Pipelining

- Parallel processing
- multiple functional unit
- flynn's classification ✓
- Pipelining
  - floating +, - in pipeling
  - concept ✓
  - Speed up equation
- Instruction level pipeling
  - instruction cycle
  - 3 and 4 instruction pipeline
  - Pipeline conflict and soln
- Vector processing
  - Application
  - Vector operation
  - Matrix Multiplication

Parallel processing

→ It is the technique which provide simultaneous data processing task for increasing computational speed of computer.

Pipelining

→ It is the process of breaking down the sequential process into sub operation, where each sub process is executed in dedicated segment that operate concurrently with all other program.

→ Collection of processing segment through which binary information flows.

→ Each segment peforms partical processing

→ The result obtained from each segment is transfered to next segment in pipeline.

→ The finial result is obtained after the data have passed through all segment.

→ Several computation can be done at a same time.

→ The overlapping of computation is made possible with the help of associative register.

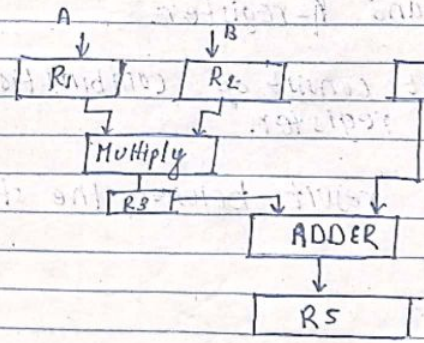
for example:-
we wanna peform operation

$A_i * B_i + C_i$ for $i = 1, 2, \ldots 7$

Here,
- If each suboperation is to be implement within a pipeline.
- each segment have 1 or 2 register and combinational circuit.

Here

$R_1 \leftarrow A_i$	Input A
$R_2 \leftarrow B_i$	Input B
$R_3 \leftarrow R_1 * R_2$	Multiply
$R_4 \leftarrow C_i$	Input C
$R_5 \leftarrow R_3 + R_4$	Addition



K-segment pipeline

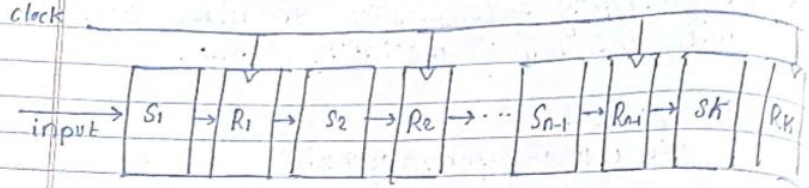


fig: K-segment pipeline

→ Any operation can be broken into a sequence of suboperation of same complexity can be implemented by pipeline processor.

→ Here, in the diagram, there consist K-segment and K-register.

→ Each segment consist of combinational circuit and register.

→ Ri holds the result between the stages

Suppose,
we have K-segment pipeline
let no of clock cycle = tp
no of task = n
Here,

1st task takes K x tp time
Remaning (n-1) task require (n-1) x tp
Thus,
The required clock cycle = k x tp + (n-1) x tp
= (k + (n-1)) . tp
= k + (n-1) clock cycle

The speed up of pipeline processing over an equivalent non-pipeline processing is defined by the ratio

$$S = \frac{n t_n}{(k+n-1) t_p}$$

1	1 2 3 4 . . . . K+(n-1)
2	T_1 T_2 T_3 T_4 . . . T_n
3	T_1 T_2 T_3 . . . . . T_n
4	T_1 T_2 T_3 . . . . . T_n
:	T_1 T_2 . . . . .
:	T_1 . . . . .
n	T_1 . . . T_n

fig: Space time diagram

T.U

Let consider,
we have 6 segment pipeline
no of task = 8
Here,

no of clock cycle require = $k + (n-1)$
$= 6 + (8-1)$
$= 13$ tp

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

The time require require = 13 tp

Instruction Pipeline

→ An instruction pipeline reads consecutive instruction from memory ~~which~~ while previous instruction are being executed in other segment.

→ This causes the instruction fetch and execute phase to overlap and peform simultaneous operation.

→ The instruction fetch segment can be implemented by means of FIFO buffer.

→ The instruction stream can be placed in queue, while it is waiting to be decoded and processed by execution segment.

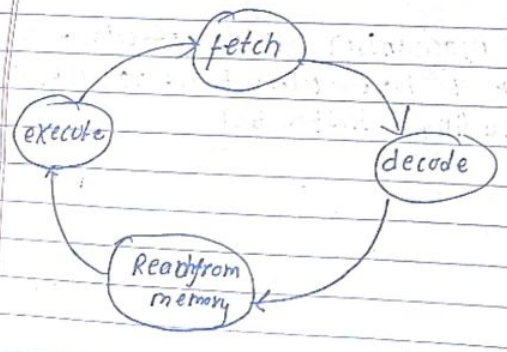
→ This queuing machanism provide effecient way for reducing average access time to memory for ~~rem~~ reading instruction.

Instruction Cycle

→ The process involved in execution of a single instruction is known as instruction cycle.

→ The computer needs to process each instruction with following sequence of step.

- fetch instruction from memory
- decode the instruction
- calculate effective address
- fetch operand from memory
- execute the instruction
- store the result.



4 segment Instruction

→ The below fig. show have instruction cycle in CPU can be processed with a four segment pipeline.

→ While instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory segment.

→ Whenever the memory is available, all subsequent instruction are fetch and placed in instruction FIFO.

→ Thus, four different instruction can be ~~progess at~~ processed at same time.

