ane 24

opcode = operational code (Add, sub ....)
SP = Stack pointer

Computer architecture

• **Instruction formats** :- It is defines the layout of the bits of an instruction. An instruction format must include an opcode and zero or more operands in one of the addressing mode.

• **Instruction classification:**

Instruction class	Example
• Three-address	ADD R_1, R_2, R_3 $R_1 \leftarrow R_2 + R_3$ or ADD A, B, C. $M[A] \leftarrow M[B] + M[C]$
• Two address	ADD R_1, R_2 $R_1 \leftarrow R_1 + R_2$ or ADD A, B $M[A] \leftarrow M[A] + M[B]$
• One address	ADD R_1 $AC \leftarrow AC + R_1$
• Zero address	ADD (SP)+, (SP)

Example:-

$$x = \frac{A + B * C}{(D - E * F + G * H)}$$

Three address instruction format:

MUL $R_1, B, C \longleftarrow R_1 \leftarrow M[B] * M[C]$

ADD $R_1, R_1, A \longleftarrow R_1 \leftarrow R_1 + M[A]$

MUL $R_2, E, F \longleftarrow R_2 \leftarrow M[E] * M[F]$

SUB R_2, D, R_2 ⟵ $R_2 \leftarrow M[D] - R_2$

MUL R_3, G, H ⟵ $R_3 \leftarrow M[G] * M[H]$

ADD R_2, R_2, R_3 ⟵ $R_2 \leftarrow R_2 + R_3$

DIV X, R_1, R_2 ⟵ $M[X] \leftarrow R_1 \div R_2$

Computer Architechture

- $x = \frac{(A+B*C)}{(D-E*F+G*H)}$

Two address:

MOV R_1, B	$R_1 \leftarrow M[B]$
MUL R_1, C	$R_1 \leftarrow R_1 * M[C]$
ADD R_1, A	$R_1 \leftarrow R_1 + M[A]$
MOV R_2, D	$R_2 \leftarrow M[D]$
MOV R_3, E	$R_3 \leftarrow M[E]$
MUL R_3, F	$R_3 \leftarrow R_3 * M[F]$
SUB R_2, R_3	$R_2 \leftarrow R_2 - R_3$
MOV R_3, G	$R_3 \leftarrow M[G]$
MUL R_3, H	$R_3 \leftarrow R_3 * M[H]$
ADD R_2, R_3	$R_2 \leftarrow R_2 + R_3$
DIV R_1, R_2	$R_1 \leftarrow R_1 \div R_2$
MOV x, R_1	$M[x] \leftarrow R_1$

$$x = \frac{(A + B * C)}{(D - E * F + G * H)}$$

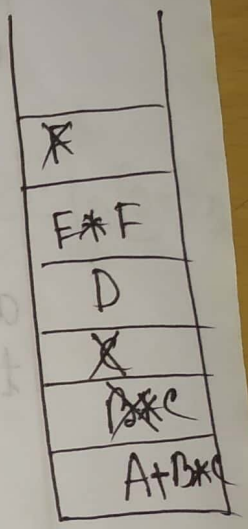
One address instruction format : (নিচ থেকে কাজ শুরু করতে হবে)

LOAD E	AC ← M[E]
MUL F	AC ← AC * M[F]
STORE T	M[T] ← E * F
LOAD D	AC ← M[D]
SUB T	AC ← AC − M[T]
STORE T	M[T] ← AC or D − E * F
LOAD G	AC ← M[G]
MUL H	AC ← AC * M[H]
ADD T	AC ← AC + T
STORE T	T ← AC or D − E * F + G * H
LOAD B	AC ← M[B]
MUL C	AC ← AC * M[C]
ADD A	AC ← AC + M[A]
DIV T	AC ← AC ÷ M[T]
STORE x	M[x] ← AC

$$x = \frac{(A+B*C)}{(D-E*F+G*H)}$$

Zero address instruction format : প্রথমে postfix এ convert করব। Then push/pop করব। Stack use করা হবে। Result আসবে infix এ

$x = ABC*+DEF*-GH*+/$

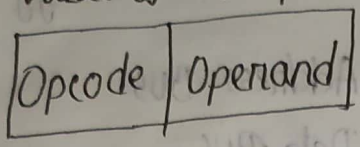


PUSH A	TOS ← A
PUSH B	TOS ← B
PUSH C	TOS ← C
MUL	TOS ← B*C
ADD	TOS ← A+B*C
PUSH D	TOS ← D
PUSH E	TOS ← E
PUSH F	TOS ← F
MUL	TOS ← E*F
SUB	TOS ← D-E*F
PUSH G	TOS ← G
PUSH H	TOS ← H
MUL	TOS ← G*H
ADD	TOS ← D-E*F+G*H
DIV	TOS ← (A+B*C)/(D-E*F+G*H)
POP X	

CA:

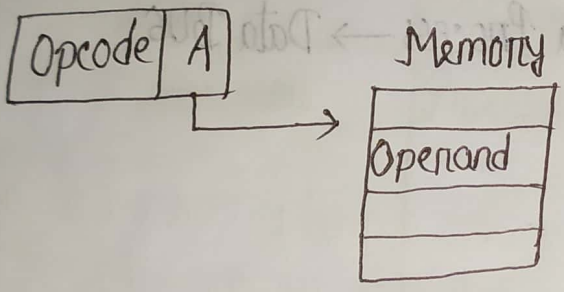
1. Implicit: CMA
2. Immediate
3. Direct
4. Indirect
5. Register
6. Register Indirect
7. Displacement
   - Relative
   - Base-Register
   - Indexing
9. Stack

Immediate: Value of the operand is in the instruction.



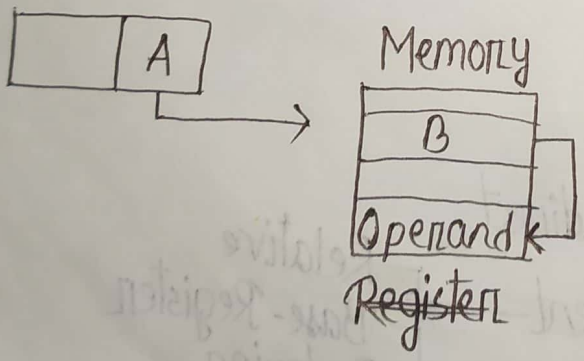
Ex- Load Ri, #1000 , Ri ← 1000

Direct: Address of operand include in instruction.



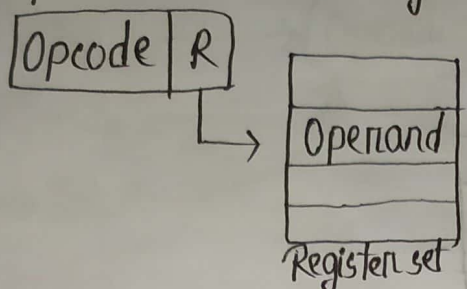
Ex- Load Ri, 1000; Ri ← M[1000]

Indirect: Address of operand is in the address mention in the instruction.



