# Unit - 1

## Introduction :-

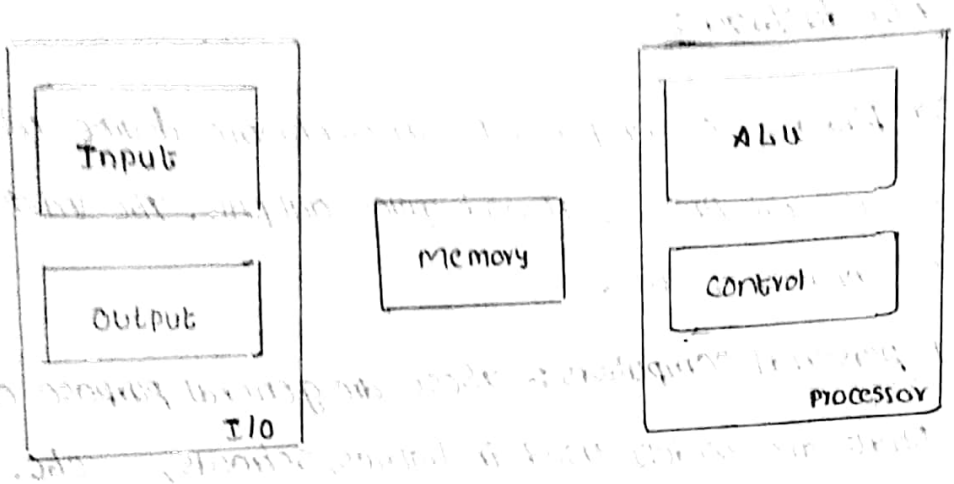
**Computer :-** A computer is an electronic device which takes input and process it and gives output. The various Types of computers are :-

1. **personal computers :-** These are general purpose computers that are widely used in homes, schools, ---- etc.
2. **Notebook Computers (laptop) :-** These are the compact version of personal computers. These are portable.
3. **Work Stations :-** These computers give more performance than pc's. These are used for scientific purpose.
4. **Main Frames :-** These computers have more storage and faster than work stations. These are used in bussiness data processing.
5. **Super computers :-** These are fastest computers that are used for large scale numerical calculations.
6. **Hand held Computers :-** It is also called as PDA (personal Digital Assistant). It can fits into packet and runs with batteries.

## Functional Units :-

A computer consists of 5 main parts as shown are :-

1. Input unit
2. Memory unit
3. Arithematic logic unit
4. Output unit
5. Control unit




1. **Input Unit :-** Computer accepts input through input units.

Eg :- Key board, mouse.

2. **Memory unit :-**

→ The function of memory unit is to store programs and data.

→ These are two classes of storage, primary storage and secondary storage. They are:-

(i) primary memory

(ii) secondary memory

→ **primary Memory :-**

* The primary memory is the fastest memory.

* Memory in which any location can be reached in short and fixed amount of time after specifying its address is called Random Access Memory (RAM).

→ **Secondary Memory :-**

* It is used when large amount of data needs to be store.

Eg :- CD's, pendrives, --- etc.

3. Arithematic logic unit :-

It performs arithematic and logical operations such as addition, subtraction, multiplication, AND, OR etc.

4. Output unit :-

Its function is to send processed result to outside world.

5. Control unit :-

Controlls all the activities in a computer by passing the control signals.

Software :-

→ Software is a collection of programs.

→ To run application software, the system must already contains some system software in the memory.

→ Software of system performs the following functions :

* Receiving and interpreting user commands.

* Entering and editing application programs and storing them as files in secondary storage.

→ Application programs are usually written in high-level programming language, such as C, C++, Java, -- etc.

→ A system software program called a compiler translates high-level language program to a suitable machine language program.

## Languages :-

→ Computer programming languages are divided into three categories. They are :

(i) High - level languages

(ii) Assembly languages

(iii) Machine languages

## High-level languages :-

→ Languages with the heighest level of abstraction are called high-level language.

→ The same program code can be converted and run the program.

Eg: C++, JAVA, — etc.

## Assembly language :-

→ Languages are at much lower level of abstraction.

→ A program written in the assembly language of one microprocessor cannot be run on computer.

→ Assembly languages are platform dependent.

## Machine language :-

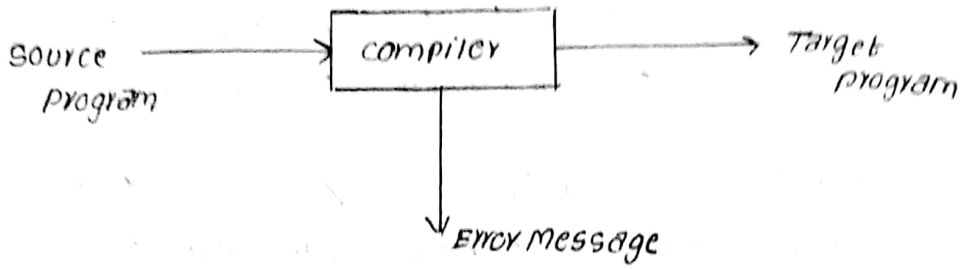
→ These language is a low-level language, which has binary values.

→ Programs never written in machine language, it will be translated into machine language.

→ Machine languages are also platform dependent; each microprocessor has its own machine language.

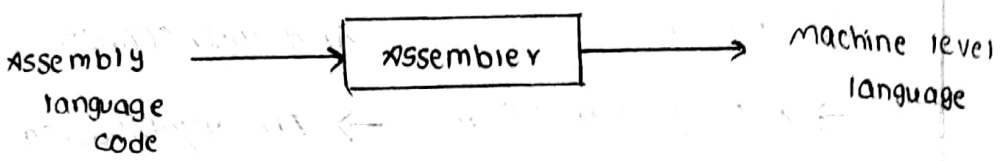
## Compiler :-

→ It is a translator which is used to convert programs in a high-level language to low level language. It translates the entire programs and also report the errors in source code.




## Assembler :-

Assembler is a translator which is used to translate the assembly language code into machine language code.




## loader :-

A loader is a part of computer operating system that is responsible for loading programs and libraries. It is one of the essential stages in process of starting program.

## linker :-

After compilation (or) Assembling process. object code is generated. Linker links the object code with other required object code.

## operating system :-

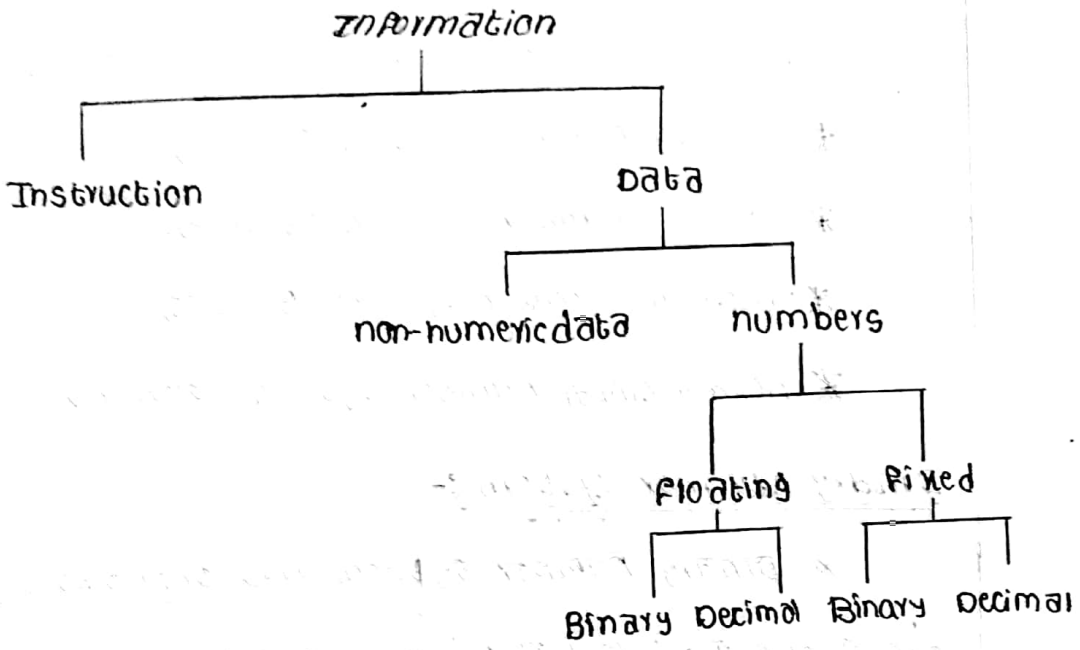
OS is a large program, or actually a collection of routines, that is used to control the sharing of and interaction among various computer units.

| Computer Architecture | Computer Organisation. |
|---|---|
| → It is concerned with the way hardware components are connected together to form a computer system. | → It is concerned with the structure and behaviour of a computer system as seen by the user. |
| → It acts as a the interface between hardware and software. | → It deals with the components of a connection in a system. |
| → It helps us to understand the functionalities of system. | → It tells us how exactly all the units are arranged and interconnected. |
| → A programmer can view architecture in terms of instructions. | → In organisation express the realization of the architecture. |
| → It involves high level design issues. | → It involves low level design issues. |

## 1. Data Types :-

* **DATA :-** DATA is set of lines (or) statements, which has some information about something homogeneous issues.

→ The basic form of information handled by a computer are instructions and data.




* **Data types :-**

→ Binary information in digital computers is stored in memory (or) processor registers. Registers contain either data (or) control information.

→ Data are numbers and binary coded information that are operated on to acheeve required.

→ Data types found in registers of digital computers may be classified as being one of following.

* numbers used in Arithematic computations.
* Letters of the alphabets used in data processing.
* Other discrete symbols used for specific purpose.

## Number System :-

→ Number systems are the technique to represent numbers in the computer system architecture, every value that you are saving or getting into/from the computer memory has a defined number system.

→ Computer Architecture support following number systems.

* Binary number system (Base 2).
* Octal number system (Base 8).
* Decimal number system (Base 10).
* Hexa decimal number system (Base 16).

## Binary Number System :-

A Binary number system has only two digits, which are 0 and 1. Every number is represented with 0 and 1 in number system. The Base of Binary numbers is 2.

Eg :- $(101101)_2 = 1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0$

$= 32 + 0 + 8 + 4 + 0 + 1$

$= (45)_{10}$

## Octal number System :-

Octal number system has eight digits i.e (0 to 7), Every number is represented with 0, 1, 2, 3, 4, 5, 6 and 7 in number system. The Base of Octal number is 8.

Eg :- $(736\cdot4)_8 = 7\times8^2+3\times8^1+6\times8^0+4\times8^{-1}$

$= 7\times64 + 3\times8 + 6\times1 + 4/8$

$= (478.5)_{10}$

## Decimal number system :-

Decimal number system has 10 digits i.e (0 to 9). Every number is represented with 0, 1, 2, 3, 4, 6, 6, 7, 8 and 9. In this decimal system the base is 10.

Eg:- $(724.5)_{10} = 7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$

$= 7 \times 100 + 2 \times 10 + 4 \times 1 + 5/10$

$= (724.5)_{10}$

## Hexa-decimal number system :-

Hexa decimal number system has 16 alpha numeric values from 0 to 9 and A to F. Every number is represented with 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. In this system, the base is 16. Here A=10; B=11; C=12; D=13; E=14; F=15.

Eg:- $(FB)_{16} = F \times 16^1 + 3 \times 16^0$

$= 15 \times 16 + 3$

$= (243)_{10}$.

## Conversions :-

The process of converting the numbers of one base to the another base is known as conversion. Basically there are three types of conversions. They are:-

1. Decimal number system to other base

   i.e $(\ )_{10} = (\ )_2 / (\ )_8 / (\ )_{16}$

2. Other system (base) to decimal number system.

   i.e $(\ )_2 / (\ )_8 / (\ )_{16} = (\ )_{10}$

3. Other Base to other Base

   $(\ )_2 / (\ )_8 / (\ )_{16} = (\ )_2 / (\ )_8 / (\ )_{16}$.

## 1- Decimal number to other Base system :-

**Step 1 :-** The seperation of number into integers and fractions.

**Step 2 :-** Divide the number part integer and multiply the fraction successively by R.

→ **Decimal to Binary :-**

* The process of continous division of a number by 2 until zero as result.

Eg :- convert 13 into binary

| | |
|---|---|
| 2 \| 13 | |
| 2 \| 6 - 1 | |
| 2 \| 3 - 0 | ↑ Answer = 1101 |
| 2 \| 1 - 1 | |
| 0 - 1 | |
| → | |

$\therefore (13)_{10} = (1101)_2$

Type-2 :- convert 10.25 into binary

integer part

| | |
|---|---|
| 2 \| 10 | |
| 2 \| 5 - 0 | |
| 2 \| 2 - 1 | |
| 1 - 0 | |

$0.25 \times 2 = 0.50$

$0.50 \times 2 = 1.00$ ↓

$(0.25)_{10} = (0.01)_2$

$(10)_{10} = (1010)_2$

$\therefore (10.25)_{10} = (1010.01)_2$

→ Decimal to octal :-

* The process of continous division of a number by 8 until zero as result.

Eg :- convert 440 into octal

| | |
|---|---|
| 8 | 440 |
| 8 | 55 - 0 |
| 8 | 6 - 7 |
| | 0 - 6 |

↑ Answer = 0670

∴ $(440)_{10} = (670)_8$

→ Decimal to Hexadecimal :-

* This process is same as that of decimal to binary except the base taken 16 instead of 2.

Eg :- convert $(765.245)_{10}$ to hexa decimal

Integer part :-

| | |
|---|---|
| 16 | 765 |
| 16 | 47 - 13 |
| 16 | 2 - 15 |
| | 0 - 2 |

↑

Fractional part :-

0.245 x 16
3.920 x 16
14.720 x 16
11 . 520

↓

∴ $(765.245)_{10} = (2FD.3EB)_{16}$

## 2. Other number system into decimal number system :-

* Multiply each digit by their weights and sum up the values.

### → Binary to Decimal number :-

In this process, we have to multiply each digit by their weights and sum up them.

**Eg :-** Convert $(101101.11)_2$ into decimal.

A) Given number $= (101101.11)_2$

$= 1\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$

$= (1\times32)+0+(1\times8)+(1\times4)+0+(1\times1)+(1\times1/2)+(1\times1/4)$

$= 32+0+8+4+1+0.5+0.25 = (45.75)_{10}$.

### → Octal to Decimal number :-

In this process, we have to follow above process.

**Eg :-** Convert $(53)_8$ into decimal.

A) Given number $= 5\times8^1+3\times8^0$

$= 5\times8+3\times1 = 40+3$

$= (43)_{10}$.

### → Hexa decimal to Decimal number :-

In this process, we have to follow above process.

**Eg :-** Convert $(3F6)_{16}$ into decimal

A) Given number $= 3\times16^2+15\times16^1+6\times16^0$

$= 3\times256+15\times16+6\times1$

$= 768+240+6$

$= (1014)_{10}$

3. Other Base to other Base :-

* There are mainly two methods are there. They are
  (I) Direct method
  (II) Indirect method

→ Direct Method :-

→ Converting Binary to octal by grouping binary bits.

Eg :- Convert 1010111100 int octal

A) Given number = $(1010\,111100)_2$

To convert binary to octal, we have to make a 3-bit groups and convert them directly.

$$\underset{\downarrow}{001}\quad \underset{\downarrow}{010}\quad \underset{\downarrow}{111}\quad \underset{\downarrow}{100}$$
$$1\quad 2\quad 7\quad 4$$

$\Rightarrow (1274)_8$.

→ In direct Method :-

→ Convert the Binary number to decimal and then decimal to octal.

Eg :- Convert $(10110)_2$ into octal

A) Given number = $(10110)_2$

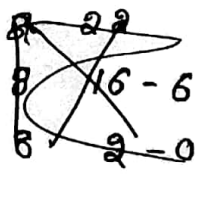
Step 1 :- Convert Binary to decimal

$$= 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$$
$$= 16 + 0 + 4 + 2 + 0$$
$$= (22)_{10}$$

Step 2 :- Convert Decimal to octal

$$= (22)_{10}$$

| 8 | 22 |
|---|---|
| 8 | 2 - 6 |
| | 0 - 2 |




$(22)_{10} = (026)_8$

## 2. Compliments :-

→ Compliments are used in digital computers for simplifying the subtraction operation and for logical manipulation.

→ There are mainly two types of compliments for each base system. They are

(i) r's - compliment

(ii) (r-1)'s - compliment.

→ When the base value r is substituted in the name, the two types are 2's and 1's complement for the Binary and 10's and 9's complement for decimal's.

### r's complement :-

The r's compliment of a n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$.

Eg:- The 10's compliment of 2389 is

$$\begin{array}{r} 9999 \\ -2389 \\ \hline 7610 \end{array} \quad +1 = 7611$$

The 2's compliment of 101100 is

101100 1's C = 010011

2's C = 1's C + 1

$$\Rightarrow \begin{array}{r} 010011 \\ +\quad 1 \\ \hline 010100 \end{array}$$

## (r-1)'s compliment :-

Given a number N having base r and having n digits, the (r-1)'s compliment of N is defined as (r-1)-N.

Eg :- For decimal numbers r=10

$r-1=9 \Rightarrow r=9+1$

So 9's compliment is $(10^n-1)-N$.

Then, 10's compliment $\Rightarrow$ 10's compliment -1

9's C of 546700 $\Rightarrow (10^6-1)-N$

$= (1000000-1)-N$

$=$ 999999
$-$ 546700
453299

## Subtraction with complements :-

→ The subtraction of two n-digit un signed numbers M-N in base r can be done as

1. Add the minued M to r's compliment of subtrahend N.
2. If $M \geq N$, the sum will produce and end carry $r^n$, which can be discarded, what is left is Result.
3. If $M < N$, the sum will not produce and end carry and is equal to $r^n$. To get answer, take the r's compliment of sum and place a negative sign in front.

Eg :- using 10's compliment subtract 3250 - 72532.

A) Here M = 03250 and N = 72532. Here N > M

M = 03250

10's compliment of N = 27468

30718

∴ The answer is -(10's C of 30718) = -69282.

→ Subtraction of unsigned numbers can also be done by means of the (r-1)'s compliment.

→ Remember that (r-1)'s compliment is one less than r's compliment.

Eg: Given two binary numbers x = 1010100 and y = 1000011. Perform the subtraction (a) x-y and (b) y-x using 2's compliment.

A) Given binary numbers are x = 1010100

y = 1000011

(a) 1's compliment of y = 0111100

x = 1010100

10010000

Here carry is added ⇒ 1

0010001

(b) y - x = 1000011 - 1010100

y = 1000011

1's compliment of x = 0101011

1101110

Here there is no end carry, Hence the answer is

y - x = -(1's complimen of 1101110) = -0010001

## 3. Fixed point Numbers :-

→ In binary numbers system, a number can be represented as an integer (or) a fraction.

