Name ⇒ SHUBHAM

Notes ⇒ Computer Organisation

Course ⇒ BCA from HPU

Unit - I

Data Representation

Number System

1) Decimal Num. System.
   Total number = 10 from (0 - 9) Base 10

2) Binary Numbers.
   (0, 1) Base 2

3) octal
   (0, 1, 2, 3, 4, 5, 6, 7) Base 8

4) Hexa Decimal
   (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

1) Conversion from
   Binary to octal $(2)^3$

1) $(1010)_2 \rightarrow (12)_8$

3 group

$\underline{001}\ \underline{010}$

$(1\ \ 2)_8$

2) $(110\,010\,110.101010)_2 = (\quad)_8$

$(6\ 2\ 6.5\ 2)_8$

octal Table

0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Base 8 → Base 2

2) Conversion from Octal (0-7) to Binary

$(67)_8 = (110\,111)_2$

$(67.32)_8 = (110\,111.011\,010)_2$

octal table

0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

3. 1) Conversion from Binary (Base 2) to Hexadecimal (Base 16)

$(1101\,0110)_2 = (D6)_{16}$

$(1101\,0110.1101) = (D6.D)_{16}$

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 = A	1	0	1	0
11 = B	1	0	1	1
12 = C	1	1	0	0
13 = D	1	1	0	1
14 = E	1	1	1	0
15 = F	1	1	1	1

note : अगर हमें कहीं Alphabet मिलता है तो हमें उसे पहले number में Convert करके लिखना होगा तभी हम operations perform कर सकते हैं।

Conversion from Hexadecimal to Binary

$(EF2)_{16} = (1110\,1111\,0010)_2$

$(CF2.D)_{16} = (1100\,1111\,0010.1101)_2$

Conversion from Octal to Hexadecimal

$(26)_8 = (\quad)_{16}$

$(26)_8 = (010\,110)_2 = (16)_{16}$

Conversion from Hexadecimal to Octal

$(AE)_{16} = (\quad)_8$

$(AE) = (1010\,1110)_2 = (256)_8$

IV Conversion from Decimal to Binary

① $(16)_{10} = (10000)_2$

2	16	
2	8	0
2	4	0
2	2	0
	1	0

1 0 0 0 0
2^4 2^3 2^2 2^1 2^0
16 + 0 + 0 + 0 + 0 = 16

V Conversion from Decimal to Octal

① $(16)_{10} = (20)_8$

8	16	
	2	0

2 0
8^1 8^0
16 + 0 = 16

VI Conversion from Decimal to Hexadecimal

$(256)_{10} = (100)_{16}$

16	256	
16	16	0
	1	0

1 0 0
16^2 16^1 16^0
256 + 0 + 0 = 256

⇒ Conversion from Decimal to Binary

$(.25)_{10} = (.01)_2$

.25 × 2 = 0.50
.50 × 2 = 1.00

0 1
2^{-1} 2^{-2}
0 + 1 × 1/4 = .25

Integer	Fraction part
0 ↓	.50
1	.00

$(.375)_{10} = (0.11)_2$

.375 × 2 = 0.750
.750 × 2 = 1.500
.500 × 2 = 1.00

⇒ Conversion from Decimal to octal

$[416.12]_{10} = (640.075341217 2)_8$

8	416	
8	52	0
8	6	4
	0	6

52 / 8 = 6 + 4/8

Now Fractional part

.12 × 8 = 0.96
.96 × 8 = 7.68
.68 × 8 = 5.44
.44 × 8 = 3.52
.52 × 8 = 4.16

.16 × 8 = 1.28
.28 × 8 = 2.24
.24 × 8 = 1.92
.92 × 8 = 7.36
.36 × 8 = 2.88

⇒ Conversion from Decimal to Hexadecimal.

$[106.0664]_{10} = (6A.10FF9)$

16	106	
16	6	10 → A in table of Hexadecimal
	0	6

$(106)_{16} = (6A)_{16}$

$(.0664)_{10} = (10FF9)_{16}$ [∵ 15 = F in table]

- $.0664 \times 16 = 1.0624$
- $.0624 \times 16 = 0.9984$
- $.9984 \times 16 = 15.9744$
- $.9744 \times 16 = 15.5904$
- $.5904 \times 16 = 9.4464$

3
16
6
96
⋮
610

VII Conversion from Binary to Decimal

⇒ $(110111)_2 = [\ 55\ ]_{10}$

$(110111)_2 = 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0$

$= 32 + 16 + 0 + 4 + 2 + 1 = 54 + 1$

$= 55$

1
32
16
4
2
54

⇒ $(11101.011)_2 = [\ \ ]_{10}$

$(11101) = 1\times2^4 + 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$= 16 + 8 + 4 + 0 + 1 = 29$

$(.011) = \frac{0}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} = 0 + 0.25 + 0.125$

$= 0 + \frac{1}{4} + \frac{1}{8} = 0.25 + 0.125 = 0.375$

0.125
$\frac{1}{4} + \frac{1}{8}$
0.25 + 0.125
0.125
0.25
0.375

VIII Conversion from octal to decimal

Octal to decimal

1) $[376]_8 = [254]_{10}$

$[376]_8 = 3\times8^2 + 7\times8^1 + 6\times8^0$

$= 3\times64 + 56 + 6$

~~$= 48 + 56 + 6 =$~~

$= 192 + 56 + 6 = 254$

2) Convert $[0.4051]_8$ to decimal

$[0.4051]_8 = [0.525]_{10}$

$[0.4051]_8 = 4\times\frac{1}{8^1} + 0\times\frac{1}{8^2} + 5\times\frac{1}{8^3} + 1\times\frac{1}{8^4}$

$= \frac{4}{8} + \frac{1}{64} + \frac{5}{512} + \frac{1}{4096}$

$= 0.5 + 0.0156 + 0.0097 + 0.00024$

$= [0.525]_{10} \approx [0.510009765625]$

0.00024
0.0097
0.0156
0.5
0.52554

16 | 3 | 48 | 56 | 48 | 6 | 1 | 0 | 64 | 3 | 192 | 11 | 192 | 54 | 2 | 2.54 | 3 | 4 | 8 | 512 | 8 | 4096 | 0.010 | 0.010 | 0.009

IX Conversion from Hexadecimal to Decimal

1) $[B6A]_H = [2922]_{10}$ do not write this in exam

$[B6A]_H = [11\ 6\ 10]_{16} = [2922]_{10}$

In exam

$= 11\times16^2 + 6\times16^1 + 10\times16^0$

$= 11\times256 + 96 + 10$

$= 2816 + 96 + 10$

$= [2922]_{16}$

2) $[AF.2F]_{16} = [\quad]$ do not write this in exam

$[AF.2F]_{16} = [10\ 15 . 2\ 15]$

In exam

$[AF.2F]_{16} = [10\times16^1 + 15\times16^0 + 2\times\frac{1}{16^1} + 15\times\frac{1}{16^2}$

$= [160 + 15 + \frac{1}{8} + \frac{15}{256}]$

$= [175 + 0.125 + 0.058]$

$= [175.183$

0.125
0.058
0.183

2560 | 256 | 5 | 2693 | 2816 | 96 | 10 | 2922 | 2560 | 160 | 15 | 176 | 16 | 16 | 96 | 16 x | 256 | 16 | 6 | 96 | 256 | 11 | 256 | 256 | 2816 | AB5 | 10 | 15 | 10 | 16 | 16 | 256 | 2560 + 113 + 5 | 15 | 15 | 183

— Binary Coded Decimal Numbers:-

	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

$8\times0 + 4\times0 + 2\times0 + 1\times0$

$8\times0 + 4\times0 + 2\times1 + 1\times0 = 2$

$8\times0 + 4\times0 + 2\times1 + 1\times1 = 3$

$2 . 3 = 4$

0010 0011

BCD Code

0010 0011

$8\times1 + 0\times4 + 2\times0 + 1\times1 = 8+1 = 9$

The representation of decimal numbers (0 - 9) into binary is called binary coded decimal.

