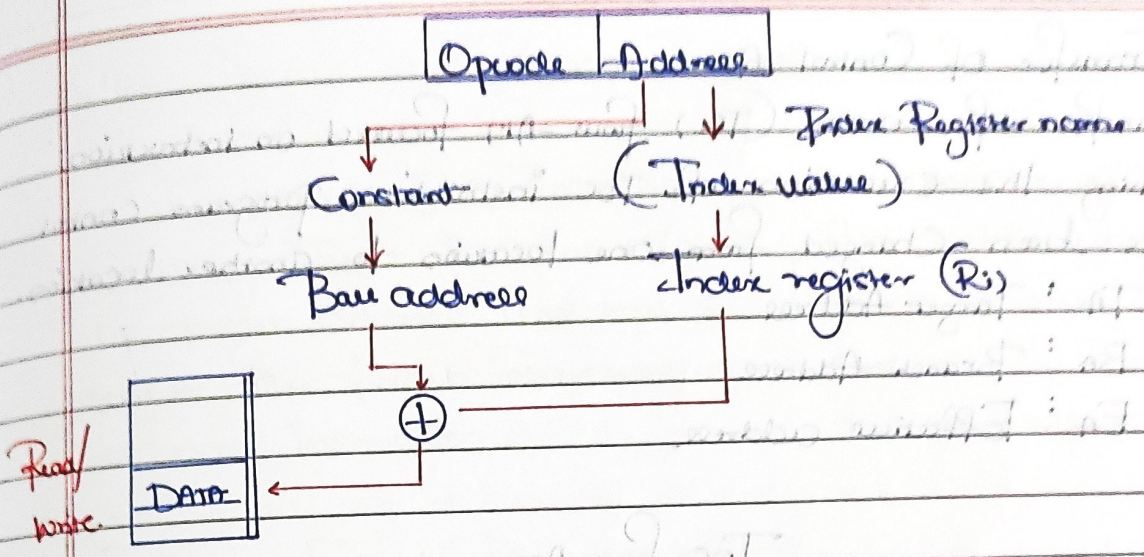


② Base Index Am: Sometime CPU contain Seperate base register and index register to store the base value and index value respectively.

eg: 8086 up

Bx reg ⇒ Base register

SI/DI : Index register (Source index / destination index)

MOV Bx, #1000

MOV SI, #20

MOV Ax [Bx][SI]

Ax ← [Bx+SI]

Ax ← M[1000+20]

③ Base Index with displacement:

MOV Ax 50([Bx][SI])

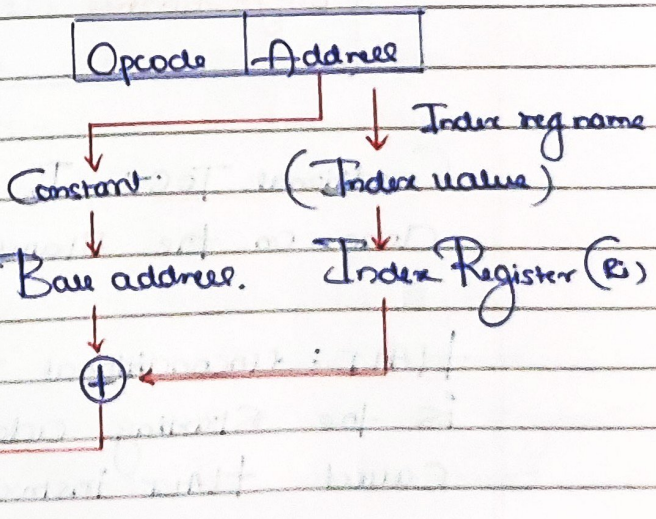
Ax ← M[(Bx)+(SI)+50]

④ Indirect Index Am.

1 Reg ref for index value

1 ALU ref for EA

1 Memory reference for Read Data.



Page No.
Date. / /

Transfer of Control Addressing Mode

* Transfer of Control (TOC) flow addressing focused on instructions.
* During the execution of TOC instructions Program Control has been changed from one location to another location.
  - → TA : Target Address
  - → BA : Branch Address
  - → EA : Effective address.

TOC flow Addressing

1. Unconditional TOC
   - eg: 1. JMP x (address)
     2. HALT.
2. Conditional TOC
   - JNZ (Jump on not zero)
   - BNZ (branch on not zero)
   - BZ (branch on zero)
   - JZ (jump on zero)

* Relative value is difference between current PC value and target address.

> Relative value = +ve (forward jumping)
> Relative value = −ve (backward jumping)

* PC relative Addressing:

> Target Address = Current PC value + OFFSET (Relative value)

Conditional TOC: In conditional TOC condition checking is done on the status of previous instruction output.

HALT: unconditional TOC instruction in which target address is the starting address of the instruction which is called HALT instruction.

Page No.
Date. / /

Branch Instruction

Conditional Jpc Instruction	Unconditional Jpc Instruction
1. Condition: True = Pc value updated as target address and now Pc execute the target instruction as a next instruction.	Pc value updated as a target address and now Pc execute the target instruction as next instruction.
2. Condition: False = Pc value is not updated as a target address & now next Sequential instruction is executed as a next instruction.	

PC Relative Am.

EA = Current Pc value + Address field value (Relative value)

* In this mode Effective Address (EA) is obtained by adding the relative value to program counter (PC).
* Relative Value means distance between Current location to target location, it is a Constant (signed constant), present in the address field of the instruction.
* Pc relative Am is used intra Segment transfer of control (branching), when target address is present in same segment then during program execution control will be transferred within the segment called intra segment branching.

Base Register Am.

EA = Base Register Value + Address Field Value

Base Register Am is used inter segment transfer of control (branching), when target address is present in different segment then during program execution control will be transferred between the segment called inter segment branching.

Page No.
Date. / /

Note: Both PC relative Am and Base register Am are suitable for program reallocation at runtime.

* Base Register Am → Reallocation
* PC Relative Am → Intra Segment branching

Q1. Consider a 4 Byte long PC relative instruction is located in the memory with the starting address of 243048 (Decimal). A -8 sign displacement is present in the address field of the instruction. What is the branch address?

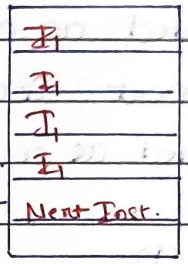
Ans Instruction Size = 4 Byte

Starting address = 24308

Current PC value = 243052

Displacement = −8 (Backward Jumping)

(Relative value)



Target address = Current PC value + relative value

= 243052 + (−8)

= 243044

Q2 Consider a 4 Byte long jump instruction stored in the word addressable memory with a word size of 16 bit. The starting address of instruction is 900 (decimal). A address field contain -32. Content of base register is 500. What is the branch address (Target address) when the instruction is designed with

(i) PC relative addressing model

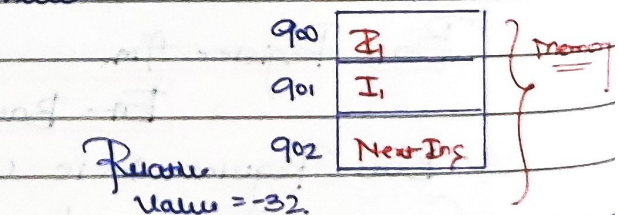
(ii) Base Register addressing model.

Ans. 4 Byte Jmp instruction

Memory word addressable

1 Word size = 16 bit = 2 Byte

Instruction Size = 4 Byte = 2 Words



Page No.
Date. / /

(i) PC relative Addr.
Target address = Current PC + AF (offset) value
= 902 - 32
= 870

(ii) Base register Addr
Target address = Base register value + relative value
= 500 + (-32)
= 468

Q3. Consider a 16 bit which support 1 word long instructions, stored in the memory with a starting address of 900 (decimal). Instructions format contain 8 bit Opcode and address field. Instruction is designed with PC relative Jmp Operation. During its execution control (branch) will be transferred to an address 614 (decimal). Then : what is :

(i) What is the relative value in address field of the instruction?

(ii) What is the PC value before instruction fetch, after instruction fetch and after execution phase?

Ans 16 bit instruction = 2 Byte

8bit	8bit
Opcode	AF

PC = 802
Target Add = 614

(i) Target address = current pc val + relative val
614 = 902 + relative value
relative value = -288

(ii) PC value : Before instruction fetch : 900
After instruction fetch : 902
After execution : 614 (Target address).

Q4. For computers based on three-address instruction format, each address field can be used to specify which of the following:
S1 : A memory operand
S2 : A processor register
S3 : An implied accumulator register

OP	AF1	AF2	AF3.

Ans Either S1 or S2.

Page No.
Date. / /

Instruction Set

1. Data transfer instructions / operation
2. Data manipulation instruction / Operation.
   (i) Arithmetic (eg ADD, SUB, DIV, MUL etc)
   (ii) Logical (OR, AND, XOR, etc)
   (iii) Shift and rotate operation (Arithmetic & logical shift, rotate with & without Carry)
3. Program Control instructions / Operation
   (i) Unconditional JOC
   (ii) Conditional JOC

Data Transfer Instructions : Data transferred from one location to another location.

(i) MOV

Destination		Source
Mem	←	Reg
Reg	←	Mem
Reg	←	Reg
Mem	←	Immediate
Reg.	←	Immediate.

Memory Specific { LOAD ⟶ Memory Read
STORE ⟶ Memory Write

Stack Specific { PUSH ⟶ Insert
POP ⟶ Delete

IO Specific { IN ⟶ IO Read
OUT ⟶ IO Write

Page No.
Date. 29/09/23

FLOATING POINT REPRESENTATION

DATA FORMAT

- Fixed Point Data
  - Magnitude Format
    - Unsigned Format (only the number)
      - n bit range
      - 0 to $2^n - 1$.
    - Signed Magnitude Format (+ve and -ve)

Sign value

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

  - Compliment Format
    - 1's Compliment Format
      - Both (+ve and -ve)
      - $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$
    - 2's Compliment Format
      - (+ve and -ve)
      - $-(2^{n-1})$ to $(2^{n-1}-1)$
- Floating Point Data
  - Single precision data (32 bit)
  - Double Precision (64 bit)

Note: If MSB bit (sign bit) is '0': +ve number and '1': negative num.

Q.1. In computer System why 2's compliment are used to represent negative number, even we have Signed and 1's compliment also?

Ans

Signed magnitude	1's Compliment
+0 : 0000	+0 : 0000
-0 : 1000	-0 : 1111

Redundant representation of '0' in Signed & 1's compliment so 2's Compliment are used in computer System.

Page No.
Date. / /

Q2. Why floating point representation?

Ans
* To represent very large number (986542.....)
* To represent very small number (0.....)

eg: 16 bit number range:
= $-(2^{16-1})$ to $(2^{16-1}-1)$
= -2^{15} to $2^{15}-1$

"if we want to store 51K then it is not possible in 16 bit."