→ In integer numbers, radix point is fixed and assumed to be to the right most digit.

→ As radix point is fixed, the number system is referred to as fixed point number system.

### Fixed point Representation :-

→ The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are

* A binary point in the extreme left of the register to make the stored number a fraction.
* A binary point in the extreme right of the register to make the stored number an integer.
* In either case, the binary number point is not actually present, but its presence is assummed from the fact the number stored in register.

### Integer Representation :-

→ When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.

→ When the number is negative, the sign is represented by 1 but the rest of numbers will be represented in three possible ways, they are

1. Signed magnitude representation.
2. Signed 1's complement representation.

3. Signed - 2's compliment representation.

Eg:- Consider an 8-bit register and number +14.

→ The way to represent is 00001110

Consider an 8 bit register and number -14

→ In signed-magnitude represented as 1 0001110

In signed - 1's compliment as 1 1110001

In signed - 2's compliment as 11110010

Arithematic addition :-

→ The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmatic.

→ Addition of two signed-magnitude numbers follow normal rules.

* If have same sign, add two numbers, use same sign.

* If they have different sign means follow the heighest number sign and assign same sign.

* Must compare the signs and magnitudes and then either add (or) subtract.

→ Addition of two signed 2's compliment number, it doesn't require a comparision. (or) subtraction, only addition and complementation.

* Add two numbers including their sign bit.

* Discard any carry out of sign bit position.

* All Negative numbers must be in 2's complement.

* If the sum obtained is negative, then it is in

2's compliment form.

Eg :- (i) +6 0000 0110 (ii) -6 1 1111010
+13 0000 1101 +13 0000 1101
+19 0001 0011 +7 0000 0111

(iii) +6 0000 0110 (iv) -6 1111 1010
-13 1111 0011 -13 1111 0011
-7 1111 1001 -19 1110 1101

→ In each of four cases, the operation performed is always addition, including the sign bit, any carry out the sign bit is discarded, and negative results are in 2's compliment form.

## Arithematic Subtraction :-

→ The subtraction of the two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated as follows: Take the 2's complement of the subtrahend and add is minuend.

→ A carry out of sign bit position is discarded.

→ This procedure steams from the fact that a subtraction operation can be changed into an other addition operation, if the sign of subtrahend is changed. These are demonstrated by following relations.

$$(\pm A) - (+B) = (\pm A) + (-B)$$

$$(\pm A) - (-B) = (\pm A) + (+B)$$

Eg:- Consider the subtraction of

$(-6) - (-13) = +7$

$-6 = 10000010$

$-13 = 10001101$ [where 13 will be 2's of 13]

$+ \quad {}^{1}{}^{1}$

$= 100010111$ [here 1 is carry]

$= 00010111$ [remove carry]

$= (+7)$.

## Decimal Fixed-point Representation:-

→ The representation of decimal numbers in registers is a function of binary code used to represent a decimal digit.

→ This takes much more space than the equivalent binary representation and the circuits required to perform decimal arithmetic are more complex.

→ Representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary.

→ To obtain the 10's compliment of a BCD number, first take the 9's compliment and then add one to least significant digit.

Eg:- $(+375) + (-240) = +135$

0375 = (0000 0011 0111 0101) BCD

9760 = (1001 0111 0110 0000) BCD

0135 = (0000 0001 0011 0101) BCD

## Floating-point representation :-

→ The floating-point representation of a number has two parts. They are :- (i) mantissa
(ii) Exponent

→ The first part represents a fixed point number, it is called mantissa. The second part designates the position of decimal point and is called exponent.

Eg :-

| Fraction | Exponent |
|---|---|
| + 0.6132789 | +04 |

↑ + 6132.789 (represented) above.

→ +04 represents is an actual decimal position of the decimal point here the 4 position to right that is indicated decimal point in the fraction as

$$0.6132789 \times 10^4 = 6132.789$$

→ The floating point is always represent as

$$m \times r^e$$

here m = mantisa, e = exponent, r = radix

→ For a binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as

| Fraction | Exponent |
|---|---|
| 01001110 | 000100 |

→ For Binary sign doesn't shown. The exponent has the equivalent binary number +4. The floating point is $m \times 2^e = +(.1001110)_2 \times 2^{+4}$

## Normalization :-

→ A floating-point number is said to be normalized if the most significant digit of mantissa is non-zero.

→ The number is normalized only if its left most digit is non-zero.

Eg :- Let us take 8-bit binary number 00011010 is not normalized because of three leading 0's. The number can be normalized by shifting it three positions towards left and discarding leading 0's.

So, the result will be 11010000.

→ Two main standard forms of floating-point numbers are from the following organisations that decide standards are :

(i) ANSI (American National Standards Institute)

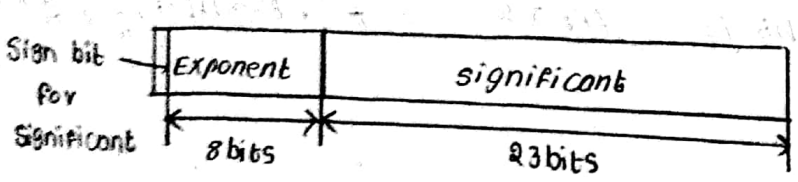
(ii) IEEE (Institute of Electrical and Electronic Engineering).

→ 32 bit floating point format.

→ left most bit = Sign bit.

→ Exponent in the next 8 bits. use a biased way of representation.

→ Final portion of word is significant.



## 4. Other Binary Codes :-

Other Binary Codes for decimal numbers and the alphanumeric characters are sometimes used. Digital computers are also employ other binary codes. They are:-

### 1. Gray Code :-

The reflected binary code (or) Gray code is an odering of two binary number system, such that two successive values differ by one bit only.

Gray codes are also useful in normal sequence of binary numbers generated by hardware.

→ Gray code is also known as reflected binary code. The Gray code is not suitable for arithematic operations.

| Decimal | Binary | Gray code |
|---|---|---|
| 0 | 000 | 000 |
| 1 | 001 | 001 |
| 2 | 010 | 011 |
| 3 | 011 | 010 |
| 4 | 100 | 110 |
| 5 | 101 | 111 |
| 6 | 110 | 101 |
| 7 | 111 | 100 |

→ Convert $(010)_2$ to Gray code.

A) 0 1 0
↓ ↗↓ ↗↓
0 1 1

Hence $(010)_2 = (011)_{GC}$

## 2. BCD (or) Binary Coded Decimal :-

→ In this method, each decimal digit is represented by a 4-bit binary number.

→ BCD is the way to express each of decimal number with binary number.

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

Eg :- Convert $(11101)_2$ to BCD.

A) First convert it into decimal

$\Rightarrow 1\times2^0+0\times2^1+1\times2^2+1\times2^3+1\times2^4$

$\Rightarrow 1+0+4+8+16$

$= (29)_{10}$

$\therefore (11101)_2 = (00101001)_{BCD}$.

## 3. Excess-3 Code (or) x33 Code :-

→ In this method, we have to add 3 to the decimal number and then we convert into BCD code.

Eg :- Convert $(426)_{10}$ to excess-3 code.

A) Each digit is added by 3 and it is converted to BCD.

$$\begin{array}{r} 426 \\ 333 \\ \hline 759 \end{array} \Rightarrow (0111\ 0101\ 1001)_{x53}.$$

4. ASC-II Code :-

It is abbreviated as American Standard Code for information Interchange, is a character encoding standard for electronic communication.

ASCII codes represents text in computers and tele-communications and other devices.

5. Error Detection code :-

→ whenever a message is transmitted, it may get scrambled by noise (or) data may corrupted.

→ To avoid this, we use error detecting codes, which are additional data added to a given message.

→ A simple example for error detection code is parity check.

Error correcting code :-

→ Along with error-detecting code, we can also pass some data to figure out original message from the received message.

→ This type of code is called Error-correcting code.

Hamming code :-

→ Hamming code is a block code that is capable of the detecting up to two simultaneous bit errors and the correcting single-bit errors.

→ It was developed by R.W Hamming for the error correction, In this coding method, the source encodes that message by inserting redundant bits within the message.

## Register Transfer language :-

→ **Micro operations** :- A micro operation is an elementary operation performed on data stored in one or more Registers.

Eg :- shift, count, clear and load.

→ The internal hardware organisation of a digital computers is best defined by spacifying :

1. The set of registers it contains and their function.
2. The sequence of micro operations performed on data stored in registers.
3. The control that initiates the sequence of the micro operations.

→ **Register Transfer language** :-

→ Micro operations can be explained in terms of words but it is a lengthy procedure, then we use the symbols to represent information transfer between them.

→ The symbolic notation used to discribe the micro operation transfer register is called as "Register Transfer language".

→ The term "register transfer" implies that the availability of hardware logic circuits that can perform a stated micro operation.

→ The term "language" implies programming language that specifies the writing of symbols to specify a given computational process.

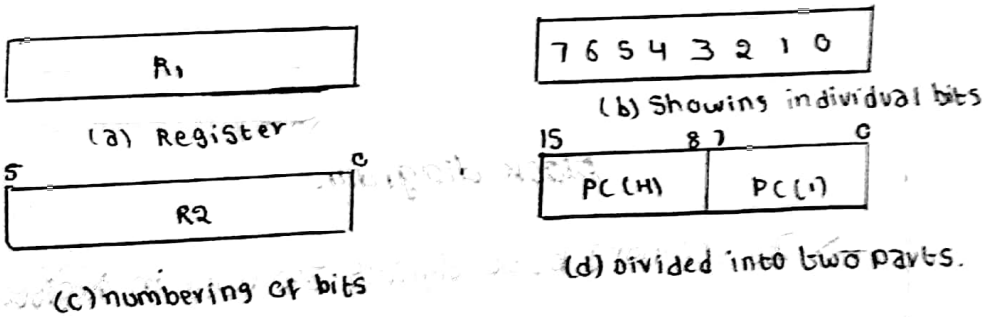
→ Register transfer language is an appropriate tool for designing the internal organisation of digital computers.

## Register TRANSFER :-

### Registers :-

→ Computer Registers are designated by capital letters to denote function of Register.

→ Eg :- MAR (Memory Address Register), PC (program counter), IR (Instruction Register), R1 (process Register).




Block diagram.

### Register Transfer :-

→ Information transfers from one Register to another Register. It is designated in symbolic form by means of Replacement operator.

$$R_2 \leftarrow R_1$$

→ If we want the transfer to occur under a predefine condition, this can be shown by if-then statement.

$$if\ (P=1)\ then\ (R_2 \leftarrow R_1)$$

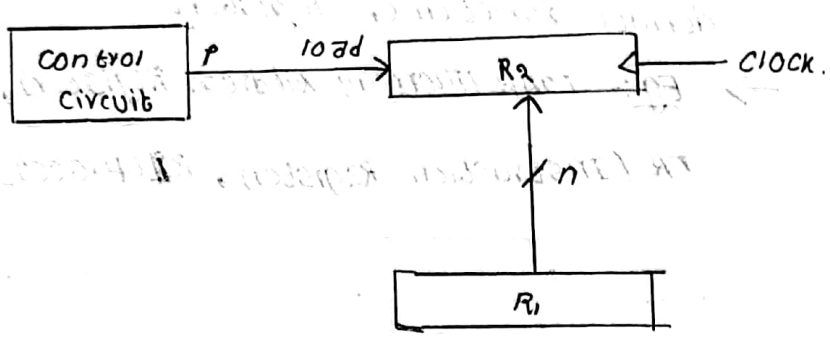
↓

Control signal.

→ This statement can also be written by using a control function as follows. A control statement is a Boolean variable that is equal to 1 (or) 0.

$$P : R_2 \leftarrow R_1$$

→ The block diagram for the register transfer from $R_1$ to $R_2$ is shown below. The n outputs of register $R_1$ is connected to n inputs of register $R_1$. Here the letter n is used to indicate number of bits for the Register.




block diagram.

→ The following basic signals are used in Register transfer function.

| Symbol | Description | Examples |
|---|---|---|
| letters and numerals | Denotes a Register | MAR, $R_2$ |
| parantheses () | Denote a part of register | $R_2(0-7)$, $R_2(L)$. |
| Arrow ← | Denotes transfer of information | $R_2 \leftarrow R_1$ |
| comma , | Seperation | $R_3 \leftarrow R_1, R_2(L)$. |

## BUS AND Memory :-

→ A computer has many registers and paths must be provided to transfer information from one register to another. The number of wires will be more if seperate lines are used between each register and all other registers.

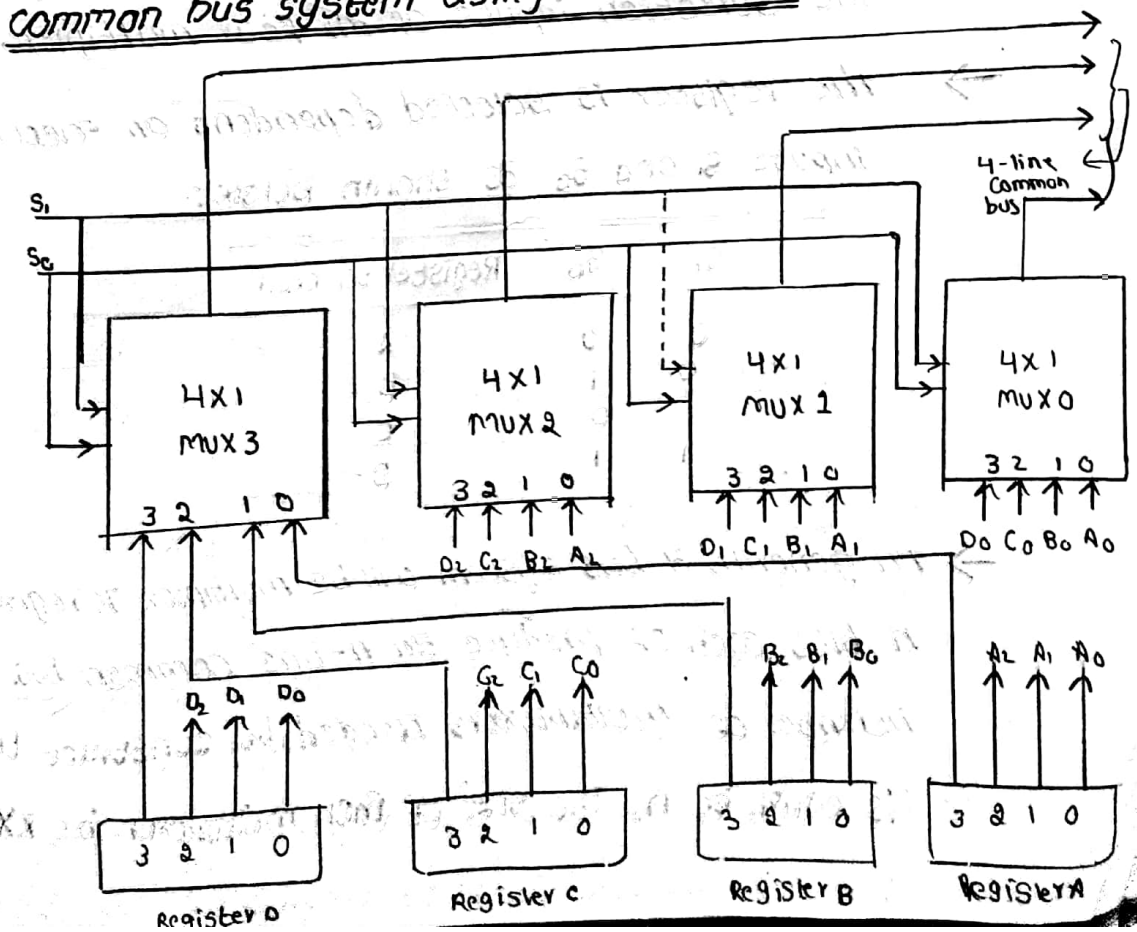
→ A more efficient scheme for transferring information between registers is a common bus system. A bus has set of common lines, one for each bit of register.

→ control signals determine which registers is selected by bus.

→ The common bus system is implemented in 2 ways:-

1) using multiplexers
2) using three-state bus buffers.

## common bus system using multiplexers :-



→ The construction of bus system for four registers is as shown in figure above.

→ Each register has four bits, numbered 0 through 3.

→ The bus consists of four 4x1 multiplexers each having four data inputs, 0 through 3, and two selection input $S_1$ and $S_0$.

→ All the 0 bits of the registers (i.e $A_0$, $B_0$, $C_0$ and $D_0$) are connected to the MUX 0. All the 1 bits of the registers are connected to MUX 1.

→ MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexers the four 1 bits of the registers, and similarity the other two.

→ The two selections lines $S_1$ and $S_0$ are connected to the selection inputs of all four multiplexers.

→ The register is selected dependent on selection inputs $S_1$ and $S_0$ as shown below:

| $S_1$ | $S_0$ | Register selected |
|---|---|---|
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

→ In general, a bus system will be multiplex k registers of n bits each of produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, The size of each multiplexer be kX1.

→ The data present in the bus can be transferred to one (or) more destination registers by connecting the inputs of the register to bus line and enabling the load control of the destination Register.

→ The symbolic statement for bus transfer is shown below : $BUS \leftarrow C$ , $R_1 \leftarrow BUS$

→ The contents of register C are placed on the bus, and the content of bus is loaded into register $R_1$ by activating load control input.

$$R_1 \leftarrow C .$$

## Three State Bus Buffers :-

→ A bus system can be constructed with three-state gates instead of multiplexers.

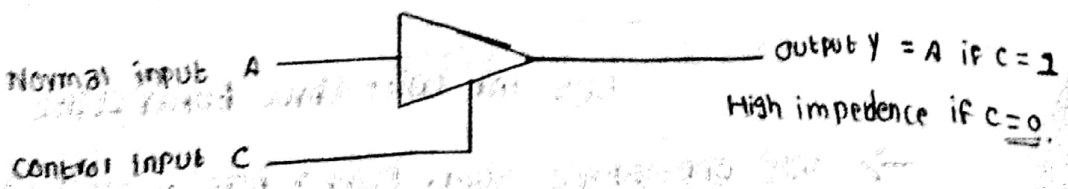
→ A three-stage gate is a digital circuit that can have 3 output state.

→ Two of the states are logic 0 and 1 as in normal gate and third state is called high impedence state.

→ High impedence state acts like an open circuit which means the output is disconnected.

→ The most commonly used gate in construction of bus system in three-state bus buffer.