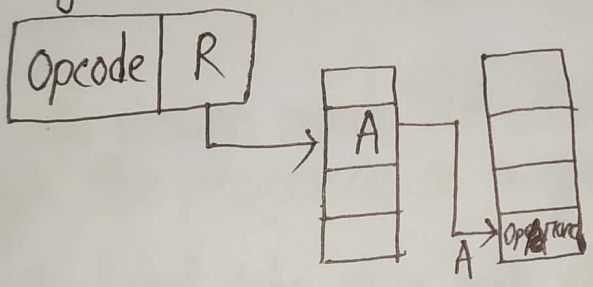
Ex- ~~Operand~~ Load Ri, (100) ; Ri ← M[M[100]]

Register: Operand is in the register mentioned in the register.



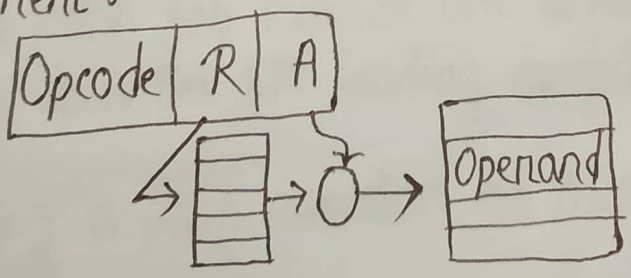
Ex- Load Ri, Rj ; Ri ← Rj.

Register Indirect: Operand address in the address mentioned in the register content.

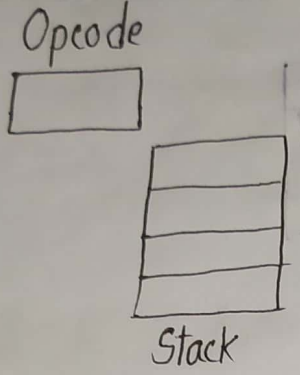


Ex- Load Ri, (Rj)
Ri ← M[Rj]

Displacement:



Stack:



10.07.2019

CA

35

Mode	Algorithm	Criteria
Immediate	Operand = A	No memory reference
Direct	EA = A ↓ Effective address/ address of operand	Simple
Indirect	EA = (A) ↓ () → আরেকটা address যেখানে data টা আছে	Large address space
Register	EA = R	Simple
Register indirect	EA = (R)	Large address space
Stack	EA = Top of stack	no memory reference

Data Dependency

↳ একটা instruction এর জন্য একটা instruction এর যে relationship.
Ex- একটার input অন্য একটার output হতে পারে।

1. Flow dependance $S1 \rightarrow S2$ [The output of S1 is the input of S2]

2. Antidependance $S1 \not\rightarrow S2$ [Flow is coming from S1 to S2 but input of S1 will be output of S2]

3. Output dependance $S1 \rightarrow\!\!\!\!\!\!\bigcirc\; S2$ [S1, S2 if both statement are writing on the same variable]

4. I/O dependance $S1 \xrightarrow{I/O} S2$ [If both the device are trying to access the same I/O device]

5. Unknown dependence

i. Subscript of a variable is itself subscripted

ii. Subscript does not contain the loop index variable.

iii. Subscript is non-linear in the loop index variable.

Data Dependency

↳ একটা instruction এর জন্য একটা instruction এর যে relationship.
Ex- একটার input অন্য একটার output হতে পারে।

1. Flow dependance $S1 \rightarrow S2$ [The output of S1 is the input of S2]

2. Antidependance $S1 \not\rightarrow S2$ [Flow is coming from S1 to S2 but input of S1 will be output of S2]

3. Output dependance $S1 \rightarrow\!\!\!\!\!\!\circ\ S2$ [S1, S2 if both statement are writing on the same variable]

4. I/O dependance $S1 \xrightarrow{I/O} S2$ [If both the device are trying to access the same I/O device]

5. Unknown dependence

i. Subscript of a variable is itself subscripted

ii. Subscript does not contain the loop index variable.

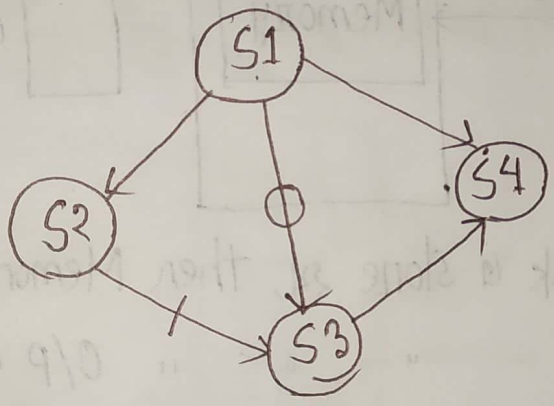
iii. Subscript is non-linear in the loop index variable.

Ex-

S1	Load R1, A	R1 ← M[A]
S2	ADD R2, R1	R2 ← R1 + R2
S3	MOVE R1, R3	R1 ← R3
S4	STORE B, R1	M[B] ← R1

Draw the dependency graph:

~~Depency~~



CA  45

Computer architecture and organization – John P. Hayes.

CA কাজ করে Machine lang. নিয়ে। Assembly lang. দুই ভাগে বিভক্ত → Opcode, Operand.

SL → Shift left
CSL → Circular shift left
SR → Shift right
CSR → Circular shift right
ASR → Arithmatic shift right

Reset = কোনো তথ্যকে 0 করে দেওয়া
Set = 1 করে দেওয়া

1 neble = 4 bit

• R ← 11010101
Upper neble, lower neble

SL. R → 10101010 [last bit এ 0 দিতে হবে। MSB delete হয়ে LSB খালি হয়, 0 হয়]

CSL R 10101011 [MSB = LSB হয়]

SR R 01101010

CSR R 11101010

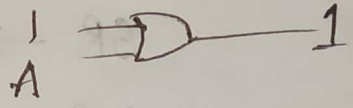
ASR R 11101010 [sign bit থাকবে, then shift হবে] sign bit নয়

• Reset upper nibble.