Floating Point Representation:

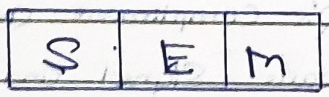
Principles:

* With a fixed point notation it is possible to represent a range of positive and negative integers centered on or near 0.
* By assuming a fixed binary or radix point, this format allows the representation of numbers with fractional component as well.

Limitations:

* Very large numbers cannot be represented nor can very small fractions.
* The fractional part of the quotient in a division of two large numbers could be lost.

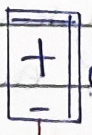
So floating point representation is to represent very large data and very small fraction and consume very less memory.



S (Sign) → 0 [+ve], 1 [-ve]

BE/E : Exponent
Bias ↓
Exponent

M : Mantissa



$0.xxxxxx \times 2^{e}$ ← e. → exponent

mantissa

Sign

$E = e + bias$

$BE = AE + bias$

Page No.
Date. / /

Number System:

2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
16	8	4	2	1	0	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$

#V. Imp

① +6.5

$6.5 = (110.1)_2$

0.1101×2^3

S m 2^e

S = 0 (+)

m = 1101

e = 3 = $(11)_2$

S	e	m
0	11	1101

$6.5 = 110.1$

$= .1101 \times 2^3$

$= [.2^{-1} + 2^{-2} + 2^{-4}] \times 2^3$

$= [2^2 + 2^1 + 2^{-1}]$

$= 6.5$

② +4.5

100.1

0.1001×2^3

S = 0 (+)

m = 1001

e = 3 [11]

S	e	m
0	11	1101

③ +4.75

100.11

$.10011 \times 2^3$

S = 0

m : 10011

$e = 3 = (11)_2$

S	e	m
0	11	10011

Note: Mantissa alignment process is used to adjust the decimal point, in this process right alignment increments the exponent and left alignment decrements the exponent.

2^{+shift} power (+) = Right alignment → Increment the exponent

2^{-shift} power (−) = left alignment → Decrease the exponent

Right alignment : $6.5 = 110.1 = .1101 \times 2^3$

$= [.2^{-1} + 2^{-2} + 2^{-4}] \times 2^3$

$= 2^2 + 2^1 + 2^{-1}$

$= 6.5$

Page No.
Date. / /

Left alignment: $0.00000000101 \times 2^{+5}$

$[1.01 \times 2^{+5-8}]$

$\therefore 1.01 \times 2^{-3}$

Note: If we take actual exponent [e] then there is no provision to represent the negative exponent. Because sign bit is telling number +ve [0] or -ve [if 1]

If we take 2's compliment of negative exponent then it creates ambiguity like:

$-2 : 1110$ } Create ambiguity.

$+14 : 1110$ }

So the solution is instead of writing in 2's compliment we use "Bias Exponent" [E/BE]

$$E = e + bias \quad \text{or} \quad BE = AE + bias$$

Biasing: to convert a number into '0' or +ve number.

How bias value is selected?

→ n bit 2's compliment range = $-(2^{n-1})$ to $+(2^{n-1}-1)$

4 bit 2's compliment = $-2(2^{4-1})$ to $(2^{4-1}-1)$

$= -8$ to 7

How bias is selected?

→ if exponent = K bit → $bias = 2^{K-1}$

If exponent K bit then 2's compliment = (-2^{K-1}) to $(2^{K-1}-1)$

4 bit 2's compliment range:

-2^{4-1} to $+2^{4-1}-1$

$\therefore -8$ to $+7$

→ In order to convert all number (+ve and -ve) into positive num take most highest negative number and add as bias.

Page No.
Date. / /

4) +0.00101

0.101×2^{-2}

m = 101

E = -2, S = 0

E = $(1110)_2$ 2's Complement.

S	E (4bit)	m (6bit)
0	1110	10100
	E	m

S	E	m
1bit	4bit	6bit

Normalised Mantissa

Explicit Normalised

Syntax: $0.1\underline{\ldots\ldots} \times 2^e$ (m.)

Formula to get number (value formula)

$(-1)^S \times 0.m \times 2^e$

$(-1)^S \times 0.m \times 2^{E\text{-bias}}$

0.1 After the point, immediate first bit should be 1.

eg: (101.11)

$= 0.10111 \times 2^3$

m = 10111, e = 3

E = e + bias.

eg: 6.75.

Explicit

S	E	m
1bit	4	5
	bias = 8	

$= 110.11 \times 2^0$

$= 0.11011 \times 2^3$

S = 0, m = 11011

Implicit Normalised

Syntax: $1.\underline{\ldots\ldots} \times 2^e$ (m.)

Formula to get number (value formula)

$(-1) \times 1.m \times 2^e$

$(-1)^S \times 1.m \times 2^{E\text{-bias}}$

Before the point 1, because 1.....

eg: (101.11)

$= 1.0111 \times 2^2$

m = 0111, e = 2

E = e + bias

S	E	m
1bit	4	5
	bias = 8	

$= 110.11$

$= 110.11 \times 2^0$

$= 1.1011 \times 2^2$

S = 0, m = 10110

Page No.
Date. / /

e = +3, bias = 8
E = e + bias = 3 + 8 = 11
E = 1011

S(1 bit)	E(4 bit)	M(5 bit)
0	1011	11011
1	7	B = (17B)16

e = +2 bias = 8
E = e + bias = 2 + 8 = 10
E = 1010

S(1 bit)	E(4)	M(5 bit)
0	1010	10110
1	5	6 = (156)16

Conversion:

S(1 bit)	E(4)	M(5)
0	1011	11011

S = 0 E = 1011 ⇒ 11
M = 11011, bias = 8
$(-1)^S \times 0.M \times 2^e$
$(-1)^0 \; 0.11011 \times 2^{11-8}$
0.11011×2^3
110.11
= 6.75

S(1)	E(4)	M(5)
0	1010	10110

S = 0, E = 1010 ⇒ 10
M = 10110, bias = 8
$(-1)^S \times 1.M \times 2^e$
$(-1)^0 \times 1.10110 \times 2^{10-8}$
1.10110×2^2
110.110
= 6.75

Q1. 4(5.5)

S	E	M
1 bit	4 bit	5 bit.

Explicit & Implicit represent?

bias = $2^{Exp.bit-1}$
$= 2^{4-1} = 8$

Explicit

(5.5) = 101.1
$= 101.1 \times 2^0$
$= 0.1011 \times 2^{+3}$
S = 0, M = 1011, E = 1011
e = 3, bias = 8, E = 3 + 8 = 11.

S(1)	E(4bit)	M(5bit)
0	1011	10110
1	7	6 = (176)16

Implicit:

+(5.5) = 101.1
$= 101.1 \times 2^0$
$= 1011 \times 2^2$
S = 0, M = 01100 E = 1010
e = 2 bias = 8 E = 2 + 8 = 10

S(1bit)	E(4bit)	M(5bit)
0	1010	01100
1	4	C ⇒ (14C)16

Page No.
Date. / /

Getting values:

Explicit:

+(5.5)

0	1011	10110

$(-1)^S 0.m \times 2^{E\text{-bias}}$

$(-1)^0 0.10110 \times 2^{11-8}$

0.10110×2^3

$101.10 = 5.5$

Implicit:

S(1 bit)	E(4 bits)	m(5 bits)
0	1010	01100

$E = 1010 = 10$

$(-1)^S 1.m \times 2^{E\text{-bias}}$

$(-1)^0 1.01100 \times 2^{10-8}$

$+1.01100 \times 2^2$

$+101.100 = +(5.5)$

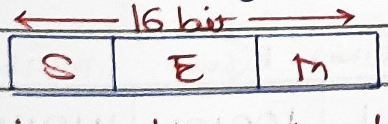
Note: *Sometimes in explicit we don't get accurate result so either increase the bit in mantissa or use implicit normalisation.

*Mantissa: giving Precision (more and more bit in mantissa giving very accurate value for small fraction also).

*Exponent: give the range (more bits in exponent means large number).

Q2. Consider a 16 bit register used to store floating point number. Mantissa is normalised signed fraction number. Exponent is in Excess-32 form then what is 16 bit for $+(13.5)_{10}$ in the register? (Using explicit & implicit)?

Ans



$bias = 2^{k-1}$ Excess = 32

$2^{k-1} = 32$ bias = 32

$k-1 = 5$

$k = 6$ bit

Explicit: (13.5)

$= 1101.1$

$= 0.11011 \times 2^{+4}$

$S=0$, $m = 11011000$

$e = 4$, bias = 32, $E = 4+32 = 36$

Implicit: (13.5)

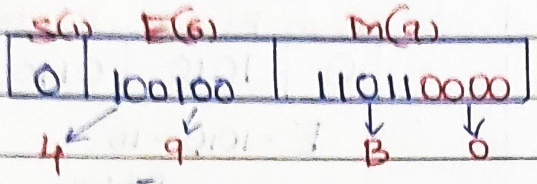
$= 1101.1$

$= 1.1011 \times 2^3$

$S=0$, $m = 1011000$

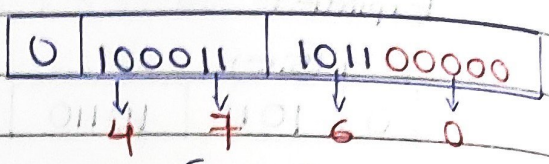
$e = 3$, $E = 3+32 = 35$

E = 100100



$(49B0)_{16}$

E = 100011



$(4760)_{16}$

Conversion : Explicit

0	100100	110110000

$(-1)^S \ 0.M \times 2^{E-bias}$

$(-1)^0 \ 0.110110000 \times 2^{36-32}$

$+0.110110000 \times 2^4$

$1101.100000 = (13.5)$

Implicit :

0	100011	101100000

$(-1)^S \ 1.M \times 2^{E-bias}$

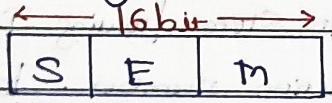
$(-1)^0 \ 1.101100000 \times 2^{35-32}$

$+1.101100000 \times 2^3$

$+1101.100000 = (13.5)$

Q3. Consider a 16 bit register used to store floating point number. Mantissa is Explicit normalised signed fraction number. Exponent is in Excess-32 form then what is 16-bit for $-(29.75)_{10}$ in the register?

Ans



Excess -32

bias = $32 = 2^{K-1}$

$2^{K-1} = 32$

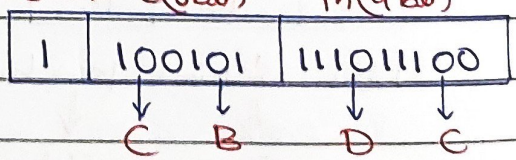
$K = 6$

$= -29.75$

$= -11101.11$

$= 0.1110111 \times 2^5$

$S = 1, \ M = 1110111, \ e = 5$



$= (CBDC)_{16}$

Q4. Consider a 16 bit register used to store floating point number. Mantissa is implicit normalised signed fraction number. Exponent is in Excess-64 then

(i) What is the first smallest positive number?