→ The graphic symbol of this three-state Buffer gate is as shown below :



→ It has both normal input and control input.

→ The control input determines the output state.

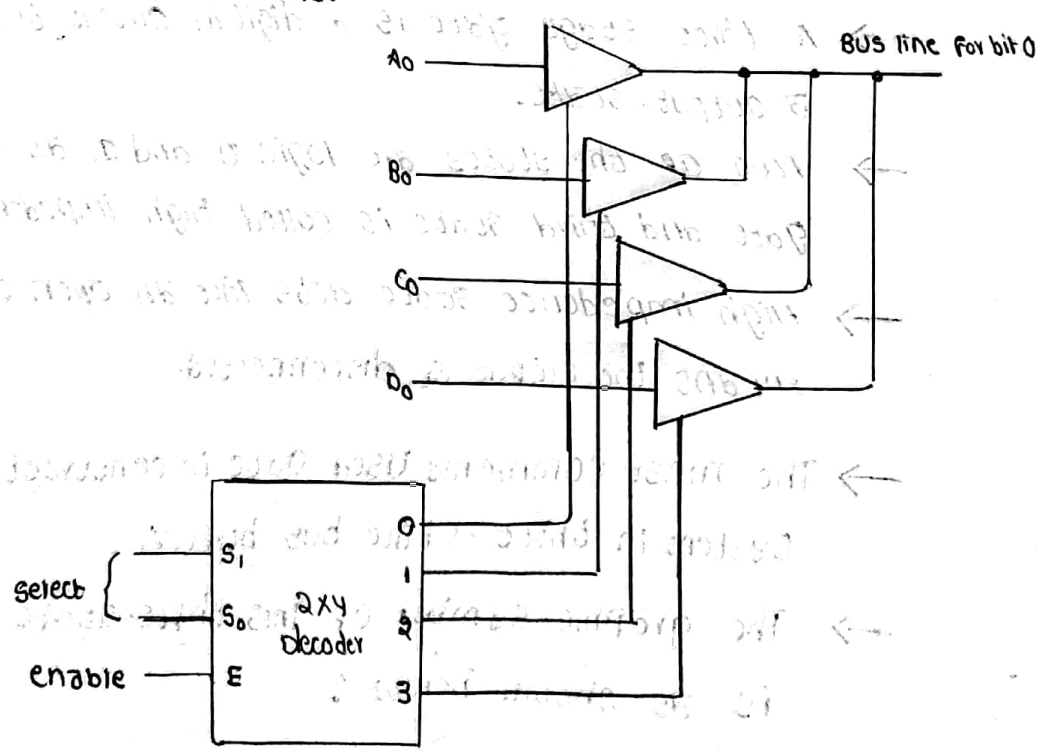
→ When the control input is equal to 1, the output is enabled and gate behaves as normal buffers.

→ When the control input is 0, then, the output is disabled and gate goes to a high impedence state.

→ The construction of a bus system with three-state buffers is demonstrated in following figures.

→ The output of four buffers are connected together to form a single bus line.

→ No more than one buffer may be in active state at a given time.




Bus line with three Buffer state

→ Only one three-state buffer has access to the bus line while all other buffers are maintained in high-impedence

State.

→ One way to make no more than one control input is active at any given time is to use a decoder, as shown in figure.

→ When the enable input of decorder is 0, all buffers are disabled.

→ When the enable input is active, one of three-state buffers will be active, depending on the binary value of the selection inputs of decoder ($S_1$ & $S_0$).

**Memory transfer :-**

→ The transfer of information from a memory word to the outside environment is called a read operation.

→ The transfer of new information to be stored in the memory is called write operation.

→ A memory word is indicated by letter M. It is necessary to specify the address of M while doing the memory transfer operation.

*** Read operation :-**

Read : DR ← M[AR]

→ This causes a transfer of information into DR (data Register) from the memory word M, selected by the address in address in AR.

*** write operation :-**

write : M[AR] ← $R_1$

→ This causes a transfer of information from $R_1$ in to the memory word selected by address in AR.

## Arithemetic micro operations :-

→ A micro operation is an elementary operation, that performed with data stored in registers. The micro operations in computers are classified as

1. **Register transfer micro operations :-**

It transfers binary information from one register to another.

2. **Arithematic micro operations :-**

It performs arithematic operations on data stored in registers.

3. **logic micro operations :-**

It performs bit manupulation operations.

4. **Shift micro operations :-**

It performs shift operation on the data stored in the registers.

→ The arithe matic micro operations is defined as

**Add micro operation :-**

It states that contencts of Register 1 are added to contents of Register R2. Sum transferred to Register $R_3$.

$$R_3 \leftarrow R_1 + R_2 .$$

**Subtraction of micro operations:-**

→ subtraction is most often implemented through the complementation and addition, instead of writing the minus operation, we can speciey it as

$$R_3 \leftarrow R_1 + \overline{R_2} + 1$$

→ In above statements $\overline{R_2}$ is symbol for 1's complement of $R_2$. Adding 1 to the 1's complement gives us a 2's complement. Hence it is equal to $R_1 - R_2$.

→ The following table shows arithematic micro operations.

| Symbolic designation | Description |
|---|---|
| $R_3 \leftarrow R_1 + R_2$ | contents of $R_1$ plus $R_2$ transferred to $R_3$. |
| $R_3 \leftarrow R_1 - R_2$ | contents of $R_1$ minus $R_2$ transfer to $R_3$. |
| $R_2 \leftarrow \overline{R_2}$ | complement the contents of $R_2$ (1's). |
| $R_2 \leftarrow \overline{R_2} + 1$ | 2's complement of contents of $R_2$. |
| $R_3 \leftarrow R_1 + \overline{R_2} + 1$ | $R_1$ plus 2's complement of $R_2$ = $R_1 - R_2$. |
| $R_1 \leftarrow R_1 + 1$ | Increment the $R_1$ contents by 1. |
| $R_1 \leftarrow R_1 - 1$ | Decrement the $R_1$ contents by 1. |

## Binary Adder :-

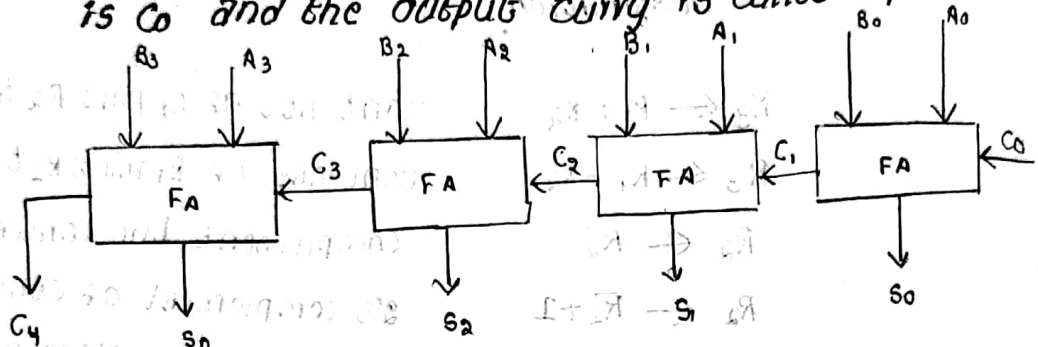
→ To implement add micro operation with hardware, we need the registers that holds the data and the digital component that performs the arithmetic addition.

→ The digital circuit that forms the arithmatic sum of the two bits and a previous carry is called a full-adder.

→ The digit circuit that generates the arithmetic sum of the binary numbers of any length is called a binary adder.

→ The binary adder is constructed with full-adder circuits connected to the input carry of the next full adder.

→ The following figure shows interconnection of four full-adders (FA) to provide a 4-bit binary adder. The input carry to the binary adder is $C_0$ and the output carry is called $C_4$.

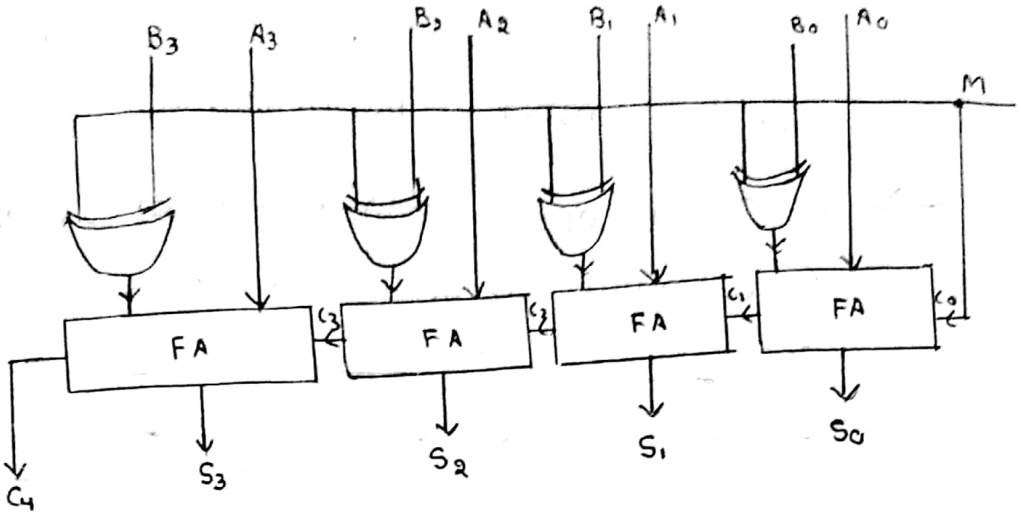



→ An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-heigher order full adder.

## Binary Adder-Subtractor :-

→ The subtraction of binary numbers can be done using complements. Remember subtraction A-B can be done by taking 2's complement of B and adding it to A.

→ The addition and subtraction operation can be combined into one common circuit by including an ex-OR gate with each full adder. A 4-bit adder subtractor circuit is as shown in figure.




4-bit adder-subtractor.

→ The mode input controls the operation, when M=0, the circuit is an adder and when M=1, the circuit will becomes subtractor.

→ Each EX-OR gate recieves input M and output which is one of them is B.

→ when M=0, we have B (xor) 0 = B. The full adders receive the value of B the input carry is 0, then the circuit performs A+B.

→ when M=1, we have B (XOR) 1 = B. The $C_0$=1, and B inputs are complemented and 1 is added through input carry. The circuit performs the 2's complement of B and then added to A. The Result is A-B.

## Binary incrementer :-

→ The increment operation adds one to a number in a register.

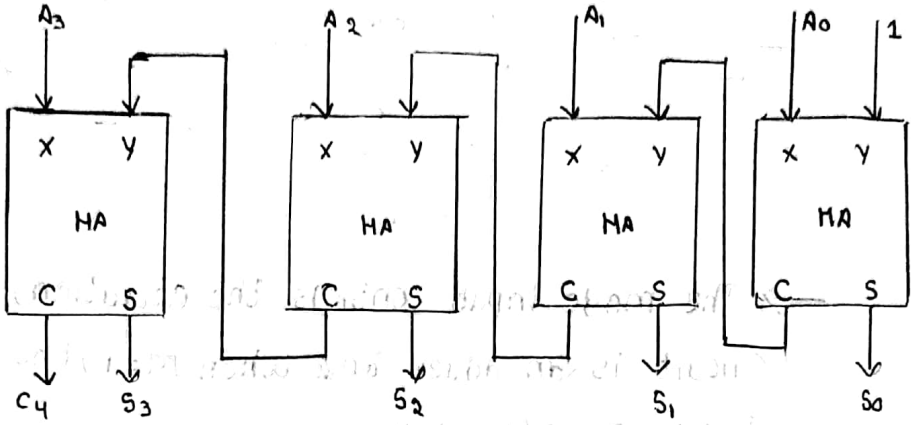
Eg :- Let a Binary number ⇒ 0110

after increment ⇒ 0111.

→ The diagram of a 4-bit combinational circuit incrementer is shown below.

→ one of the inputs to the least significant bit of the one number to be incremented.

→ The output carry from one half-adder is connected to one of inputs to the next higher order half-adder.




4-bit binary incrementer.

→ The circuit can be extended to an n-bit binary incrementer by extending the diagram by adding n-half adders.

*Arithmetic Circuit :-*

→ The arithmetic operations listed in above table (addition, subtraction, increment and decrement) can be implemented in one composite arithemetic circuit.

→ The basic component of the arithmatic circuit is the parallel adder. By controlling the data inputs to the adder.

→ The diagram for 4-bit arithmetic circuit is as shown in figure below. It has 4 fun adder circuits that has 4-bit adder.

→ There are two four bit inputs A and B and a four-bit output D.

→ The four inputs from A go directly to the X inputs of the binary adder.

→ The four inputs from B are connected to the data inputs of multiplexers.

→ The multiplexer's data inputs also receive the complement of B. The other two data inputs are connected to logic 0 (or) logic 1.

→ The four multiplexers are controlled by two selection inputs $S_1$ and $S_0$.

→ The input carry $C_{in}$ goes to carry input of the full adder.

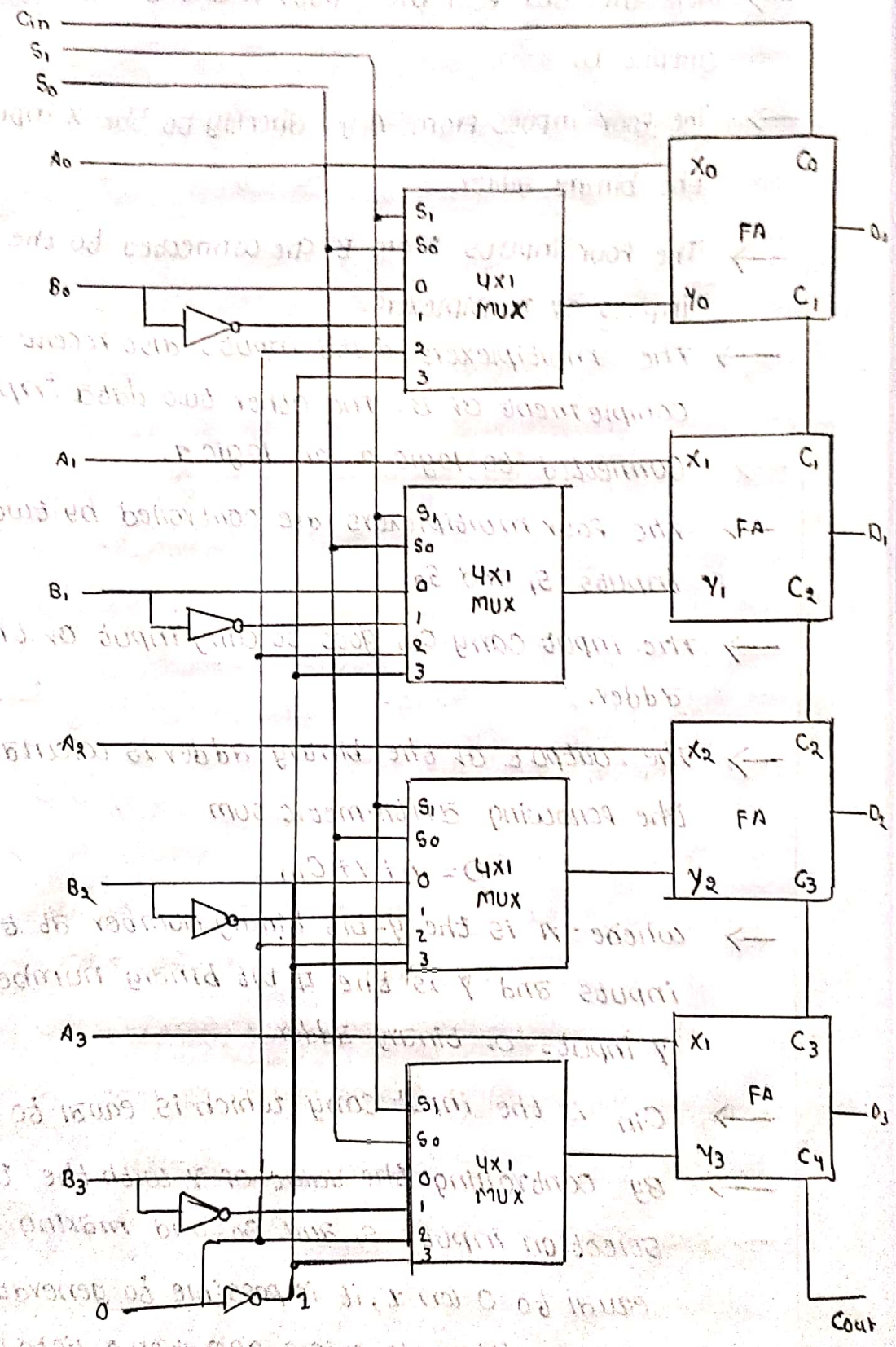
→ The output of the binary adder is calculated from the following arithemetic sum

$$D = X + Y + C_{in}.$$

→ where X is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of binary adder.

→ $C_{in}$ is the input carry which is equal to 0 (or) 1.

→ By controlling the value of Y with the two selection inputs $S_1$ and $S_0$ and making $C_{in}$ equal to 0 (or) 1, it is possible to generate the eight arithmetic micro operations listed in below table.




4-bit arithmetic circuit.

| Select | | | Input | Output | Microoperation |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $C_{in}$ | Y | $D = A + Y + C_{in}$ | |
| 0 | 0 | 0 | B | $D = A + B$ | Add |
| 0 | 0 | 1 | B | $D = A + B + 1$ | Add with carry |
| 0 | 1 | 0 | $\overline{B}$ | $D = A + \overline{B}$ | Subtract with borrow |
| 0 | 1 | 1 | $\overline{B}$ | $D = A + \overline{B} + 1$ | Subtract |
| 1 | 0 | 0 | 0 | $D = A$ | Transfer A |
| 1 | 0 | 1 | 0 | $D = A + 1$ | Increment A |
| 1 | 1 | 0 | 1 | $D = A - 1$ | Decrement A |
| 1 | 1 | 1 | 1 | $D = A$ | Transfer A |

→ **Addition** :- When $S_1 S_0 = 00$, the value of B is applied to the Y inputs of the adder. If $C_{in} = 0$, the output $D = A + B$. If $C_{in} = 1$, output $\Rightarrow D = A + B + 1$. Both cases perform add operations.

→ **Subtraction** :- When $S_1 S_0 = 01$, the complement of B is applied to the Y inputs of the adder. If $C_{in} = 1$, the output $D = A + B' + 1$. This produces the A + 2's C of B. which is equal to $A - B$. When $C_{in} = 0$, then $D = A + B'$.