AND R, OF H → Hexa decimal

R → 1 1 0 1 0 1 0 1
AND 0 0 0 0 1 1 1 1
---
0 0 0 0 0 1 0 1

• Rest lower nibble



AND R, F 0 H

R → 1 1 0 1 0 1 0 1
AND 1 1 1 1 0 0 0 0
---
1 1 0 1 0 0 0 0

• Set upper nibble

OR R, F 0 H

R → 1 1 0 1 0 1 0 1
OR 1 1 1 1 0 0 0 0
---
1 1 1 1 0 1 0 1

Q- What is the final Content of Register R of the end of the following assembly language code ?

```
MOVE  R , E7H
CSR   R
ASR   R
CSL   R
AND   R , F0H
HLT
```

Soln:

R ← 11100111

CSR R ← 11110011

ASR R ← 11111001

CSL R ← 11110011

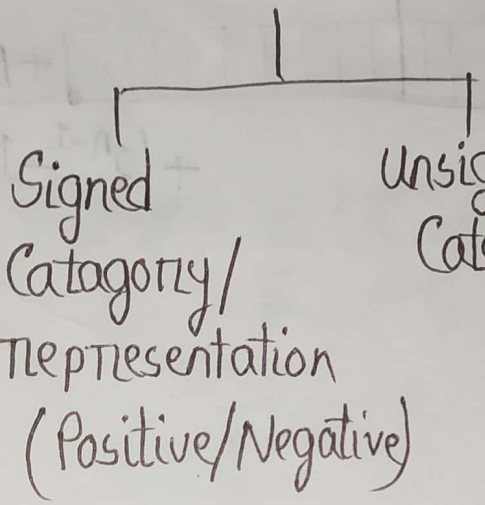
AND R, F0H

```
   R ← 11110011
       11110000
  ----------------
   R ← 11110000
```

R ← F0H

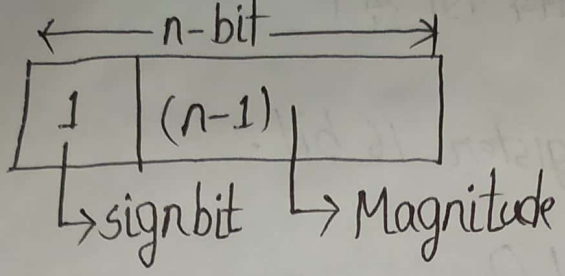
17.07
CA

Number Representation



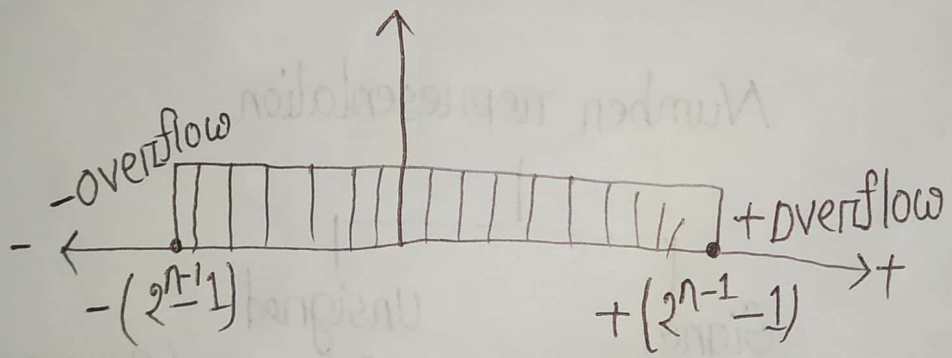
- Difference btwn signed and unsigned?

i. Signed Representation :-



0 ← +ve
1 ← -ve

• Range specification graph :-



57

16	8	4	2	1
1	0	1	1	0

- Represent -22 in 8-bit memory.

+22 → 00010110

1's – 11101001

+1

2's – 11101010

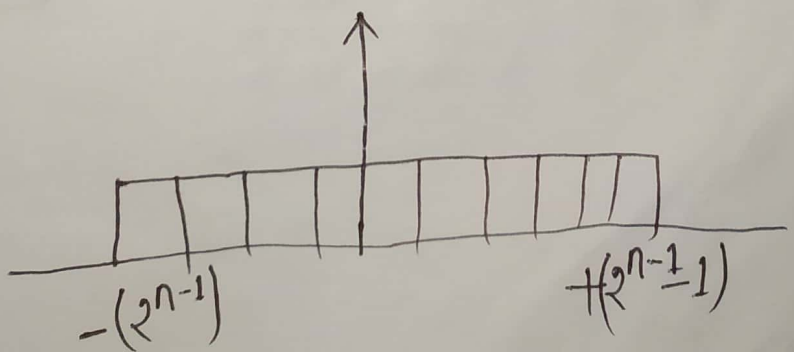
Another way to 2's → (1111 1111 – 00010110) + 1

" " → 2^n – number

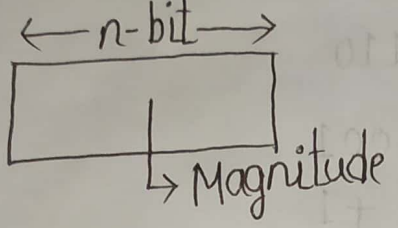
- Signed representation (using 2's complement)

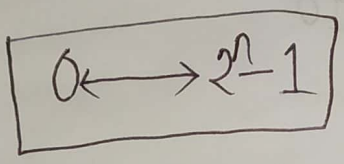
Upper range = + $(2^{n-1} - 1)$

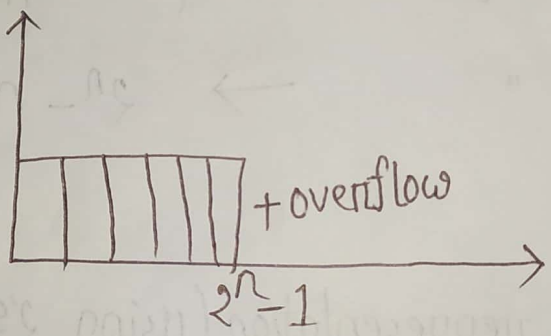
Lower " = − (2^{n-1})



ii. Unsigned Representation:







22.06

CA 77

Overflow:-

- Overflow in unsigned addition → Result occupies more number of bits.
- Overflow in signed-magnitude/2's complement addition :→
  - i. Result occupies more number of bits.
  - ii. Addition of two positive number resulting negative number.
  - iii. Sum of two negative number " positive number. → Underflow

Overflow	Underflow
i. When addition of two binary number yields a result that is greater than the maximum positive value.	i. When the addition/substraction of two binary number yields a result that is less than the minimum possible value.
i. Assume 4-bit restriction and 2's complement.	ii. Assume 4-bit restriction and 2's complement.
iii. Maximum possible value $= 2^{n-1} - 1$ $= 2^{4-1} - 1 = 7$	iii. Minimum possible value $= -(2^{n-1})$ $= -8$

7 →	0 →
6 →	-1 → 1111
5 →	-2 → 1110
4 →	-3 → 1101
3 →	-4 → 1100
2 →	-5 → 1011
1 →	-6 → 1010
0 →	-7 → 1001
	-8 → 1000

$6_{(10)} + 3_{(10)}$

6 → 0110
3 → 0011
-7 → 1001

$-5_{(10)} + (-5)_{10}$

-5 → 1011
-5 → 1011
+6 → 1 0110

```
  1 1 0 1 (-3)
  1 0 1 1 (-5)
-------------
(1) 1 0 0 0 (-8)
```

(carry generate)

```
  1 1 0 1 (-3)
  1 0 1 0 (-6)
-------------
(1) 0 1 1 1
```

(carry + underflow)

```
  1 1 0 1 (-3)
  0 0 1 0 (+2)
-------------
  1 1 1 1
```

(Nothing)

```
  0 1 1 1 (+7)
  0 0 1 1 (+3)
-------------
[1] 0 1 0
```

(Overflow)

CA

Unsigned Multiplication (Paper-Pencil Method :-

Interface :

i. Multiplication involves the generation of partial product, one for each digit in the multiplier. These partial products are then summed to produce the final product.

ii. The partial products are easily defined. When the multiplier bit is '0', the partial product is '0'. When the multiplier is 1, the partial product is multiplicand.

iii. The total product is produced by summing the partial product. For this operation, each successive partial product is shifted one position left relative to the preceding partial product.

iv. The multiplication of two n-bit binary integers results in a product of upto 2n bits in length.