Page No.
Date 30/9/23

(i) What is the Second Smallest positive number?
(ii) What is the difference between first Smallest and Second Smallest positive number?

Ans

1 bit	7 bit	8 bit
S	E	M

Excess-64 → bias = 64

$2^{k-1} = 64 \rightarrow 2^{k-1} = 2^6$

$k-1 = 6$

$k = 7$ bit

(i) First Smallest positive number:

S(1)	E(7bit)	M(8bit)
0	0000000	00000000

bias = 64

E = 0

$(-1)^S 1.M \times 2^{E-bias}$

$(-1)^0 1.00000000 \times 2^{0-64}$

I. Smallest number = $1.00000000 \times 2^{-64}$

(ii) Second Smallest positive number:

S(1bit)	E(7bit)	M(8bit)
0	0000000	00000001

$(-1)^S 1.M \times 2^{E-bias}$

$(-1)^0 1.00000001 \times 2^{0-64}$

II. Smallest = $1.00000001 \times 2^{-64}$

(iii) Difference between I & II Smallest

difference = $1.00000001 \times 2^{-64} - 1.00000000 \times 2^{-64}$

$= [1.00000001 - 1.00000000] \times 2^{-64}$

$= [0.00000001] \times 2^{-64}$

$2^{-8} \times 2^{-64}$

$\Rightarrow \boxed{2^{-72}}$

Note:

(i) 0.111 → $\boxed{1 - \frac{1}{2^3}}$ or $\boxed{1 - 2^{-3}}$

(ii) 0.1111 → $1 - \frac{1}{2^4}$

(iii) 0.11111 → $1 - \frac{1}{2^5}$

Page No.
Date. / /

Q2. 0.11111

$\Rightarrow 11111.0 \times 2^{-5}$

$\Rightarrow (2^5 - 1) \times 2^{-5}$

$\Rightarrow 2^5 \times 2^{-5} - 2^{-5}$

$\Rightarrow \boxed{1 - 2^{-5}}$

0.11111

GP series: $\frac{1}{2} \; \frac{1}{2^2} \; \frac{1}{2^3} \; \frac{1}{2^4} \; \frac{1}{2^5}$

$a = \frac{1}{2}, \; r = \frac{1}{2}$

$$\frac{a(1-r^n)}{1-r} = \frac{1/2\left(1-1/2^5\right)}{(1-1/2)}$$

$$\Rightarrow \boxed{1 - \frac{1}{2^5}}$$

Note: We cannot represent '0'.

→ Implicit = 1..... ≠0 (not zero)

→ Explicit = 0.1..... ≠0 (not zero)

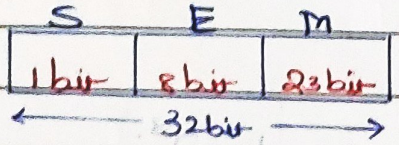
In both implicit and explicit representation we cannot represent '0'. In IEEE 754 floating point we can represent 0.

Disadvantage of Conventional Representation: It cannot store 0. Explicit = 0.1, Implicit = 1.0. It cannot represent '∞' (infinity). It cannot represent number which is not normalised.

IEEE 754 Floating Point Representation

Single Precision (32 bit)

Excess 127



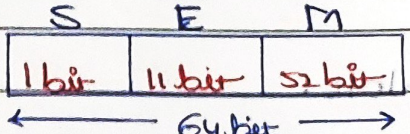
$bias = 2^{k-1} - 1$

$= 2^8 - 1$

$= 127$

Double Precision (64 bit)

Excess 1023



$bias = 2^{11-1} - 1$

$= 1023$

Page No.
Date. / /

Note: In IEEE 754 default is implicit normalisation.

Q1. How to represent +(119) into IEEE 754 single precision and double precision floating point representation.

Ans **Single Precision**

$119 : 1110111 \times 2^0$

$= 1.0110111 \times 2^{+6}$

S	E	M
1 bit	8 bit	23 bit

$S = 0$, $M = 10111000000000\ldots$ $\quad bias = 2^{8-1} - 1$

$e = 6$, $bias = 127$ $\quad E = e + bias$ $\quad bias = 127$

$= 6 + 127 = 133$

$E = 10000101$

0	10000101	11011100000000000000000
4	2	E E 0 0 0 0

$+119 : (42EE0000)_{16}$

Double Precision

$119 : 1110111 \times 2^0$

$+1.110111 \times 2^{+6}$

S	E	M
1 bit	11 bit	52 bit

$bias = 2^{11-1} - 1$

$= 1023$

$S = 0$, $M = 11011100000000\ldots$

$e = 6$, $bias = 1023$

$E = 1023 + 6 = 1029. = \boxed{10000000101}$

S (1 bit)	E (11 bit)	M (52 bit)
0	10000000101	11011100000000000000\ldots
4	0 5	D C 00000

$+119 : (405DC00000000000)_{16}$

Q2 Consider a 32 bit register which stored floating numbers in IEEE single precision format (implicit). The value of number, of 32 bit are given below ie

Sign (1 bit)	Expo (8 bit)	Mantissa (23 bit)
0	10000100	11000000000000...

Ans. E = 132, bias = 127

$(-1)^S 1.m \times 2^{E-bias}$

$(-1)^0 1.11000000\ldots \times 2^{132-127}$

$+1.11000000\ldots \times 2^{+5}$

$+111000.000000\ldots \times 2^0$

$\Rightarrow 56$

Q3. The value of float type variable is represented using the single precision 32 bit floating point IEEE 754 standard use 1 bit sign, 8 bit for E and 23 bit mantissa. A float type variable X is assigned the decimal value (−14.25) Then representation of X in Hexa decimal notation is?

Ans. -14.25

$1110.01 = 1.1101 \times 2^3$

$S = 1,\ M = 11001,\ e = 3$

1 bit	8 bit	23 bit
S	E	M

$Bias = 2^{8-1} - 1 = 127$

$E = e + bias = 3 + 127$

$= 130$

S	E	M
1	10000010	11001000...

Hexadecimal = $(C1640000)_{16}$

Single Precision (32 bit)

S	E	M
1 bit	8 bit	23 bit

Sign (1 bit)	E (1 bit)	M (23 bit)	Value
0 or 1	00000000 E = 0	000000.... M = 0	∓ 0
0 or 1	11111111 E = 255	0000..... M = 0	∓ ∞
0 or 1	1 ≤ E ≤ 254	M = ...	Implicit Normalised form $(-1)^S \times 1.M \times 2^{E-127 bias}$

0 or 1	E = 0	M ≠ 0	Denormalised number $(-1)^S \times 0.M \times 2^{E-127 \text{ bias}}$
0 or 1	E = 255	M ≠ 0	Not a Number (NAN)

Q. Why bias = 2^{k-1} ?

Ans Single Precision:

1 bit	8 bit	23 bit
S	E	M

If in single precision, if bias = $2^{8-1} = 2^7$ (bias = 128) then there is a possibility of getting E = 255.

$E = e + \text{bias}$ e = +127 then E = 127 + 128 = 255.

if E = 255 ? $\pm\infty$ / NAN

8 bit 2's complement range

$= -2^{8-1}$ to $2^{8-1} - 1$

$= -128$ to 127

So bias = 2^{k-1}.

Double Precision:

Excess = 1023.

1 bit	11-bit	52-bit
S	E	M

Sign (1 bit)	E (11 bit)	M (52 bit)	Value
0 or 1	00000000000 E = 0	000000.... M = 0	± 0
0 or 1	11111111111 E = 2047	000000.... M = 0	± ∞
0 or 1	1 ≤ E ≤ 2046	M = ...	Implicit Normalisation $(-1)^S \times 1.M \times 2^{E-1023}$.
0 or 1	E = 0	M ≠ 0	Denormalised Number $(-1)^S 0.M \times 2^{E-1023}$
0 or 1	E = 2047	M ≠ 0	Not a Number.

Page No.
Date. / /

Note: When E=0 then value=0 when m=0 and when m≠0 then value is in fractional form.

When E=2047 then value = ∞ when m=0 and when m≠0 it is not a number (NAN)

Features IEEE 754 are Special Symbols to represent Unusual events...

1. For example instead of interrupting on a divide by 0, Software can set the result to a bit pattern representing +∞, -∞. The largest exponent is reserved for these Special Symbols
2. IEEE 754 has a Symbol for the result of invalid operations such 0/0, or Subtract infinity from infinity. This Symbol is NAN for not a number.

The purpose of NAN is to allow Programmer to postpone some test and decisions to a later time in this program (when it is convinient).

Denormalised Number:

Minimum Possible value of E = 00000001, E=1

E = e + bias

e = E - bias, 1 - 127 = 126, so $\boxed{e = -126}$

That means in worst case e = -126, if value of e is small than -126, then number is not able to normalise and we store as denormalise number.

eg: 1.1001×2^{-127}, e = -127 E = -127 + 127 = 0 so E=0 not able to normalise.

eg: 1.1001×2^{-128} e = -128 so E = -1, not able to normalise

or 1.1001×2^{-129} Here m=1001,

e = -129, E = e + bias, -129 + 127 = -2

So E = -2 not able to normalise because E must be 1 after biasing.

Page No.
Date. / /

For eg. The Smallest Positive (true) Single Precision normalised number is : 1.0000..... (23 times) × 2^{-126}.
But the Smallest Single Precision denormalised number is: 0.000 .... 01 × 2^{-126} or 1.0×2^{-149}
and double precision range is : 1.0×2^{-1022} to 1.0×2^{-1074}

Q1. How to represent +(1.0) into IEEE 754 Single Precision floating point representation?

Ans.

1 bit	8 bit	23 bit
S	E	M

bias = $2^{8-1}-1$
= 127

1.0×2^0

m = 0, e = 0, E = e + bias
= 0 + 127 = 127.

0	01111111	00000000 ..

↓ 3 F 8 0 0

$(3F800000)_{16}$

Q2. Consider a 64 bit register which store floating point number in IEEE 754 Double Precision format. What is the value of the number if 64 bits are given as

0	111 1111 1111	000000 000 ....
S	E (11 bit)	M (52 bit)

Ans.

Here all bits in E are 1
E = 2047, M = 0 ∴ its ± ∞.

Add / Subtract Rule:
1. Choose the number with the smallest exponent and shift its mantissa right a number of steps equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition / subtraction on the mantissa and determine the sign of the result.
4. Normalise the resulting value if necessary.

Page No.
Date. 02/10/23

Note: Multiplication and division are somewhat easier than addition and subtraction, in that no alignment of mantissas is required.

Multiply Rule

1. Add the exponents and subtract 127.
2. Multiply the mantissas and determine the sign of the result.
3. Normalise the resulting value, if necessary.

Divide Rule

1. Subtract the exponents and add 127.
2. Divide the mantissa and determine the sign of the result.
3. Normalise the resulting value if necessary.