→ **Increment** :- When $S_1 S_0 = 10$, the inputs from B are neglected and instead, all 0's are inserted into Y inputs. The outputs becomes $D = A + 0 + C_{in}$. This gives $D = A$ when $C_{in} = 0$ and $D = A + 1$ when $C_{in} = 1$. In this first case we have direct transfer of input A to output D.

→ **Decrement :-** When $S_1S_0 = 11$, all 1's are inserted in to the Y inputs to the adder to produce the decrement operation $D = A-1$, when $C_{in} = 0$. This is because a number with all 1's equal to 2's compliment of 1.

## Logic micro operations :-

→ logic micro operations specify micro binary operations for strings of bits stored in registers. These operations are considered each bit of the register separately and treated them as binary.

→ *For Eg :-* The exclusive-OR micro operation with the contents of two register $R_1$ and $R_2$ is symbolized by statements

$$P: R_1 \leftarrow R_1 \oplus R_2$$

→ It specifies a logic micro operation to be executed on the individual bits of registers. As numeric example, assume each register has 4 bits.

→ **Special Symbols :-**

The symbol $\vee$ is used to denote an OR operation and the symbol $\wedge$ is used to denote an AND operation. The compliment micro operation is same as 1's compliment and uses a bar on top of the register name.

## list of logic micro operations :-

→ There are 16 different operations that can be performed with two binary variables. They can be determined from all possibilities truth table obtained with two binary variables as $F_0$ to $F_{15}$.

| x | y | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Truth table for 16 functions of 2 variables.

→ The 16 Boolean functions of two variables x and y are expressed in the algebric form in the first column of table below, The relationship between two binary contents of two registers A and B. Name of the micro operation is in the third column.

| Boolean function | Microoperation | Name |
|---|---|---|
| $F_0 = 0$ | $F \leftarrow 0$ | Clear |
| $F_1 = xy$ | $F \leftarrow A \wedge B$ | AND |
| $F_2 = xy'$ | $F \leftarrow A \wedge \bar{B}$ | |
| $F_3 = x$ | $F \leftarrow A$ | Transfer A |
| $F_4 = x'y$ | $F \leftarrow \bar{A} \wedge B$ | |
| $F_5 = y$ | $F \leftarrow B$ | Transfer B |
| $F_6 = x \oplus y$ | $F \leftarrow A \oplus B$ | Exclusive-OR |
| $F_7 = x + y$ | $F \leftarrow A \vee B$ | OR |
| $F_8 = (x+y)'$ | $F \leftarrow \overline{A \vee B}$ | NOR |
| $F_9 = (x \oplus y)'$ | $F \leftarrow \overline{A \oplus B}$ | Exclusive-NOR |
| $F_{10} = y'$ | $F \leftarrow \bar{B}$ | Complement B |

| | | | |
|---|---|---|---|
| $F_{11} = x + y'$ | $F \leftarrow A \vee \overline{B}$ | |
| $F_{12} = x'$ | $F \leftarrow \overline{A}$ | complement A |
| $F_{13} = x' + y$ | $F \leftarrow \overline{A} \vee B$ | |
| $F_{14} = (xy)'$ | $F \leftarrow \overline{A \wedge B}$ | NAND |
| $F_{15} = 1$ | $F \leftarrow$ all 1's | set to all 1's. |

## Hardware Implementation :-

→ The hardware implementation of logic micro operations requires that logic gates be inserted for each bit (or) pair of bits in registers.

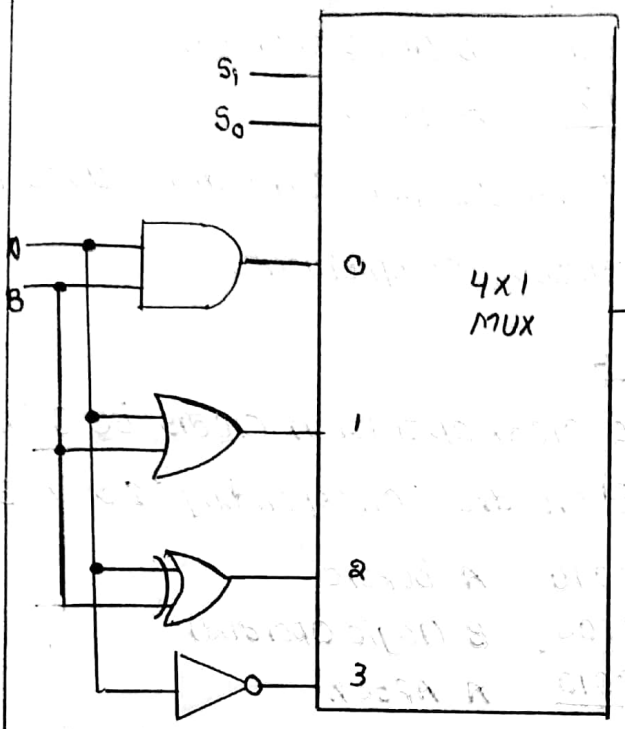
→ Although there are 16 logic micro operations, most computers use only four - AND, OR, XOR and complement from which all others can be derived.

→ Figure shows one stage of circuit that generates the four basic logic micro operations, It has four gates and a multiplexer.

→ Each of the four logic operations is generated through a gate that perform required logic.

→ The outputs of the gates are applied to the data inputs of multiplexers.

→ The two selection inputs choose one of the data inputs of the multiplexer and direct its value to the output.




(a) logic diagram

| $S_1$ | $S_0$ | output | operation |
|---|---|---|---|
| 0 | 0 | $E = A \wedge B$ | AND |
| 0 | 1 | $E = A \vee B$ | OR |
| 1 | 0 | $E = A \oplus B$ | XOR |
| 1 | 1 | $E = \overline{A}$ | complement |

Function table

## Some Applications :-

1. **Selective-set :**

→ The selective set operation sets the bits in the Register A to 1, where there are corresponding 1's in the register B. It does not effect bit position that have 0's in B.

Eg :-

```
  1010    A before
  1100    B (logic operand)
 ------
  1110    A after.
 ------
```

→ "OR" microoperation can be used for performing selective-set.

2. **Selective complement :-**

→ The selective complement operation complements bits in A where there are corresponding 1's in B. It doesnot effect bit positions that have 0's in B.

Eg :- 1010 A before
1100 B (logic operand)
0110 A after.

→ "Exclusive OR" micro operation can be used for performing selective complement.

3. Selective clear :-

→ The selective clear operation clears to 0 the bits in A where there are corresponding 1's in B.

Eg :- 1010 A before
1100 B (logic operand)
0010 A After.

→ The corresponding microoperations is $A \leftarrow A \wedge \bar{B}$.

4. Mask :-

→ The mask operation is similar to selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B.

Eg :- 1010 A before
1100 B (logic operand)
1000 A after masking

→ The mask operation is "AND" micro operation.

5. Clear :-

→ The clear operation compares the word in A and B and produces an all 0's result if the two numbers are equal. This is "Exclusive OR" micro operation.

Eg :- 1010 A
1010 B
0000 $A \oplus B$

## Shift Micro Operations :-

→ Shift micro operations are used to shift the contents of register to the left (or) right shift microoperations. They are used for serial transfer of data.

→ There are three types of shifts :

1. logical shift
2. circular shift
3. Arithematic shift.

→ The symbolic notation for shift micro operations are :-

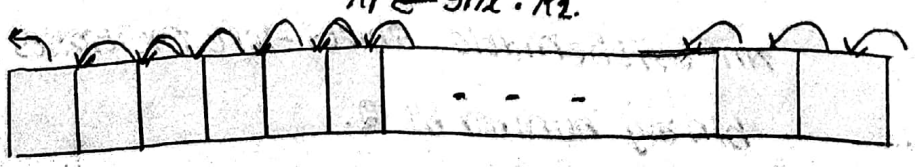
| Symbolic designation | description |
|---|---|
| R ← shl R | Shift-left register R. |
| R ← shr R | Shift-right register R. |
| R ← cil R | Circular left-shift register R |
| R ← cir R | Circular right-shift Register R. |
| R ← ashl R | Arithematic shift-left R |
| R ← ashr R | Arithematic shift-right R |

## logical Shift :-

→ A logical shift is one that transfer 0 through the serial input.

→ Logical shift (shl) :- Specify one bit shift to the left of content of register $R_1$.

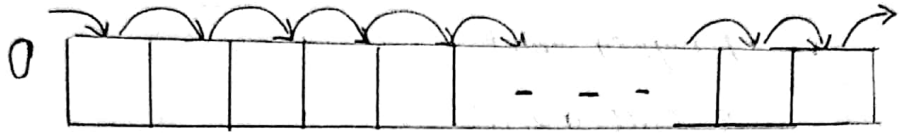
$R_1 \leftarrow shl \cdot R_1$.



→ logical shift right (Shr) :-

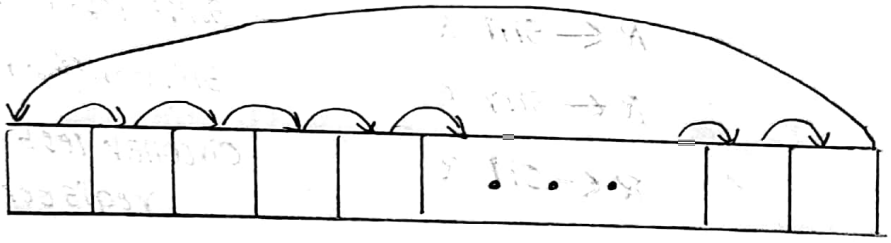
→ Specify one bit shift to the right of content of register $R_2$.

$R_2 \leftarrow$ Shr $R_2$

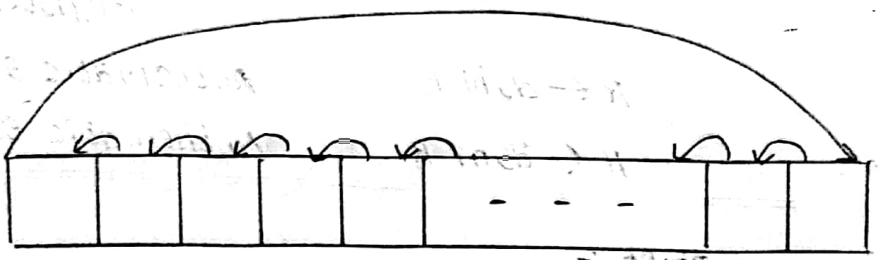



Circular shift :-

→ The circular shift circulates the bits of the register around the two ends without loss of information. We will use the symbol Cil and Cir for circular left shift and right shift.



Circular right shift.



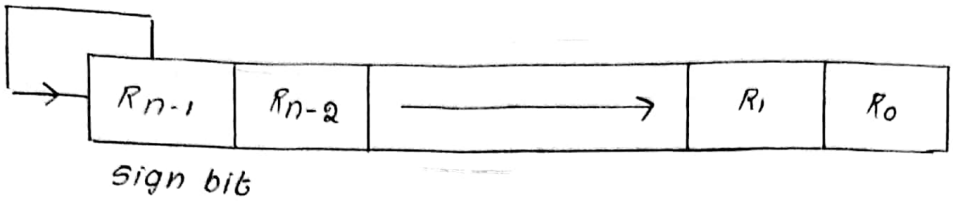
Circular left Shift.

Arithematic Shift :-

→ An Arithematic shift is a micro operation that shifts a signed binary number to the left (or) right. An arithematic shifts - left multiplies a signed binary number by 2.

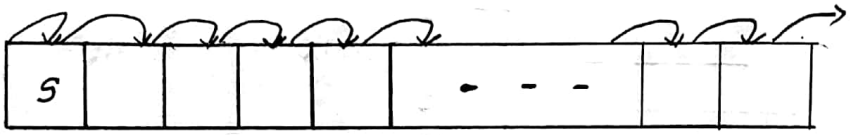
→ Arithematic shifts must leave the sign bit unchanged.

→ The left most bit in the register holds the sign bit and the remaining bits hold the numbey.




**Arithematic Shift right:-**

→ The arithematic shift-right leaves the sign bit as unchanged. Thus $R_{n-1}$ remains same as $R_{n-2}$ receives the bit from $R_{n-1}$. The bit $R_0$ is lost




Arithematic right shift.

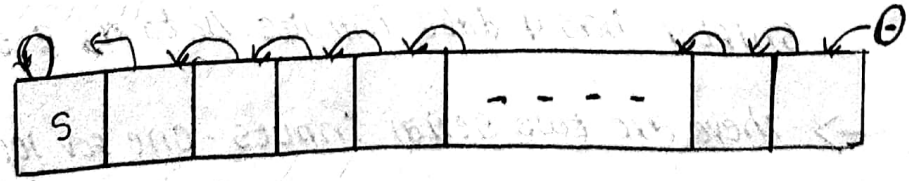
**Arithematic Shift-left:-**

→ The arithematic shift-left inserts a 0 in to $R_0$ and shifts all other bits to the left. $R_{n-1}$ is replaced by $R_{n-2}$. A sign reversal occur if the bit in $R_{n-1}$ changes.

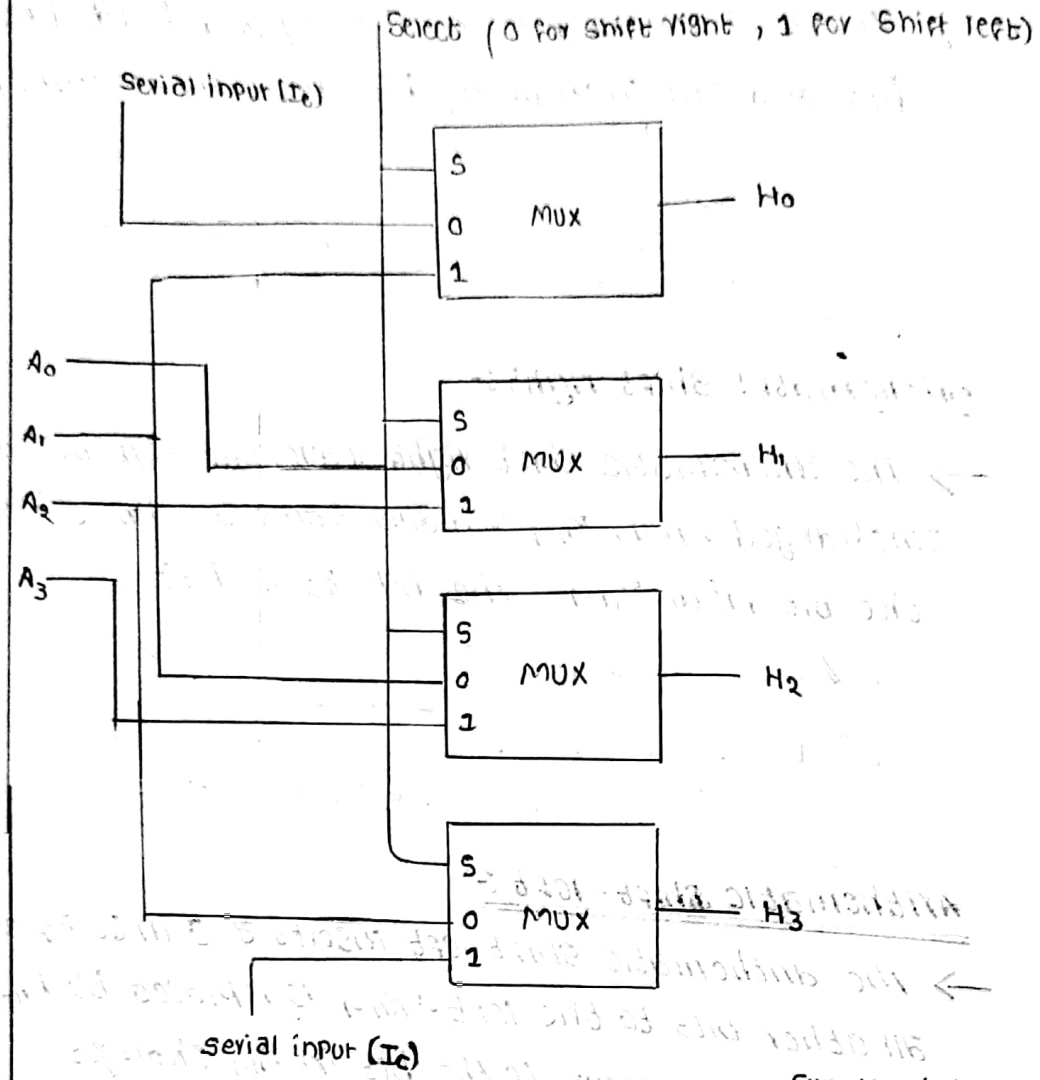
→ An overflow occurs if the bit in sign bit position ($R_{n-1}$) after performing shift operation doesn't match with previous value.