Ex- 1011 ⟶ Multiplicant
1101 ⟶ Multiplier

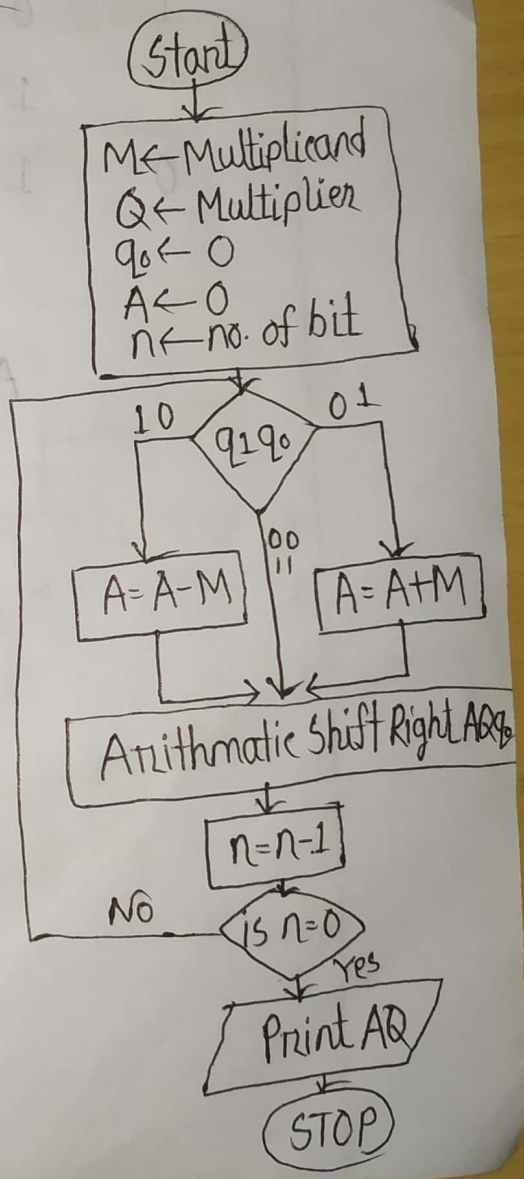
1011
0000
1011
1011 } Partial product

10001111 ⟶ Final Product

(95%) Signed Multiplication (Booths Algorithm)

Ex- $(-6) \times (+4) = (-24)$

$M = 1010, -M = 0110$
$Q = 0100$
$q_0 = 0$
$A = 0000$
$n = 4$



Tracing Table

n	A	Q (q_4 ↑ Q_1)	q_0	Action
4	0000	0100	0	Initialization
3	0000	0010	0	ASR AQq_0
2	0000	0001	0	ASR AQq_0
	0110	0001	0	A=A-M
1	0011	0000	1	ASR AQq_0
	1101	0000	1	A=A+M
0	1110	1000	0	ASR AQq_0

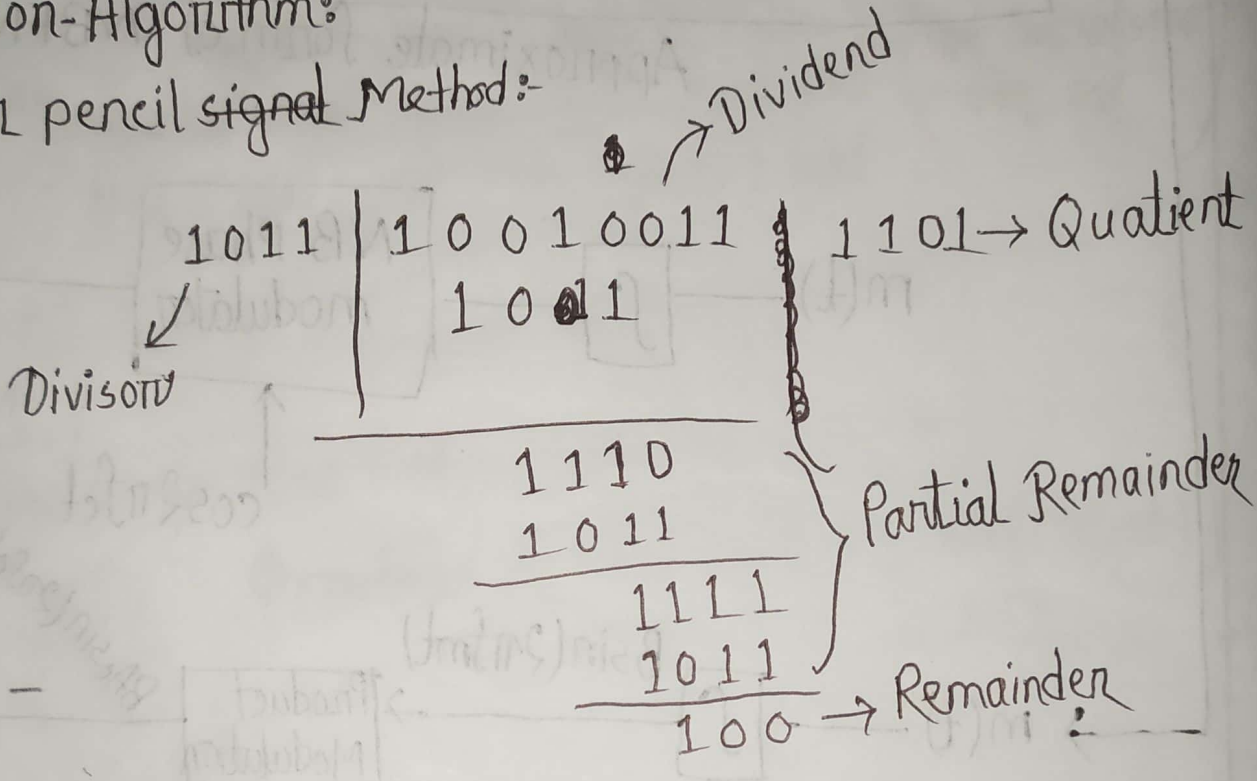
AQ = 1110 10000

29.06

CA

Division-Algorithm:

Paper pencil signal Method:-



Restoring Method :-

→ Partial Remainder is restored by adding the divisor to the negative difference

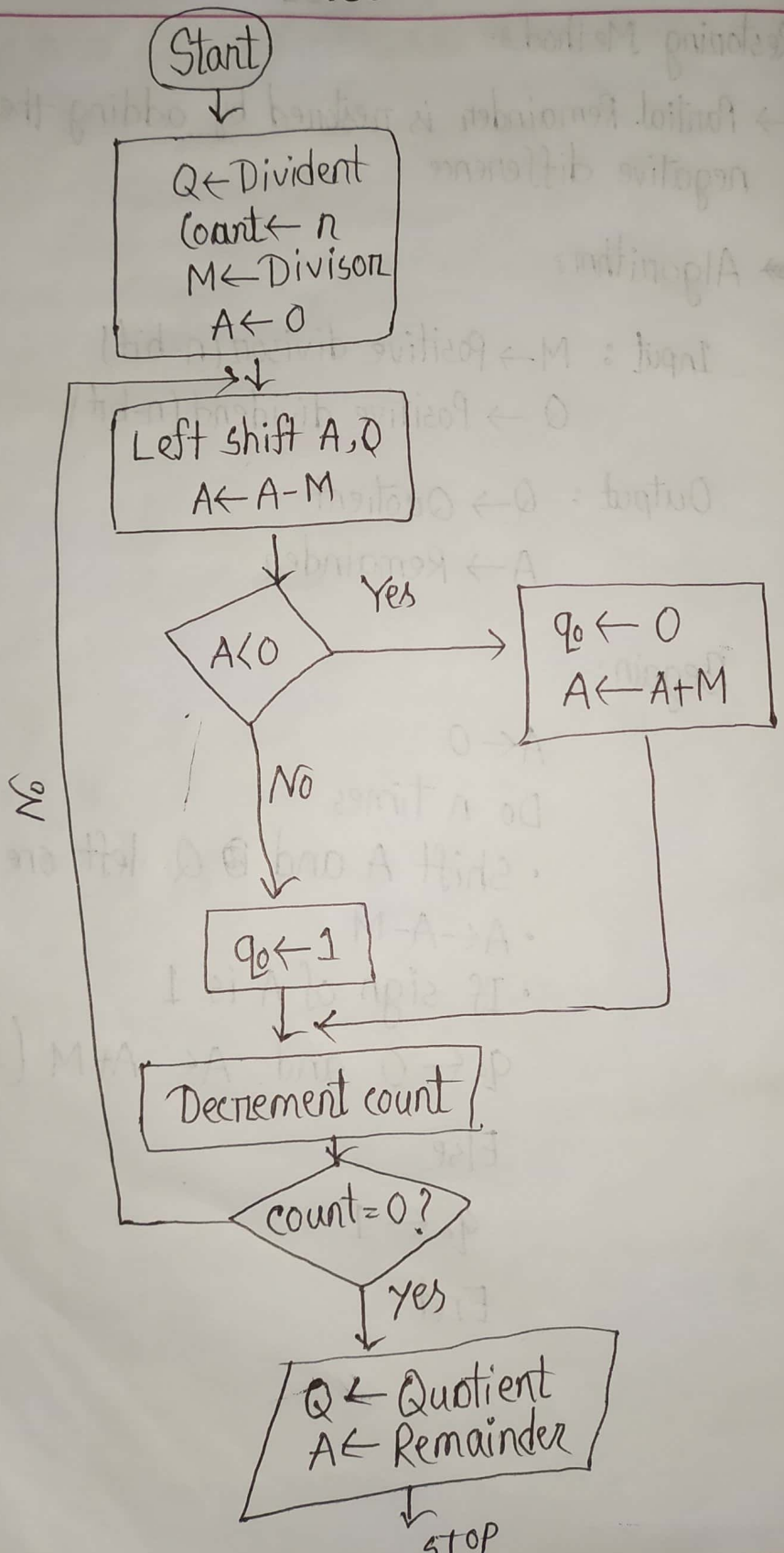
⇒ Algorithm:

Input : M → Positive divisor (n-bit)
Q → Positive dividend (n-bit)

Output : Q → Quotient
A → Remainder

Begain:-

A ← 0
Do n times
• Shift A and Q left one binary position
• A ← A-M
• If sign of A is 1
$q_0 \leftarrow 0$ and A ← A+M (restore A)
Else
$q_0 \leftarrow 1$
End.



Example

Tracing

0011
1101
0000

0001
1101
1110

1110
0011
0001

Example - 7/3

M → 0011

-M → 1101.

Exam q

Tracing table :

count	A	Q	Comment
4	0000	0111	Initialization
	0000	1110	SL AQ
	1101	1110	A ← A-M
3	0000	1110	$q_0 \leftarrow 0$, A ← A+M
	0001	1100	SL AQ
	1110	1100	A ← A-M
2	0001	1100	$q_0 \leftarrow 0$, A ← A+M
	0011	1000	SL AQ
	0000	1000	A ← A-M
1	0000	1001	q ← 1
	0001	0010	SL AQ
	1110	0010	A ← A-M
	0001	0010	$q_0 \leftarrow 0$, A ← A+M

A = 10

Q = 1

0011
1101
10000

0001
1101
1110

1110
0011
10001

1010 +1 = 1011

Parity Bit → Even (1 এর সংখ্যা যুগ্ম)
→ Odd

1101 [1] 1001 [1]

কোন সংকেত bit change হলে error হচ্ছে তা না বুঝলে ধারণা না।

Sender

$M = D_8, D_7, D_6, D_5, D_4, D_3, D_2, D_1$

$D_8 - D_1\ P_4 P_3 P_2 P_1$

$P_4 \rightarrow D_8 \oplus D_5 \oplus D_3 \oplus D_1$

$P_3 \rightarrow D_7 \oplus D_5 \oplus D_3 \oplus D_1$

$P_2 \rightarrow D_8 \oplus D_6 \oplus D_1$

$P_1 \rightarrow D_6 \oplus D_5 \oplus D_4 \oplus D_1$

Receiver

$D_8' - D_1\ P_4' P_3' P_2' P_1'$

$P_4'' =$

$P_3'' =$

P_2''

P_1''

X-OR করলে 0 পাইলে data genuine হবে।

Single error correction 2

Hamming Code (SEC DED Method)

Double ~~data~~ error detection

Possible parity combination = 2^P

max Error data = $2^P - 1$

Message bit = Data bit + Parity bit

$N = D + P$

$N \leq 2^P - 1$ → এটা দিয়ে parity bit এর সংখ্যা নির্ণয় করা হয়

$n = 8 \rightarrow P = 4$

$= 14 \quad P = 4$

$= 20 \quad P = 5$

Message bit : $M_{12}\ M_{11}\ M_{10}\ M_9\ M_8\ M_7\ M_6\ M_5\ P_4\ M_3\ M_2\ M_1$