The addition or subtraction of 127 in the multiply and divide rule results from using the excess −127 notation for exponents.

eg: −0.75 in Single Precision format.

$\rightarrow -0.11 \times 2^{0}$

$\rightarrow -1.1 \times 2^{-1}$.

1 bit	8 bit	23 bit
S	E	M

$e = -1$ bias = 127

$E = -1 + 127 = 126$ $E = 01111110$

Sign.	E.	M.
1.	01111110	10000....

$\Rightarrow (BF400000)_{16}$

eg: Add $9.999 \times 10^{+1}$ and 1.610×10^{-1}.

now, 9.999

$+ 0.016$

$10.015 \times 10^{+1}$

Normalise → $1.0015 \times 10^{+2}$

1.610×10^{-1}

↓

0.1610×10^{0}

$0.0161 0 \times 10^{1}$ (now equivalent exponent so perform addition)

$1.002 \times 10^{+2}$

Roundoff

Page No.
Date. / /

Try adding the numbers 0.5 and −0.4375 in binary then using algorithm.

Ans: $0.5 = \frac{1}{2} = 0.1 \times 2^0$

$= (1.000 \times 2^{-1})_2$

$-0.4375 = \frac{-7}{16} = \frac{-7}{2^4} = -0.0111$

$= -0.0111 \times 2^0$

$= (-1.110 \times 2^{-2})_2$

Step 1: The significand of the number with the lesser exponent $(-1.11_2 \times 2^{-2})$ is shifted right until its exponent matches the larger number: $-1.110_2 \times 2^{-2} = -0.111 \times 2^{-1}$.

Step 2: Add the significands:

$(1.000)_2 \times 2^{-1} + (-0.111_2 \times 2^{-1}) = 0.001_2 \times 2^{-1}$.

Step 3: Normalizing the sum, checking for overflow or underflow

$(0.001)_2 \times 2^{-1} = 0.010_2 \times 2^{-2}$

$= 0.100 \times 2^{-3}$.

$= 1.000_2 \times 2^{-4}$.

Since $127 \geq -4 \geq -126$ there is no overflow or underflow. (The biased exponent would be $-4 + 127$, or 123, which is between 1 and 254, the smallest and largest unreserved biased exponents.

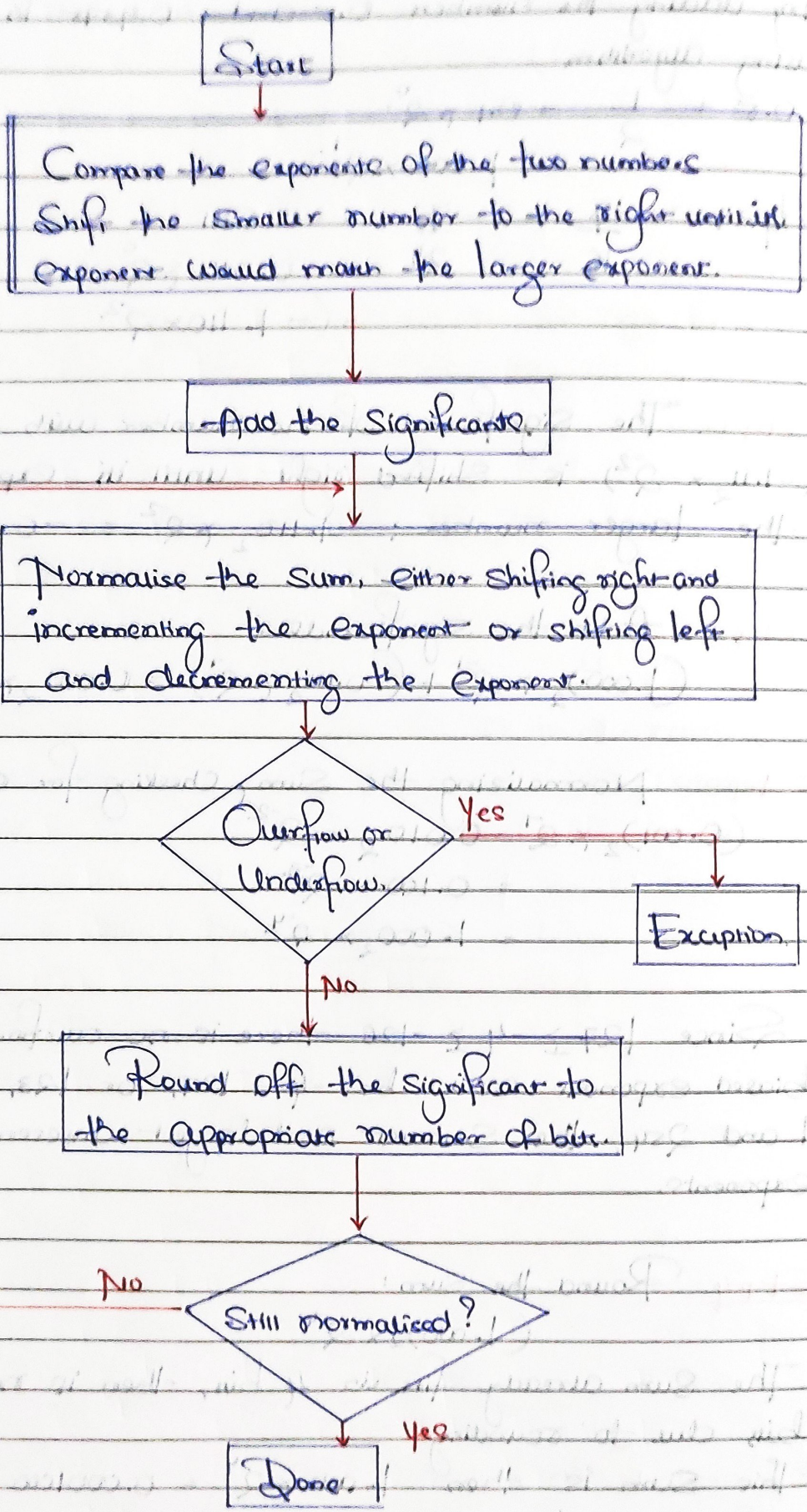
Step 4: Round the sum:

$(1.000)_2 \times 2^{-4}$

The sum already fits in 4 bits, there is no change to the bits due to rounding.

This sum is then $1.000 \times 2^{-4} = 0.0001000 = 0.0001$

$= \frac{1}{2^4} = \frac{1}{16} = 0.0625$.


Start
Compare the exponents of the two numbers
Shift the smaller number to the right until its
exponent would match the larger exponent.
Add the Significands
Normalise the sum, either shifting right and
incrementing the exponent or shifting left
and decrementing the exponent.
Overflow or
Underflow
Yes
Exception
No
Round off the significand to
the appropriate number of bits.
No
Still normalised?
Yes
Done.

Page No.
Date. / /

Note: The normal path is 10 execute steps 3 and 4 once, if rounding causes the sum to be abnormalised, we must repeat Step 3

→ Multiply 1.110×10^{10} and 9.2000×10^{-5}.

Step 1: Unlike addition, we calculate the exponent of the product by simply adding the exponents of both operands

New exponent $= 10 + (-5) = 5$

Let's do the same with biased exponent to make sure we obtain the same result:

$10 + 27 = 137$, and $-5 + 127 = 122$, so

new exponent $= 137 + 122 = 259$.

This is too large for 8 bit exponent field, so something is amiss. The problem is with the bias because we are adding the biases as well as new exponents:

New exponent: $(10+127) + (-5+127) = (8\times2^5 + 2\times127) = 259$

Accordingly, to get the correct biased sum when we add biased numbers, we must subtract bias from the sum

New Exponent $= 137 + 122 - 127$

$= 259 - 127 = 132$ $(5+127)$ and 5 is indeed the exponent we calculated initially.

Step 2: Multiply the significands

$$\begin{array}{r} 1.110 \\ \times\ 9.2000 \\ \hline 0000 \\ 0000+ \\ 2220++ \\ 0990+++ \\ \hline 10212000 \end{array} \quad = 10.212 \times 10^5$$

Page No.
Date. / /

Step 3: The product is unnormalised, so we need to normalise it:

$10.212 \times 10^5 = 1.0212 \times 10^6$

Step 4: We assumed that the significand is only four digits long (excluding the sign), so we must round the number.

The number 1.0212×10^6 is rounded to four digits in the significand to 1.021×10^6

Step 5: The sign of the product depends on the signs of the original operands. Here they have both same sign hence the product is: $+1.021 \times 10^6$

Binary floating point multiplication

Multiply $(0.5)_{10}$ and $(-0.4375)_{10}$

Step 1: In binary 1.000×2^{-1} by -1.110×2^{-2}

Adding the exponents (without bias): $-1 + (-2) = -3$

Or, using the biased representation:

$\Rightarrow (-1 + 127) + (-2 + 127) - 127$

$\Rightarrow (-1 - 2) + (127 + 127 - 127)$

$\Rightarrow -3 + 127 = 124$

Step 2: Multiply the Significands

$$\begin{array}{r} 1.000 \\ \times\ 1.110 \\ \hline 0000 \\ 1000+ \\ 1000++ \\ 1000+++ \\ \hline 1110000 \end{array}$$

The product is 1.110000×2^{-3} but we need to keep it to 4 bits, so it is 1.110×2^{-3}.

Page No.
Date. / /

Step 3: Now we check the product to make sure it is normalised, and then check the exponent for overflow or underflow. The product is already normalised and, since $127 \geq -3 \geq -126$, there is no overflow or underflow.

Step 4: Rounding the product makes no changes:

$$-1.0110 \times 2^{-2}$$

Step 5: Since the signs of the original operands differ, make the sign of the product negative, hence the product is

$$-1.0110 \times 2^{-3}$$

Converting to decimal to check our results:

$$-1.0110 \times 2^{-3} = -0.00110 = -0.00111$$

$$= -\frac{7}{2^5} = -\frac{7}{32} = -0.21875$$

The product of $(0.5)_{10}$ and $(-0.4375)_{10}$ is indeed $(-0.21875)_{10}$

Note: The normal path is to execute steps 3 and 4 once, but if rounding caused the sum to be unnormalised, we must repeat step 3.

Little Endian and Big Endian

* The exact number of bytes that constitute a word depends on the system, for eg: in system Pentium, a word refers to four bytes or 32 bits. On the other hand, eight bytes are grouped into a word in the Itanium Processor.
* Bits in a word are usually ordered from right to left.

1 Byte = 8 bit

Word Size = Word length of the Processor.

* Default configuration in the memory chip is Byte addressable

So in the memory data is stored in byte wise.

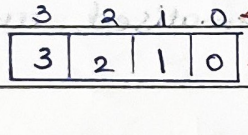
* In the processor operations are performed on a word format, so when the word length of the processor is greater than 8 bit then multiple cells accessing is required to access the data from memory simultaneously.