BCD Addition

add (binary addition)

1) Sum ≤ 9 Carry = 0 [Correct] [valid BCD]

2) Sum ≤ 9 Carry = 1 incorrect then add [Sum + 0110 (6)]

3) Sum > 9 Carry = 0 incorrect then add [sum + 0110 (6)]

Ex $2 + 4 = 0010 + 0100$

$\checkmark\ 6 = 0110$

Ans is correct satisfied case 1)

8	4	2	1	
0	0	1	0	← 2
0	1	0	0	← 4

0010
0100
0110

Ex 2 2+8

0010
+1000
1010 = 10

It is incorrect it satisfied case 3>

then add 0110

1010
+ 0110
10000 = 0001 0000

Now 2+8 = 10 is the correct answer

Ex 3 23+22

2 3 → 0010 0011
22 → 0010 0010

Now add
23 → 0010 0011
22 → 0010 0010
0100 0101
4 5

Ans = 45 is correct

Ex 4 55+55

55 → 0101 0101

now add 5> 0101 0101
5> 0101 0101
1010 1010
It is greater than 9

1010 1010
+0110 0110
10001 0000

It is greater than 9
So add 0110 in it

= 0001 0001 0000
1 1 0 = 110

0001 0001 0000 is the correct answer

Binary Arithmetic

1> Binary Addition

rules
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 (Carry Bit)

1010 = 10
+ 0111 = 7
10001 = 17

2> Binary Subtraction

0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
0 - 1 = 1
10 - 1 = 1

10 1 0 = 10
- 0 1 1 1 = 7
0 0 1 1 = 3

2 + 1 = 3

1 0 1 0 = 8
0 1 1 1 = 7
0 0 0 1 = 1

1010
0111
0011

(3) Binary Multiplication

```
     101        1×1 = 1
    × 11        1×0 = 0
    ----        0×1 = 0
     101        0×0 = 0
  + 101×
   -----
    1111
```

5 × 3 = 15

(4) Binary Division

```
          11
100 ) 1100
     -100↓
      ×100
      -100
      ----
       ×××
```

1's Complement and 2's Complement

1's Complement subtraction

1's Complement

flip

Binary no. → $(7)_{10} = (0111)_2$

1's Complement → 1000 1's Complement

1's Complement of a binary no is obtained by flipping all bits.

7 → 0111 → 1000

2's Complement → 7 → (0111) → 1000 + 1 = 1001

Signed / magnitude refer.

positive → 0
−ve → 1

7 → 0111 → 1000

Sign	number Binary
0	0111

−7 | 1 | 0111 |

1's Comp → 1 1000

2's Comp → 1 1001

7 → 10111
−7 → 1 1000 (1's)
−7 → 1 1001 (2's)

This is for changing binary to dec and dec to binar

1 00101 (−5)
−ve 5

0 0101 (+5)
+ve 5

−5 = 1 101
= 1 010 ← 1's Complement

in −ve first bit don't change

Addition of +ve and -ve number when +ve is greater

1's Complement Subtraction

Ex → $(1100)_2 - (1010)_2$

(12) − (10) = (2)

Step 1 — 1's Complement of 1010 is 0101

Step 2

```
  1100
+ 0101
------
 10001
    +1
------
 0010   = 2
```

Carry → 1

result → 0010

① Take ones Complement of II no.
② add
③ check Carry
- 1 is Carry then add 1 to the result
- no Carry result is -ve and in 1's Complement

when -ve number is greater

Ex ⇒ $(1010)_2 - (1100)_2$

1's complement

Step 1 ⇒ 0011

Step 2 ⇒

```
  1010
+ 0011
------
  1101   1's Complement
```

There is no Carray
so this result is -ve and in 1's Complement
= − 0010 Binary number
= −2

1's Complement addition

~~Ex 40 + 9~~
~~(101000) + (1001)~~

Ex −0110 and −0111

Solution

−0110 ⇒ 1001 (1's Complement)
−0111 ⇒ 1000 (1's Complement)

```
  1001
+ 1000
------
 10001
    +1   ← 1 Carry
------
 10010
```

1's Complement of 0010 is 1101 and the sign bit is 1
Hence the required sum is −1101

Ex 0110 and 0111

Sol

0110 ⇒
0111 ⇒

6
7
13

```
0110
0111
----
1101
```

2's Complement Subtraction

[when the no is greater]

7 → 0111

2 → 0010

$A - B = (0111)_2 - (0010)_2$

0010

2's → 1101

+1

2's 1110

① Take 2's Complement of no to be subtracted

② Perform addition

③ check if there is end carry (1) then discard it and you got the answer otherwise answer is negative take 2's Complement again

add 0111

+ 1110

10101

discard carry and you got the answer

ans = 0101 = 5

[when -ve number is greater]

7 → 0111

2 → 0010

$B - A = (0010)_2 - (0111)_2$

0110

2's → 1000

+1

2's 1001

add 0010

+1001

1011

take 2's Complement again

0100

+1

0101

Ans = 0101

= -5

- 0011 and - 0101

- 0011 => 1 1101 (2's Complement)

- 0101 => 1 1011 (2's Complement)

(Adding 1 bit(carry)) 1 1000

2's Complement of 1000 is 0111
+1
1000

9's and 10's Complement

Understanding how to do 9's and 10's Complement of a Decimal Number

1) How to find the 9's Complement?
=> Subtract each digit of a decimal number from 9

Ex
Q) find the 9's Complement of 78
Sol

99
- 78
21

21 = 9's Complement of 78

2) How to find the 10's Complement
=> add 1 to the 9's Complement of the given number.

Ex
Q) find the 10's Complement of 78
Sol

99
- 78
21

21 = 9's Complement of 78

To find 10's Complement add 1 into the 9's Complement

21
+1
22

22 = 10's Complement of 78

Subtraction using 9's Complement

① Case 1234 − 1000

9's

$10^n - 1$

n = no. of digits

$10^4 - 1 = 9999$

9999 − 1000 = 8999

```
  1234
+ 8999
 (1)0233
     +1
   234   Ans
```

② Case 1234 − 2000

↓ 9's

7999

```
  1234
+ 7999
  9233  ← 9's
- 0766
```

Ans −766

Subtraction by 10's Complement

Case ①

$(215)_{10} - (155)_{10}$

↓ 9's + 1 = 10's

```
  999      844
- 155      + 1
  844      845   ← 10's of 155
```

```
  215
  845
 1060
```

← carry

= 60 Ans

Case ② 155 − 215 → 10's

```
  999
  215
  784
   +1
  785  ← 10's
```

```
  155
  785
  940
```

Take 10's Complement

```
  999      59
  940      +1
  059      60
```

Ans = −60

Floating point Numbers − Representation

Small fraction −

0.0000976 → 9.76×10^{-5}

97600000 → 9.76×10^{7}

General structure

1) Sign
2) Mantissa
3) Exponent

$$\pm M \times B^{\pm E}$$

2 − Binary
10 − Decimal

Represent in floating point format:

⇒ 111 0111 . 10010

$1.110111|0010 \times 2^{6}$

$1.1101110010 \times 2^{110}$

1) Sign = 0
2) Mantissa = 1101110010
3) Exponent = 110

6 = 110 in binary

IEEE

IEEE Standards of floating Point Representation

Single Precision − 32 bit
Double Precision − 64 bit

Single Precision

Floating pt. No. → $\pm M \times B^{\pm E}$

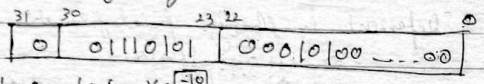
Single Precision

Sign: 0 → +ve, 1 → −ve

8 bit signed Exp. in excess −127 Rep.

23 bit Mantissa

Q) Represent the floating point in the IEEE Standard for given no: 1.0001010×2^{-10}



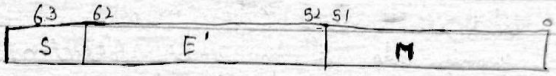
⇒ 1.0001010×2^{-10}

S = 0

M = 00010100

E' = E + bias = −10 + 127 = $117_{10} \leftrightarrow (1110101)_2$

Double Precision – 64 bit



0 → +, 1 → −

11-bit Excess −1023

8 bit 127

52-bit Mantissa

$\pm M \times B^{\pm E}$

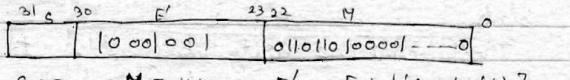
Q) Represent 1460.125_{10} in single & double precision format.

$1460.125_{10} \leftrightarrow (10110110100.001)_2$

Normalise the no.