~~Parity bit~~

Data bit : $D_8\ D_7\ D_6\ D_5\ \quad D_4\ D_3\ D_2\ \quad D_1$

Parity bit : $\quad\quad\quad\quad P_4\quad\quad\quad P_3\quad P_2\ P_1$

Error No	Parity bit P_4	P_3	P_2	P_1	Comment
1	—	—	—	1	P_1 bit error
2	—	—	1	—	P_2 " "
3	—	—	1	1	D_1 " "
4	—	1	—	—	P_3 " "
5	—	1	—	1	D_2 " "
6	—	1	1	—	D_3 " "
7	—	1	1	1	P_4 " "
8	1	—	—	—	D_4 " "
9	1	—	—	1	D_5
10	1	—	1	—	D_6
11	1	—	1	1	D_7
12	1	1	—	—	D_8

Msg bit error Data bit

$P_4 \rightarrow \{9, 10, 11, 12\} = \{5, 6, 7, 8\}$

$P_3 \rightarrow \{5, 6, 7, 12\} = \{2, 3, 4, 8\}$

$P_2 \rightarrow \{3, 6, 7, 10, 11\} = \{1, 3, 4, 6, 7\}$

$P_1 \rightarrow \{3, 5, 7, 9, 11\} = \{1, 2, 4, 5, 7\}$

DFD = Sending data

8	7	6	5	4	3	2	1
Data = 1	1	0	1	0	1	0	GP = 1

যে কলাম যতটি parity bit আছে তাকে generate bit বলে

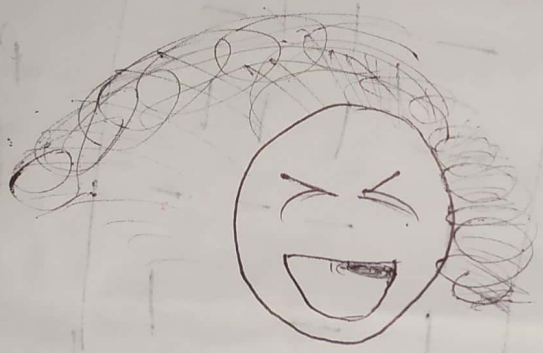
$P_1 = \{1, 0, 1, 1\} = 1$

$P_2 = \{0, 1, 0, 1\} = 0$

$P_4 = \{1, 1, 0, 1\} = 1$

$P_8 = \{1, 0, 1, 1\} = 1$

DED Sending



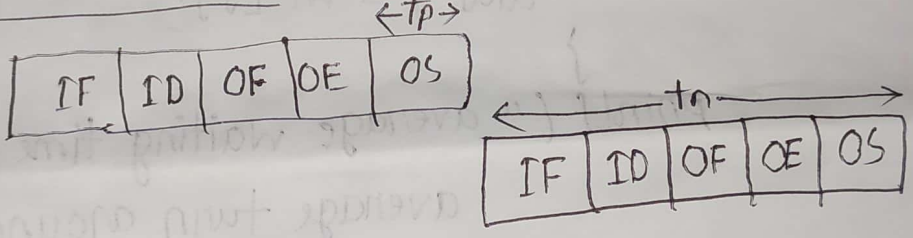
19.08)

CA 12

instruction গুলো fetch phase wise কাজ করে।

Phase → IF → Instruction fetch (Opcode নিয়ে কাজ)
fetch
ID → " Decode (" এর প্রসেস. চালাবে)
OF → Operand Fetch
OE → " Execute
OS → " Store

Non-Pipeline Architecture :-



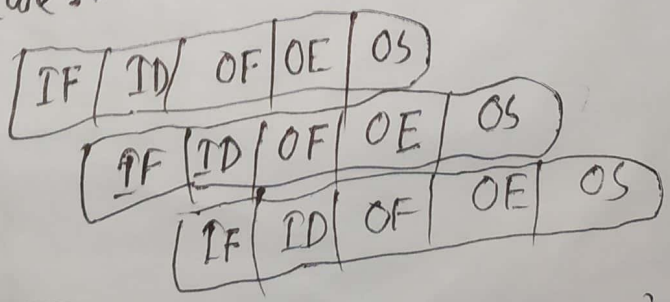
Time বেশি লাগে

IF ID ---

instructionwise কাজ করে। slower
একটা instruction এর কাজ শেষ হবে then অন্যটা শুরু হবে।

t_n = total time = $t_p \times k$
t_p = প্রতি fetch এর কাজ শেষ time লাগে
$t = n \times t_n$

Pipeline Antitecture :-



fetchwise কাজ করে।
একসাথে একই fetch execute করবে না। একটা fetch হয়ে
যাওয়ার পর অন্য instruction এর শুরু হবে।

Time Overlapping
So much fast
→ Differen

- Time overlapping হচ্ছে।
- So much faster

→ Difference btwn non and pipeline architechture.

For pipeline

Segment No	clock Cycle 1	2	3	4	5	6	7
1	t_1	t_2	t_3				
2		t_1	t_2	t_3			
3			t_1	t_2	t_3		
4				t_1	t_2	t_3	
5					t_1	t_2	t_3

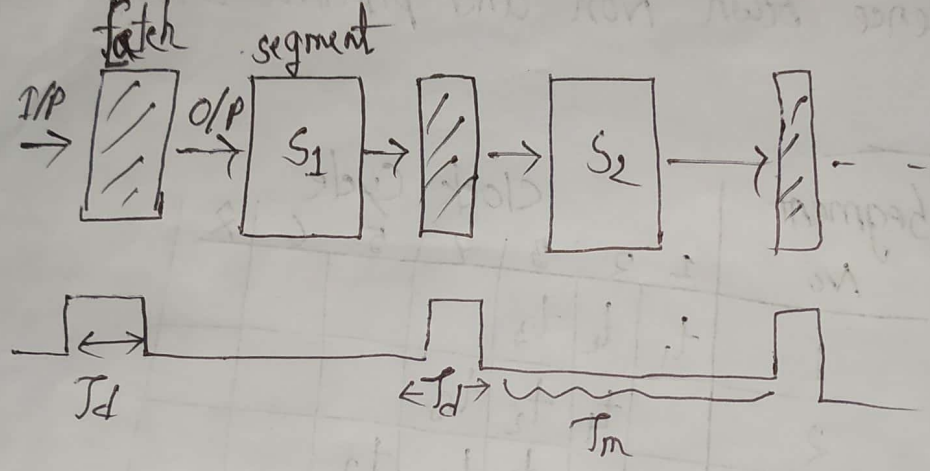
n = no. of instruction
k = " " fetch

Total Unit time = $\boxed{k+n-1}$

$= 5+3-1$

$= 7$

• fetch একটা segment এর সাথে related. একটার input অন্যটার o/p হয়



T_d = delay time এক seg থেকে অন্যটাতে যাইতে

T_m = seg. সমূহের যে tym লাগে

T = total time

$$= \max\left(T_{i=1}^{k}\right) + T_d$$

$$= T_m + T_d$$

→ clock signal use করে এটা use করা হয় তাই একে synchronous ও বলা হয়।

15

pipline
- Synchnonous
- Asynchnonous

• Xm এর math আসবে।

↳ Speed up ratio:

$$S_K = \frac{\text{Non-Pipeline Execution time}}{\text{Pipline} \quad " \quad "}$$

$$= \frac{n \times t_n}{(K+n-1)\tau}$$

↳ Frequency, $f = \frac{1}{\tau}$

↳ Efficiency : $E_K = \frac{S_K}{K}$

$$S_K = \frac{n \times K \times t_p}{(K+n-1) \times \tau}$$

$$= \frac{n \times K}{K+n-1} \quad [t_p = \tau \text{ same}]$$

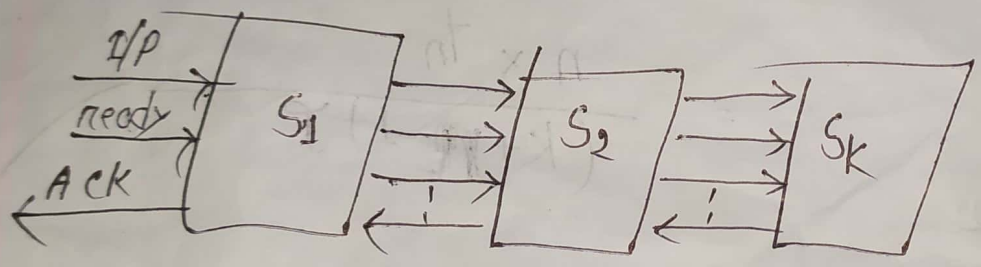
$$E_K = \frac{n}{K+n-1}$$

Throughput → Per unit tym ৫ কতগুলা i/s সম্পন্ন হয়

$$H_k = \frac{n}{(k+n-1)\tau}$$

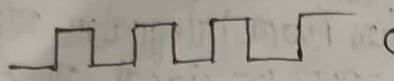
$$= \frac{nf}{k+n-1} \quad \left[f = \frac{1}{\tau}\right]$$

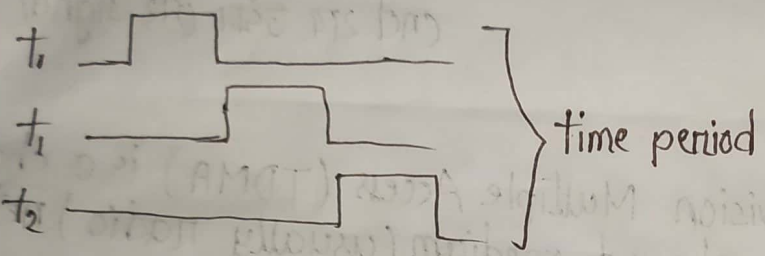
Asynchonous pipline (cp যাকবে না)



একটা i/s সম্পন্ন হওয়ার পর ~~ও~~ ওর O/P অন্যটাতে যাওয়ার জন্য ready থাকে। একটা ক্ষ্P যেওয়ার জন্য এরু অন্যটা I/P নেওয়ার জন্য ready হলে then যাবে।

Control unit : It decides the main operation signal to the data path. It flows between CPU, I/O etc.