→ To solve the problem of ambiguity there is need of memory address interpretition mechanism (m/c instructions) called "Endian m/c instructions":

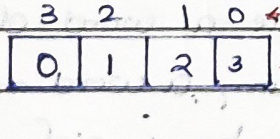
① Little Endian
② Big Endian.

Byte ordering or
Endian m/c instructions: Show the order of data storage in memory.

① Little Endian: Lower address contain lower byte and higher address contain higher byte.



② Big Endian: Lower address contain higher byte and higher address contain lower byte.



* Default m/c instructions ie: Little Endian used in processor design. But some processor use Big Endian also.

* Little Endian: Right to left
Big Endian: Left to right.

Page No.
Date. / /

Byte Ordering:

* Storing data often requires more than a byte.
* There are two byte ordering schemes referred to as little endian and big endian.

eg: $(11, 12, 13, 14)_{16}$ Stored at starting address 100.

32 bit processor

Little Endian
(Right to left)

103	102	101	100
11	12	13	14

r0 = 11 12 13 14

Big Endian
(left to right)

103	102	101	100
14	13	12	11

r0 = 14 13 12 11

eg: (51, 21, 15, 25)H Stored at starting address 400.

"32 bit processor"

Little Endian
(Right to left)

403	402	401	400
51	21	15	25

r2 = 51 21 15 25

Big Endian
(left to right)

403	402	401	400
25	15	21	51

r2 = 25 15 21 51

Page No.
Date. 08 / 10 / 23

ALU & CONTROL UNIT

Components of a Computer:

① CPU, ② Memory and ③ IO,

- Memory
- Register
- ALU
- Timing signals, Control Signals
- Flags (Psw)

Register: [Flip Flop]: Collection of bits / sequence of bits stored in flip flop.

+ Register is temporary storage.

Register

Special purpose register.

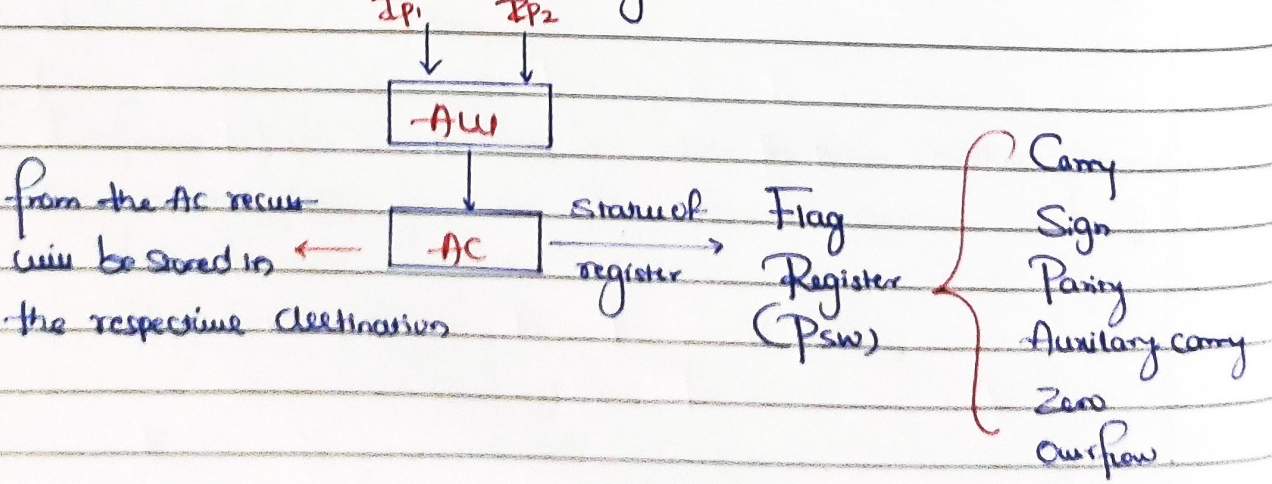
General purpose register

LD INC CLR

LD: Load
INC: Increment
CLR: Clear.

ALU: Arithmetic and logic Unit.

- Hardware
- Perform arithmetic and logical operations, comparisions and condition checking.
- Perform multiple type operations.



Page No.
Date. / /

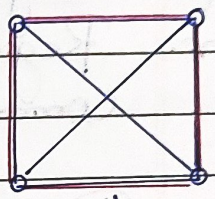
Timing Signal: To execute the Operation in Propes Sequence.

eg: 1. Fetch
2. Decode
3. Execute

Fetch
T_1: PC → MAR
T_2: M[MAR] → Memory
Memory → MBR
MBR → IR

eg If we have 16 Registers, 1 ALU, 1 Memory, 1 PSW and 1 Compensat. i.e total 20 Component

$^{20}C_2$ = 200 + ~~Component~~ connections



So the Solution instead of using 200+ Connections is to Connect all Component to a Common bus. (Internal Bus)

But only two pairs (2 component) can communicate at a time and which Component will communicate for that control Signal is required.

> The number of multiplexer = No. of bit in the register (Size of the register)
>
> Size of the multiplexer = No. of registers

Q.) If we have 32 registers and each register Size is 8 bit then

Ans → Number of multiplexer (MUX) = 8 bit
→ Size of MUX = 32 (32×1)

Note: If we have n registers and each register Size is 'n' bit then
→ Number of MUX = N
→ Size of MUX = n (n×1)

Page No.
Date. / /

Q1. How data is transferred?
Register A to Register B.

$[R_A \rightarrow R_B]$

Process: Register 'A' content given to mux (multiplexer), mux to common bus then, Common bus to load into Register B (R_B).

Q2. Why Rout and Rin is needed?

Ans. Register output is connected to the mux then mux to common bus then common bus to respective register.

Process: When Rout is set to '1' then that respective register content (data) load into the mux. Then mux to common bus and here common bus is connected to all registers, then the register having Rin is set to 1 in that respective register bus content is loaded in that register.

$R_A \rightarrow R_B$

RA to RB; R_{Aout}, R_{Bin}

$[R_{Aout} = 1]$ Content of register A loaded into mux then mux to common Bus

Now $[R_{Bin} = 1]$ then content of Bus load into register RB

MAR/AR [memory Address Register]: It contains address of memory location. It is connected to the Address line (Aw of the system bus.

MBR/MDR/DR [memory Data Register]: It contain memory content (instruction and data). It is connected to the data line of the bus.

Page No.
Date. / /

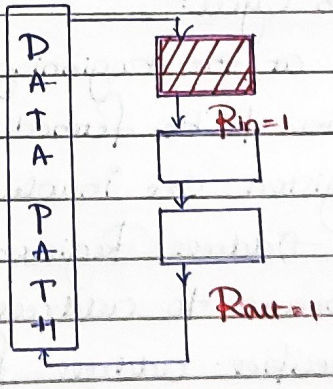
Instruction Register (IR): Contains the instruction currently being executed by the CPU.

In the Processor some general purpose and some special purpose register are available. All the register are connected to a common path called Data Path (Bus). Data Path is a collection of functional units (ALU and MUX).
Every register has two switch Rin and Rout:

Rin: Rin is set to 1, if the content of the bus loaded into register Ri.

Rout: Rout is set to 1, the content from the register Ri will be placed on bus.

Data Path: ALU, MUX, Register and other component, how they process, & path for processing.



Micro Operation:

- Micro operation is a elementary operation in the hardware.
- eg: register to register transfer is a one kind of micro operation
- Micro Operation consume 1 cycle to complete the execution.
- Control Signal are required to execute the micro Operation.

eg: Machine Instruction: MOV R0, R1
RTL (Register Transfer language): R0 ← R1
Micro operation: T1: R1 out R2 in.

Micro Operations:

- The functional, or atomic, operations of a Processor

Page No.
Date. / /

* Series of steps, each of which involves the processor registers
* Micro refers to the fact that each step is very simple and accomplishes very little.
* The execution of a program consist of the sequential execution of instructions.
* Each instruction is executed during an instruction cycle made up of shorter sub cycle (fetch, indirect, execute, interrupt).
* The execution of each sub cycle involves one or more shorter operations (micro-operations).

Micro Program (μ-Program): Sequence of micro operations (μ-operations) to perform some operation in hardware level is called as micro program.

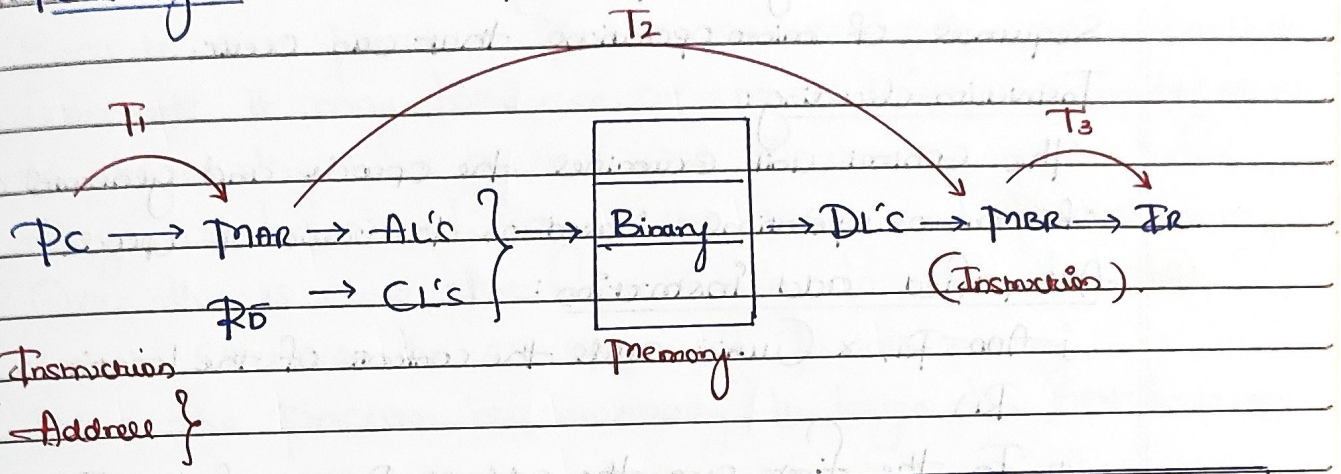
The Fetch Cycle:
Occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory.
Four registers are involved:

1. Memory Address Register (MAR):
   * Connected to address bus
   * Specifies address for read or write operations.
2. Memory Buffer Register (MBR):
   * Connected to data bus
   * Holds data to write or last data read.
3. Program Counter (PC):
   * Holds address of next instruction to be fetched.
4. Instruction Register (IR):
   * Holds the last instruction fetched.