$1.0110110100001 \times 2^{10}$

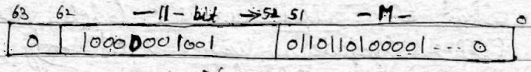
Single Precision



S = 0, M = 01101101..., E' = E + bias = 10 + 127

$= 137_{10} \leftrightarrow 10001001_2$

Double Precision



S = 0, M = 011..., E' = E + bias = 10 + 1023

$= 1033_{10} \leftrightarrow 10000001001$

Q) what is Hamming Code

Ans Hamming Code is an error correction system that can detect and correct errors when data is stored or transmitted. It requires adding additional parity bits with the data. It is commonly used in error correction code (ECC) RAM. It's named after its inventor. Richard W. Hamming.

Hamming Code can correct single-bit errors and detect the presence of two-bit errors in a data block.

The amount of parity data added to Hamming Code is given by the formula $2^P \geq d + P + 1$, where P is the number of parity bits and d is the number of data bits.

Q) what is alphanumeric Codes.

Ans Earlier Computers were used only for the purpose of Calculations i.e. they were only used as a Calculating device. But now Computers are not just used for numeric representations, they are also used to represent information such as names, addresses, item descriptions etc. Such information is represented using letters and symbols. Computer is a digital system and can only deal with 1's and 0's, So to deal with letters and symbols they use alphanumeric Codes.

BCD Arithmetic ⇒ In Computing and electronic systems, binary-Coded decimal (BCD) is a class of binary encodings of decimal numbers where each digit is represented by a fixed number of bits, usually four or eight.
Sometimes, special bit patterns are used for a sign or other indications

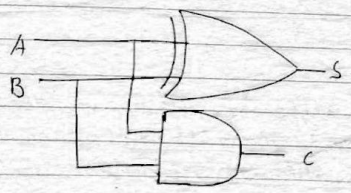
Q) Half Adder

A Half adder is a type of Adder, an electronic circuit that performs the addition of numbers. The half adder is able to add two single binary digits and provide the output plus a Carry value. It has two inputs, Called A and B, and two outputs S (Sum) and C (Carry).

The adder works by Combining the operations of basic logic gates, with the simplest form using only a XOR and an AND gate. This Can also be Converted into a Circuit that only has AND, OR

and NOT gates. This is especially useful since these three simpler logic gate ICs (integrated circuits) are more common and available than the XOR IC. though this might result in a bigger circuit since three different chips are used instead of just one.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

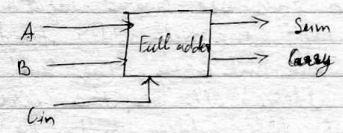


XOR $\Rightarrow A\bar{B} + \bar{A}B$

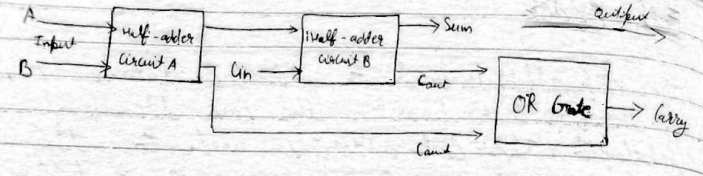
Full Adder

The Half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and Carry C. The full adder has three input states and two output states i.e, Sum and Carry

Block diagram



A	B	Cin	Sum Output	Carry Output
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



The above block diagram describes the construction of the full adder circuit. In the above circuit, there are two half adder circuits that are combined using the OR gate. The first half adder has two single-bit binary input A and B. As we know that, the half adder produces two outputs, i.e, Sum and Carry. The 'Sum' output of the first adder will be the first input of the second half adder, and the 'Carry' ~~output of the first adder will be the second input of the second half adder~~.
Cin is the third input is the second input of the second half adder. The second half adder will again provide 'Sum' and 'Carry'. The final outcome of the full adder circuit is the 'Sum' bit. In order to find the final output of the 'Carry', we provide the 'Carry' output of the first and second adder into the OR gate. The outcome of the OR gate will be the final carry out of the full adder circuit.

Hexadecimal alphanumeric representation is done in computer.
Computer programmers can use a series of 0's and 1's to represent any character they wish. For example, in binary, the letter 'A' would be written as 01000001. Another way computer programmers represent alphanumeric characters is to use ASCII. ASCII stands for American standard code for information interchange.

Unit-II

Register Transfer Language

Micro operations ⇒ operations executed on data stored in registers are called micro-operations.

A micro operation is an elementary operation performed on information stored in one or more registers. Result of operation may replace the previous binary information of a register or may be transfered to another register.

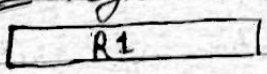
Eg of micro-operations are shift, Count, Clear and load.

Register Transfer Language ⇒ The symbolic notation used to describe the micro-operation transfers among registers is called a register transfer language.

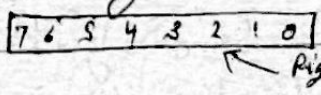
It is convenient tool for describing the internal organisation of digital Computers in Concise & precise manner. It can also be used to facilitate the design process of digital systems

 # Register transfer → Is defined as the transfer of information between registers with the help of Common bus.

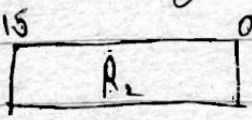
(a) Register R



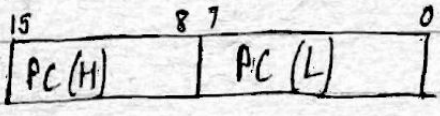
(b) Showing Individual bits



(c) Numbering of bits



(d) Divided into two parts



Information transfer from one register to another is designated in symbolic form by means of replacement operator.

$$R_2 \leftarrow R_1$$

Under a predetermined control condition

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

Symbol → $P: R_2 \leftarrow R_1$

Symbol	Description	
letters (and numerals)	Denotes a register	R_1, R_2, PL, IR
Parentheses ( )	Denotes a part of register	PC(0-7), PC(IL)
Arrow ←	" a transfer of info.	$R_2 \leftarrow R_1$
Comma ,	Separates two micro-operations	$R_2 \leftarrow R_1$, $R_3 \leftarrow R_1$

Arithmetic Micro-operations

The basic arithmetic operation are addition, Substraction, Complement, increament and decrement.

Some symbolic representation of arithmetic microoperation is

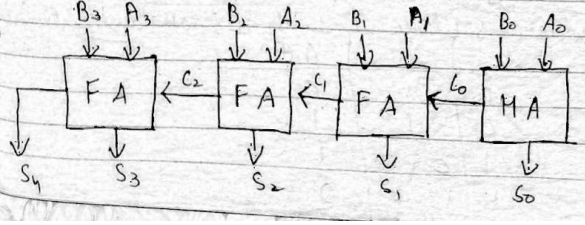
operation	
Contents of R_1 plus R_2 transferred to R_3	$R_3 \leftarrow R_1 + R_2$
Contents of R_1 minus R_2 transferred to R_3	$R_3 \leftarrow R_1 - R_2$
1's Complement of the contents of R_2	$R_2 \leftarrow \overline{R_2}$
2's Complement of contents of R_2	$R_2 \leftarrow \overline{R_2} + 1$
Increment	$A \leftarrow A+1$
Decrement	$B \leftarrow B-1$

Multiplication and division are valid operation but it is not the basic micro-operation.

7.9 4- bit binary Adder / binary Adder

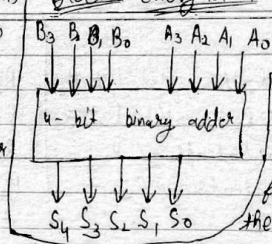
A 1101
B 1011

A → $A_3 A_2 A_1 A_0$ = 1101
B → 1011
$B_3 B_2 B_1 B_0$
1100 (2)
$S_4 S_3 S_2 S_1 S_0$



The digital circuit that generates the arithmetic addition of two binary numbers of any length is called binary adder. Also, the digital circuit that performs the addition of two binary bits and a previous carry is called a full-adder

Block diagram of 4 bit binary adder



The binary adder circuit is constructured using full-adder circuits with the output carry from one full adder is connected to the input carry of the next full adder. An n-bit binary adder requires n-full-adders, where in data bits for the A inputs come from one register (say R_1) and the in data bits for the B inputs come from another register (say R_2) and the result can be stored into any of these two registers (R_1 or R_2) replacing its previous content or it stored in the third register

7.9.2 Binary - Adder Subtractor

The substraction of binary numbers can be done most conveniently by means of complements.