→ An overflow flipflop $V_s$ can be used to detect the arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$



## Hard ware Implementation :-




4-bit Combinational circuit shifter

Function table

| Select | Output | | | |
|---|---|---|---|---|
| S | $H_0$ | $H_1$ | $H_2$ | $H_3$ |
| 0 | $I_R$ | $A_0$ | $A_1$ | $A_2$ |
| 1 | $A_1$ | $A_2$ | $A_3$ | $I_L$ |

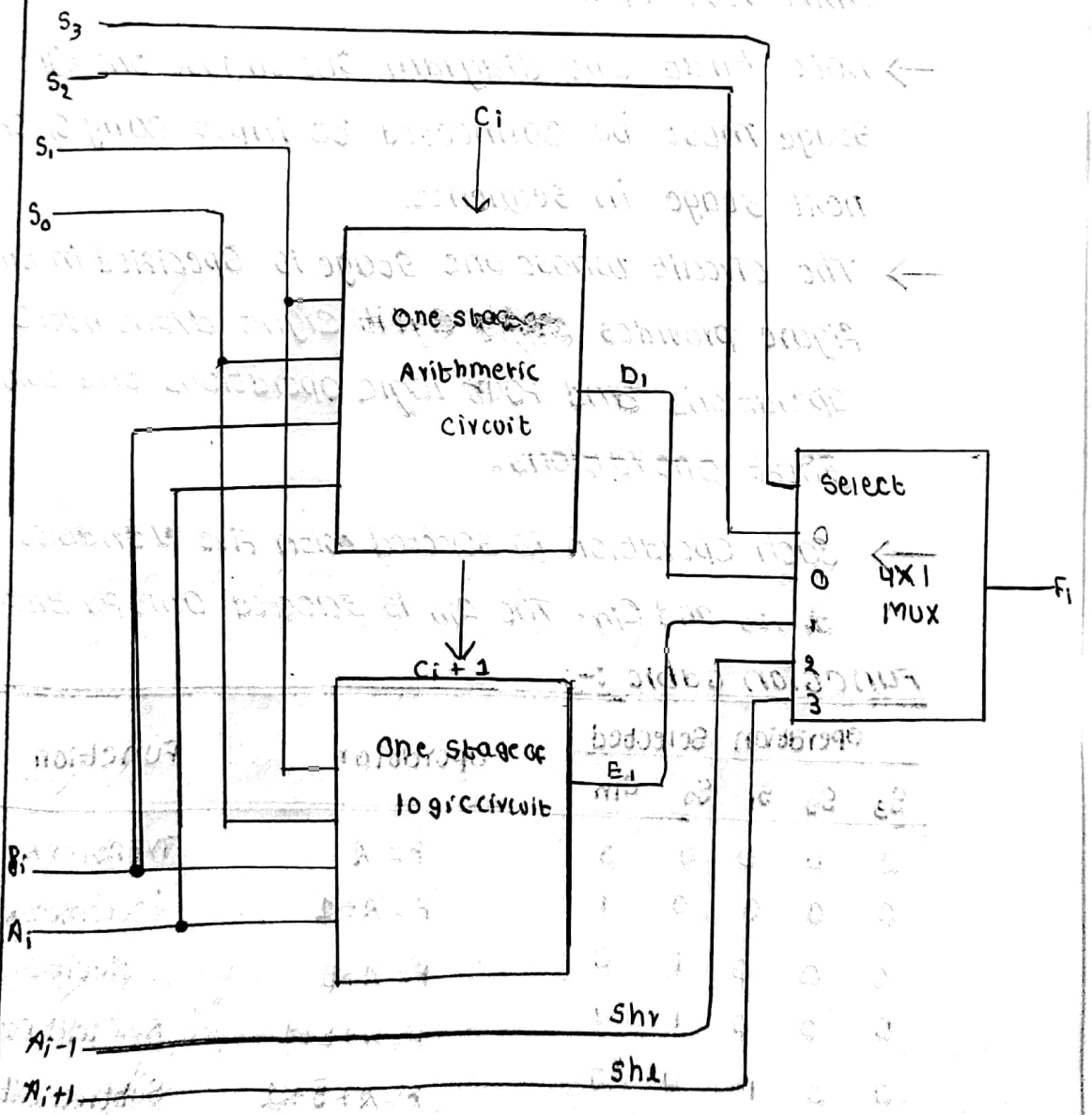
→ A combinational circuit shifter can be constructed with multiplexers as shown in figure. The 4-bit shifter has 4-data inputs $A_0$ to $A_3$ & outputs $H_0$ to $H_3$.

→ There are two serial inputs. One for left shift and another for right shift.

→ When the selection input s=0, the data are shifted right. When s=1, the data are shifted left.

## Arithmetic logic shift unit :-

→ The arithmetic logic and shift units can be combined in to one ALU with common selection variables. One stage of ALU is as shown in below.




→ Inputs $A_i$ and $B_i$ are placed / applied to both the arithmetic and logic units. A particular micro operation is selected with input $S_1$ and $S_0$.

→ A 4x1 multiplexer at the output chooses between

an arithmatic output in $E_i$ and logic output in $H_i$. The data in multiplexer are selected with input $S_3$ and $S_2$.

→ The other two data inputs to the multiplexer receive inputs $A_{i-1}$ for shift right operation and $A_{i+1}$ for shift left operation.

→ Note that the diagram shown just one typical stage must be connected to input carry $C_i$ of next stage in sequence.

→ The circuit whose one stage is specified in the figure provides eight arthematic operations and four logic operations and two shift operations.

→ Each operation is selected with five variables $S_3, S_2, S_1, S_0$ and $C_{in}$. The $C_{in}$ is selected only for arithmetic.

**Function table :-**

| Operation Selected | | | | | Operation | Function |
|---|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | $C_{in}$ | | |
| 0 | 0 | 0 | 0 | 0 | $F = A$ | Transfer A |
| 0 | 0 | 0 | 0 | 1 | $F = A+1$ | increment A |
| 0 | 0 | 0 | 1 | 0 | $F = A+B$ | Addition |
| 0 | 0 | 0 | 1 | 1 | $F = A+B+1$ | Add with carry |
| 0 | 0 | 1 | 0 | 0 | $F = A+\bar{B}$ | Subtract with borrow |
| 0 | 0 | 1 | 0 | 1 | $F = A+\bar{B}+1$ | Subtraction |
| 0 | 0 | 1 | 1 | 0 | $F = A-1$ | Decrement A |
| 0 | 0 | 1 | 1 | 1 | $F = A$ | Transfer A |
| 0 | 1 | 0 | 0 | X | $F = A \wedge B$ | AND |
| 0 | 1 | 0 | 1 | X | $F = A \vee B$ | OR |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | X | $F = A \oplus B$ | XOR |
| 0 | 1 | 1 | 1 | X | $F = \overline{A}$ | Complement A |
| 1 | 0 | X | X | X | F = shr A | shift right A into F |
| 1 | 1 | X | X | X | F = shl A | shift left A into F |

**Example 1.8.1** *Consider the memory system of a computer storing the following data :*

| Address in Hex | Data stored (binary) |
|---|---|
| 2000 | 00111000 |
| 2001 | 00110100 |
| 2002 | 00110010 |
| 2003 | 00111001 |

*Interpret the storage as numbers in the manner indicated below and find their decimal values in each case.*

*i) Big-endian storage of 2 hex words of 4-digits each.*

*ii) Big-endian storage of 2 BCD words of 4 -digits each.*

*iii) Little-endian storage, in ASCII, of a 4-digit signed hex word.*

*iv) Little endian storage, in ASCII, of a 4-digit BCD word.*

**Solution :** We know that, big endian storage uses lower byte address for more-significant bytes and little-endian storage uses lower byte address for less-significant bytes. Therefore, in big-endian storage given two words are : 3834H, 3239H and in little-endian storage given two words are : 3932H, 3438H.

**i)** Big-endian storage of 2 hex words of 4-digit each are : 3834H, 3239H

$$3834H = 3\times16^3+8\times16^2+3\times16^1+4\times16^0 = 14388 \text{ decimal}$$

$$3239H = 3\times16^3+2\times16^2+3\times16^1+9\times16^0 = 12857 \text{ decimal}$$

---

ii) Big-endian storage of 2 BCD words of 4-digit each are : 3834, 3239.

3834 (BCD) = 3834 decimal

3239 (BCD) = 3239 decimal

iii) Little-endian storage in ASCII, of 4-digit signed hex word is : 9248H. **Note :** We subtract 30H from ASCII code to get the digit value in hex.

$$9248H = 9\times16^3+2\times16^2+4\times16^1+8\times16^0 = 37448 \text{ decimal}$$

iv) Little-endian storage, in ASCII, of a 4-digit BCD word is : 9248

9248 BCD = 9248 decimal

**Example 1.8.2** *Convert the following binary number to decimal number.*

*a)* $110101_2$ *b)* $111100_2$ **May-09, Marks 2**

**Solution :**

$$(110101)_2 = 1\times2^5+1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$$

$$= 32+16+0+4+0+1 = (53)_{10}$$

$$(111100)_2 = 1\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0$$

$$= 32+16+8+4+0+0 = (60)_{10}$$

**Example 1.9.1** *Find 1's complement of* $(11010100)_2$.

**Solution :**

| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | Number |
|---|---|---|---|---|---|---|---|---|
| ⏚ | ⏚ | ⏚ | ⏚ | ⏚ | ⏚ | ⏚ | ⏚ | NOT operation |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1's complement of number |

**Example 1.9.2** *Find 2's complement of* $(11000100)_2$.

**Solution :** Let us see how – 6 is represented in these three formats. Consider the number 6 represented in binary with eight bits.

Signed-magnitude representation : 10000110

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | **Number** |
| | | | | | **1** | **1** | | **Carry** |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | **1's complement of number** |
| + | | | | | | | 1 | **Add 1** |
| **0** | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **2's complement of number** |

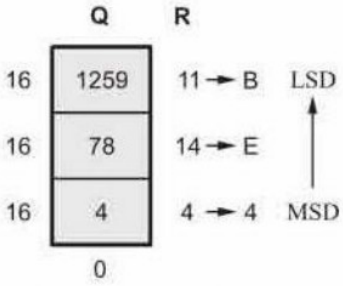
Signed-1's complement representation : 11111001

Signed-2's complement representation : 11111010

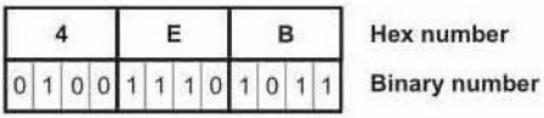
**Example 1.10.1** *Represent $1259.125_{10}$ in single precision and double precision formats*

**Solution : Step 1 : Convert decimal number in binary format**

**Integer part :**

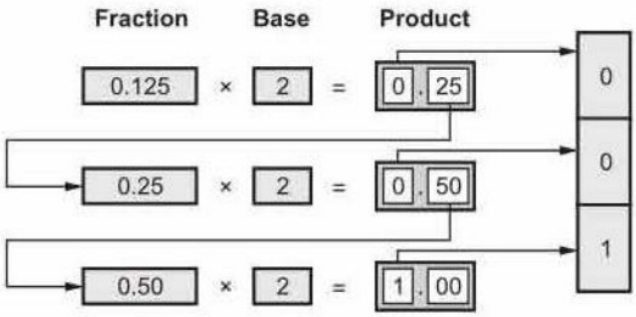



$$\therefore (1259)_{10} = (4EB)_{16}$$




$$\therefore (1259)_{10} = (100\ 1110\ 1011)_2$$

**Fractional part :**




$$\therefore (0.125)_{10} = (0.001)_2$$

$$\therefore \text{Binary number} = 1\,0\,0\,1\,1\,1\,0\,1\,0\,1\,1 + 0.0\,0\,1$$

$$= 1\,0\,0\,1\,1\,1\,0\,1\,0\,1\,1.0\,0\,1$$

**Step 2 : Normalize the number**

$$1\,0\,0\,1\,1\,1\,0\,1\,0\,1\,1.0\,0\,1 = 1.0\,0\,1\,1\,1\,0\,1\,0\,1\,1\,0\,0\,1 \times 2^{10}$$

Now we will see the representation of the numbers in single precision and double precision formats.

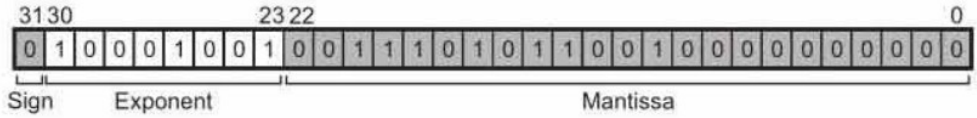
**Step 3 : Single precision representation**

For a given number S = 0, E = 10 and M = 0 0 1 1 1 0 1 0 1 1 0 0 1

Bias for single precision format is = 127

$\therefore \quad E' = E + 127 = 10 + 127 = 137_{10} = 1\,0\,0\,0\,1\,0\,0\,1_2$

$\therefore$ Number in single precision format is given as




**Step 4 : Double precision representation**

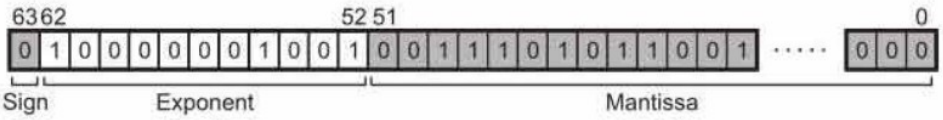
For a given number

S = 0, E = 10, and M = 0 0 1 1 1 0 1 0 1 1 0 0 1

Bias for double precision format is = 1023

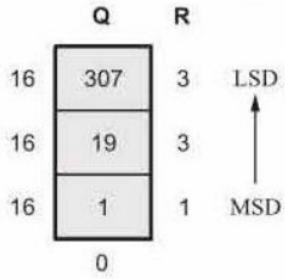
$\therefore \quad E' = E + 1023 = 10 + 1023 = 1033_{10} = 1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,1_2$

$\therefore$ Number in double precision format is given as

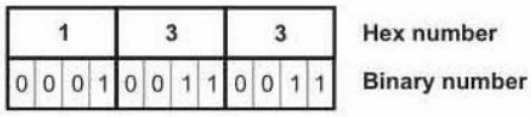



**Example 1.10.2** *Represent* $-307.1875_{10}$ *in single precision and double precision formats.*

**Solution : Step 1 : Convert decimal number in binary format integer part**

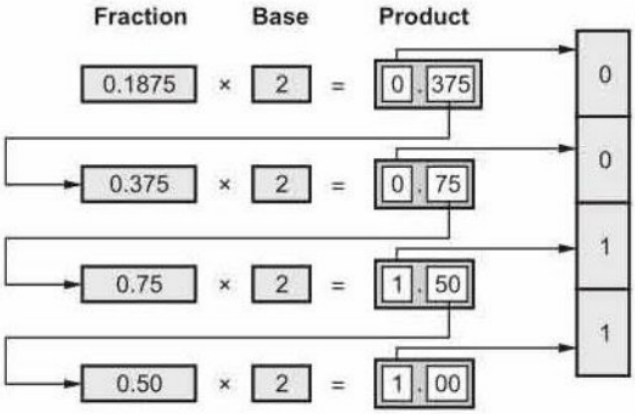



**Integer part :**




$\therefore \quad (307)_{10} = (133)_{16}$

$\therefore \quad (307)_{10} = (100110011)_2$




**Fractional part :**

$$\therefore \quad (0.1875)_{10} = (0.0011)_2$$

$$\therefore \quad \text{Binary number} = -1\,0\,0\,1\,1\,0\,0\,1\,1 + .\,0\,0\,1\,1$$

$$= -1\,0\,0\,1\,1\,0\,0\,1\,1 \quad .\,0\,0\,1\,1$$

**Step 2 : Normalize the number**

$$-1\,0\,0\,1\,1\,0\,0\,1\,1\,.\,0\,0\,1\,1 = -1.0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1 \times 2^8$$

Now we will see the representation of the numbers in single precision and double precision formats.

**Step 3 : Single precision representation**

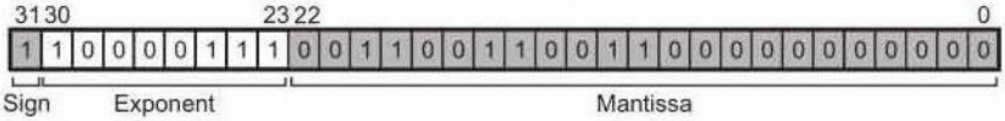
For a given number

S = 1, E = 8, and M = 0011 0011 0011

Bias for single precision format is = 127

$$\therefore \quad E' = E + 127 = 8 + 127 = 135_{10} = 10000111_2$$

$\therefore$ Number in single precision format is given as




**Step 4 : Double precision representation**

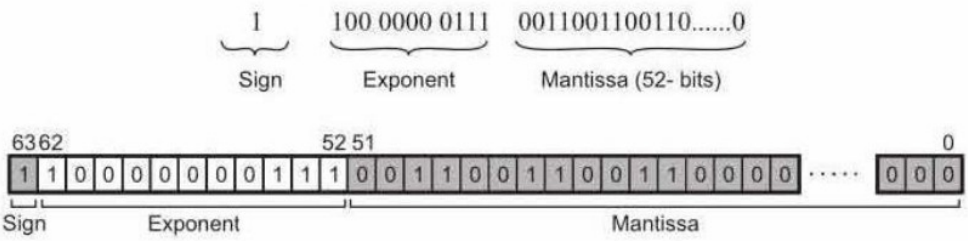
For a given number

S = 1, E = 8 and M = 0011 0011 0011

Bias for double precision format is = 1023

$$\therefore \quad E' = E + 1023 = 8 + 1023 = 1031_{10} = 10000000111_2$$

∴ Number in double precision format is given as

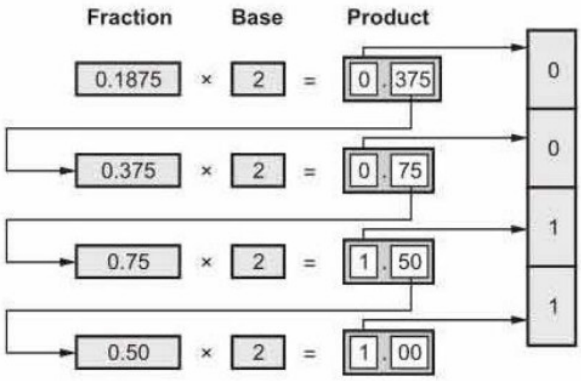



**Example 1.10.3** *Represent* $0.0625_{10}$ *in single precision and double precision formats.*

**Solution : Step 1 : Convert decimal number in binary format**

**Integer part :** 0

**Fractional part :**




$$(0.0625)_{10} = (0.0001)_2$$

∴ Binary number = 0.0001

**Step 2 : Normalize the number**

$$0.0001 = 1.0 \times 2^{-4}$$

Now we will see the representation of the numbers in single precision and double precision formats

**Step 3 : Single precision representation**

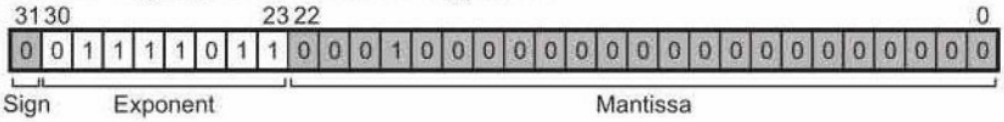
For a given number

S = 0, E = – 4, and M = 0001

Bias for single precision format is = 127

$\therefore \quad E' = E + 127 = -4 + 127 = 123_{10} = 01111011_2$

$\therefore$ Number in single precision format is given as




**Step 4 : Double precision representation**

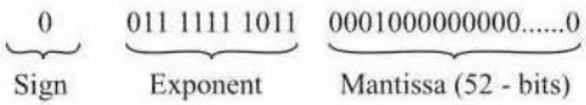
For a given number

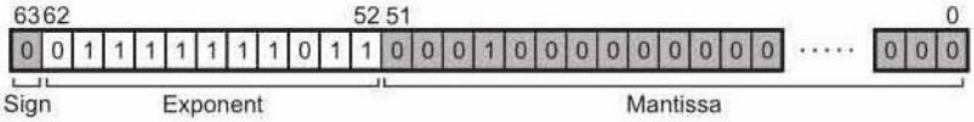
$S = 0, \quad E = -4, \quad$ and $M = 0001$

Bias for double precision format is = 1023

$\therefore \quad E' = E + 1023 = -4 + 1023 = 1019 = 01111111011_2$

$\therefore$ Number in double precision format is given as




**Example 1.13.1** *Registers R1 and R2 of a computer contain the decimal values 1200 and 2400 respectively. What is the effective address of the memory operand in each of the following instructions?*

*1) Load 20 (R1), R5* *2) Add - (R2), R5*

*3) Move # 3000, R5* *4) Sub (R1) +, R5*

**May-06, Marks 6**

**Solution :** R1 1200 R2 2400

| Sr. No. | Instruction | Effective address of memory |
|---|---|---|
| 1 | Load 20(R1), R5 | 1200 + 20 = 1220 |
| 2 | Add – (R2), R5 | 2400 – 1 = 2399 |
| 3 | Move #3000, R5 | 3000 |
| 4 | Sub (R1)+, R5 | 1200 |

**Example 1.13.2** *Registers R1 and R2 of a computer contain the decimal values 1200 and 4600. What is the effective address of the memory operand in each of the following instructions ?*

*a) Load 20 (R1), R5* *b) Add– (R2), R5.*

**May-08, Marks 2**

**Solution :** a) EA : 1200 + 20 b) EA : 4599.

**Example 1.13.3** *What must the address field of an indexed addressing mode instruction be to make it the same as a register indirect mode instruction ?*

**Solution :** The effective address in the indexed mode is given by EA = Offset + R and the effective address in the register indirect mode is given by EA = R. Thus, when offset is 0, the indexed addressing mode instruction is same as that of register indirect mode instruction.