Page No.
Date. / /

Instruction Cycle:

(1.) Fetch Cycle: Instruction Fetch



Micro Program ⟹	T_1 : PC → MAR	PCout MARin
	T_2 : M[MAR] → MBR;	MARout MBRin } System Bus
	PC + 1 → PC	PCout PCin } Local Bus
	Increment	
	T_3 : MBR → IR	MBRout IRin

*At the end of fetch cycle instruction is loaded into IR register

(2.) Execute Cycle:

1. Decode the instruction.
2. Operand address calculation.
3. Operand fetch with the help of addressing mode
4. Data processing.
5. Result Storage

Rules for Micro Operations Grouping.

① Proper Sequence must be followed.
MAR ← (PC) must Proceed MBR ← (memory)

② Conflicts must be avoided
* must not read and write the same register at same time
* MBR ← (memory) and IR ← (MBR) must not be in same cycle
* If different components then perform in same cycle.

Page No.
Date. / /

Execute Cycle:

* Because of variety of opcodes, there are a number of different sequences of micro operations that can occur.

1. **Instruction Decoding:**
   The control unit examines the opcode and generates a sequence of micro-operations based on the value of opcode.

2. **A simplified add instruction:**
   (i) ADD R_1, X (which adds the contents of the location X to register R_1)
   (ii) In the first step the address portion of the IR is loaded into the MAR.
   (iii) Then the referenced memory location is read.
   (iv) Finally the contents of R_1 and MBR are added by the ALU
   (v) Additional micro operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some immediate registers.

   (a) **Is Stage:** Enable the hardware to perform the operation (Instruction Decode).

   (b) **OF Stage:** Any (addressing mode) are required to access (Operand Fetch).

Interrupt Cycle:

* At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred, and if so, the interrupt cycle occurs.
* The nature of this cycle varies greatly from one machine to another in a simple sequence of events:
  1. In the first step contents are transferred to the MBR so that they can be saved from return from the interrupt.
  2. Then MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with

the address of the start of the interrupt processing routine.

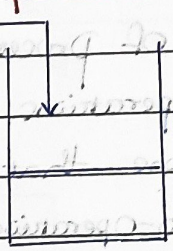
(3) These two actions maybe a - Single micro operation.

(4) Because most Processor Provide multiple types and/or levels of interrupt, it may take one or more additional micro-operation to obtain the save address and the routine address before they can be transferred to the MAR and PC respectively.

(5) Once this is done, the final step is to store the PSW, which contains the old value of PC into memory.

(6) Now the processor is now ready to begin the next instruction cycle.

Whenever interrupt occur after the completion of current instruction execution interrupt will be serviced.



PC value stored in the Stack as return address and Control transfer to ISR (Interrupt Subroutine)

Vector Address → PC

Q1. The following are some events that occur after a device controller issues an interrupt while process L is under execution.

(P) The processor pushes the processor status of L onto the control stack.

(Q) The processor finishes the execution of the current instruction

(R) The processor executes the interrupt service routine.

(S) The processor pops the process status of L from the control stack

(T) The processor loads the new PC value based on the interrupt

Which is the correct order: QPTRS

Page No.
Date. / /

Q2. Consider the following sequence of micro-operations:

MBR ← PC
MAR ← X
PC ← Y
MEMORY ← MBR

Which one of the following is a possible Operation performed by this sequence?

Ans. Initiation of interrupt Service.

Control Unit Functional Requirements

* By reducing the Operation of the Processor to its most fundamental level we are able to define exactly what is that the Control unit must cause to happen.
* Three step process to lead to a Characterization of the Control Unit:
  1. Define basic elements of processor.
  2. Describe the micro-Operations processor performs.
  3. Determine the functions that the Control Unit must perform to cause the micro-operations to be performed.
* The Control unit performs two basic tasks:
  * Sequencing
  * Execution.

Control Unit:

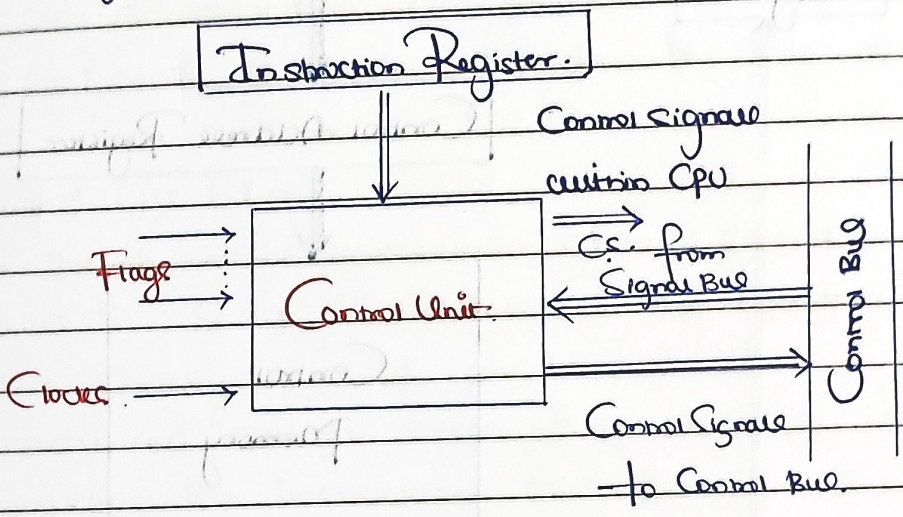
* Control unit is the Supervisor in the System that Control each and every activity.
* Control Unit takes various input but produce only one output (Control Signals).
* These Control Signals are required to execute the micro Operations.
* Control Signals are directly executed on a Base Hardware so hardware generated the desired output.
* Computer System functionality is Program Execution.

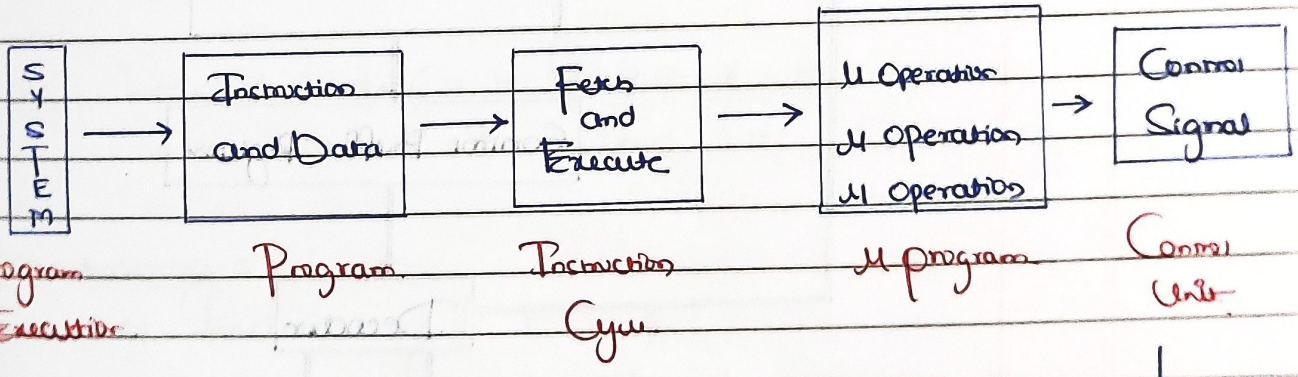
Page No.
Date. / /

Q. Why Control Unit is set after decoder?

Ans In fetch and decode same type of operation is performed. After the decode, input given to the control unit, so according to the operations control signal is generated to perform micro operations.

Block Diagram of the Control Unit





Control Unit : Control word

Ti : PC → MAR ; Read MAR in

Program Output

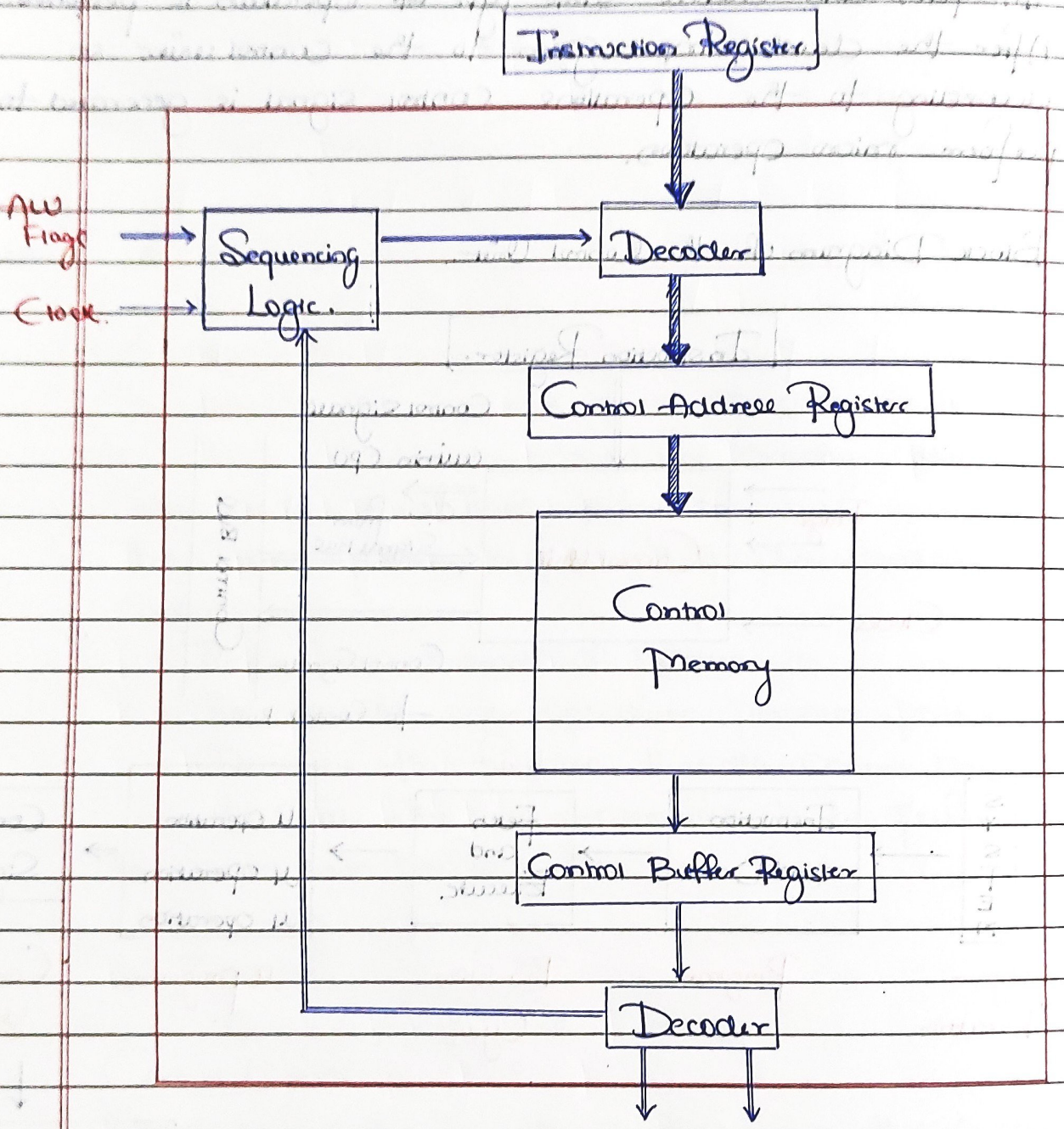
Bus Hardware (H/W)

Pcin	Pcout	MARin	MARout	IRin	IRout	MBRin	MBRout	...	Acin	Acout
0	1	1	0	0	0	0	0	...	0	0

1 Control word for 1 micro operation

Page No.
Date. / /

Functioning of Microprogrammed Control Unit:



→ Control Signals can be implemented into the Control Unit by using the following approach:

1. Hardwired CU design
2. Micro programmed CU design.

Page No.
Date: 03/10/23

Working of Control Unit:
* Control unit generate the control signals, these control signals are same for fetch and decode.
* After the decode, different type control signal (control word) generated according to different type of operations.
* At the control unit design time, designer decide which control are generated in which cycle (T1, T2 ... Tn) for different type of instructions and that is stored in the table.
* Later we make a boolean function for implementations.

