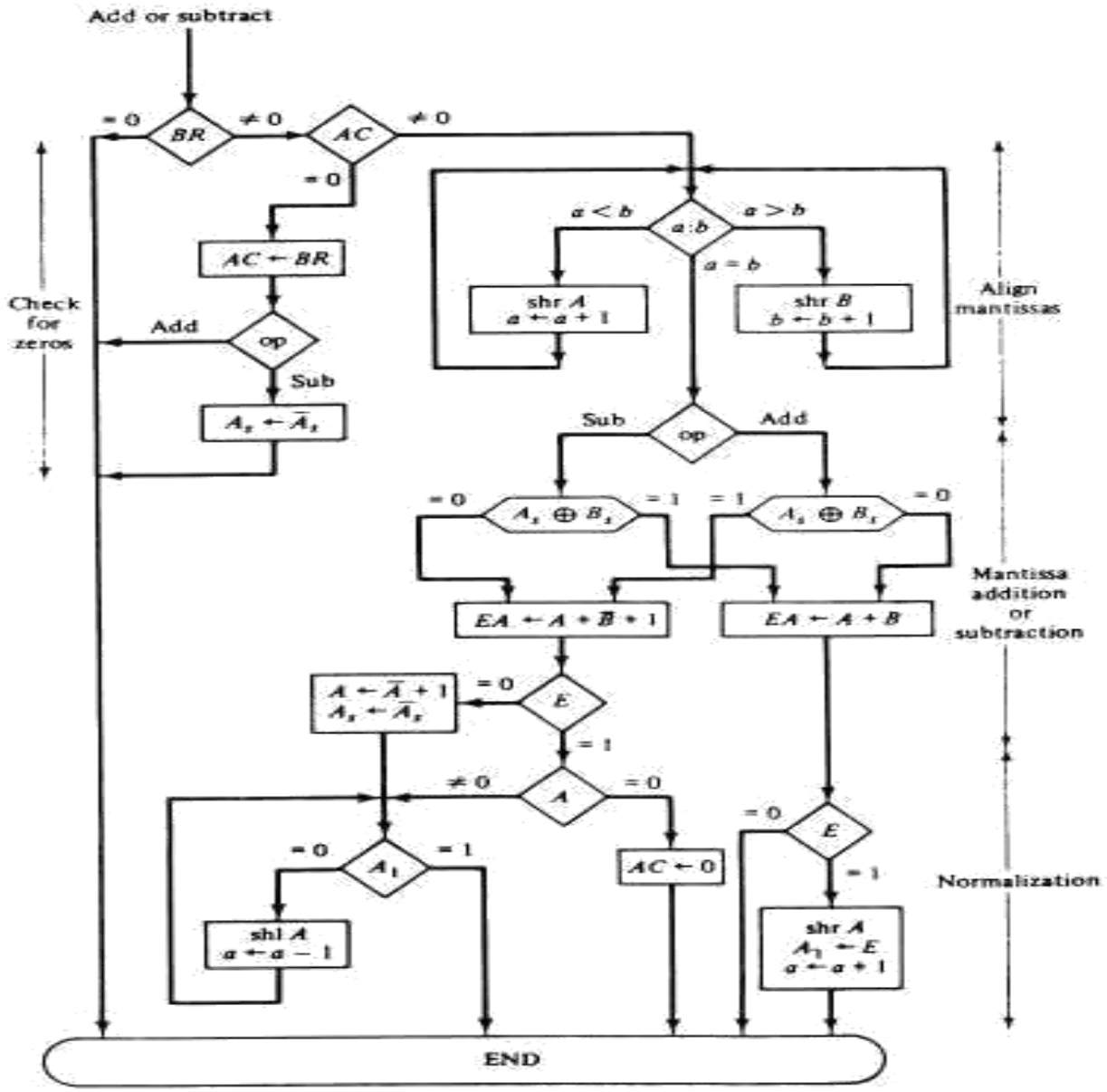
During addition or subtraction, the two floating-point operands are kept in AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts:

1. Check for zeros.
2. Align the mantissas.
3. Add or subtract the mantissas
4. Normalize the result

A floating-point number cannot be normalized, if it is 0. If this number is used for computation, the result may also be zero. Instead of checking for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary. The alignment of the mantissas must be carried out prior to their operation. After the mantissas are added or subtracted, the result may be un-normalized. The normalization procedure ensures that the result is normalized before it is transferred to memory.

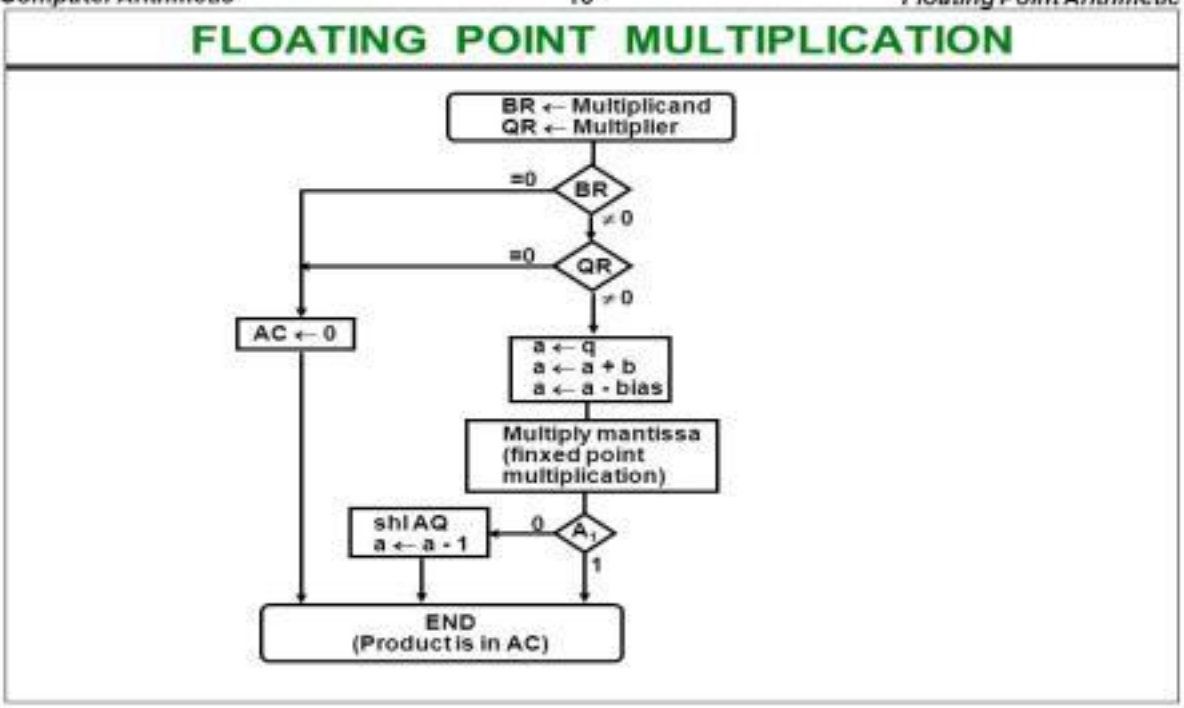
If the magnitudes were subtracted, there may be zero or may have an underflow in the result. If the mantissa is equal to zero the entire floating-point number in the AC is cleared to zero. Otherwise, the mantissa must have at least one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A1, is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A1 is checked again and the process is repeated until A1 = 1. When A1 = 1, the mantissa is normalized and the operation is completed.