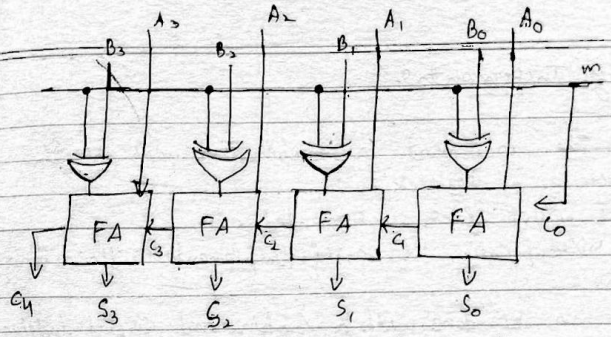
A-B can be done by taking the 2's complement of B and adding it to A

The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits.

The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry

adder - Subtractor with Ex-OR and full-adder

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0



mode I/p m controls the operation

m = 0 the circuit is an adder

m = 1 the circuit becomes a Subtractor

Each Ex-or gate receives I/p m and one of the I/p of B

when m = 0, we have $B \oplus 0 = B$, the full adder receive the value of B, the input carry is 0, and the circuit performs A Plus B

when m = 1, we have $B \oplus 1 = B'$ and $C_0 = 1$ The B inputs are all complemented and A 1 is added through the input carry. The circuit performs the operation A Plus the 2's complement of B.

Binary Incrementor

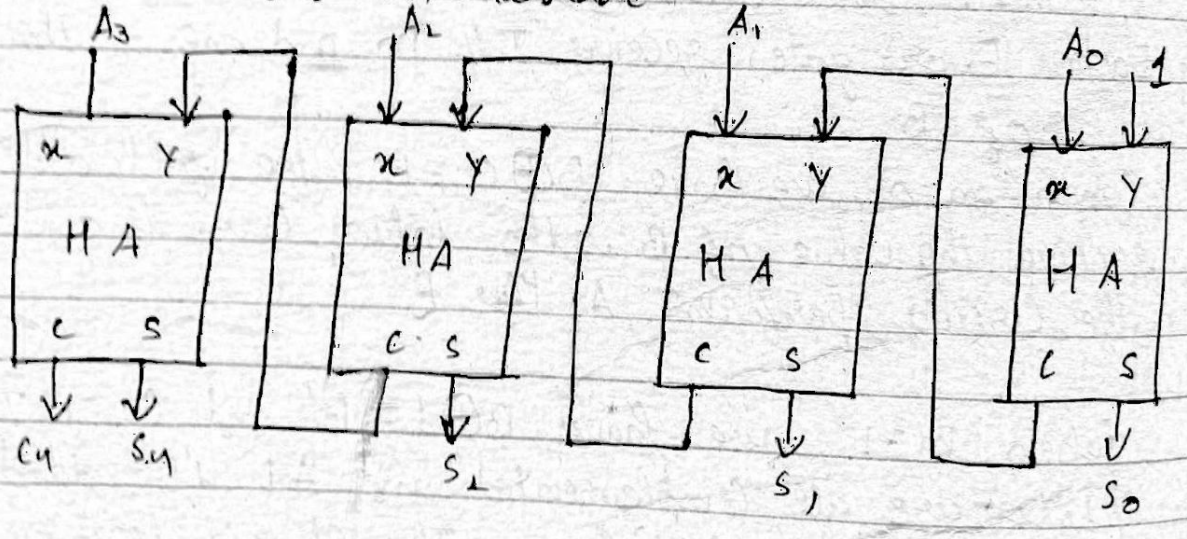
The increment microoperation adds one to a number in a register.

0110 after incremented 0111

This microoperation is easily implemented with a binary Counter.

This can be done with a combinational circuit independent of a particular register.

This can be accumplished by means of half adders connected in cascade.



$A_3\ A_2\ A_1\ A_0$ → no. to be incremented

$S_3\ S_2\ S_1\ S_0$ → output

The o/p carry C_4 will be 1 only after incrementing binary 1111. This also cause o/p. So through S_3 to gate 0.

Logic Micro-operations

A group of bits called strings is stored in the register, the logic micro operation consider each bit of register as compared to the arithmetic micro-operation where we take whole content of register to perform arithmetic operation.

Symbol for AND gate $C \leftarrow A \wedge B$

OR gate $C \leftarrow A \vee B$

Logical micro-operation are those microoperations that are used to perform logical operations such as AND, OR, NOT, etc.

The micro-operation which perform operation such as AND, OR, NOT, XOR, etc. on individual pairs of bits stored in registers are said to be logic micro operations.

The XOR (exclusive OR) operation between the content of two registers R_1 and R_2 can be represented as:

$P: R_1 \leftarrow R1 \oplus R_2$.

Symbol $\oplus$ represent the XOR operation. Also, special Symbols are used for logic microoperations OR, AND and Complement operations.

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

($x\,y'$; $x\,y$)

F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1

$F_0 = 0000 = 0$

$F_1 = 0001 = x \cdot y$

$F_2 = 0010 = x \cdot y'$

$F_3 = 0011 = xy' + x \cdot y = x$

$F_4 = x' \cdot y = 0101$

$F_5 = 0101 = x'y + xy = y \cdot 1$

$F_6 = x' \cdot y + x \cdot y'$

$F_7 =$

$x \backslash y$	0	1
0		1
1	1	1

$x + y$

$F_8 = 1000 = x' \cdot y' = (x+y)'$

$F_9 = x' \cdot y' + x \cdot y$

$F_{10} = x' \cdot y' + x \cdot y' = (x' + x)y'$
$= y' \cdot 1$

$F_{11} =$

$x \backslash y$	0	1
0	1	
1	1	1

$x + y'$

$F_{12} = x' \cdot y' + x' \cdot y = x' \cdot 1$

$F_{13} =$

$x \backslash y$	0	1
0	1	1
1		1

$x' + y$

$F_{14} =$

$x \backslash y$	0	1
0	1	1
1	1	

$x' + y'$

$F_{15} = 1$

▷ list of logic micro-operations

Sixteen logic Microoperations

Boolean Function	Logic microoperations	meaning
$F_0 = 0$	$F \leftarrow 0$	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
$F_6 = yx' + y' \cdot x$	$F \leftarrow A \oplus B$	Exclusive -OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow (A \vee B)'$	NOR
$F_9 = x' \cdot y' + x \cdot y$	$F \leftarrow$	
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (x' + y')$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ All 1's	Set all to 1's

$x' + y' = (x\,y)'$ [demorgans theorem ke according]

Arithmetic logic shift Unit

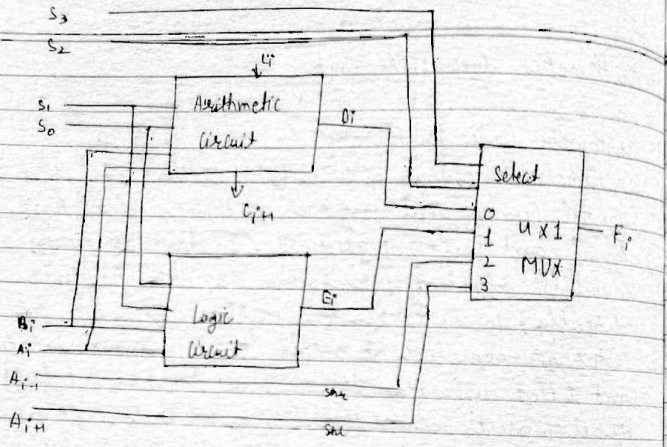
Arithmetic logic shift Unit (ALSU) is a part of Arithmetic Logic Unit of a Computer System. It is a digital circuit that performs arithmetic calculations, logical and shift operations. Instead of having invidual registers performing the micro-operations directly, Computer system employs a number of storage registers connected to a common operational Unit called an arithmetic logic unit, abbreviated ALU. ALSU consists of arithmetic, logical and shift circuits. In each stage, the circuit can perform 8 arithmetic operations, 16 logical operations and 2 shift operations.

Characteristics of the ALSU

1) Arithmetic logic shift unit in digital circuit performs all the logical and arithmetic perations.

2) The operations like addition, subtraction, multiplication and division are referred as Arithmetic operations.