**Example 1.13.4** *At memory address 200, two word instruction, load to AC is stored with a mode bit as a most significant bit. At location 201 the address stored is 500. At location 202 next instruction is stored. The following numbers are stored at different memory locations as shown ahead.*

| Memory location (Address) | Memory contents |
|---|---|
| 399 | 450 |
| 400 | 700 |
| 500 | 800 |
| 600 | 900 |
| 702 | 325 |
| 800 | 300 |

*If the content of PC is 200, while the contents of register R1 is 400. XR register is 100. If all the numbers and addresses are in decimal number, find out contents of AC and effective address for the following addressing modes.*

*i) Direct address ii) Indirect address*

*iii) Relative address iv) Indexed address*

*v) Register indirect addressing mode.*

**Solution :** The Fig. 1.13.3 shows the given information for our convenience.

**i) Direct address** : EA = 500 ∴ AC = **800**

**ii) Indirect address** : EA = [500] = **800** ∴ AC = **300**

**iii) Relative address** : EA = PC + Address part of instruction
= 202 + 500 = **702**
∴ AC = **325**

**iv) Indexed address** : EA = XR + Address part of instruction
= 100 + 500 = **600**
∴ AC = **900**

**v) Register indirect address** : EA = R1 = **400**
∴ AC = **700**

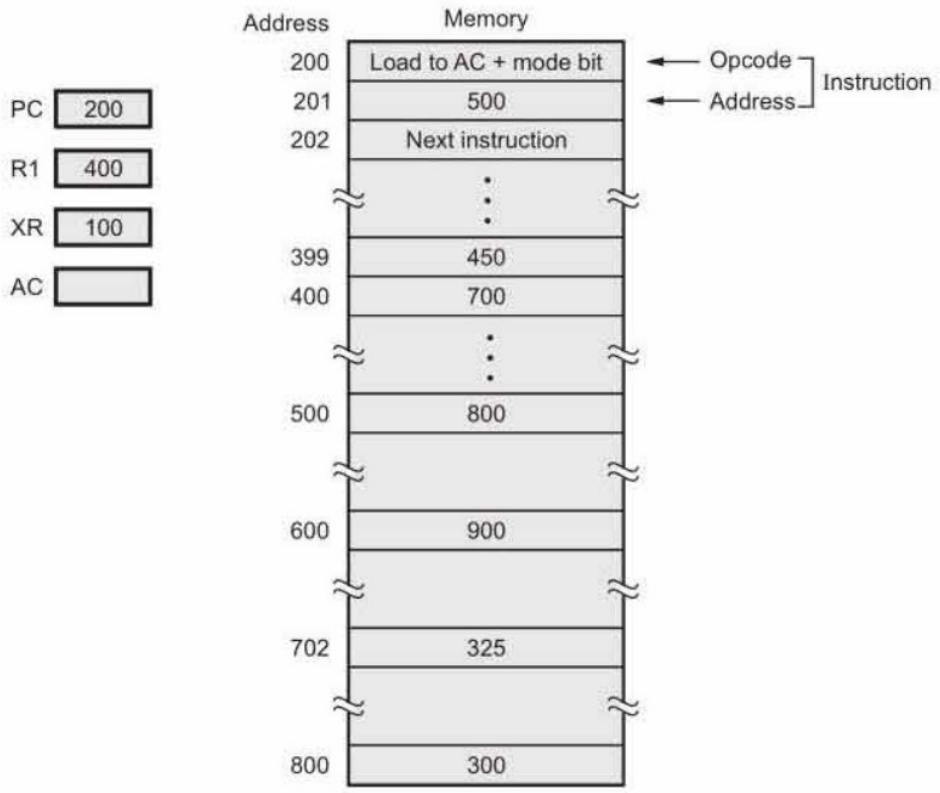


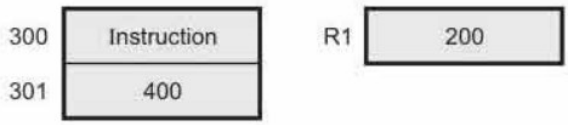

**Fig. 1.13.3**

**Example 1.13.5** *An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is :*

*i) Immediate ii) Direct iii) Register indirect*

*iv) Relative v) Index with R1 as the index register.*

**Solution :**




**i) Immediate mode :** In immediate addressing mode, the address bits of an instruction code specify the actual operand. Hence for this addressing mode the address of the operand i.e. effective address is 301.

**ii) Direct mode** : Effective Address (EA) : **400**

**iii) Register indirect node** : Effective Address (EA) : **200**

**iv) Relative mode** : Effective Address (EA) : PC + Address part of instruction = 302 + 400 = **702**

**v) Index with R1 as the index register** : Effective address (EA)

= R1 + Address part of the instruction = 200 + 400

= **600**

**Example 2.2.3** *Perform arithmetic operation with binary numbers with negative numbers in signed 2's complementary form (– 35) + (– 40).* **Dec.-03, Marks 2**

**Solution :**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 35 in binary |
| | 1 | 0 | 1 | 1 | 1 | 0 | 0 | – 35 in 1's complement form |
| | 1 | 0 | 1 | 1 | 1 | 0 | 1 | – 35 in 2's complement form |
| | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 40 in binary |
| | 1 | 0 | 1 | 0 | 1 | 1 | 1 | – 40 in 1's complement form |
| | 1 | 0 | 1 | 1 | 0 | 0 | 0 | – 40 in 2's complement form |
| ∴ (–35) | | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 2's complement of – 35 |
| + (–40) | + | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 2's complement of – 40 |
| | **1** | **0** | **1** | **1** | **0** | **1** | **0** | **1** | 2's complement of 75, i.e. (–75) |
| **Verification :** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | |
| | + | | | | | | | 1 | |
| | **0** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | (+ 75) |

**Example 2.2.1** *Perform addition of $(11001100)_2$ and $(11011010)_2$.*

**Solution :**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **1** | **1** | | **1** | **1** | | | | | **Carry** |
| | | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | **Number 1** |
| + | | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | **Number 2** |
| | **1** | **1** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **Result (S)** |

In the above example, we have seen the addition of two unsigned binary numbers. In case of the signed numbers, we have to consider the sign of the number and we may require sign extension to avoid possible overflow.

**Example 2.2.2** *Add $(28)_{10}$ and $(15)_{10}$ by converting them into binary.*

**Solution :** Using decimal to binary conversion technique we have,

$$(28)_{10} = (011100)_2 \text{ and } (15)_{10} = (01111)_2$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **1** | **1** | **1** | | | | **Carry** |
| Sign Extension → | **0** | 0 | 1 | 1 | 1 | 0 | 0 | **Binary equivalent of $(28)_{10}$** |
| + | **0** | **0** | 0 | 1 | 1 | 1 | 1 | **Binary equivalent of $(15)_{10}$** |
| Sign → | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **Result : Binary equivalent of $(43)_{10}$** |

$$\begin{array}{r} (28)_{10} \\ + \ (15)_{10} \\ \hline (43)_{10} \end{array}$$

**Example 2.2.4** *Perform 1010100 – 1000100 using 1's and 2's complement.*

**May-08, CSE, Marks 2**

**Solution :** Subtraction using 1's complement

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

Subtrahend

1's complement of subtrahend

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1** | **1** | **1** | | | | | Carry |
| | | 1 | 0 | 1 | 0 | 1 | 0 | 0 | Minuend |
| + | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1's complement of subtrahend |
| | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | Result |
| | | | | **1** | **1** | **1** | **1** | | Carry |
| | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **1** | Result |
| + | | | | | | | | **1** | Add end-around carry |
| | | **0** | **0** | **1** | **0** | **0** | **0** | **0** | Final result |

Subtraction using 2's complement

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| + | | | | | | | **1** |
| | **0** | **1** | **1** | **1** | **1** | **0** | **0** |

Subtrahend

1's complement of subtrahend

2's complement of subtrahend

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1** | **1** | **1** | **1** | **1** | | | | Carry |
| | | 1 | 0 | 1 | 0 | 1 | 0 | 0 | Minuend |
| + | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 2's complement of subtrahend |
| Discard carry | ~~1~~ | **0** | **0** | **1** | **0** | **0** | **0** | **0** | Result |

**Example 2.2.5** *Using 2's complement method perform* $(1010_2 - 1111_2)$.

**Dec.-08, CSE/IT, Marks 2**

**Solution :**

| | | | | | |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 1's complement of 1111 |
| + | | | | 1 | Add 1 |
| | **0** | **0** | **0** | **1** | 2's complement of 1111 |

| | | | | | |
|---|---|---|---|---|---|
| | 1 | 0 | 1 | 0 | |
| + | 0 | 0 | 0 | 1 | 2's complement of 1111 |
| | **1** | **0** | **1** | **1** | No carry, answer is negative and in 2's complement form |

## Two Marks Questions with Answers

**Q.1** *Define computer architecture.*

**Ans. :** Computer architecture is defined as the functional operation of the individual H/W unit in a computer system and the flow of information among the control of those units.

**Q.2** *Define computer H/W.*

**Ans. :** Computer H/W is the electronic circuit and electro mechanical equipment that constitutes the computer.

**Q.3** *Name the functional units of a computer.* **Dec.-08**

**Ans. :** The functional units of a computer are :

1. Input unit
2. Output unit
3. Control unit
4. Memory unit
5. Arithmetic and logical unit

**Q.4** *What is meant by Central Processing Unit (CPU) ?* **Dec.-09**

**Ans. :** The arithmetic and logic unit in conjunction with control unit is commonly called **Central Processing Unit (CPU)**.

**Q.5** *What is the function of input unit ?*

**Ans. :** A computer accepts a digitally coded information through input unit using input devices such as keyboard and mouse.

**Q.6** *What are the functions of control unit ?*

**Ans. :** The control unit co-ordinates and controls the activities amongst the functional units. The basic function of control unit is to fetch the instructions stored in the main memory, identify the operations and the devices involved in it, and accordingly generate control signals to execute the desired operations.

**Q.7** *What is the function of arithmetic and logic unit ?*

**Ans. :** The arithmetic and logic unit (ALU) is responsible for performing arithmetic operations such as add, subtract, division and multiplication, and logical operations such as ANDing, ORing, Inverting etc.

**Q.8** *What is meant by the stored program concept ? Discuss.* **May-07**

**Ans. :** According to stored-program concept memory can contain the program (source code), the corresponding compiled machine code, editor program and even the compiler that generated the machine code.

**Q.39** *Convert the following binary number to decimal number.*

*a)* $110101_2$ *b)* $111100_2$ **May-09**

**Ans. :**

$$(110101)_2 = 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$
$$= 32 + 16 + 0 + 4 + 0 + 1 = (53)_{10}$$
$$(111100)_2 = 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 0\times 2^0$$
$$= 32 + 16 + 8 + 4 + 0 + 0 = (60)_{10}$$

**Q.40** *How do you relate addition and subtraction ?*

**Ans. :** We can relate addition and subtraction operations of numbers by the following relationship :

$(\pm A) - (+B) = (\pm A) + (-B)$ and

$(\pm A) - (-B) = (\pm A) + (+B)$

**Q.41** *What is one's complement of numbers ?*

**Ans. :** The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

**Q.42** *What is 2's complement of numbers ?*

**Ans. :** The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as 2's complement = 1's complement + 1.

**Q.43** *Define IEEE floating point single and double precision standard.*

**Dec.-06, 07**

**Ans. :** The 32-bit standard representation (**single-precision representation**) occupies a single 32-bit word. The 32-bits are divided into three fields as shown below :

(field 1) Sign ← 1 - bit

(field 2) Exponent ← 8 - bits

(field 3) Mantissa ← 23 - bits

The 64-bit standard representation (**double-precision representation)** occupies two 32-bit words. The 64-bits are divided into three fields as shown below :

(field 1) Sign ← 1 - bit

(field 2) Exponent ← 1 - bit

(field 3) Mantissa ← 52 - bits

**Q.1** *Define half adder and full adder.*

**Ans. :**

- **Half adder :** The logic circuit which performs the arithmetic sum of two bits is called a half adder.
- **Full adder :** The logic circuit which performs the arithmetic sum of 3 bits (bit 1 : input 1, bit 2 : input 2, bit 3 : carry from the previous addition) is called a full adder.

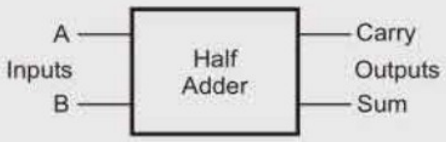
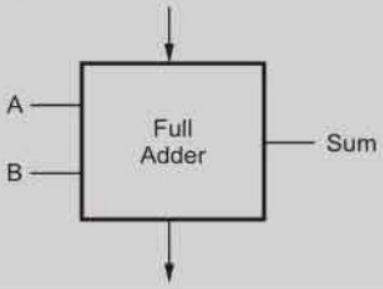
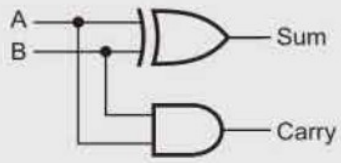
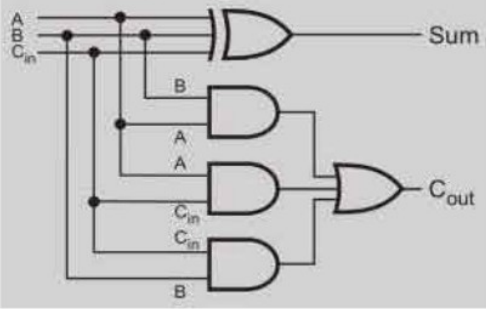
**Q.2** *Define half subtractor and full subtractor.*

**Ans. :**

- **Half subtractor :** It is a combinational circuit that subtracts two bits and produces their difference and borrow.
- **Full subtractor :** It is a combinational circuit that performs a subtraction between 2 bits. It also takes into account borrow of the lower significant stage.

**Q.3** *What is the difference between half adder and full adder?* **Dec.-07**

**Ans. :**

| Sr. No. | Half adder | Full adder |
|---|---|---|
| 1. | Half-adder takes two binary-inputs i.e. augend and addend bits and gives out two binary outputs as sum and carry. | Full-adder, alongwith augend and addend takes third additional bit Cin as input. Cin represents the carry from the previous lower significant position. |
| 2. | Half-adder is not used in practice | Full adder is used in practice |
| 3. | **Block diagram :**   | **Block diagram :**   |
| 4. | **Logic diagram :**   | **Logic diagram :**   |

generator circuit. As shown in Fig. 3-3, this circuit consists of one exclusive-OR and one exclusive-NOR gate. Since $P$(even) is the exclusive-OR of $x$, $y$, $z$, and $P$(odd) is the complement of $P$(even), it is necessary to employ an exclusive-NOR gate for the needed complementation. The message and the odd-parity bit are transmitted to their destination where they are applied to a parity checker. An error has occurred during transmission if the parity of the four bits received is even, since the binary information transmitted was originally odd. The output of the parity checker would be 1 when an error occurs, that is, when the number of 1's in the four inputs is even. Since the exclusive-OR function of the four inputs is an odd function, we again need to complement the output by using an exclusive-NOR gate.

It is worth noting that the parity generator can use the same circuit as the parity checker if the fourth input is permanently held at a logic-0 value. The advantage of this is that the same circuit can be used for both parity generation and parity checking.

It is evident from the example above that even-parity generators and checkers can be implemented with exclusive-OR functions. Odd-parity networks need an exclusive-NOR at the output to complement the function.

## PROBLEMS

**3-1.** Convert the following binary numbers to decimal: 101110; 1110101; and 110110100.

**3-2.** Convert the following numbers with the indicated bases to decimal: $(12121)_3$; $(4310)_5$; $(50)_7$; and $(198)_{12}$.

**3-3.** Convert the following decimal numbers to binary: 1231; 673; and 1998.

**3-4.** Convert the following decimal numbers to the bases indicated.
- **a.** 7562 to octal
- **b.** 1938 to hexadecimal
- **c.** 175 to binary

**3-5.** Convert the hexadecimal number F3A7C2 to binary and octal.

**3-6.** What is the radix of the numbers if the solution to the quadratic equation $x^2 - 10x + 31 = 0$ is $x = 5$ and $x = 8$?

**3-7.** Show the value of all bits of a 12-bit register that hold the number equivalent to decimal 215 in (a) binary; (b) binary-coded octal; (c) binary-coded hexadecimal; (d) binary-coded decimal (BCD).

**3-8.** Show the bit configuration of a 24-bit register when its content represents the decimal equivalent of 295: (a) in binary; (b) in BCD; (c) in ASCII using eight bits with even parity.

**3-9.** Write your name in ASCII using an 8-bit code with the leftmost bit always 0. Include a space between names and a period after a middle initial.



Scanned by CamScanner

**3-10.** Decode the following ASCII code:

1001010 1001111 1001000 1001110 0100000 1000100 1001111 1000101

**3-11.** Obtain the 9's complement of the following eight-digit decimal numbers: 12349876; 00980100; 90009951; and 00000000.

**3-12.** Obtain the 10's complement of the following six-digit decimal numbers: 123900; 090657; 100000; and 000000.

**3-13.** Obtain the 1's and 2's complements of the following eight-digit binary numbers: 10101110; 10000001; 10000000; 00000001; and 00000000.