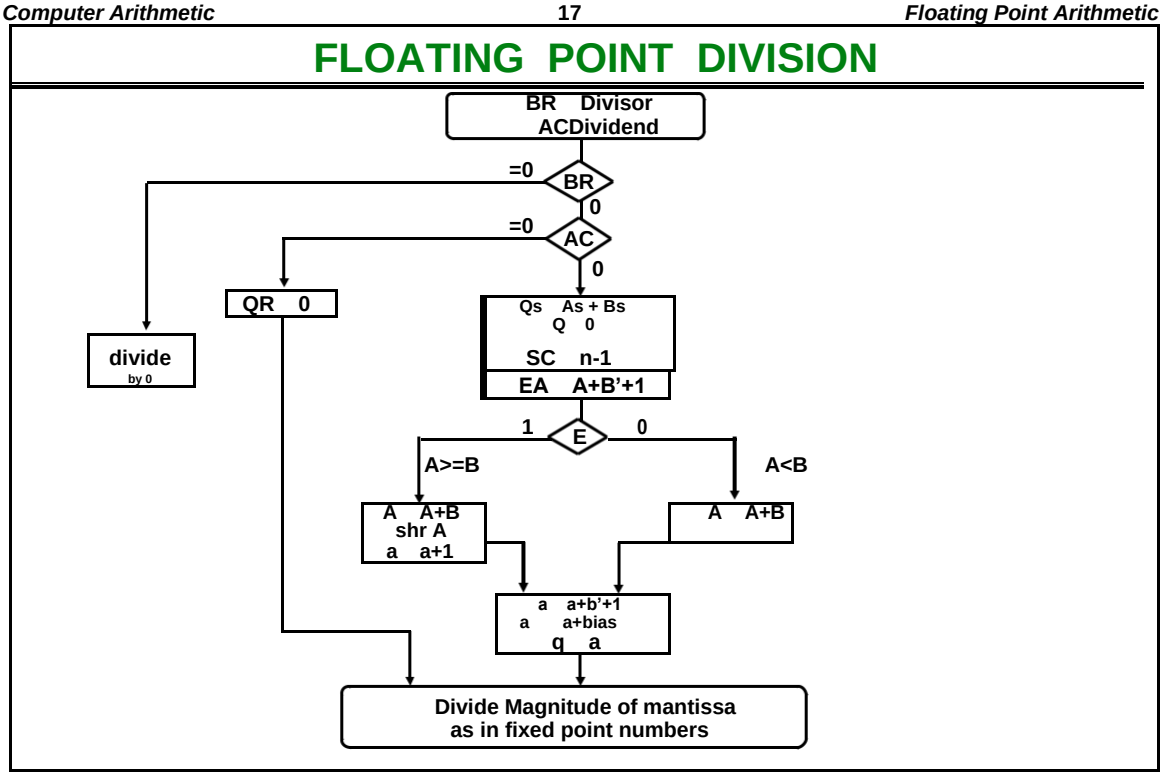




Algorithm for Floating Point Addition and Subtraction

Multiplication:

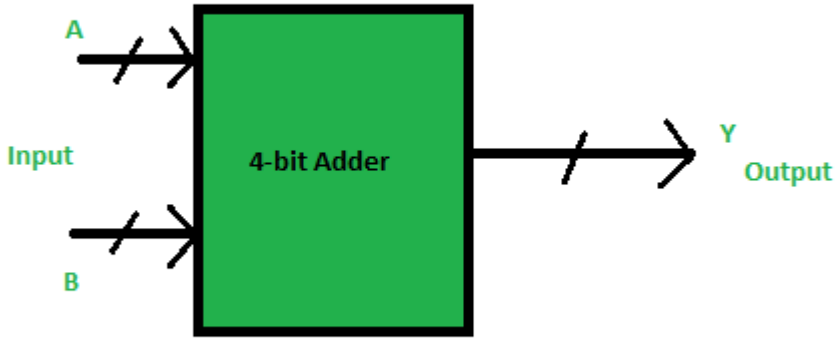




Decimal Arithmetic Unit
BCD ADDER

BCD Adder in Digital Logic

BCD stands for binary coded decimal. Suppose, we have two 4-bit numbers A and B. The value of A and B can varies from 0(0000 in binary) to 9(1001 in binary) because we are considering decimal numbers.



The output will varies from 0 to 18, if we are not considering the carry from the previous sum. But if we are considering the carry, then the maximum value of output will be 19 (i.e. 9+9+1 = 19).
When we are simply adding A and B, then we get the binary sum. Here, to get the output in BCD

The output will varies from 0 to 18, if we are not considering the carry from the previous sum. But if we are considering the carry, then the maximum value of output will be 19 (i.e. 9+9+1 = 19)

.EXAMPLE

```
Input:

A = 0111  B = 1000

Output:

Y = 1 0101


Explanation: We are adding A (=7) and B (=8).

The value of binary sum will be 1111(=15).

But, the BCD sum will be 1 0101,

Where 1 is 0001 in binary and 5 is 0101 in binary.
```

Note – If the sum of two number is less than or equal to 9, then the value of BCD sum and binary sum will be same otherwise they will differ by 6(0110 in binary).
Now, let's move to the table and find out the logic when we are going to add "0110

Decimal	Binary Sum					BCD Sum				
	C'	S3'	S2'	S1'	S0'	C	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

We are adding "0110" (=6) only to the second half of the table.

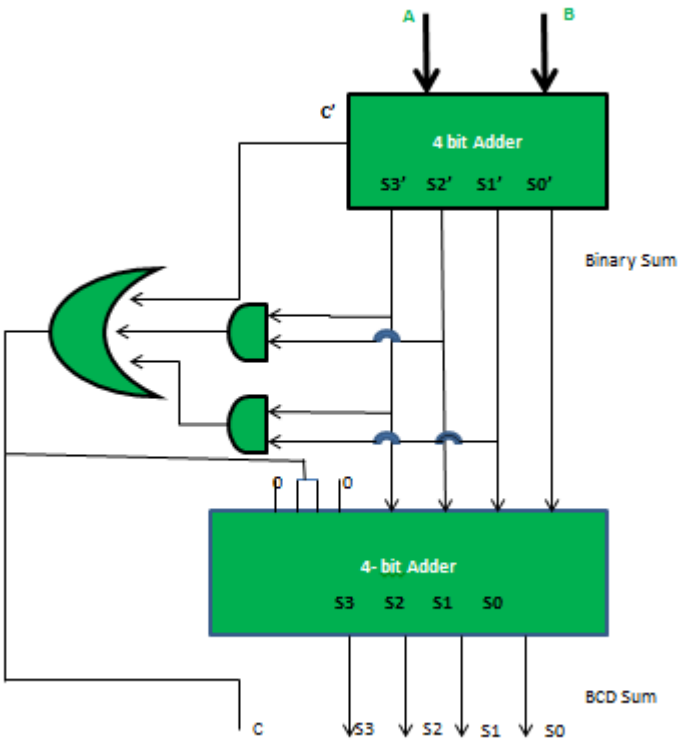
The conditions are:

1. If C' = 1 (Satisfies 16-19)
2. If S3′.S2′ = 1 (Satisfies 12-15)
3. If S3′.S1′ = 1 (Satisfies 10 and 11)

So, our logic is

```
C' + S3'.S2' + S3'.S1' = 1
```

Implementation:



Decimal BCD Subtractor:

Decimal BCD Subtractor – Addition of signed BCD numbers can be performed by using 9's or 10's complement methods. A negative BCD number can be expressed by taking the 9's or 10's complement. Let us see 9's and 10's complement numbers and subtraction process using it.