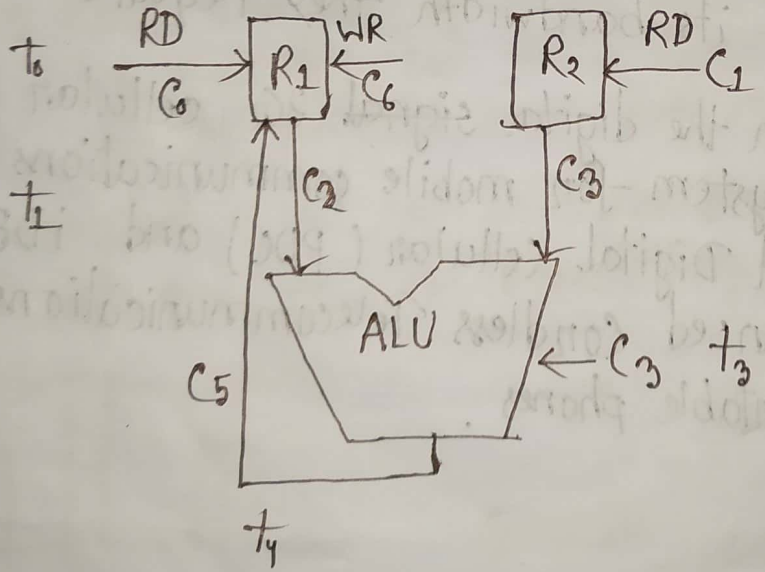




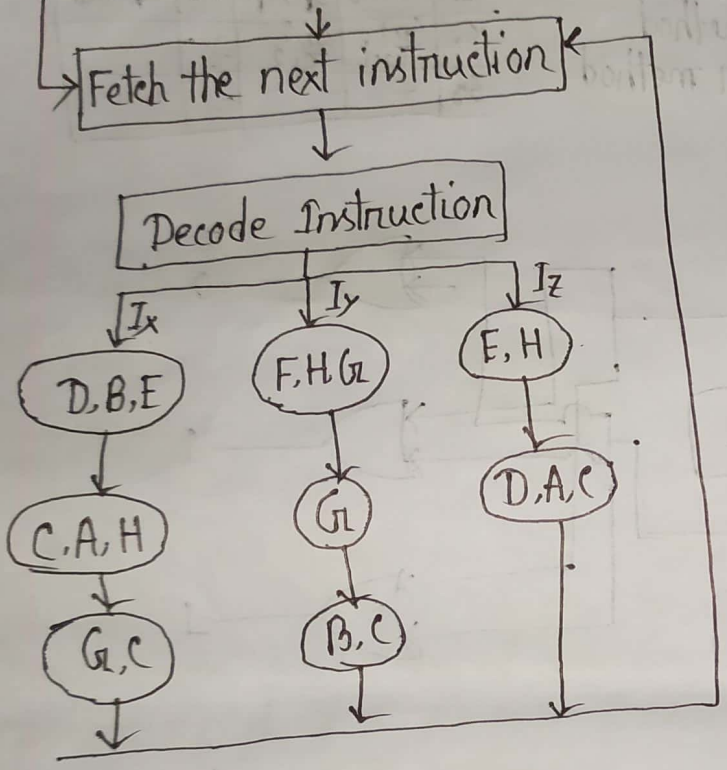
Example:-

ADD R_1, R_2

$R_1 \leftarrow R_1 + R_2$

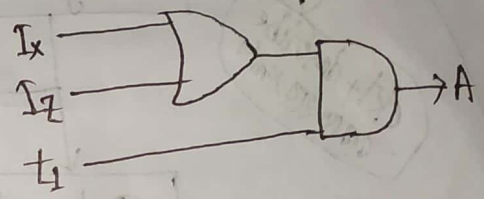


→ Micropnogrammed control unit
→ Handware control unit

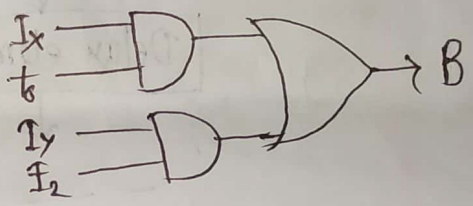


$A = I_x \cdot t_1 + I_z \cdot t_1$

$= (I_x + I_z) t_1$

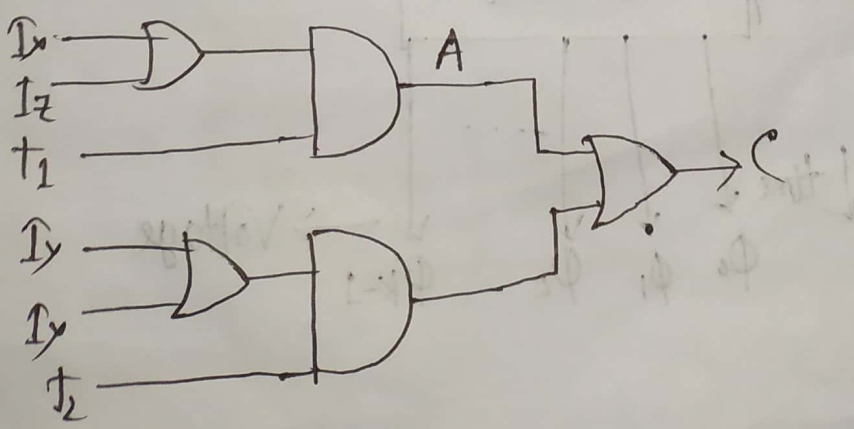


$B = I_x t_0 + I_y t_2$



$C = I_x \cdot t_2 + I_y t_2 + A$

$= (I_y + I_x) t_2 + A$

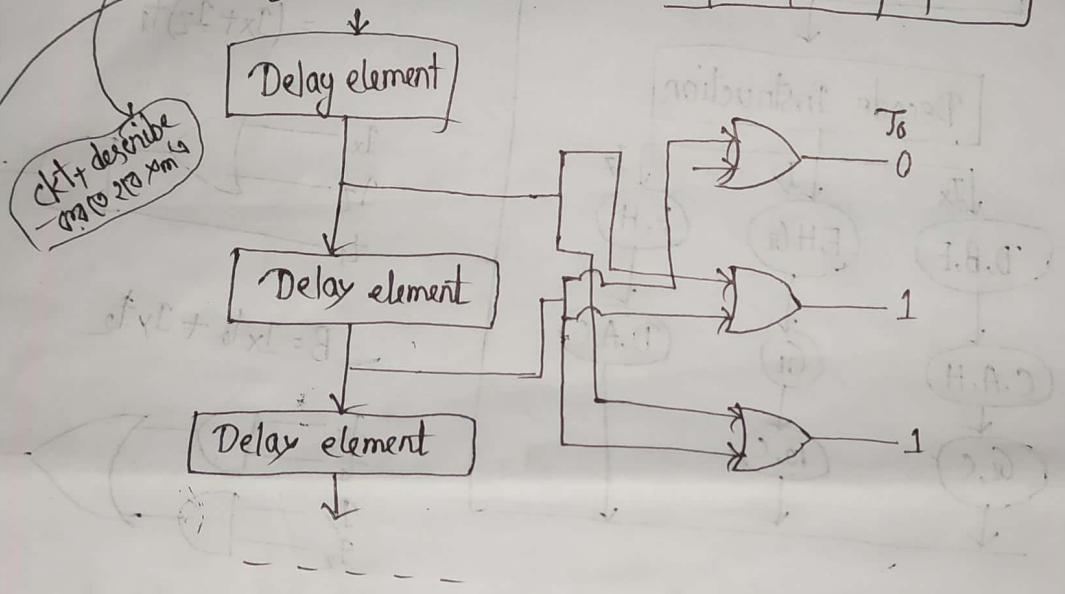


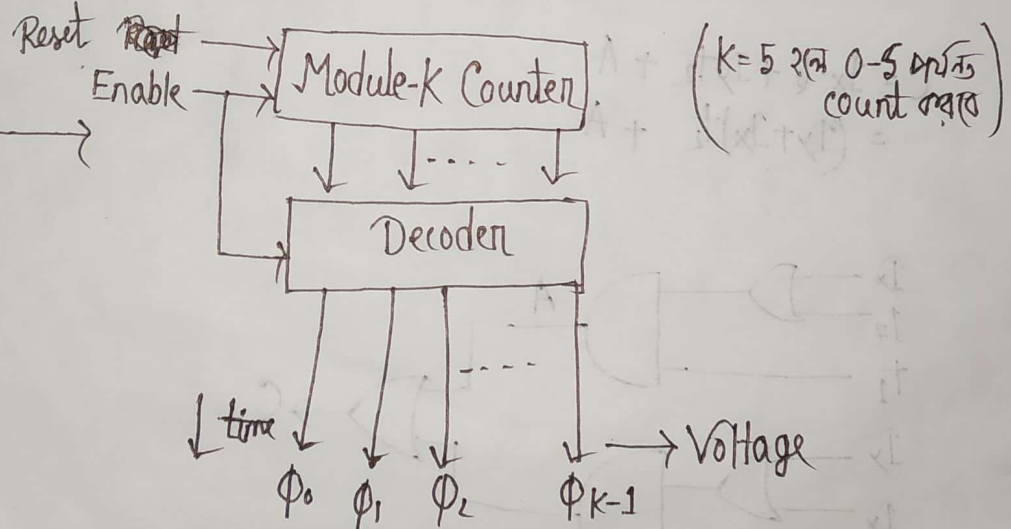
Hardware Control Unit design :

1. State table method
2. Delay element method
3. Sequence counter method

(I/P, current state এর উপর O/P, next state depend করে)

States	Input combinations			
	I_1	I_2	--	I_m
S_1	S_{11}, Z_{11}	S_{12}, Z_{12}	--	S_{1M}, Z_{1M}
S_2	S_{21}, Z_{21}	S_{22}, Z_{22}	--	
S_3	-	-	--	-





(K=5 হলে 0-5 পর্যন্ত count করবে)