3) The logical operations refer to operations on numbers and special character operations.



Functions of the ALSU

A particular micro-operation is selected with inputs S_1 and S_0. A 4x1 multiplexer at the output chooses between an arithmetic output in E_i and a logic output in H_i. The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.

The circuit must be repeated n times for an n-bit ALU. The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence. The input carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations.

Shift Micro-operation

Shift micro-operation transfer binary bits b/w the register serially, this shift operation is also used to perform binary multiplication & binary division.

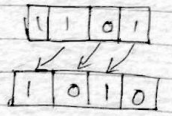
Register can be shift to left or to right There are no conventional symbol for shift operation. The shift micro-operation are represented by Shl & Shr to shift the content to left & to right

The one bit shift is represented by symbolic representation & is given in form of a statement.

$A \leftarrow Shl\ B$ — (1)

$C \leftarrow Shr\ A$ — (2)

The first statement specifies as one bit shift to the left of register B & second statement specifies one bit shift to right of register A

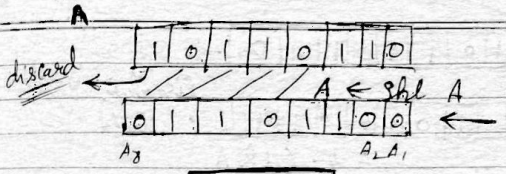


logical — left
— Right

circular — L
— R

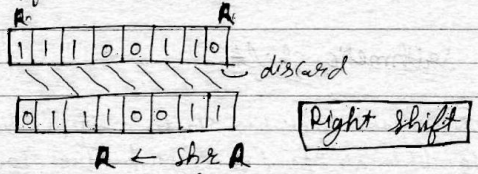
Arithmetic

▷ logical Shift



Left shift

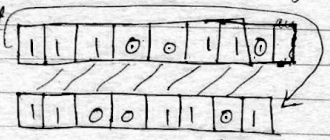
It is the easiest to perform as the bits are simply to the right or to the left. If it is shift left, a 0 bit is transferred at the right most position, through serial input and the left most bits.



Right shift

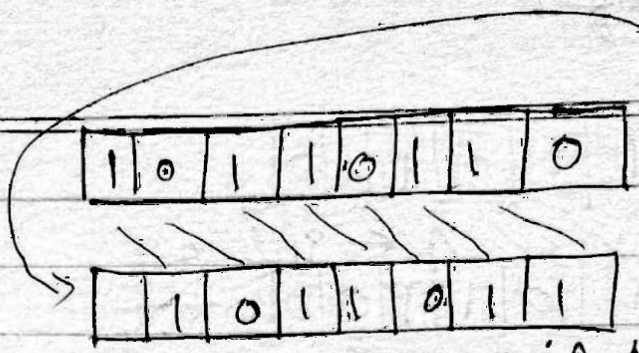
Similarly, if it is shift right, the right most bit is lost and a 0 bit is added at the left most place. The above statements specify a 1-bit shift to the left of the content of register A and a 1-bit to the right of the content of register R respectively.

▷ Circular Shift



also known as rotate operation it simply circulates the bits of the register around the two end. No bit is added or lost.

- If it is left circular shift, the bits are shifted in left direction and the right most bit takes the left most bit position.



Circular Right shift

$A \leftarrow CIRA$

$A \leftarrow shr\ A,\ A_n \leftarrow A_1$

Similarly, if it is right circular shift, the bits shift in right direction and the left most bit takes the right most bit position.

3) Arithmetic Shift

This micro-operation shifts a signed binary number to the left or to the right position. In an arithmetic shift-left, it multiplies a signed binary number by 2 and In an arithmetic shift-right, it divides the number by 2

7.7> Bus & Memory Transfers

A digital Computer has many registers and rather than Connecting wires between all register to transfer information between them, a Common bus is used.

Bus is a path over which information is transfers from any of several sources to any of several destinations

From a register to Bus: Bus ← R

The transfer from bus to register: R_1 ← Bus

The Content of selected register is placed on the Bus and the Content of the bus is loaded into register R1 by activating its load Control input.

① Data Bus:

The most Common bus is the data bus. A data bus Carries data.

- It is an electrical path that Connects the CPU, memory, Input/output devices & secondary storage devices.
- The Bus Contain parallel group of lined.
- The number of lines in bus affects the speed at which the data travels b/w different Components.

② Address Bus

An address Bus carries address information. It is set of wires similar to the data bus but it only connects CPU & memory.

- whenever the processor needs data from the memory. It places the address of data on the address bus.
- The address is carried to the memory where the data from the requested address is fetched & placed on the data bus the data bus carries to CPU.

③ Control Bus

The Control Bus carries control information from the control unit to the other units

- The control information is used for directing the activities of all units
- The control unit control the functioning of other units eg: input/output devices, secondary storage etc.

Three State Bus buffer

⇒ Bus System — address line
Data line
Control line

⇒ A bus can be constructed in 2 ways
→ using multiplexer
→ using three state bus buffer.

Three state bus buffers

It is useful device that allows us to control when current passes through a device & when it does not.

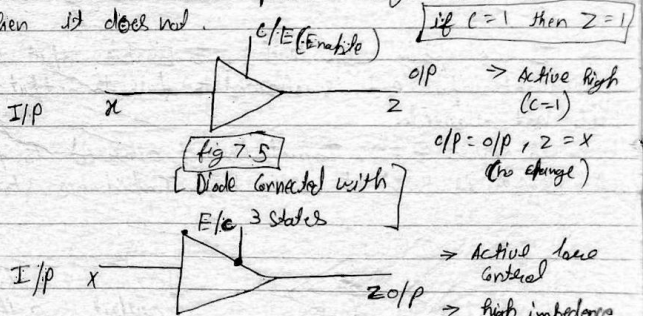
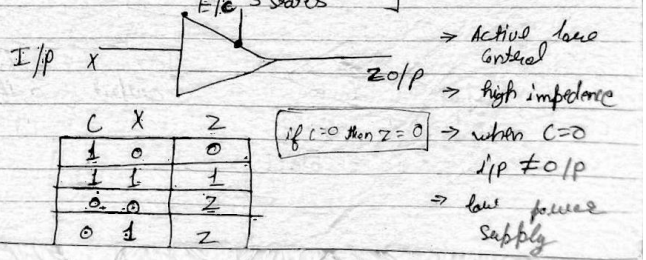


fig 7.5

[Diode connected with E/C 3 states]



C	X	Z
1	0	0
1	1	1
0	0	Z
0	1	Z

if C=0 then Z=0 → when C=0 I/P ≠ O/P

→ low power Supply

→ At a time a bus can write once but can read many data.

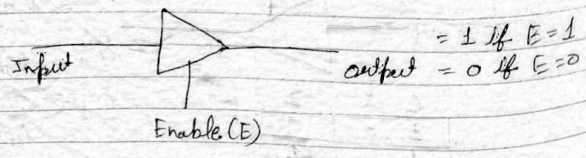
→ As many devices are connected to bus to make the task easier three state bus buffer is used.

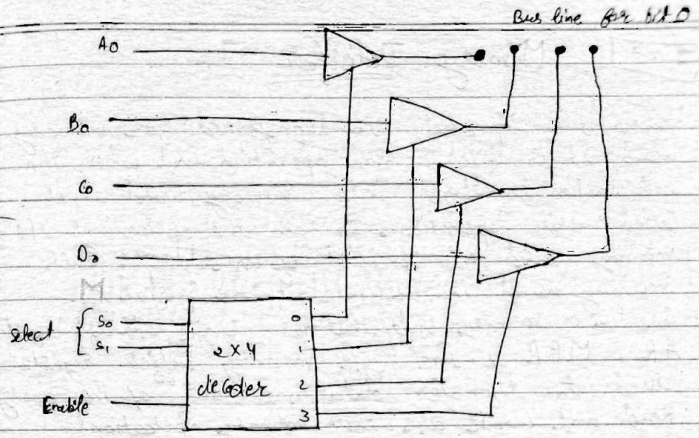
Three state bus Buffers

A bus system can be developed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 and the third state is a high-impedance state. It acts like an open circuit i.e. the output is disconnected and does not have a logic significance. The graphic symbol of a three state buffer is shown in fig 7.5.

If we apply logic level 1, to Enable input, the input present at input end is allowed to pass to output, else it is not allowed.

whenever E=0, Input is not allowed to pass and buffer is said to be in High Impedance state.