9's Complement Subtraction:

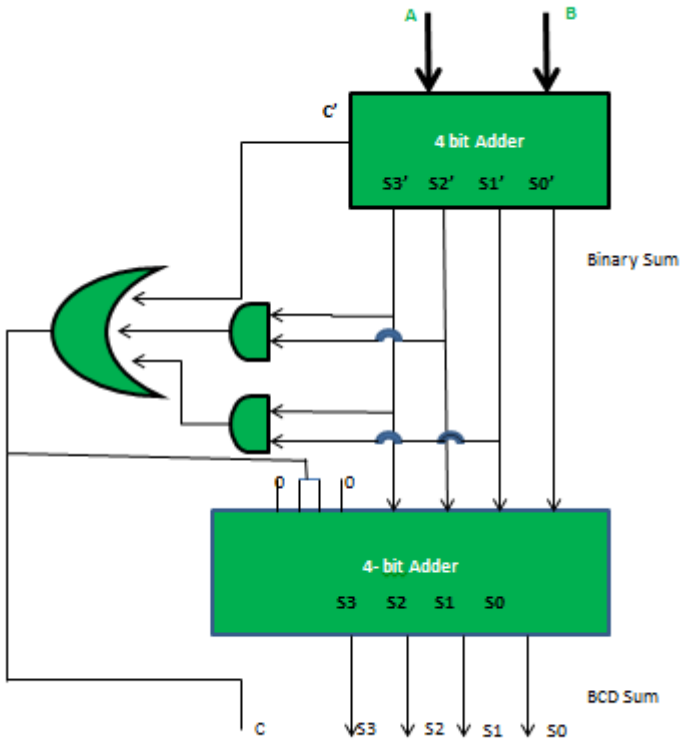
In 9's complement subtraction when 9's complement of smaller number is added to the larger number carry is generated. It is necessary to add this carry to the result. (this is called an end-around carry). When larger number is subtracted from smaller one, there is no carry, and the result is in 9's complement form and negative. This is illustrated in following examples :

Regular Subtraction **9's Complement Subtraction**

(a)

$$\begin{array}{r} 8 \\ -\ 2 \\ \hline 6 \end{array}$$

$$\begin{array}{rl} 8 & \\ +\ 7 & \text{9's complement of 2} \\ \hline (1)\ 5 & \\ \quad +\ 1 & \text{Add carry to result} \end{array}$$



ement of 13

y to the result

ement of 24

lement of result
indicates that the
negative and in
ent form)

UNIT-IV

INPUT-OUTPUT ORGANIZATION

Peripheral Devices:

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are:

i) Monitor

ii) Keyboard

iii) Mouse

iv) Printer

v) Magnetic tapes

The devices that are under the direct control of the computer are said to be connected online.

Input - Output Interface

Input Output Interface provides a method for transferring information between internal storage and external I/O devices.

Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.

The purpose of communication link is toresolve the differences that exist between the central computer and each peripheral.

The Major Differences are:-

1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.

2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.

3. Data codes and formats in the peripherals differ from the word format in the CPU and memory.

4. The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfers

- These components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals.

The I/O Bus consists of data lines, address lines and control lines.

The I/O bus from the processor is attached to all peripherals interface.

To communicate with a particular device, the processor places a device address on address lines.

Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.

It is also synchronizes the data flow and supervises the transfer between peripheral and processor.

Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing

The control lines are referred as I/O command. The commands are as following:

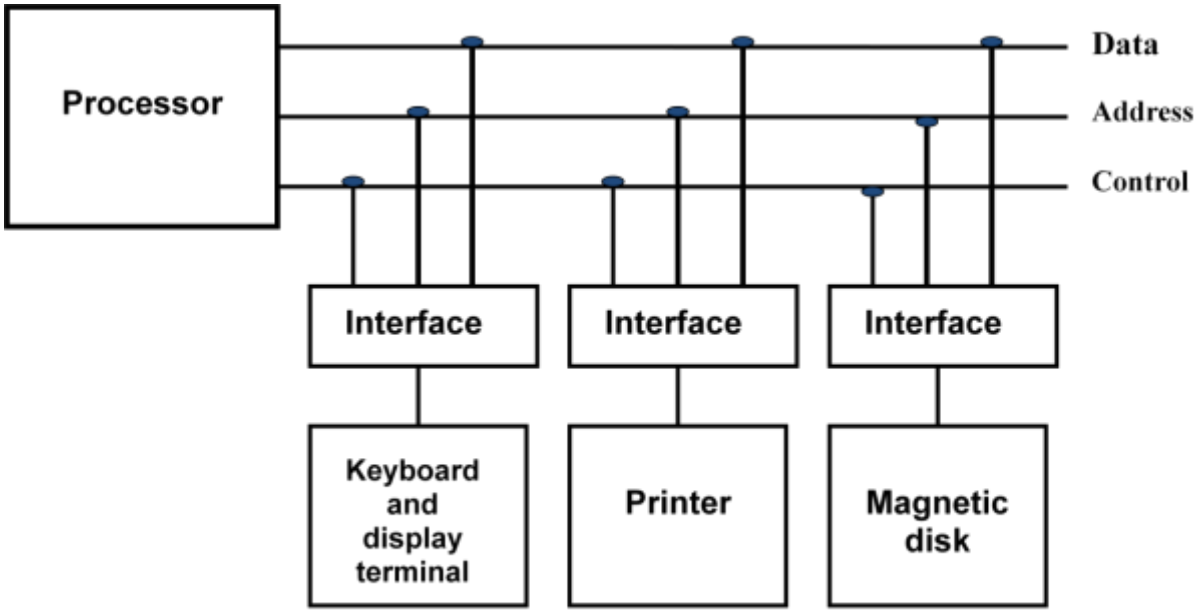
Control command-A control command is issued to activate the peripheral and to inform it what to do.

Status command-A status command is used to test various status conditions in the interface and the peripheral.

Data Output command-A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus



Connection of I/O bus to input-output devices

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

i. Use two Separate buses , one for memory and other for I/O.

ii. Use one common bus for both memory and I/O but separate control lines for each.

iii. Use one common bus for memory and I/O with common control lines.

I/O Processor

In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provides a separate I/O processor (IOP). The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory.

Asynchronous Data Transfer :

This Scheme is used when speed of I/O devices do not match with microprocessor, and timing characteristics of I/O devices is not predictable. In this method, process initiates the device and check its status. As a result, CPU has to wait till I/O device is ready to transfer data. When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.

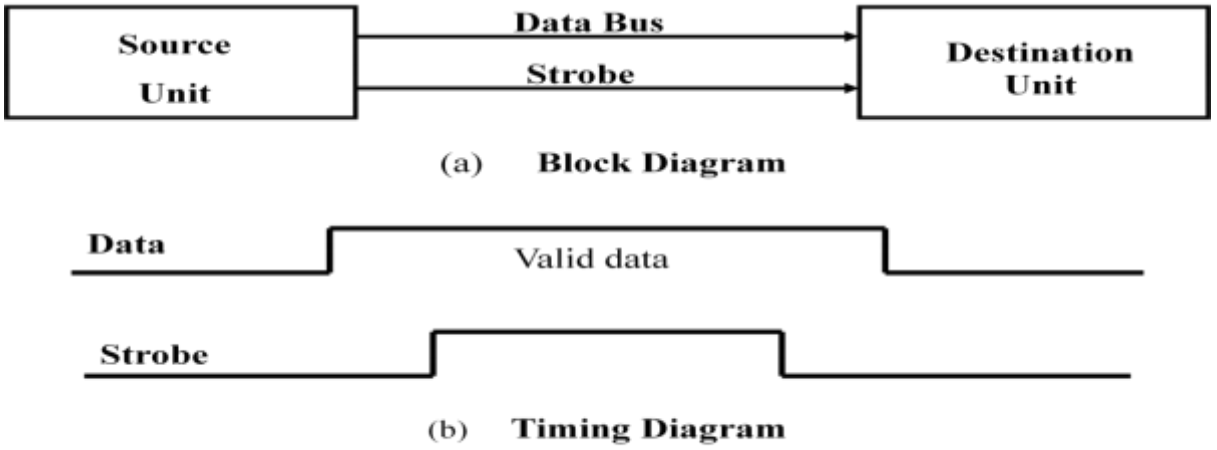
i. Strobe Control

ii. Handshaking

Strobe Signal:

The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.