**3-14.** Perform the subtraction with the following unsigned decimal numbers by taking the 10's complement of the subtrahend.

a. 5250 − 1321
b. 1753 − 8640
c. 20 − 100
d. 1200 − 250

**3-15.** Perform the subtraction with the following unsigned binary numbers by taking the 2's complement of the subtrahend.

a. 11010 − 10000
b. 11010 − 1101
c. 100 − 110000
d. 1010100 − 1010100

**3-16.** Perform the arithmetic operations (+42) + (−13) and (−42) − (−13) in binary using signed-2's complement representation for negative numbers.

**3-17.** Perform the arithmetic operations (+70) + (+80) and (−70) + (−80) with binary numbers in signed-2's complement representation. Use eight bits to accommodate each number together with its sign. Show that overflow occurs in both cases, that the last two carries are unequal, and that there is a sign reversal.

**3-18.** Perform the following arithmetic operations with the decimal numbers using signed-10's complement representation for negative numbers.

a. (−638) + (+785)
b. (−638) − (+185)

**3-19.** A 36-bit floating-point binary number has eight bits plus sign for the exponent and 26 bits plus sign for the mantissa. The mantissa is a normalized fraction. Numbers in the mantissa and exponent are in signed-magnitude representation. What are the largest and smallest positive quantities that can be represented, excluding zero?

**3-20.** Represent the number $(+46.5)_{10}$ as a floating-point binary number with 24 bits. The normalized fraction mantissa has 16 bits and the exponent has 8 bits.

**3-21.** The Gray code is sometimes called a reflected code because the bit values are reflected on both sides of any $2^n$ value. For example, as shown in Table 3-5, the values of the three low-order bits are reflected over a line drawn between 7 and 8. Using this property of the Gray code, obtain:

a. The Gray code numbers for 16 through 31 as a continuation of Table 3-5.
b. The excess-3 Gray code for decimals 10 to 19 as a continuation of the list in Table 3-6.

**3-22.** Represent decimal number 8620 in (a) BCD; (b) excess-3 code; (c) 2421 code; (d) as a binary number.



Scanned by CamScanner

**3-23.** List the 10 BCD digits with an even parity in the leftmost position (total of five bits per digit). Repeat with an odd-parity bit.

**3-24.** Represent decimal 3984 in the 2421 code of Table 3-6. Complement all bits of the coded number and show that the result is the 9's complement of 3984 in the 2421 code.

**3-25.** Show that the exclusive-OR function $x = A \oplus B \oplus C \oplus D$ is an odd function. One way to show this is to obtain the truth table for $y = A \oplus B$ and for $z = C \oplus D$ and then formulate the truth table for $x = y \oplus z$. Show that $x = 1$ only when the total number of 1's in $A$, $B$, $C$, and $D$ is odd.

**3-26.** Derive the circuits for a 3-bit parity generator and 4-bit parity checker using an even-parity bit. (The circuits of Fig. 3-3 use odd parity.)



Scanned by CamScanner

# CHAPTER 3

## 3-1

$(101110)_2 = 32+8+4+2 = 46$

$(1110101)_2 = 64+32+16+4+1 = 117$

$(110110100)_2 = 256+128+32+16+4 = 436$

## 3-2

$(12121)_3 = 3^4 + 2\times3^3 + 3^2 + 2\times3 + 1 = 81+54+9+6+1 = 151$

$(4310)_5 = 4\times5^3 + 3\times5^2 + 5 = 500 + 75 + 5 = 580$

$(50)_7 = 5\times7 = 35$

$(198)_{12} = 12^2 + 9\times12 + 8 = 144+108+8 = 260$

## 3-3

$(1231)_{10} = 1024+128+64+15 = 2^{10}+2^7+2^6+2^3+2^2+2+1 = (10011001111)_2$

$(673)_{10} = 512+128+32+1 = 2^9+2^7+2^5+1 = (1010100001)_2$

$(1998)_{10} = 1024+512+256+128+64+8+4+2$

$= 2^{10}+2^9+2^8+2^7+2^6+2^3+2^2+2^1 = (11111001110)_2$

## 3-4

$(7562)_{10} = (16612)_8$

$(1938)_{10} = (792)_{16}$

$(175)_{10} = (10101111)_2$

## 3-5

$(F3A7C2)_{16} = (1111\ 0011\ 1010\ 0111\ 1100\ 0010)_2$

$= (74723702)_8$

## 3-6

$(x^2-10x+31)_r = [(x-5)(x-8)]_{10}$

$= x^2 - (5+8)_{10}\,x + (40)_{10}\,x$

Therefore: $(10)_r = (13)_{10}$ $\quad r = 13$

Also $(31)_r = 3\times13+1 = (40)_{10}$ $\quad (r=13)$





Scanned by CamScanner

3-7 $(215)_{10} = 128+64+16+7 = (11010111)_2$

(a) 0000 1101 0111 Binary

(b) 000 011 010 111 Binary coded octal
0 3 2 7

(c) 0000 1101 0111 Binary coded hexadecimal
0 D 7

(d) 0010 0001 0101 Binary coded decimal
2 1 5

3-8 $(295)_{10} = 256+32+7 = (100100111)_2$

(a) 0000 0000 0000 0001 0010 0111

(b) 0000 0000 0000 0010 1001 0101

(c) 10110010 00111001 00110101

3-10 JOHN DOE

3-11 87650123; 99019899; 09990048; 999999.

3-12 876100; 909343; 900000; 000000

3-13 01010001; 01111110; 01111111; 11111110; 11111111
01010010; 01111111; 10000000; 11111111; 00000000

3-14

(a) 5250 + 8679 = 13929

(b) 1753 + 1360 = 03113
10's complement → −6887

(c) 020 + 900 = 0920
10's complement → −080

(d) 1200 + 9750 = 10950

3-15

(a) 11010 + 10000 = 101010 (26−16=10)

(b) 11010 + 10011 = 101101 (26−13=13)

(c) 000100 + 010000 = 010100
−101100 (4−48=−44)

(d) 1010100 + 0101100 = 10000000 (84−84=0)





Scanned by CamScanner

3-16

+42 = 0101010 +13 = 0001101
−42 = 1010110 −13 = 1110011

(+42) 0101010 (−42) 1010110
(−13) 1110011 (+13) 0001101
(+29) 0011101 (−29) 1100011

3-17 01 ← last two carries → 10

+70 01000110 -70 10111010
+80 01010000 -80 10110000
+150 10010110 -150 01101010

greater than +127 ← negative less than −128 ← positive

3-18

(a) (−638) 9362 (b) (−638) 9362
(+785) +0785 (−185) +9815
(+147) 0147 (−823) 9177

3-19

Mantissa: | + | 26 bits |
Exponent: | S | 8 bits | 36 bits

Largest: +0.1111...1 + 11111111 $(1-2^{-26})\times 2^{+255}$
$1-2^{-26}$ +255

Smallest: +0.1000...0 −11111111
(normalized) $2^{-1}$ −255 $2^{-256}$

3-20

$46.5 = 32+8+4+2+0.5 = (101110.1)_2$

Sign
0 10111010000000000 00000110
24-bit mantissa 8-bit exponent (+6)




Scanned by CamScanner

3-21 (a)

| Decimal | Gray code |
|---|---|
| 16 | 11000 |
| 17 | 11001 |
| 18 | 11011 |
| 19 | 11010 |
| 20 | 11110 |
| 21 | 11111 |
| 22 | 11101 |
| 23 | 11100 |
| 24 | 10100 |
| 25 | 10101 |
| 26 | 10111 |
| 27 | 10110 |
| 28 | 10010 |
| 29 | 10011 |
| 30 | 10001 |
| 31 | 10000 |

(b)

| Decimal | Excess-3 Gray |
|---|---|
| 9 | 0010 1010 |
| 10 | 0110 1010 |
| 11 | 0110 1110 |
| 12 | 0110 1111 |
| 13 | 0110 1101 |
| 14 | 0110 1100 |
| 15 | 0110 0100 |
| 16 | 0110 0101 |
| 17 | 0110 0111 |
| 18 | 0110 0110 |
| 19 | 0110 0010 |
| 20 | 0111 0010 |

3-22 8620

(a) BCD 1000 0110 0010 0000

(b) XS-3 1011 1001 0101 0011

(c) 2421 1110 1100 0010 0000

(d) Binary 10000110101100 (8192+256+128+32+8+4)

3-23

| Decimal | BCD with even parity | BCD with odd parity |
|---|---|---|
| 0 | 0 0000 | 1 0000 |
| 1 | 1 0001 | 0 0001 |
| 2 | 1 0010 | 0 0010 |
| 3 | 0 0011 | 1 0011 |
| 4 | 1 0100 | 0 0100 |
| 5 | 0 0101 | 1 0101 |
| 6 | 0 0110 | 1 0110 |
| 7 | 1 0111 | 0 0111 |
| 8 | 1 1000 | 0 1000 |
| 9 | 0 1001 | 1 1001 |





Scanned by CamScanner

3-24

3984 = 0011 1111 1110 0100

1100 0000 0001 1011 = 6015

3-25

| A | B | $y = A \oplus B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| C | D | $z = C \oplus D$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| y | z | $x = y \oplus z$ | |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 1 | 1 | ← AB = 00 or 11, CD = 01 or 10 |
| 1 | 0 | 1 | ← AB = 01 or 10, CD = 00 or 11 |
| 1 | 1 | 0 | |

ABCD

0001, 0010, 1101, 1110

0100, 0111, 1000, 1011

Always odd number of 1's

3-26

Same as in Fig. 3-3 but without the complemented circles in the outputs of the gates.

$P = x \oplus y \oplus z$

$\text{Error} = x \oplus y \oplus z \oplus P$





Scanned by CamScanner

## PROBLEMS

**4-1.** Show the block diagram of the hardware (similar to Fig. 4-2a) that implements the following register transfer statement:

$$yT_2: \quad R2 \leftarrow R1, \quad R1 \leftarrow R2$$

**4-2.** The outputs of four registers, $R0$, $R1$, $R2$, and $R3$, are connected through 4-to-1-line multiplexers to the inputs of a fifth register, $R5$. Each register is eight bits long. The required transfers are dictated by four timing variables $T_0$ through $T_3$ as follows:

$$\begin{aligned} T_0&: \quad R5 \leftarrow R0 \\ T_1&: \quad R5 \leftarrow R1 \\ T_2&: \quad R5 \leftarrow R2 \\ T_3&: \quad R5 \leftarrow R3 \end{aligned}$$

The timing variables are mutually exclusive, which means that only one variable is equal to 1 at any given time, while the other three are equal to 0. Draw a block diagram showing the hardware implementation of the register transfers. Include the connections necessary from the four timing variables to the selection inputs of the multiplexers and to the load input of register $R5$.

**4-3.** Represent the following conditional control statement by two register transfer statements with control functions.

$$\text{If } (P = 1) \text{ then } (R1 \leftarrow R2) \text{ else if } (Q = 1) \text{ then } (R1 \leftarrow R3)$$

**4-4.** What has to be done to the bus system of Fig. 4-3 to be able to transfer information from any register to any other register? Specifically, show the connections that must be included to provide a path from the outputs of register $C$ to the inputs of register $A$.

**4-5.** Draw a diagram of a bus system similar to the one shown in Fig. 4-3, but use three-state buffers and a decoder instead of the multiplexers.

**4-6.** A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.

a. How many selection inputs are there in each multiplexer?
b. What size of multiplexers are needed?
c. How many multiplexers are there in the bus?

**4-7.** The following transfer statements specify a memory. Explain the memory operation in each case.

a. $R2 \leftarrow M[AR]$
b. $M[AR] \leftarrow R3$
c. $R5 \leftarrow M[R5]$



Scanned by CamScanner

**4-8.** Draw the block diagram for the hardware that implements the following statements:

$$x + yz: \quad AR \leftarrow AR + BR$$

where $AR$ and $BR$ are two $n$-bit registers and $x$, $y$, and $z$ are control variables. Include the logic gates for the control function. (Remember that the symbol + designates an OR operation in a control or Boolean function but that it represents an arithmetic plus in a microoperation.)

**4-9.** Show the hardware that implements the following statement. Include the logic gates for the control function and a block diagram for the binary counter with a count enable input.

$$xyT_0 + T_1 + y'T_2: \quad AR \leftarrow AR + 1$$

**4-10.** Consider the following register transfer statements for two 4-bit registers $R1$ and $R2$.

$$\begin{aligned} xT: &\quad R1 \leftarrow R1 + R2 \\ x'T: &\quad R1 \leftarrow R2 \end{aligned}$$

Every time that variable $T = 1$, either the content of $R2$ is added to the content of $R1$ if $x = 1$, or the content of $R2$ is transferred to $R1$ if $x = 0$. Draw a diagram showing the hardware implementation of the two statements. Use block diagrams for the two 4-bit registers, a 4-bit adder, and a quadruple 2-to-1-line multiplexer that selects the inputs to $R1$. In the diagram, show how the control variables $x$ and $T$ select the inputs of the multiplexer and the load input of register $R1$.

**4-11.** Using a 4-bit counter with parallel load as in Fig. 2-11 and a 4-bit adder as in Fig. 4-6, draw a block diagram that shows how to implement the following statements:

| | | |
|---|---|---|
| $x$: | $R1 \leftarrow R1 + R2$ | Add $R2$ to $R1$ |
| $x'y$: | $R1 \leftarrow R1 + 1$ | Increment $R1$ |

where $R1$ is a counter with parallel load and $R2$ is a 4-bit register.

**4-12.** The adder-subtractor circuit of Fig. 4-7 has the following values for input mode $M$ and data inputs $A$ and $B$. In each case, determine the values of the outputs: $S_3$, $S_2$, $S_1$, $S_0$, and $C_4$.

| | $M$ | $A$ | $B$ |
|---|---|---|---|
| a. | 0 | 0111 | 0110 |
| b. | 0 | 1000 | 1001 |
| c. | 1 | 1100 | 1000 |
| d. | 1 | 0101 | 1010 |
| e. | 1 | 0000 | 0001 |



Scanned by CamScanner

**4-13.** Design a 4-bit combinational circuit decrementer using four full-adder circuits.

**4-14.** Assume that the 4-bit arithmetic circuit of Fig. 4-9 is enclosed in one IC package. Show the connections among two such ICs to form an 8-bit arithmetic circuit.

**4-15.** Design an arithmetic circuit with one selection variable $S$ and two $n$-bit data inputs $A$ and $B$. The circuit generates the following four arithmetic operations in conjunction with the input carry $C_{in}$. Draw the logic diagram for the first two stages.

| $S$ | $C_{in} = 0$ | $C_{in} = 1$ |
|---|---|---|
| 0 | $D = A + B$ (add) | $D = A + 1$ (increment) |
| 1 | $D = A - 1$ (decrement) | $D = A + \overline{B} + 1$ (subtract) |

**4-16.** Derive a combinational circuit that selects and generates any of the 16 logic functions listed in Table 4-5.

**4-17.** Design a digital circuit that performs the four logic operations of exclusive-OR, exclusive-NOR, NOR, and NAND. Use two selection variables. Show the logic diagram of one typical stage.

**4-18.** Register $A$ holds the 8-bit binary 11011001. Determine the $B$ operand and the logic microoperation to be performed in order to change the value in $A$ to:

**a.** 01101101

**b.** 11111101

**4-19.** The 8-bit registers $AR$, $BR$, $CR$, and $DR$ initially have the following values:

$$AR = 11110010$$
$$BR = 11111111$$
$$CR = 10111001$$
$$DR = 11101010$$

Determine the 8-bit values in each register after the execution of the following sequence of microoperations.

| | |
|---|---|
| $AR \leftarrow AR + BR$ | Add $BR$ to $AR$ |
| $CR \leftarrow CR \wedge DR,\ BR \leftarrow BR + 1$ | AND $DR$ to $CR$, increment $BR$ |
| $AR \leftarrow AR - CR$ | Subtract $CR$ from $AR$ |

**4-20.** An 8-bit register contains the binary value 10011100. What is the register value after an arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow.

**4-21.** Starting from an initial value of $R = 11011101$, determine the sequence of binary values in $R$ after a logical shift-left, followed by a circular shift-right, followed by a logical shift-right and a circular shift-left.



Scanned by CamScanner

**4-22.** What is the value of output $H$ in Fig. 4-12 if input $A$ is 1001, $S = 1$, $I_R = 1$, and $I_L = 0$?

**4-23.** What is wrong with the following register transfer statements?

a. $xT$: $AR \leftarrow \overline{AR}$, $AR \leftarrow 0$

b. $yT$: $R1 \leftarrow R2$, $R1 \leftarrow R3$

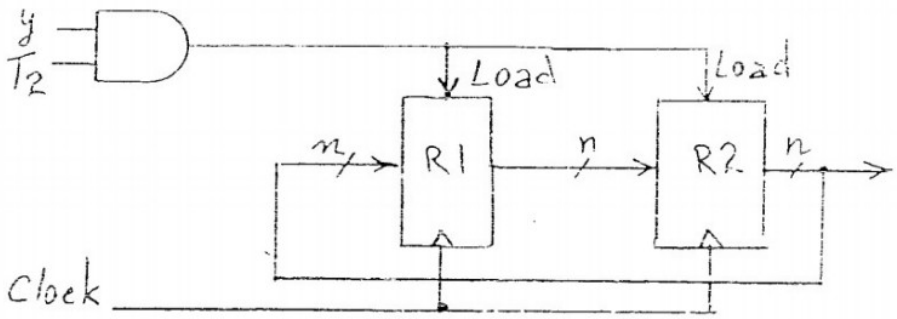
c. $zT$: $PC \leftarrow AR$, $PC \leftarrow PC + 1$



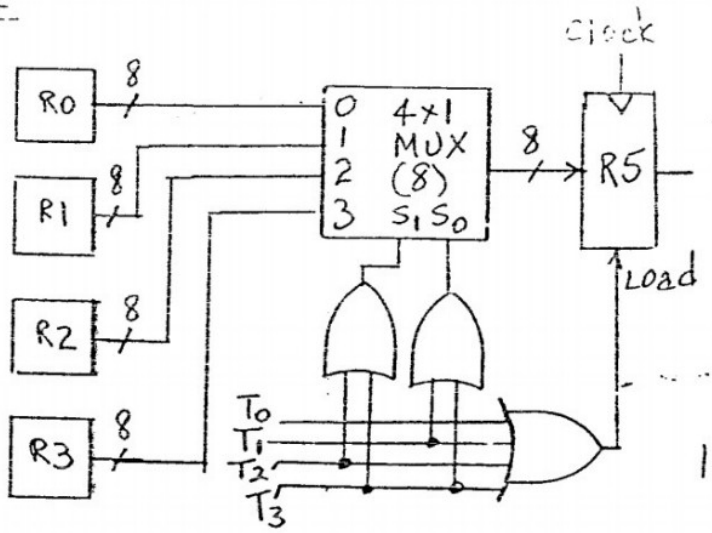
Scanned by CamScanner

# CHAPTER 4

## 4-1




## 4-2




| $T_0$ $T_1$ $T_2$ $T_3$ | $S_1$ $S_0$ | R3 load |
|---|---|---|
| 0 0 0 0 | X X | 0 |
| 1 0 0 0 | 0 0 | 1 |
| 0 1 0 0 | 0 1 | 1 |
| 0 0 1 0 | 1 0 | 1 |
| 0 0 0 1 | 1 1 | 1 |

$S_1 = T_2 + T_3$

$S_0 = T_1 + T_3$

$load = T_0 + T_1 + T_2 + T_3$

## 4-3

P: R1 ← R2

P'Q: R1 ← R3

## 4-4

Connect the 4-line common bus to the four inputs of each register.
Provide a "load" control input in each register.
Provide a clock input for each register.