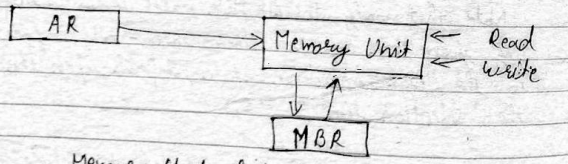


Note that only three-state gates is in active state at any given time and the other three-state buffers remain in the high-impedence state, depend upon the value of the selection lines S_0 & S_1.

The above fig can be extended to include all the four line of the bus. we shall require four buffers for each register. The 0^{th} output of the decoder will be connected to all the four buffer of register A, the 1st output of the decoder will be connected to all the four buffers of the register B, the 2nd output of the decoder will be connected to all the four buffers of register C and 3^{rd} output of the decoder will be connected to all four buffers of the register D.

7.2 Memory Transfer

When the information is transferred for memory word, it is called a read operation and when the information is stored into a memory word it is called write operation. In both the operations the memory word is specified by an address. This memory word is designated by the symbol M. Consider a memory unit that has single address register AR. MBR is the single memory buffer register used to transfer data in or out of the memory. Read and write are two memory operations.



Memory Unit that Communicates with the External Registers

Consider a memory unit that receives the address from the register AR, called address register and the data is to be transferred into register DR, called data register. The read operation can be:

Read : DR ← M[AR]

The above statement causes the information from the memory word M selected by the address AR to be transferred into the destination register DR

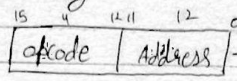
The information is to be transferred from register R_1 into the memory word M at location given by address register AR. The operation can be stated as:

Write : M[AR] ← R_1

UNIT → III

Instruction Codes

⇒ Group of bits that tell the computer what to do basically.

format → 

→ Tells the address of the operand on which the operation is going performed

opcode ⇒ operation
ADD, SUB, MUL

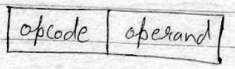
operand ⇒ A, B, C, R_1, R_2

word → 1024 bits = 2^{10} →

Format → 16 bits → 12 bits = address
→ 4 bits = opcode

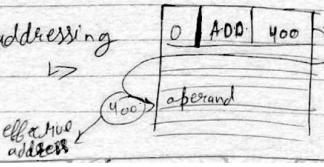
→ Immediate operand
→ Direct addressing (0)
→ Indirect addressing (1)

Immediate operand

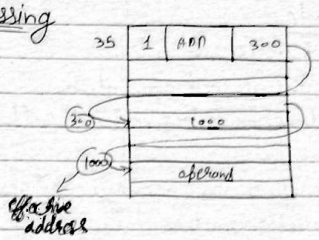


(address acts as a operand)

Direct addressing



Indirect addressing



Addressing mode ⇒ addressing mode is basically a technique used for determining the operand that associates with any given instruction. A multiprocessor mainly functions to execute all the instructions in a group (that their memory stores) in order to conduct any specified task.

out of all the addressing modes, the direct and indirect modes help in specifying the way in which we can access the data and info from a given memory. There exists a major difference between direct and indirect addressing modes.

In the case of a direct mode, the given address field directly refers to the memory location where we have stored the data. Contrary to this, the address field in an indirect mode first refers to the register. After that, it directs to the intended memory location.

Direct Mode

The direct addressing mode contains the concerned operand in the instruction code's address field.

It requires no memory references for accessing the data

Indirect Mode

In the case of an indirect addressing mode, the operand's address stays in the address field of any instruction.

It requires multiple memory references for searching an operand.

IV. Unit

Relative addressing mode

Relative addressing is the technique of addressing instructions and data areas by designating their location in relation to the location counter or to some symbolic location.

What is a Computer instruction?

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A Computer performs tasks on the basis of the instructions provided

An instruction comprises of groups called field. These fields include:

- The operation code (opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e. register or memory location.
- The mode field which specifies how the operand will be located.

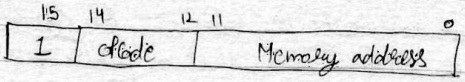
Mode	opcode	operand/address of operand

The basic Computer has 16-bit instruction register (IR) which can denote either memory reference or register or reference or input-output instruction.

1) Memory Reference ⇒ These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode and 1-bit addressing mode for

IR => Instruction Register

direct and indirect addressing.



Example

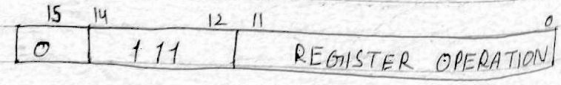
IR register contain = $\overbrace{0001xxxxxxxxxxxx}^{16-bit}$

i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction

Hence, DR ← M[AR]

AC ← AC + DR, SC ← 0

2) Register Reference => These instructions perform operations on registers rather than memory addresses. The IR(14-12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiate it from input/output instructions). The rest 12 bits specify register operation.

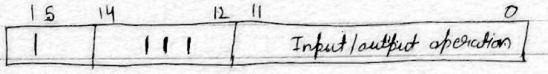


Example

IR register contain = 0111001000000000, i.e CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

Hence, AC ← ~AC

3) Input/output => These instructions are for communication between computer and outside environment. The IR(14-12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



Example

IR register contains = 1111100000000000, i.e INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.

Q what is an instruction code and its parts.

Ans An instruction code is a group of bits that instruct the computer to perform a specific operation. The operation code of an instruction is a group of bits that define operations such as addition, subtraction, shift, complement.

It consists of three parts called fields, which include: The mode field defines how the location of the operand can be found by the computer.

The operation code field or Opcode specifies the operation, such as addition, subtraction, shift, and complement, to be performed.

In general, each instruction code is 16-bit and consists of three parts called field, which include the mode, the opcode, and the address

Instruction Cycle

An instruction cycle, also known as fetch-decode-execute cycle is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer

following are the steps that occur during an instruction cycle.

1. Fetch the instruction
The instruction is fetched from memory address that is stored in PC (Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed

2. Decode the instruction
The instruction in the IR is executed by the decoder.

3. Read the effective address
if the instruction has an indirect address, the effective address is read from the memory otherwise operands are directly read in case of immediate operand instruction.

d) Execute the instruction
The Control Unit passes the information in the form of Control Singnals to the functional Unit of CPU. The result generated is stored in main memory or sent to an output device

The cycle is then repeated by fetching the next instruction. Thus in this way the instruction cycle is repeated continuously

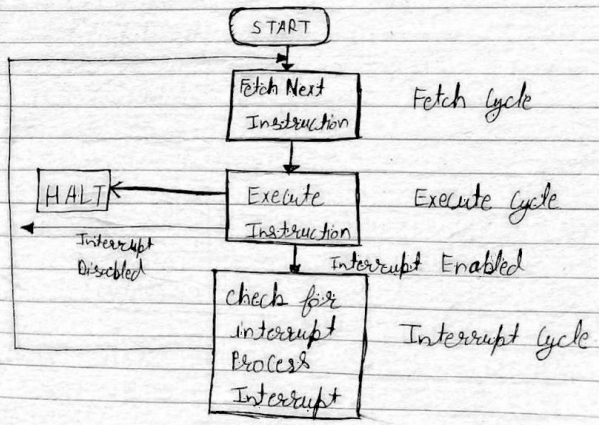
again 2

Decode: The Decode operation is used for interpreting the instructions means the instructions are decoded means the CPU will find out which operation is to be performed on the instructions.

Instruction Cycle

A program stays in the memory unit of the computer and has a sequence of instructions. The program is executed by going through a cycle for each instruction. Each instruction cycle is now sub divided into a sequence of sub cycle of phases. In the basic computer each instruction cycle has the following parts:

a) Fetch an instruction from the memory.
b) Decode the instruction
c) Read the effective address from memory if the instruction has an indirect address.
d) Execute the instruction



Instruction cycle with interrupts

After the completion of step 4, the control goes back to step 1 to fetch, decode and execute the next instruction

This process continues indefinitely unless a HALT instruction is encountered. In an improved instruction execution cycle, we can introduce a third cycle known as the interrupt cycle.