Data Transfer Initiated by Source Unit:



Source-Initiated strobe for Data Transfer

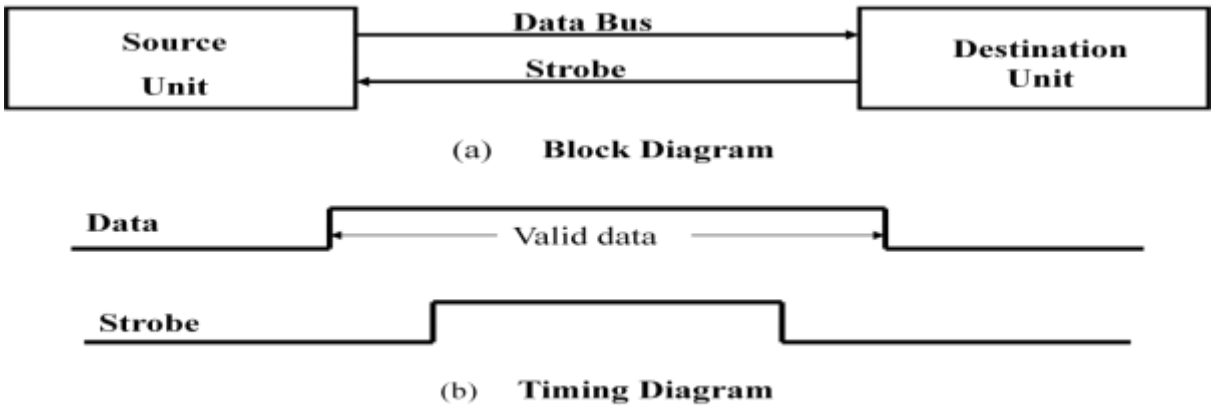
In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.

The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data.

Data Transfer Initiated by Destination Unit:

In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond by placing the requested binary information on the data bus.

The data must be valid and remain in the bus long enough for the destination unit to accept it. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.



Destination-Initiated strobe for Data Transfer

Disadvantage of Strobe Signal:

The disadvantage of the strobe method is that, the source unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was places in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus. The Handshaking method solves this problem.

Handshaking:

The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

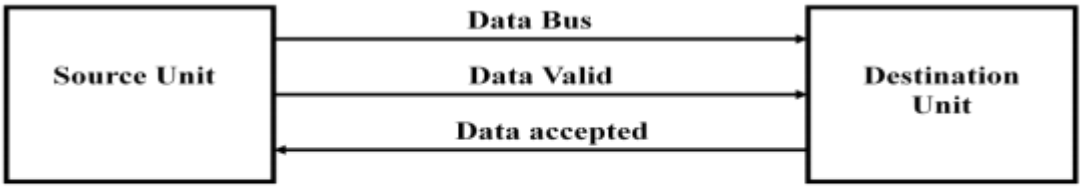
Principle of Handshaking:

The basic principle of the two-wire handshaking method of data transfer is as follow:

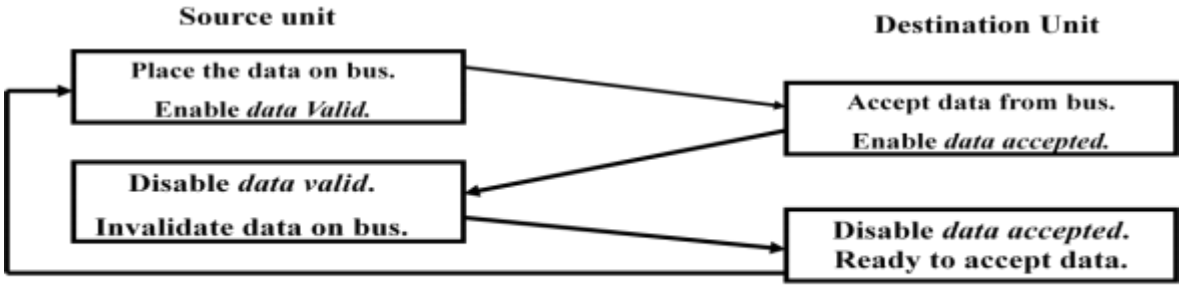
One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its *data valid* signal. The *data accepted* signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its *data accepted* signal and the system goes into its initial state.



(a) Block Diagram

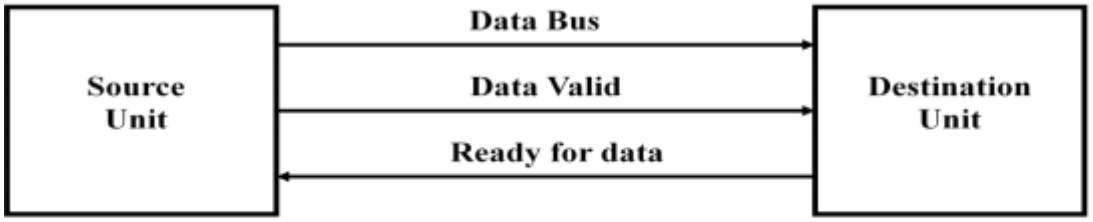


(b) Sequence of events

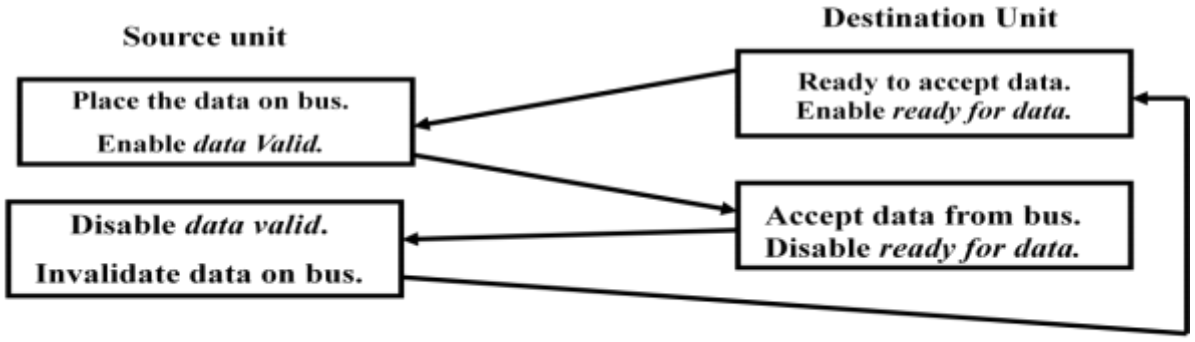
Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflects its new meaning. The source unit in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.



(a) Block Diagram



(b) Sequence of events

Destination-Initiated transfer using Handshaking

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *Timeout mechanism* which provides an alarm if the data is not completed within time.

Asynchronous Serial Transmission:

The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.

In this technique each character consists of three points :

i. Start bit

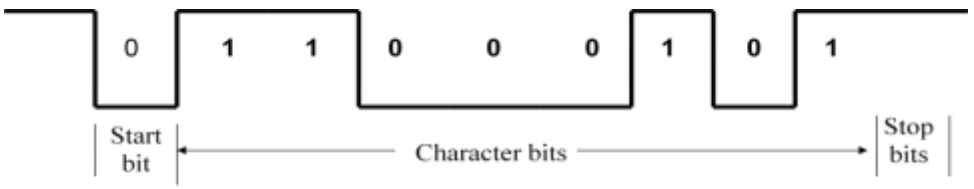
ii. Character bit

iii. Stop bit

i. Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.

ii. Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.

iii. Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by two ways:

a) Asynchronous Communication Interface

b) First In First out Buffer

Asynchronous Communication Interface:

It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register.