1. Hardwired CU Design:
* In the hardwired control unit design control signals are expressed in S.O.P. Expression (sum of product).
* They are directly realised on hardware.
* In hardwired control unit they used fixed logic circuit to interpret the instructions and generate the control signal.

Note:
* Hardwired CU is fastest CU design.
* RISC is a Hardwired CU.

Disadvantages:
* It is not flexible.
* Even a minor modifications require redesigning and rewiring.
* It does not support new operations (one designed).

Process:
Step 1: Search where the control signals Ain and Bout are present.
Step 2: Options are in I.I format or T.I format.
Step 3: For any particular time interval. Is the control signal present for all the instructions?

Page No.
Date. / /

Q1. Consider the following hypothetical CPU which uses 3 data register A, B and C, and supports 2 instructions I_1, I_2 and I_3. Obtain the logic function that will generate the hardwired control for the signal Ain and Bout using the following data.

	I_1	I_2	I_3
T_1	Ain, Bout	Ain, Cin, Bout	Bin, Bout
T_2	Bin, Cin, Aout	Ain, Aout	Ain, Bout, Cout
T_3	Bin, Bout	Bin, Bout	Bin, Bout
T_4	Cin, Aout	Bin, Aout	Ain, Aout
T_5	End	End	End

$Bout = T_1I_1 + T_1I_2 + T_1I_3 + T_2I_1 + T_2I_2 + T_3R_2 + I_3i_n$

$= T_1(I_1 + I_2 + I_3) + T_3(I_1 + I_2 + I_3)$

$Bout = T_1 + T_3$

$Ain = T_1I_1 + T_1I_2 + T_2I_2 + T_2I_3 + T_4I_3$

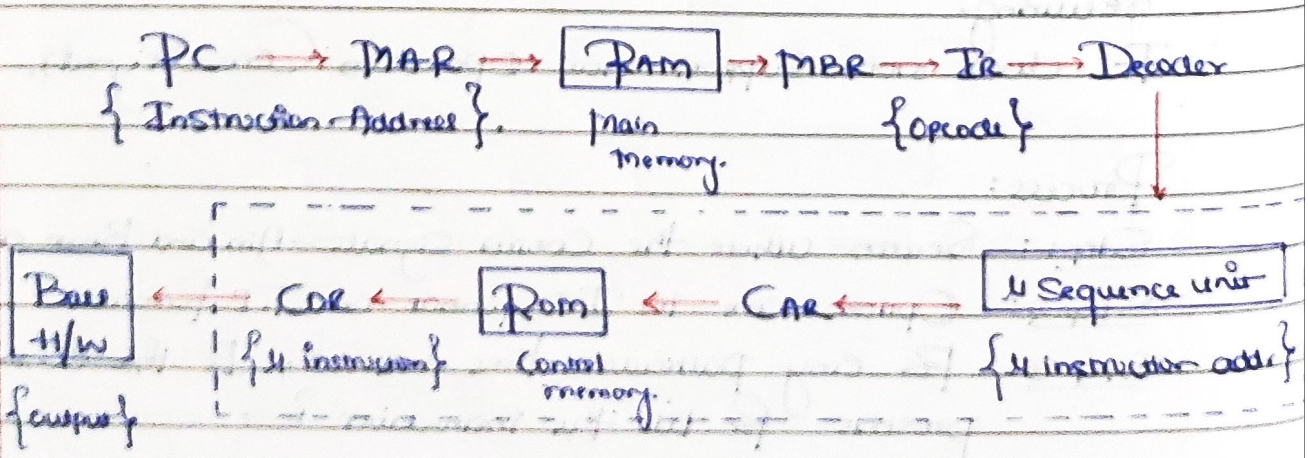
$Ain = T_1I_1 + (T_1 + T_2)I_2 + (T_2 + T_4)I_3$

(2) Micro Programmed Control Unit:

- In micro-programmed CU design control words are stored in the control memory.
- According to the type of operation control signals are generated.
- Control memory is associated with CAR and CDR Register to contain control memory address and data respectively.

CAR: Control Address Register

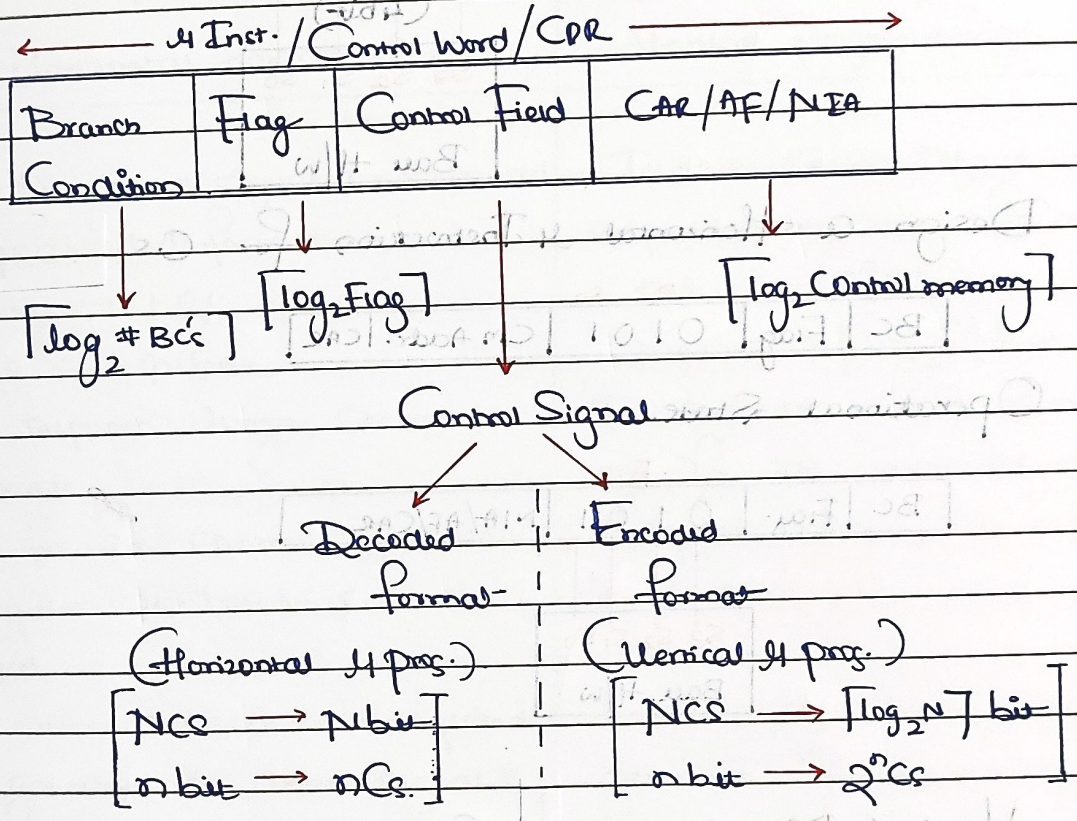
CDR: Control Data Register.



Page No.
Date. / /

Process:

* The set of micro instructions stored in the control memory.
* The [CAR] Control Address Contain the address of next instruction or micro instruction to be read.
* Then a micro instruction is read from the control memory and it's transferred to control buffer register.
* Control buffer register is connected to the control line so is this micro instruction execution occurs.



1. Hardwired CU Design
   - S.O.P expression
   - Logic function
   - Fastest (RISC)
   - Not flexible

2. Microprogrammed CU
   - Micro programs stored in control memory

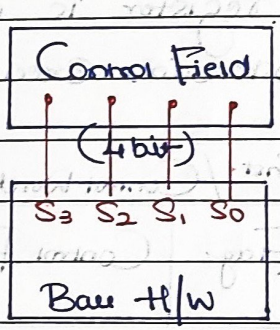
Page No.
Date. / /

→ Micro Instructions are implemented by 2 approaches:
① Horizontal μprogramming
② Vertical μ programming

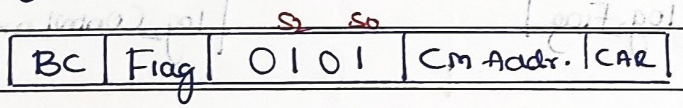
Horizontal Programming:

1. Number of control in the hardware = [S_0 S_1 S_2 S_3]
2. Decoded for CS ⟹

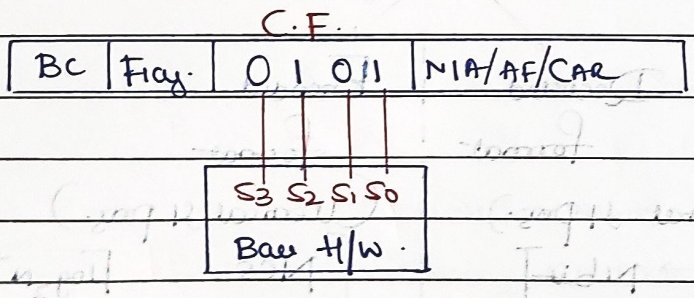
Enable = 1
Disable = 0



3. Design a Horizontal μ Instruction for C.S = (S_0 & S_2)

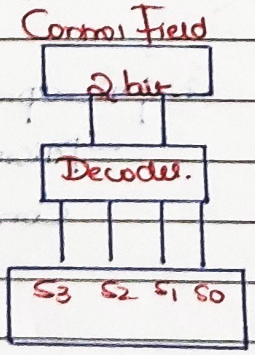


4. Operational State



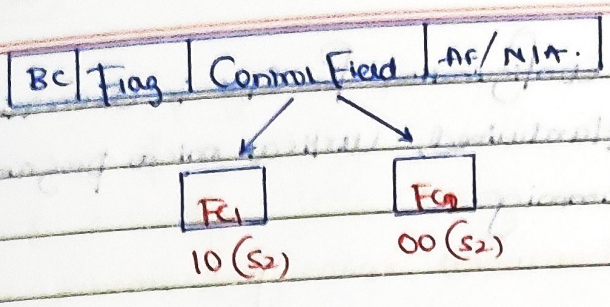
Vertical Microprogrammed CU:

1. Number of control signal in H/W = (S_0 S_1 S_2 S_3)
2. Encoded form of CS ⟹ eg: 4CS = $\lceil \log_2 4 \rceil$ = 2 bit



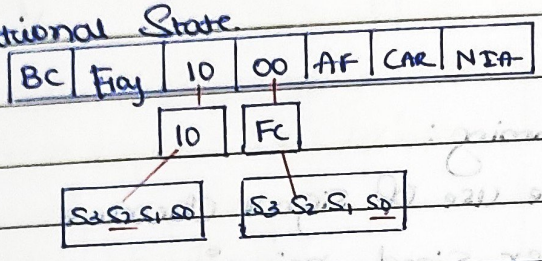
Note: External decoder is required

3. Design a vertical micro instruction for CS = [S_0 S_2]



Function Code [FC] generated by the Control Unit to give the signal to CPU to perform Operation.