Computer Registers

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

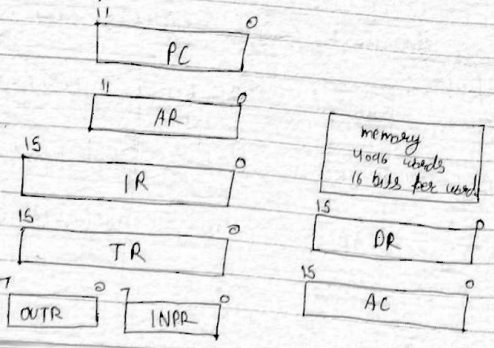
The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Some of the most common registers used in a basic computer:

Register	Symbol	Number of bit	Function
Data register	DR	16	Hold memory operand
Address register	AR	12	Holds address for the memory

Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program Counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input Register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

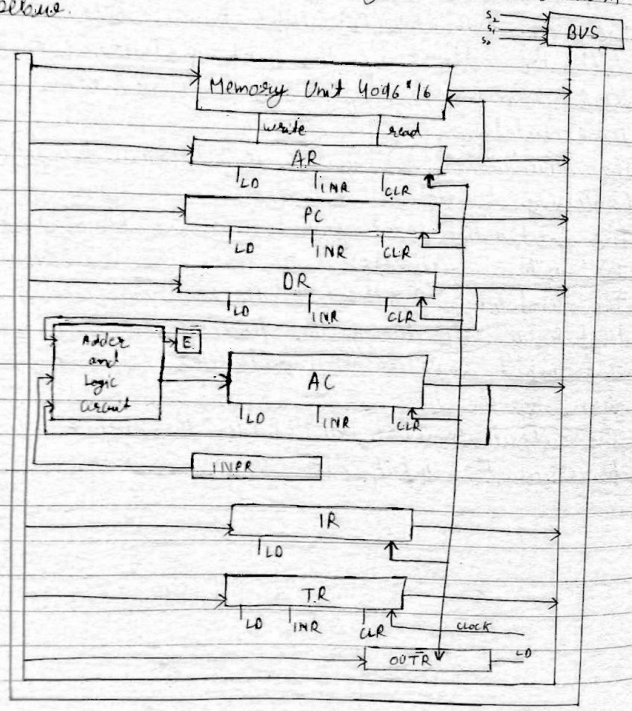
Register and memory configuration of a basic computer.



- The memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The input Registers (IR) holds the input characters given by the user.
- The output Registers (OR) holds the output after processing the input data.

Common Bus System

We shall study the common bus system of a very basic computer in this article. A basic computer has 8 registers, memory unit and a control unit. The diagram of the common bus system is as shown below.



Connections:

The outputs of all the registers except the OUTR (output register) are connected to the common bus. The output selected depends upon the binary value of variable S_2, S_1 and S_0. The lines from common bus are connected to the inputs of the registers and memory. A register receives the information from the bus when its LD (load) input is activated while in case of memory the write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various Registers:

4 register DR, AC, IR and TR have 16 bits and 2 registers AR and PC

~~Kal da Sada Parso da nainda~~ yashpal gatbai

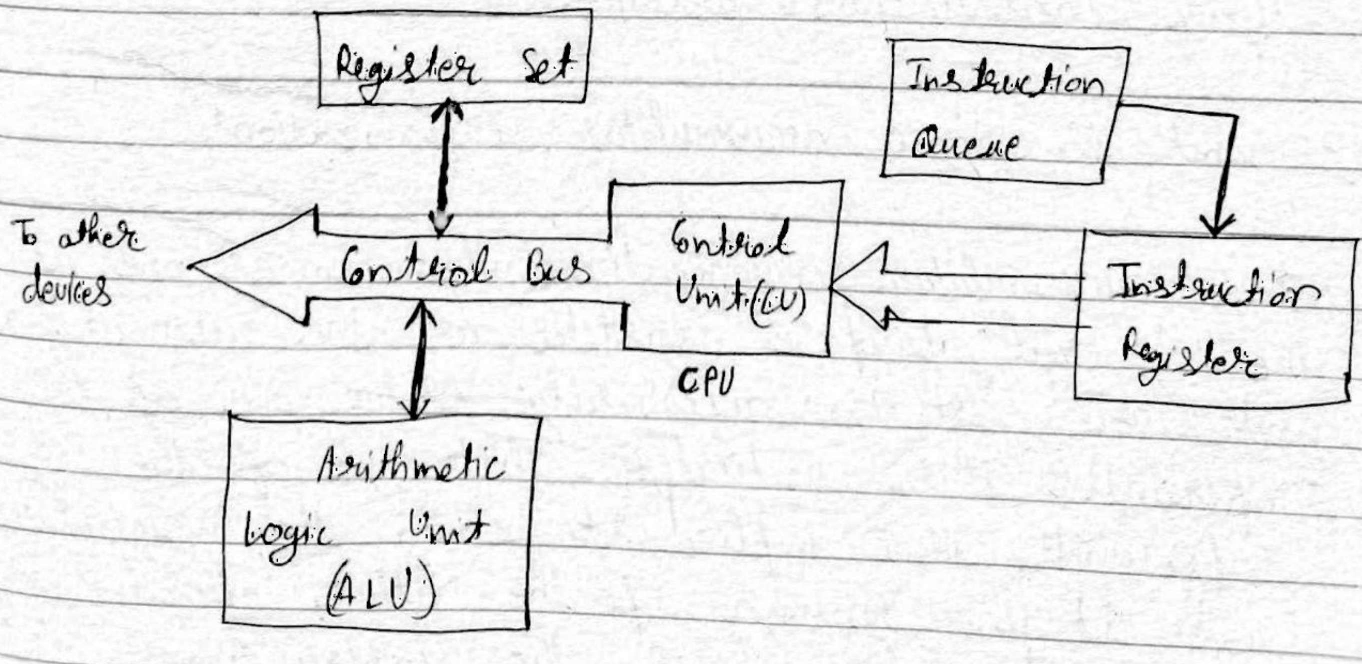
Unit ⇒ IV

I) Central Processing Unit.

The part of the computer that performs the bulk of data processing operations is known as central processing unit (CPU). CPU is the brain of the computer. Although its main function lies in executing programs, it also controls input devices, output devices and other components of a computer. Under its control, programe and data are stored in memory and displayed on the screen.

It is made up of three major parts as shown in the following figure 10.1:

(a) Register Set
(b) ALU (Arithmetic logic Unit)
(c) CU (Control Unit)



Block degram of CPU

Register Set is used to store intermediate results obtained during the execution of instructions. ALU performs the arithmetic and logic micro-operations required for executing the instructions. Control Unit instructs the ALU to performs the desired operation and supervises the transfer of information among different Registers.

Q. What is general general register organisation

A. When we are using multiple general-purpose registers, instead of a single accumulator register in the CPU organization then this type of organization is known as General register-based CPU organization. In this type of organization, the computer uses two or three address fields in their instruction format.

Q. What is single accumulator organization?

Single accumulator organization, which names one of the general purpose registers as the accumulator and uses it to necessarily store one of the operands. This indicates that one of the operands is implied to be in the accumulator and it is enough if the other operand is specified along with the instruction.

Q. What is stack organization in computer architecture?

The computers which use stack-based CPU organization are based on a data structure called a stack. The stack is a list of data words. It uses the Last in First out (LIFO) access method which is the most popular access method in most of the CPU.

features of stack organization

Reverse Polish notation

You are familiar with the arithmetical expressions and their evaluation. Normally, an arithmetical expression is written as follows:

a + b - C * d / e

Note that the binary operations (i.e +, -, *, /) are placed between two operands as is expected. This notation of the arithmetic expressions where binary operations are placed between the operands is called infix notation. The BODMAS (Bracket, of, Division, Multiplication, Addition and Subtraction) rule is applicable as usual.

Though, infix notation is comprehensible and easily amenable to evaluation by humans, it is not as easy for the digital machines to do so as it. It is easier for a computer. The expression using digital system of the form of the expression is slightly modified. In one of the forms, the binary operators are placed at the end of the respective pair of operands. This notation is called Reverse Polish notation or Postfix notation. The BODMAS rule is equally applicable in this case also.

Thus the a+b-C*d/e expression can be written in the postfix form as follows:

a+b-Cd*/e : * postfixed with operand C and d
a+b-Cd*e/ : / postfixed with operand Cd* and e
ab+ -cd*e/ : + postfixed with operand a and b
ab+cd*e/- : - postfixed with operands ab+cd*e/

This is required post fix notation for the given expression.