The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission.

The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register.

CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO):

A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates.

When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate.

If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously.

Modes of Data Transfer :

Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is Internal Operation.

All the internal operations in a digital system are synchronized by means of clock pulses supplied by a common clock pulse Generator. The data transfer can be

i. Synchronous or

ii. Asynchronous

When both the transmitting and receiving units use same clock pulse then such a data transfer is called Synchronous process. On the other hand, if the there is not concept of clock pulses

and the sender operates at different moment than the receiver then such a data transfer is called Asynchronous data transfer.

The data transfer can be handled by various modes. some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

i. Programmed I/O

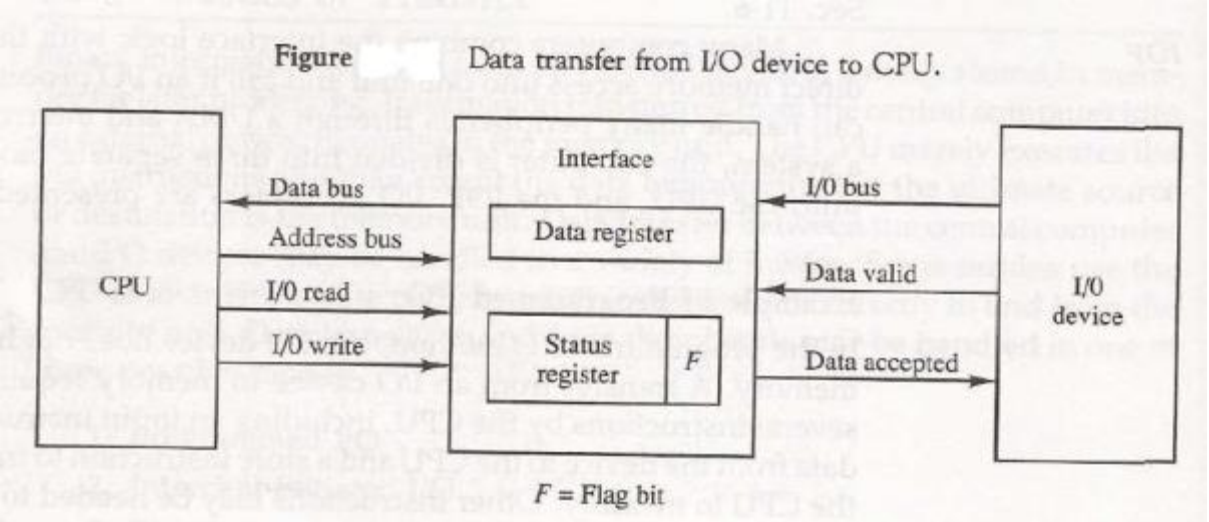
ii. Interrupt-Initiated I/O

iii. Direct Memory Access (DMA)

Programmed I/O Mode:

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa.

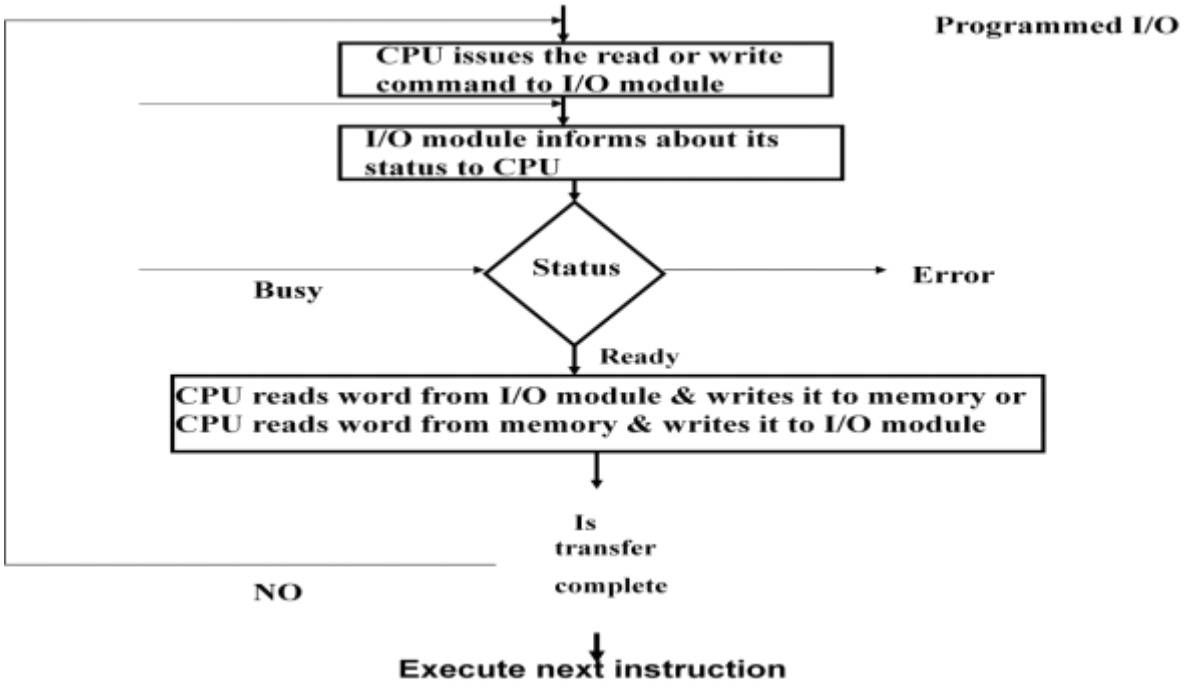
Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.



- The transfer of data requires three instructions:

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

In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Flowchart-:



Drawback of the Programmed I/O :

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process and the CPU time is wasted a lot in keeping an eye to the executing of program.

To remove this problem an Interrupt facility and special commands are used.

Interrupt-Initiated I/O :

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

- In this type of IO, computer does not check the flag. It continue to perform its task.

Whenever any device wants the attention, it sends the interrupt signal to the CPU.

- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.
- There are two ways of choosing the branch address:
  - Vectored Interrupt
  - Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

Priority Interrupt:

- There are number of IO devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
  - Using Software
  - Using Hardware
- A pooling procedure is used to identify highest priority in software means.

Polling Procedure :

- There is one common branch address for all interrupts.
- Branch address contain the code that polls the interrupt sources in sequence. The highest priority is tested first.
- The particular service routine of the highest priority device is served.
- The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

Using Hardware:

- Hardware priority system function as an overall manage

- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.
- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU.
- CPU then sends the INTACK signal which is applied to PI(priority in) of the first device.
- If it had requested the attention, it place its VAD(vector address) on the bus. And it block the signal by placing 0 in PO(priority out)
- If not it pass the signal to next device through PO(priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