4. Operational State.



Horizontal Programming	Vertical Programming
1. In this control signal are expressed in a decoded format.	In this control signal are expressed in encoded format.
2. 1 bit → 1 CS N CS → N bit	n bit → 2^n CS N CS → $\lceil \log_2 N \rceil$ bits
3. It supports longer control words. eg: for 200 control signal we require 200 bit in control field.	It supports shorter control word. eg: for 200 control signal we require 8 bit $\lceil \log_2 200 \rceil$ or 2^8 in control field.
4. No external decoder is required to generate the control signal	External decoder is required to generate the control signal
5. It is flexible compared to hardwired CU.	It is more flexible
6. It support high degree of parallelism (more/more than 1)	It support low degree of parallelism (none/one)

Speed: Hardwired > Horizontal > Vertical
Time Consumption: Vertical > Horizontal > Hardwired.

Page No.
Date. / /

Q2. Arrange the following configuration for CPU in decreasing order of operating speeds: Hardwired, Vertical micro programming, Horizontal micro programming.

Ans Hardwired control, horizontal micro programming, Vertical micro programming.

Q3. Horizontal micro programming:

a. does not require use of signal decoders
b. results in larger sized microinstructions than Vertical micro programming
c. Use one bit each control signal
d. all of the above

Q4. An instructions set of a processor has 125 signals which can be divided into 5 groups of mutually exclusive signals as follows:

Group 1: 20 signals, Group 2: 70 signals, Group 3: 2 signals
Group 4: 10 signals, Group 5: 23 signals.

How many bits of the control words can be saved by using Vertical micro programming over horizontal?

Ans.

	Horizontal	Vertical	
G1: 20CS	20	5 bit	# bit = 125-22
G2: 70CS	70	7 bit	Saved = 103
G3: 2CS	2	1 bit	
G4: 10CS	10	4 bit	
G5: 23CS	23	5 bit	
	125 bit	22 bit	

Q5. Consider a hypothetical CU it has 1024 control words memory It support 48 control signals and 16 Flags. What is the size of the control words in bits and control memory in byte using:

(i) Horizontal Programming
(ii) Vertical Programming

Control memory = 1024 control word = 2^{10} CW → Control address / AF = 10 bit

Flags = 16 Flag = 2^4 flag
= 4 bit

(i) Horizontal Programming

48 Control signal = CF = 48 bit.

Flag	CF	AF
4 bit	48 bit	10 bit

1 CW = 4 + 48 + 10 = 62 bit

Control memory = 1024 CW
= 1024 × 62 bit
≈ 8K Byte

(ii) Vertical Programming

48 Control Signal = CF = 16 bit

Flag	CF	AF/NIA
4 bit	6 bit	10 bit

1 Control Word = 4 + 6 + 10 = 20 bit

Control memory = 1024 CW
= 1024 × 20 bit
≈ 3K Byte

Q6. Control field of a micro instruction support 2 groups of control signal in which Group1 indicate None/one of 400 control signal and Group2 (Horizontal) indicate 6 signals. Hardwire contain 16 Flags and 32 branch condition.
If CAR register size is 20 bit then what is COR in bit and Control memory in bit?

Ans. 16 Flag ⇒ Flag = 4 bit. 32 Branch Condition = BC = 5 bit. CAR/AF = 20 bit

BC	Flag	Control Field	AF (CAR/NIA)
5 bit	4 bit		20 bit

Vertical G_1 — 400 CS ⇓ 9 bit

G_2 Horizontal — 6 CS ⇓ 6 bit

+ = 15 bit

1 CW Size = 5 + 4 + 15 + 20 = 44 bit

Control memory = 2^{20} CW = 2^{20} × 44 bit
= 44 mb bit

	RISC Reduced Instn. Set computer.	CISC Complex Instr. Set Computer.
1.	It supports less number of addressing modes.	It supports more number of Am.
2.	It supports smaller instruction Set.	It support large instruction Set.
3.	It supports more number of registers.	It supports less number of register.
4.	It supports fixed length instr.	It support variable length instr.
5.	It supports 1 instruction per cycle (CPI = 1)	It does not support 1 instruction Per cycle. (CPI ≠ 1)
6.	It supports pipeline successfully.	It supports unsuccessful pipeline.
7.	It is the expensive Processor used in real time applications.	It is less expensive processor.
8.	It is a Super Computer.	General purpose Computer.
9.	It uses hardwired Control unit (Motorola Processor, Power processor, Arm processor).	It used microprogrammed (vertical) control unit (Pentium processor).

Q7. A micro Program Control Unit is required to generate a total of 25 control signals. Assume that during any micro instruction at most two control signals are active. Minimum number of bit required in the control word to generate the required Control Signal.

Ans 25 C ⇒ 5 bit.

⇒ 10 bit

5 bit	5 bit
FC1	FC2

Q8. A hypothetical processor contain word length of 12 bit. It support 1 word Opcode. Each instruction take 8 cycle to comp-lete the execution. Processor contain 256 control signal & 16

Page No.
Date: 06/10/24

Stage Processor used horizontal control unit. 64 branch is used then what is the size of CAR & CBR in bit of control memory?

Ans Word length = 12 bit
Opcode = 1 word = 12 bit
Total # of instructions = 2^{12}
cycle/instruction = 8 cycle/instruction
Total no. of μ Instructions = $2^{12} \times 8$
$= 2^{15}$ μ Instructions/CW.

Control memory = 2^{15} CW
No. of CW addr = 15 bit.

16 Flag => Flag = 4 bit
64 Branch Condition = BC = 6 bit

← 1 CW/CBR →

Flag	BC	Control Field	CAR
4 bit	6 bit	256 bit	15 bit

Horizontal : 256 CS = 256 bit

1 CW = 281 bit

Control Memory = 2^{15} CW
$= 2^{15} \times 281$ bit

Q9- Consider the following processor design characteristics:
I. Register-to-register arithmetic operations only
II. Fixed length instruction format
III. Hardwired control unit
Which of the characteristics above are used in the design of a RISC processor?

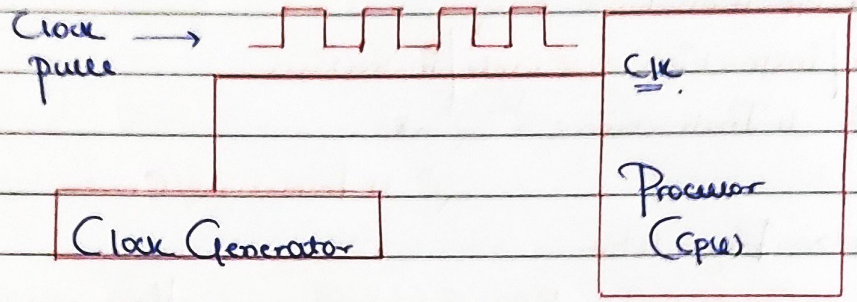
Ans I, II and III

CPU Time Calculation:

* CPU time calculation Program execution time.
* Program execution time is based on the clock.
* Processor contain clock pins and these clock pin is externally connected with the clock generator. Or
So In the computer System all the operation are controlled by the clock, so CPU contain clock pins which is externally connected with clock generator.

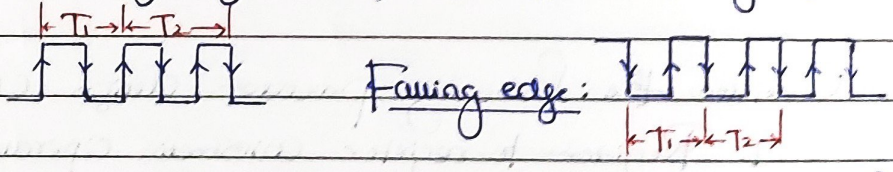
Page No.
Date. / /

* Clock generator is operating with a constant frequency to generate the clock pulse (clock signal).
* These clock signals are carried into the CPU through (with the help of) clock pin. So CPU operations are controlled by the clock signal.



* Program execution time is calculated based on 2 factors:
  1. Cycle
  2. Cycle time

(1.) Cycle: Cycle is defined as clock pulse transition either from rising edge to rising edge (or) falling edge to falling edge.



(2) Cycle Time: The time required to transfer the pulse either from rising edge to rising edge or falling edge to falling edge is called as cycle time.
Cycle time depends on the clock frequency.

$$\text{Cycle time} \propto \frac{1}{\text{Clock frequency}}$$

Q. In program we have 100 instructions, and each instructions takes 6 clock cycles. Operating at 2GHz processor program E.T = ?

Ans 100 Instructions, each takes 6 cycles
Total = 100 × 6 = 600 cycles
Cycle time = $\frac{1}{2GHz}$ = 0.5 nsec
Program E.T = 600 × 0.5 nsec = 3000 nsec

Page No.
Date. / /

CPU Time Calculation / Program ET

Program ET = # seconds/Program
= # Instruction/Program × # cycle/instruction × # second/cycle

- # Instruction/Program → Instruction Count
- # cycle/instruction → Cycle per instruction
- # second/cycle → Cycle time

$$\text{Program ET} = Ic \times CPI \times \text{Cycle time}$$

* Program is a combination of data transfer, data manipulation, & transfer of control (TOC) instruction. Different instruction takes (consume) different cycle to complete the execution so,

$$\text{Program ET} = \left[\sum (ICi \times CPIi)\right] \text{Cycle time}$$

i = instruction.

Let CPI_i be the number of cycles required for the instruction type i and I_i be the number of executed instructions of type i for a given program. Then we can calculate an overall (average) CPI as follows:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{Ic}$$

The processor time T needed to execute a given program can be expressed as $T = Ic \times CPI \times \tau$ (where $\tau = 1/f$, and Ic = instr. count).

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{Ic}{T \times 10^6} = \frac{f}{CPI \times 10^6} \quad \text{MIPS}$$