The motivation behind converting an expression postfix notation is that expressions written in post fix notation are easy to evaluate using a stack.

Let us consider an expression x+y. The plus operation is placed in between the two operands x and y. Such a notation is known as infix notation. If the operator is placed before the operands as +xy, the notation is said to be prefix notation, also known as polish notation. If the operator is placed after the two operands as xy+, the notation is said to be postfix notation, also known as Reverse polish notation as discussed above.

The three notations are:

x + y	infix notation
+ xy	Prefix or Polish notation
xy +	Postfix or reverse Polish notation

How to Convert infix Expression into postfix expression

Here is a simple procedure to convert an expression from infix ~~to~~ ~~form~~ into postfix form. Read through the expression and identify the operator with the highest precedence.

Take the operands of this operator and place it after the two operands in the expression. Treat the two operands and the operator as a single operand. Now parentheses may be used to increase clarity however if used, they must be removed from the final expression

Repeat the above steps until all the operators have been taken care of
This provides the postfix notation of the given infix expression.

Ex example, Suppose that the input expression is a a + b * c + d.

a + b * c + d

1) Scanning through the expression we find that * is the highest precedence operator with b and c as its operands. Thus, we postfix it to get the following expression a + (bc*) + d

2) Scanning again we find the first + to be the next operation that must be postfixed. Its operands are a and (bc*): postfixing this operator gives the following expressions.

(a (bc*) +) + d

3) Postfixing the last + operator with operands (a(bc*)+) and d, we get the following expression

(a(bc*)+d) +

4) Removing the parentheses, the final postfix expression is given below.

abc*+d+

1

9 Convert postfix / Reverse Polish notation form into the Infix form

b> bcd + * a = c+d

ba *

b * a

b * (c+d)

⇒ pqw/ca *+ = p/w = b

p bca *+ = c*a = d

p bd + = b+d = e

p e = e

p = b + d

p = b + c*a

p = (p/w) + (c*a)

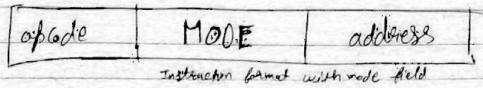
ADDRESSING MODE

For executing an instruction, data is required. This data may be present in the accumulator (AC) or stored in some location in the memory. There are various ways to specify the address of data are more precisely operands. These are known as addressing modes. The techniques for specifying the address of the operand are known as addressing modes. The address of an operand is known as effective address. Each instruction needs data on which it has to perform the specified operation. The operand (data) may be in accumulator, general purpose register or at some specified memory location. Thus, there are various ways of specifying the address of the data, known as addressing modes.

Instruction cycle

(i) Fetch the instruction from the memory
(ii) Decode the instruction
(iii) Execution → Execute the instruction

PC i.e Program Counter keep track of the instructions in the program stored in the memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from the memory. The decoding determines the operation to be performed, the addressing mode of the instruction and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.



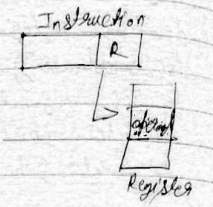
Instruction format with mode field

There are two addressing modes that need no address field at all. They are implied and immediate. Zero - address instructions in a stack - organized computer are implied - mode instruction since the operands are implied to be on the top of the stack.

Direct or absolute addressing

In this mode of addressing, the instruction contains the address of the data in the memory. The operand will be present at some location in the memory and its address in memory is provided in the instruction itself. That is, the address field of the instruction contains the effective address.

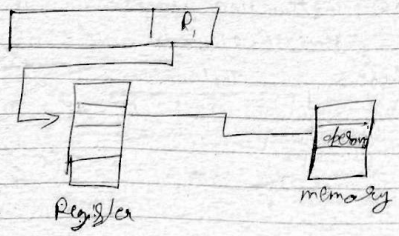
Register Addressing mode



In Register addressing mode, the operands are in registers that resides within the CPU i.e. the content of the register is the operand itself

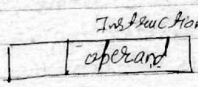
Register indirect mode addressing

The register indirect addressing mode, the instruction specifies the register which contain the address of the operand in memory. The operand is present at some location in memory and its address is contained in a register. In the indirect addressing mode, the instruction specifies the address of this register. That is the operand address is given indirectly by the instruction. Hence, this method is called register indirect Addressing mode



immediate addressing

In this mode, the operand is specified in the instruction itself, i.e. in immediate addressing mode, instruction has an operand field rather than an address field. The operand field contain the actual operand. This mode is useful for initializing registers to a constant value



Relative addressing mode

In relative addressing mode, the contents of program counter are added to the address field of instruction. The sum of these values then provides the effective address of the operand

Instruction Formats

The reference manuals provided with a computer system contain a description of its physical and logical structure. These manuals explain the internal construction of the CPU, including the processor register available and their logical capabilities. All hardware implemented instructions, their binary code format, and a precise definition of each instruction is very well described by these manuals. Given the fact that a computer has a variety of instruction code formats, it is the control unit within the CPU that interprets each instruction code and provides the necessary control functions needed to process the instruction.

An instruction format defines the layout of bits of an instruction in terms of its constituent parts. It has already been explained that the bits of an instruction can be grouped into parts called field. An instruction is a command given to a computer to perform a specified operation on some given data and the format in which the instruction is specified is known as instruction format. In a computer, the instruction format usually consists of 3 fields.

(a) operation code (op code) field. An operation code field that specifies the operation to be performed. It is known as opcode field.

b) Address field: An address field that specifies the designates the register address or a memory address.

c) Mode field: A mode field that specifies the way the operands or the effective address is determined.

A computer can contain multiple instruction formats. It is the duty of the control unit of the CPU to determine the instruction format and send appropriate signals to various parts of the CPU.

Instruction Types

Instructions to a CPU are of different kinds based on the number of operands they contain or require to operate on. These are as follows:

Three address instructions

In three address instruction, two operand addresses are specified and a third address is provided for storing the result. For example ADD R_1, A, B leads to addition of operands at location A and B and the sum of the two operands are stored in Register R_1.

ADD R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL X_1, R_1, R_2	$X \leftarrow R_1 * R_2$

The Symbol M[A] denotes the operand, at memory address symbolized by A. The advantage of the three address format is that in starting programs when evaluating arithmetic expressions.

Two Address Instruction

In this type of instruction addresses of the two operands are specified. The result of the operation is stored in one of the given operand addresses. For example, mov, R_1, R_2 is an example of 2-address instruction, where R_1 and R_2 store addresses

MOV R1, A	$R1 \leftarrow M[A]$
ADD R1, B	$R1 \leftarrow R1 + M[B]$
MOV R2, C	$R_2 \leftarrow M[C]$
MUL R_1, R_2	$R_1 \leftarrow R_1 * R_2$

One address instruction

In this type of instruction, a single operand address is specified. The other operands lie in the accumulator and the result is also stored back in the accumulator. LDA A, STA A, ADD A and PUSH are some example of 1-address instruction. One address instruction uses an accumulator (AC) system for all data manipulation.

LOAD A	$AC \leftarrow M[A]$
ADD B	$AC \leftarrow AC + M[B]$
STORE T	$M[T] \leftarrow AC$
LOAD C	$AC \leftarrow M[C]$
ADD D	$AC \leftarrow AC + M[D]$
MULT T	$AC \leftarrow AC * M[T]$
STORE X	$M[X] \leftarrow AC$

All operations we did between the AC register and a memory operand. T is the address of a temporary memory location required for storing intermediate result.

Zero-address Register

Zero-address type of instructions do not certain any operand or operand address. The operand addresses are implied. The instruction CLA is a 0-address instruction CLA stands for clear an accumulator. Here, the instruction itself specifies that the operation "clear" is to be performed on the Accumulator.

To evaluate arithmetic expression for zero-address machine and the expression must be in reverse polish notation. Also, the instructions like ADD MUL do not require an operand field. It is simply pop-up the two top most operands from the stack, perform the operation and place the result on the top of the stack

TOS → Top of Stack

The reverse polish notation of expression

$X = (A+B) * (C+D)$

$= AB+) * (CD+)$

$= AB+CD+*$

PUSH A	TOS ← A
ADD	TOS ← A+B
MUL	TOS ← (A+B) * (C+D)