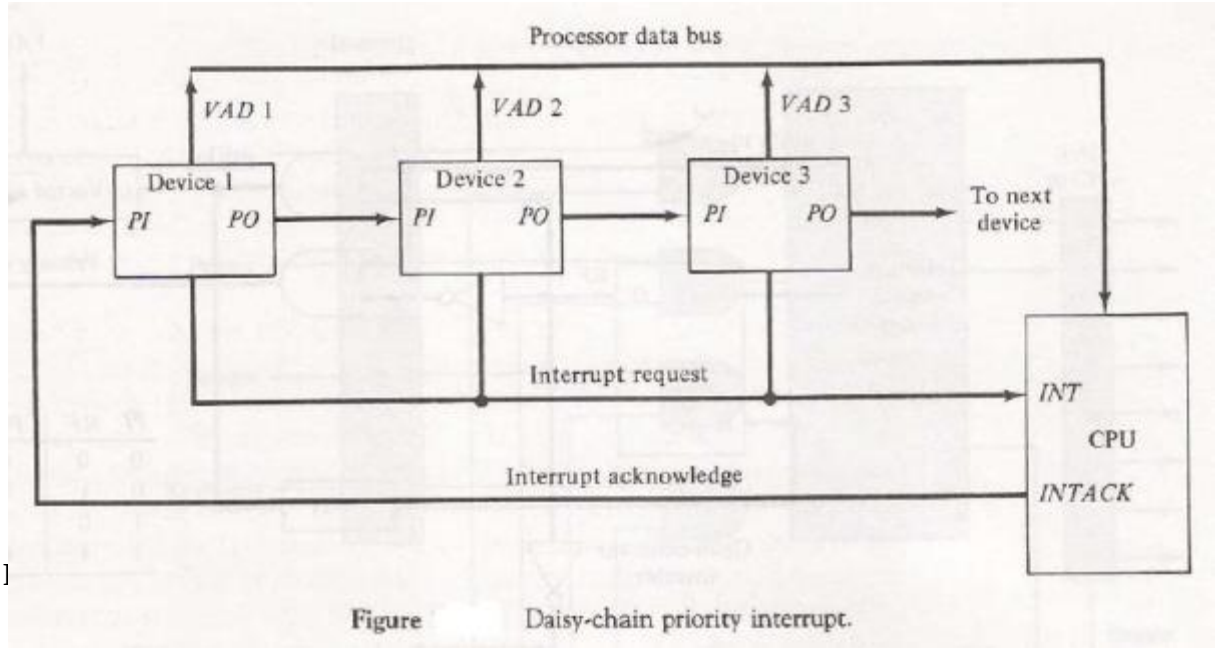
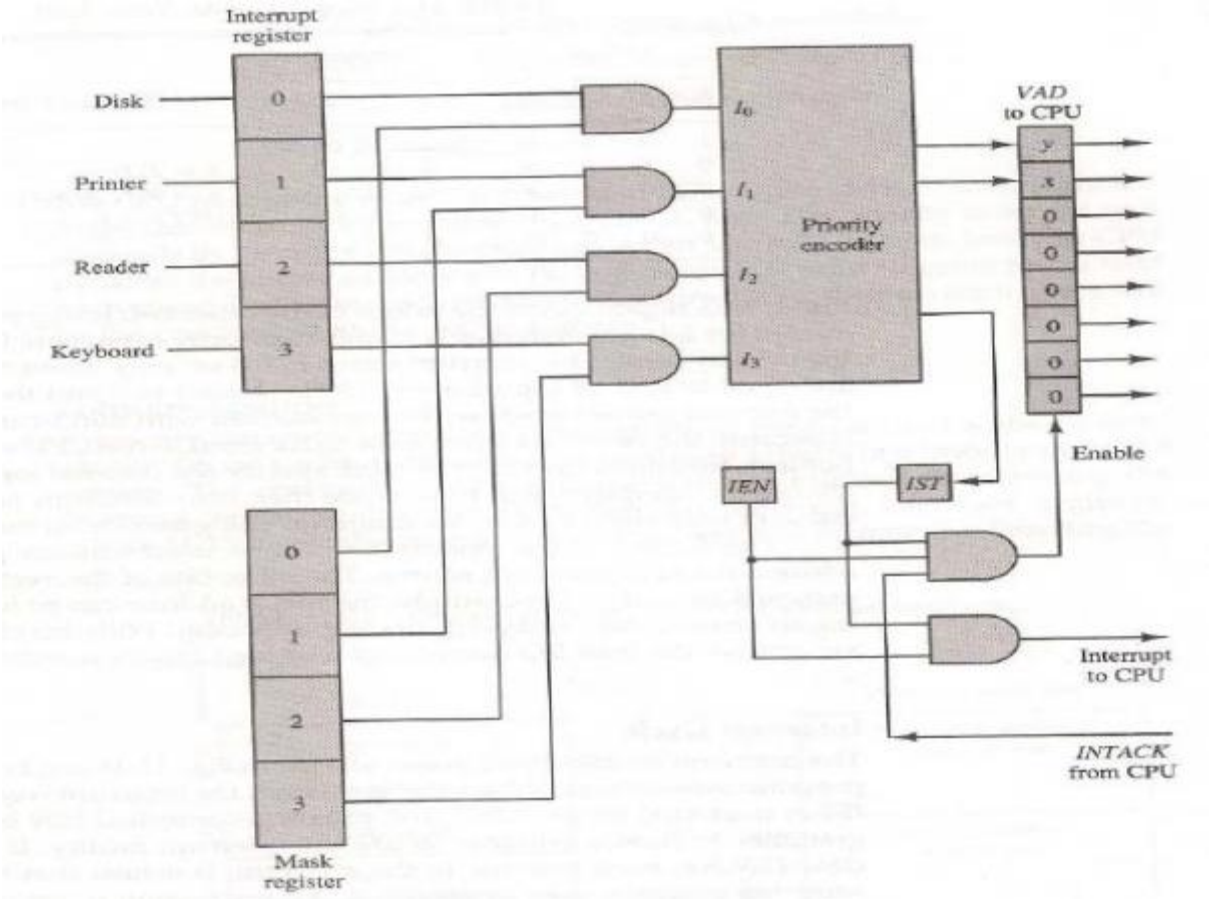


Figure Daisy-chain priority interrupt.

- Priority is established according to the position of the bits in the register.

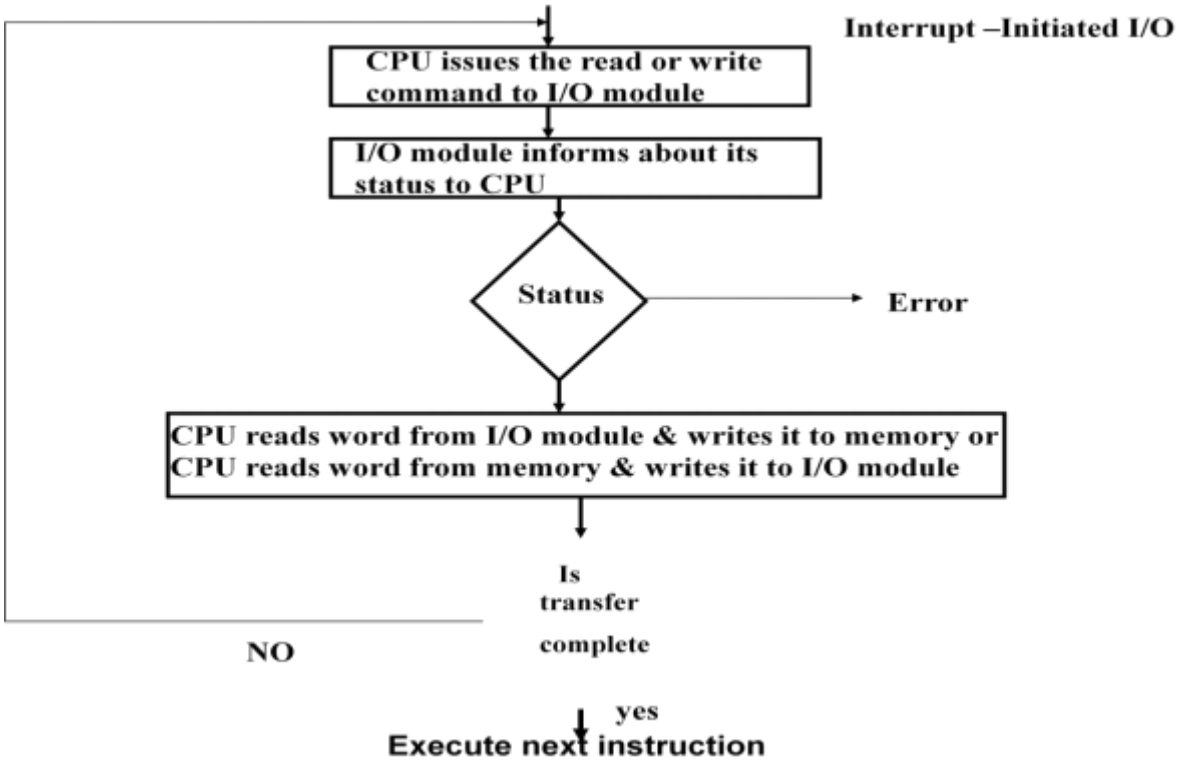
- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.
- Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.
- Priority encoder generates two bits of vector address.
- Another output from it sets IST(interrupt status flip flop).



Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
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 Execution process of Interrupt–Initiated I/O is represented in the flowchart:



Direct Memory Access (DMA):

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. 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 manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

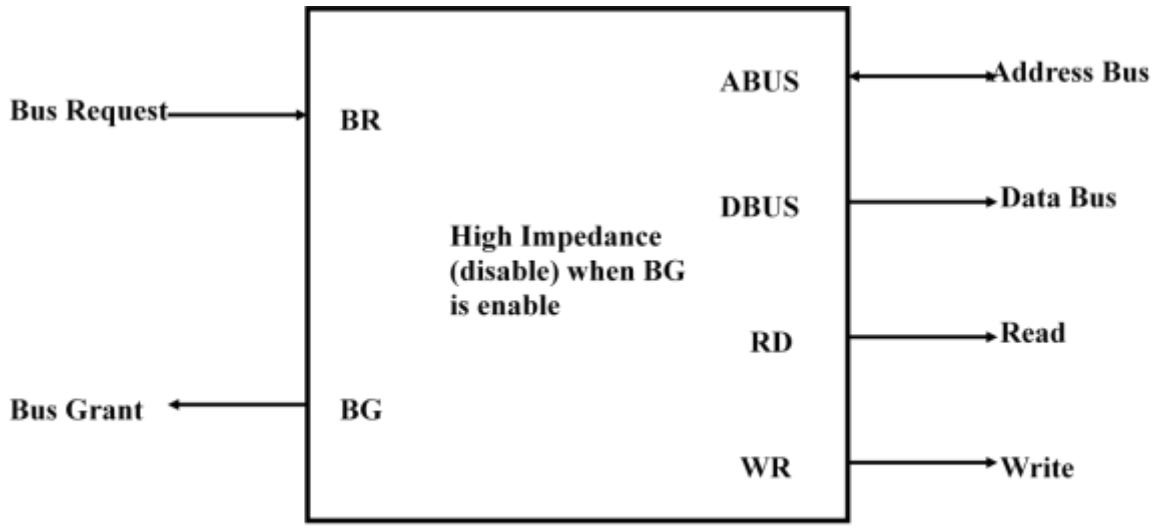
During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- Bus Request (BR)
- Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request (BR)* input is used by the *DMA controller* to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus

and read write lines into a *high Impedance state.* High Impedance state means that the output is disconnected.



CPU bus Signals for DMA Transfer

The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.

When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPU disables the *Bus Grant (BG)*, takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

i. DMA Burst

ii. Cycle Stealing

i) DMA Burst: - In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

ii) Cycle Stealing: - Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

i. Address Register

ii. Word Count Register

iii. Control Register

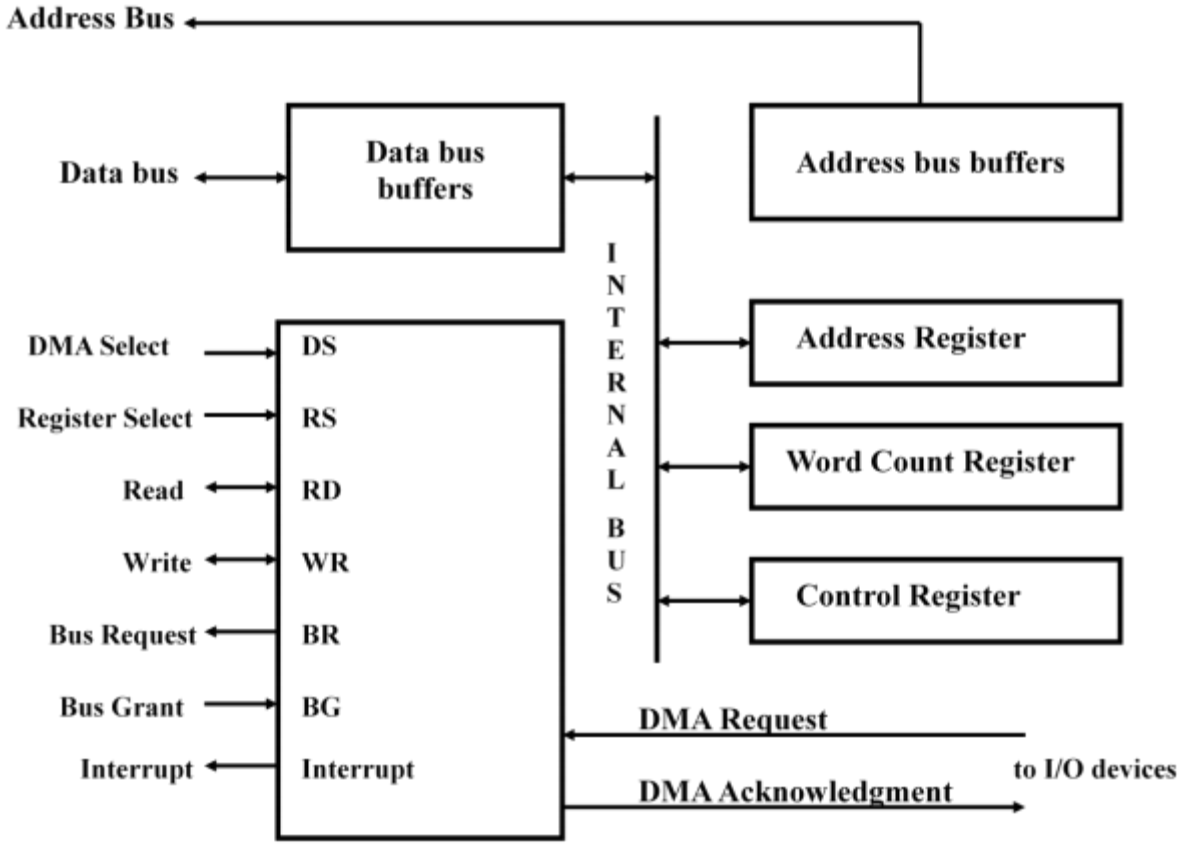
i. Address Register :- Address Register contains an address to specify the desired location in memory.

ii. Word Count Register :- WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.

i. Control Register :- Control Register specifies the mode of transfer

The unit communicates 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 bidirectional.

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 DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



Block Diagram of DMA Controller

DMA Transfer:

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 transfer between the peripheral and the memory.

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 of data.

Summary :

- Interface is the point where a connection is made between two different parts of a system.
- The strobe control method of Asynchronous data transfer employs a single control line to time each transfer.
- The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.
- Programmed I/O mode of data transfer the operations are the results in I/O instructions which is a part of computer program.
- In the Interrupt Initiated I/O method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer.
- In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus.

Input-Output Processor:

- It is a processor with direct memory access capability that communicates with IO devices.
- IOP is similar to CPU except that it is designed to handle the details of IO operation.
- Unlike DMA which is initialized by CPU, IOP can fetch and execute its own instructions.
- IOP instructions are specially designed to handle IO operation.