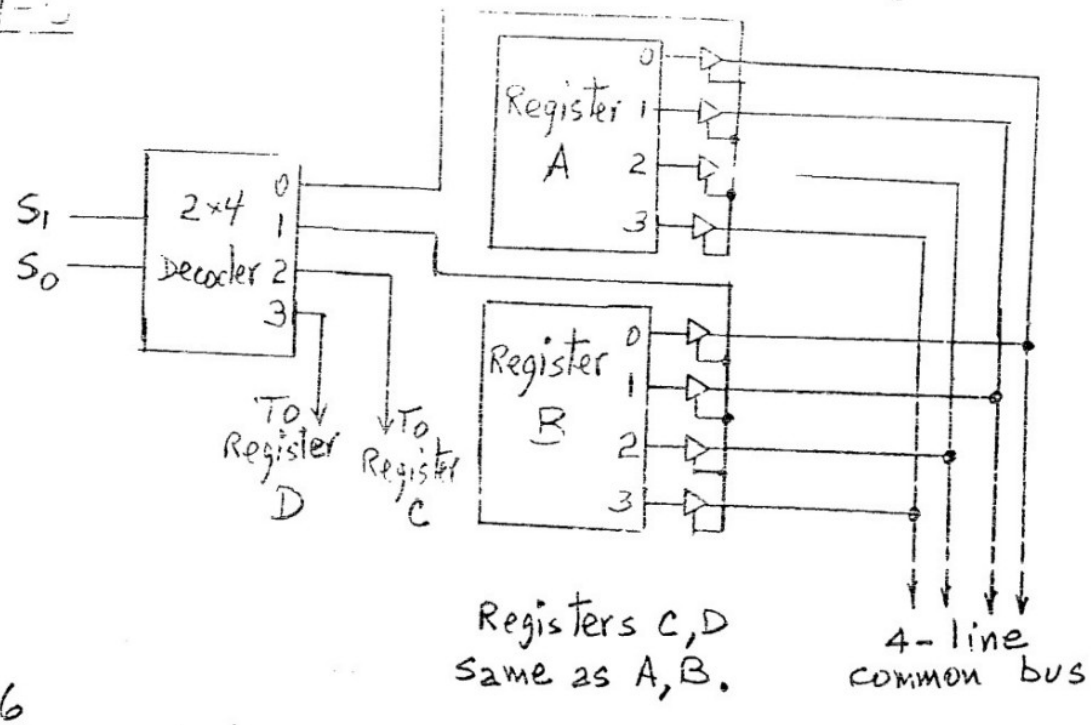
To transfer from register C to register A:
Apply $S_1S_0 = 10$ (to select C for the bus.)
Enable the load input of A
Apply a clock pulse.





Scanned by CamScanner

4-5




4-6

(a) 4 selection lines to select one of 16 registers,

(b) 16 × 1 multiplexers

(c) 32 multiplexers, one for each bit of the registers.

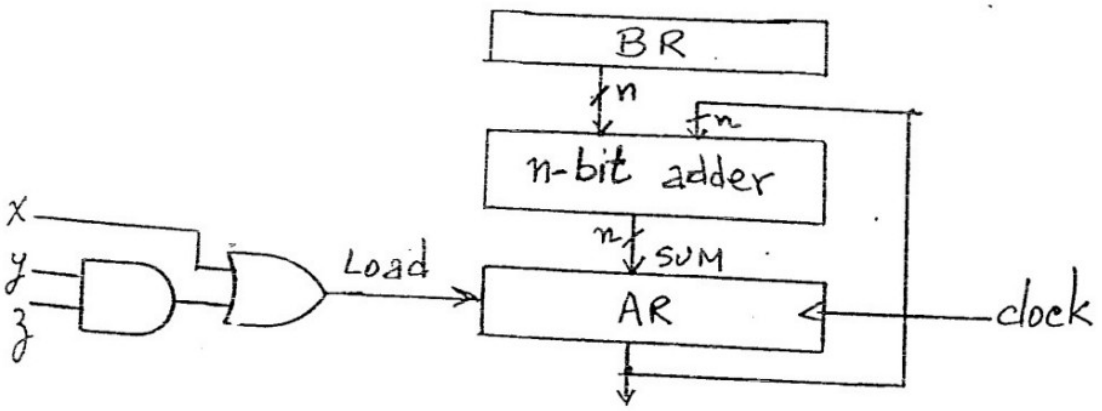
4-7

(a) Read memory word specified by the address in AR into register R2.

(b) Write content of register R3 into the memory word specified by the address in AR.

(c) Read memory word specified by the address in R5 and transfer content to R5 (destroys previous value)

4-8

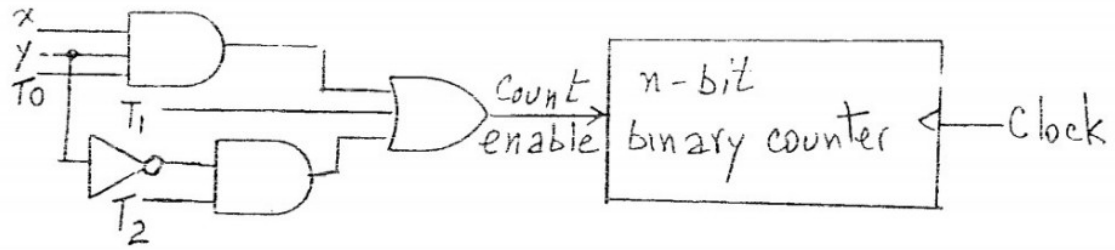




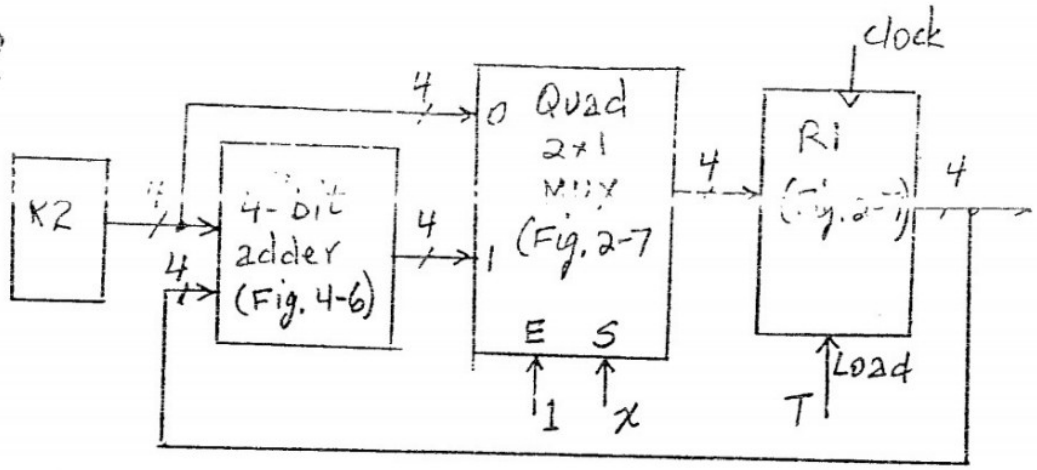



Scanned by CamScanner

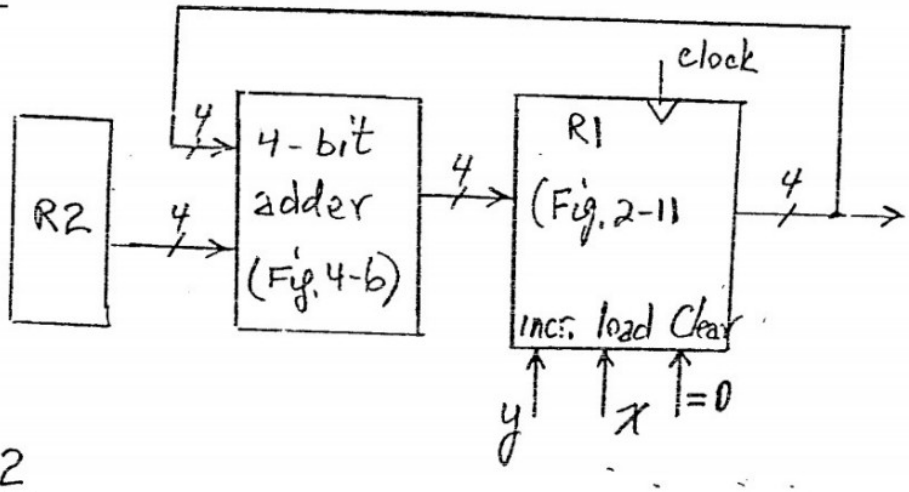
4-9




4-10




4-11




4-12

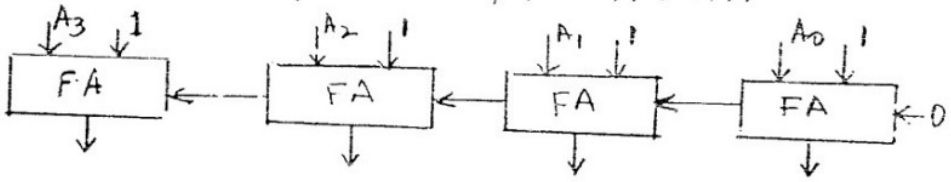
| M | A | B | Sum | $C_4$ | |
|---|---|---|---|---|---|
| 0 | 0111 | + 0110 | 1101 | 0 | 7+6 = 13 |
| 0 | 1000 | + 1001 | 0001 | 1 | 8+9 = 16+1 |
| 1 | 1100 | − 1000 | 0100 | 1 | 12−8 = 4 |
| 1 | 0101 | − 1010 | 1011 | 0 | 5−10 = −5 (in 2's comp.) |
| 1 | 0000 | − 0001 | 1111 | 0 | 0−1 = −1 (in 2's comp.) |



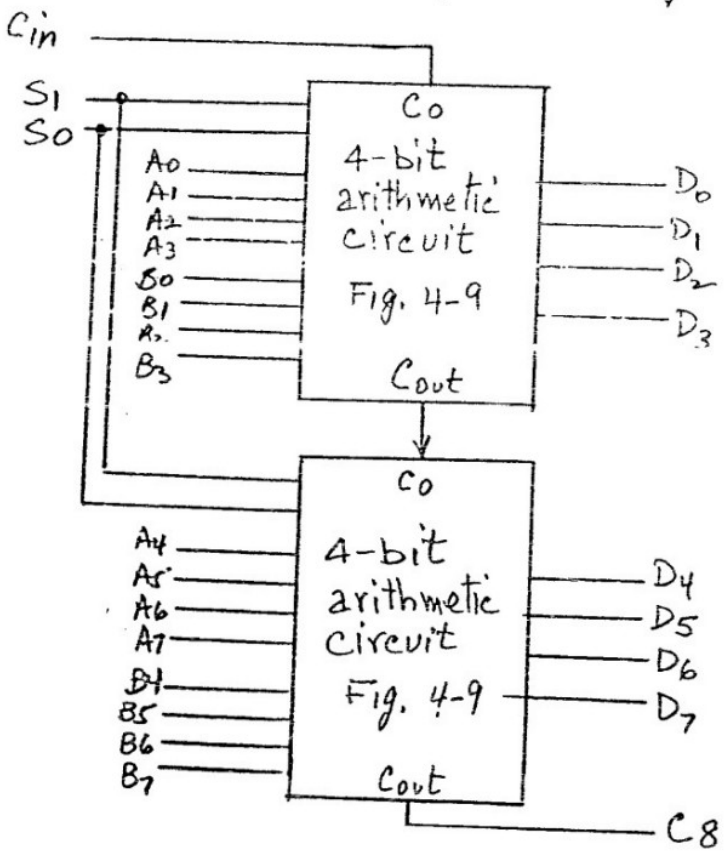


Scanned by CamScanner

4-13 A−1 = A + 2's complement of 1 = A + 1111

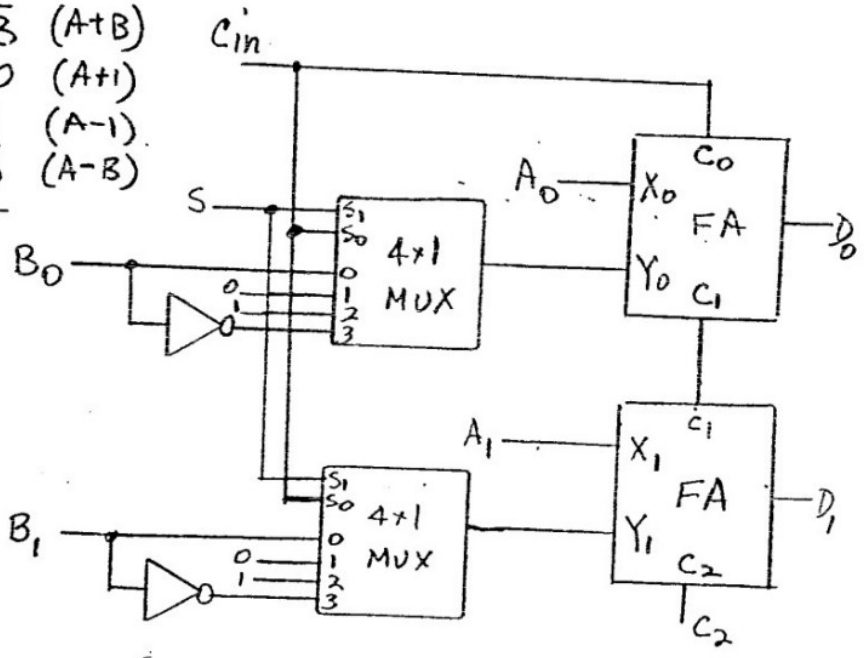



4-14




4-15

| S | Cin | X | Y | |
|---|---|---|---|---|
| 0 | 0 | A | B | (A+B) |
| 0 | 1 | A | 0 | (A+1) |
| 1 | 0 | A | 1 | (A-1) |
| 1 | 1 | A | $\overline{B}$ | (A-B) |

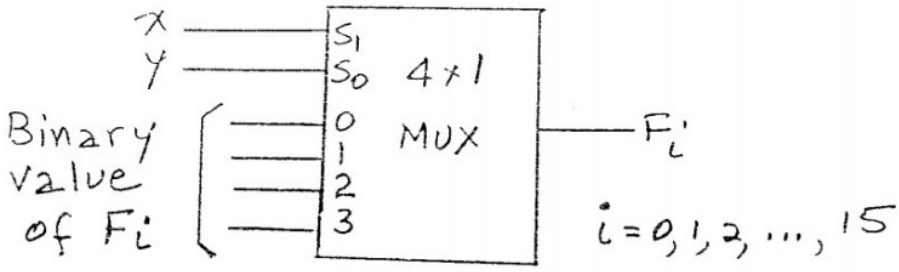




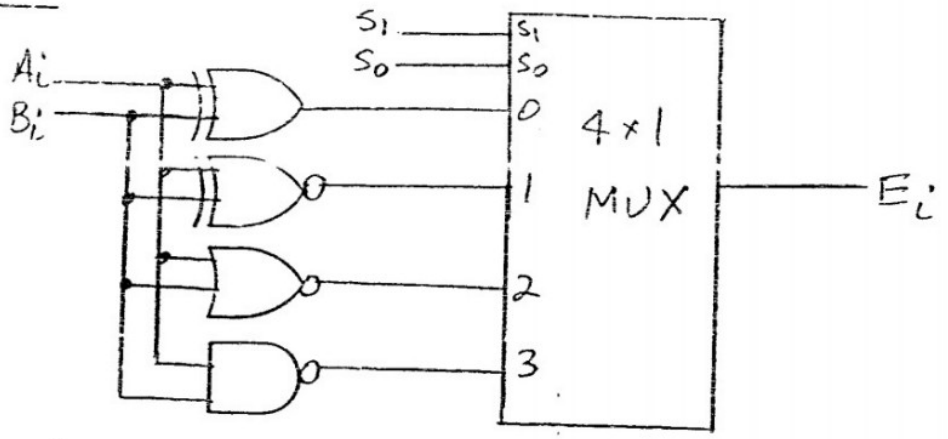



Scanned by CamScanner

4-16




4-17




4-18

(a) A = 11011001
B = 10110100 ⊕
A ← A⊕B 01101101

A = 11011001
B = 11111101 (OR)
11111101 A ← AVB

4-19

(2) AR = 11110010
BR = 11111111 (+)
AR = 11110001 BR = 11111111 CR = 10111001 DR = 11101010

(b) CR = 10111001
DR = 11101010 (AND)
CR = 10101000

BR = 11111111
+ 1
BR = 00000000 AR = 11110001 DR = 11101010

(c) AR = 11110001
CR = 10101000 (−)
AR = 01001001 ; BR = 00000000 ; CR = 10101000 ; DR = 11101010





Scanned by CamScanner

4-20

R = 10011100

Arithmetic shift right : 11001110

Arithmetic shift left : 00111000 overflow because a negative number changed to positive

4-21

R = 11011101

logical shift left : 10111010

Circular shift right : 01011101

logical shift right : 00101110

Circular shift left : 01011100

4-22

S=1 Shift left

$A_0\ A_1\ A_2\ A_3\ I_L$

$H = \begin{matrix} 1 & 0 & 0 & 1 & 0 \\ \swarrow & \swarrow & \swarrow & \swarrow & \swarrow \\ 0 & 0 & 1 & 0 & \end{matrix}$ shift left

4-23

(a) Cannot complement and increment the same register at the same time.

(b) Cannot transfer two different values ($R_2$ and $R_3$) to the same register (R1) at the same time.

(c) Cannot transfer a new value into a register (PC) and increment the original value by one at the same time.





Scanned by CamScanner