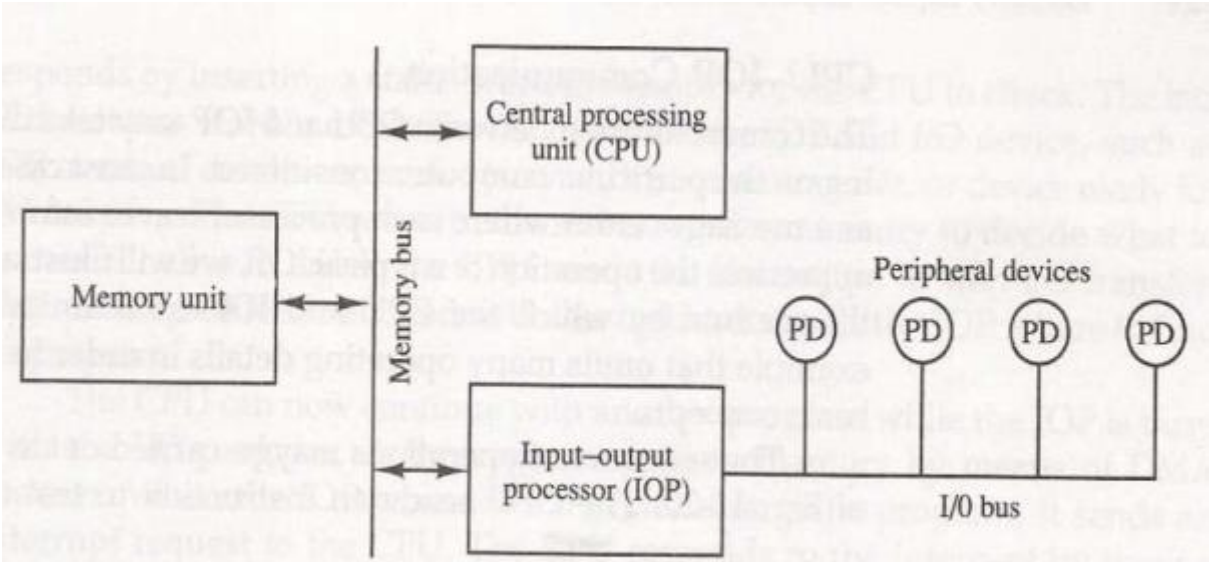


Figure Block diagram of a computer with I/O processor.

Memory occupies the central position and can communicate with each processor by DMA.

- CPU is responsible for processing data.
- IOP provides the path for transfer of data between various peripheral devices and memory.
- Data formats of peripherals differ from CPU and memory. IOP maintain such problems.
- Data are transfer from IOP to memory by stealing one memory cycle.
- Instructions that are read from memory by IOP are called commands to distinguish them from instructions that are read by the CPU.
-
-

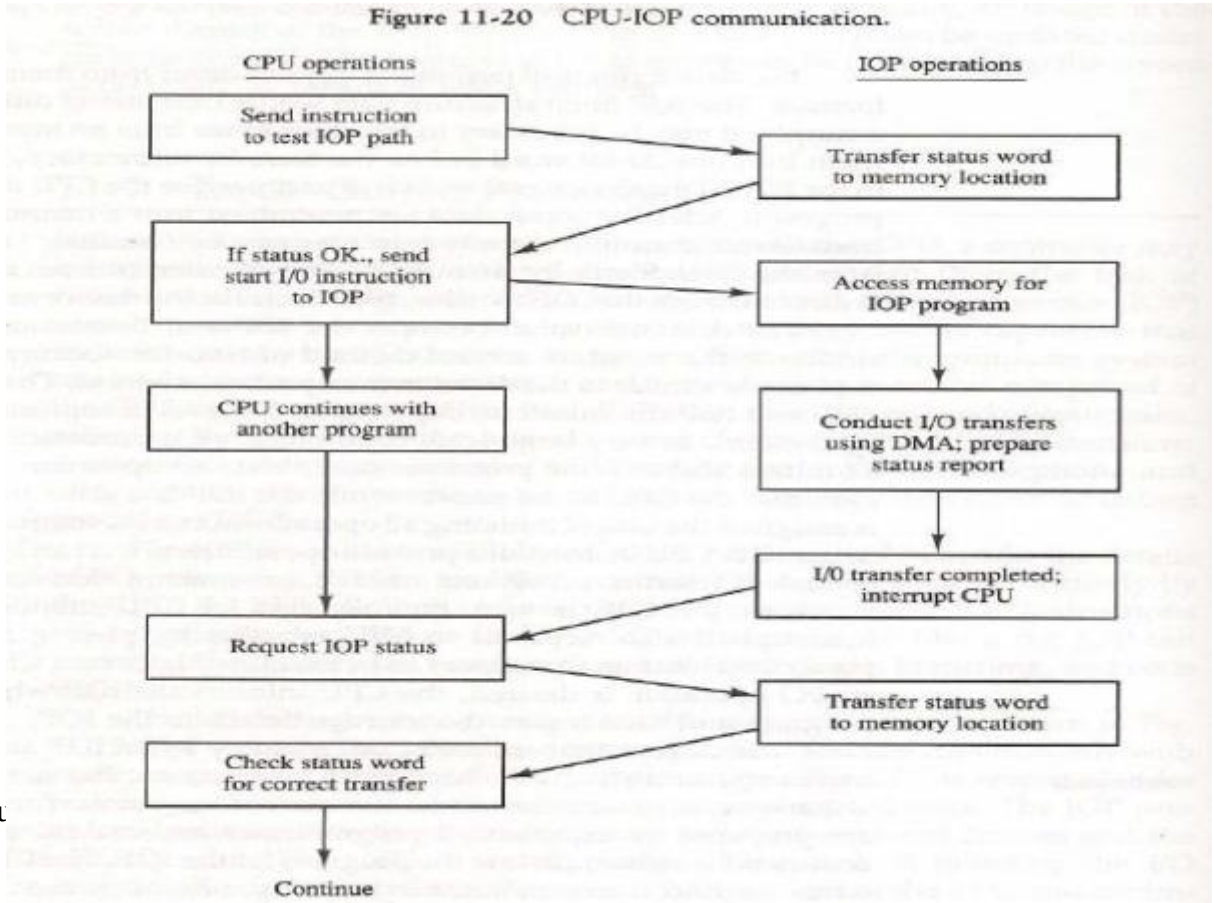


Figure 11-20 CPU-IOP communication.

Instru

» Commands are prepared by experienced programmers and are stored in memory

» Command word = IOP program

Serial Communication

In telecommunication and data transmission, **serial communication** is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.

Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical. Serial computer buses are becoming more common even at shorter distances, as improved signal integrity and transmission speeds in newer serial technologies have begun to outweigh the parallel bus's advantage of simplicity (no need for serializer and deserializer, or SerDes) and to outstrip its disadvantages (clock skew, interconnect density). The migration from PCI to PCI Express is an example.

The way that remote terminals are connected to a data communication processor is via telephone lines ,since telephone lines were originally designed for voice communication and computers communicate in terms of digital signals ,some form of conversion must be used .The converter are called modem

In synchronous transmission where an entire block of characters is transmitted, each character has a parity bit for the receiver to check. After the entire block is sent, the transmitter sends one more character that constitutes a parity over the length of the message. This character is called longitudinal redundancy check (LRC) .The receiving station calculate the LRC as it receives characters and compare it with the transmitted LRC

Another method used for checking errors is transmission is the cyclic redundancy check(CRC) .This is polynomial code obtained from the message bits by passing them through a feedback shift register containing a number of exclusive OR gates

Data can be transmitted between two points in three different modes :simplex ,half-duplex or full duplex modes

The communication lines,modems and other equipment used in the transmission of information between two or more station is called data link

The ordely transfer of information in a data link is accomplished by means of a protocol