Four segment are represent as

1) FI → Fetch instruction
2) DA → Decode instruction and calculate effective address
3) FO → Fetch operand
4) EX → Execute instruction

	1	2	3	4	5	6	7	8
1	FI	DA	FO	EX	-	-	-	-
2	-	FI	DA	FO	EX	-	-	-
3	-	-	FI	DA	FO	EX	-	-
4	-	-	-	FI	DA	FO	EX	-
5	-	-	-	-	FI	DA	FO	EX

Three-Segment Instruction pipeline

- The instruction cycle can be divided into 3-suboperation and implemented in three segment

1) I: Instruction fetch
2) A: ALU operation
3) E: Execute instruction

Example

LOAD : $R_1 \leftarrow M$ [address 1]
LOAD : $R_2 \leftarrow M$ [address 2]
ADD : $R_3 \leftarrow R_1 + R_2$
STORE : M [address 3] $\leftarrow R_3$

Clock cycle	1	2	3	4	5	6
Load R1	I	A	E	-	-	-
Load R2	-	I	A	E	-	-
ADD R1+R2	-	-	I	A	E	-
STORE R3	-	-	-	I	A	E

fig: 3 segment instruction timing with data conflict.

Arithmetic Pipeline (floating point addition and subtraction)

→ high speed
→ used for implementing floating point operation
→ used in array multiplier.

say, we have input,

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

A and B are fraction
a and b are exponent

so, we can peform addition and subtraction in four segment, and they are

- compare the exponents
- align the mantissa
- add or subtract
- normalize the result.

Difficulties in Pipeline / Pipeline conflict

ipeline conflict can be defined as difficulty
instruction pipeline to deviate from
s normal operation

Pipeline Conflict / Difficulties in Pipeline

→ Pipeline conflict can be defined as difficulty in instruction pipeline to peform its normal operation.

The three major causes are

1) Resoure conflict
→ It is the condition where, two segment are trying to access the memory at same time.

2) Data dependency
→ It is the condition where, an instruction is depended on result of previous instruction, but the reasult in no available at that moment.

3) Branch difficulties
→ It is the condition that arries when there is change in value of PC (Program Counter) in Branch or other instruction.

Pipeline Hazard

→ Pipeline Hazard is the condition that occur in pipelined machine that causes delay in execution of subsequent instruction in a particular cycle.

There are 3 types of pipeline Hazard

1) Data Hazard
   → Condition where source or destination operand of instruction are not availabe at expected time
   - Read before write
   - write before Read
   - Write ~~before~~ write

2) Instruction Hazard
   → caused due to delay in availability of instruction

3) Structural hazard
   → condition where two instruction require same hardware resoure at same time.

Data Conflict and its solution

example-

I_1 : $R_1 \leftarrow R_2 + R_3$
I_2 : $R_4 \leftarrow R_1 + R_3$
I_3 : $R_5 \leftarrow R_6 + R_3$

Let consider, we have 3 segment instruction pipeline
ie. I : instruction fetch
A : ALU operation
E : execute

	1	2	3	4	5
I	I_1	I_2	I_3		
A		I_1	I_2	I_3	
E			I_1	I_2	I_3

Here, R_1 is only available after execution of I_1 i.e after 4th clock cycle.
But, In 3rd clock cycle I_2 requires $R_4 \leftarrow R_1 + R_3$ Here, but R_1 is not available, which cause s the data conflict.

Solving using No operation / Delayed load

$I_1 : R_1 \leftarrow R_2 + R_3$
No-operation
$I_2 : R_4 \leftarrow R_1 + R_3$
$I_3 : R_5 \leftarrow R_4 + R_5$

Stage \ clock	1	2	3	4	5	6
I	I_1	N	I_2	I_3		
A		I_1	N	I_2	I_3	
E			I_1	N	I_2	I_3

Here,
1 addition clock cycle is required, which we used 1 No operation.
Here I_2 can peform ALU operation using R_1, where in previous case we were unable to utilize R_1.

Solving using Rearranging

$I_1 : R_1 \leftarrow R_2 + R_3$
$I_3 : R_5 \leftarrow R_4 + R_5$
$I_2 : R_4 \leftarrow R_1 +$

Stage \ clock	1	2	3	4	5
I	I_1	I_3	I_2		
A		I_1	I_3	I_2	
E			I_1	I_3	I_3

Here,
we have simply arranged the instruction, thus we didn't required addition al clock cycle.
which is more advantage using delayed load.

Vector Operation and Processing

- Vector processing:

→ Vector processing is the central processing unit that peform the complete vector input in individual instruction.

Application

- Long range weather forecasting
- Seismic data analysis
- Medical diagonosis
- space flight simulation

Matrix Multiplication

→

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

let A and B be two matrix.
The result will be C which is also 2x2 matrix.

Flynns Classification

Unit 7: Computer Arithmetic

Addition and Subtraction using Signed-Magnitude Data

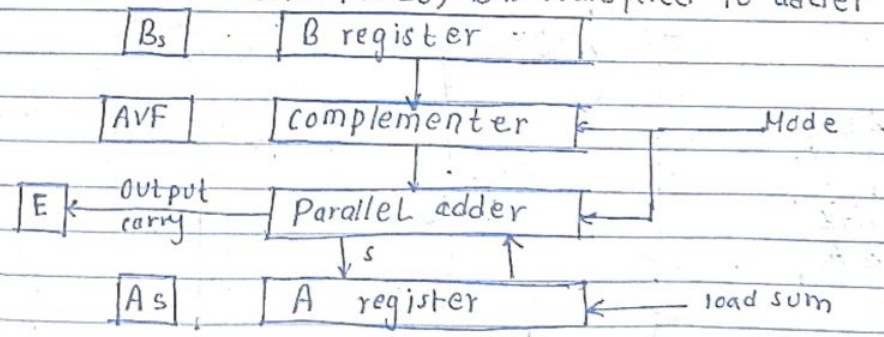
- Hardware Implementation

for addition :- A + B = addition

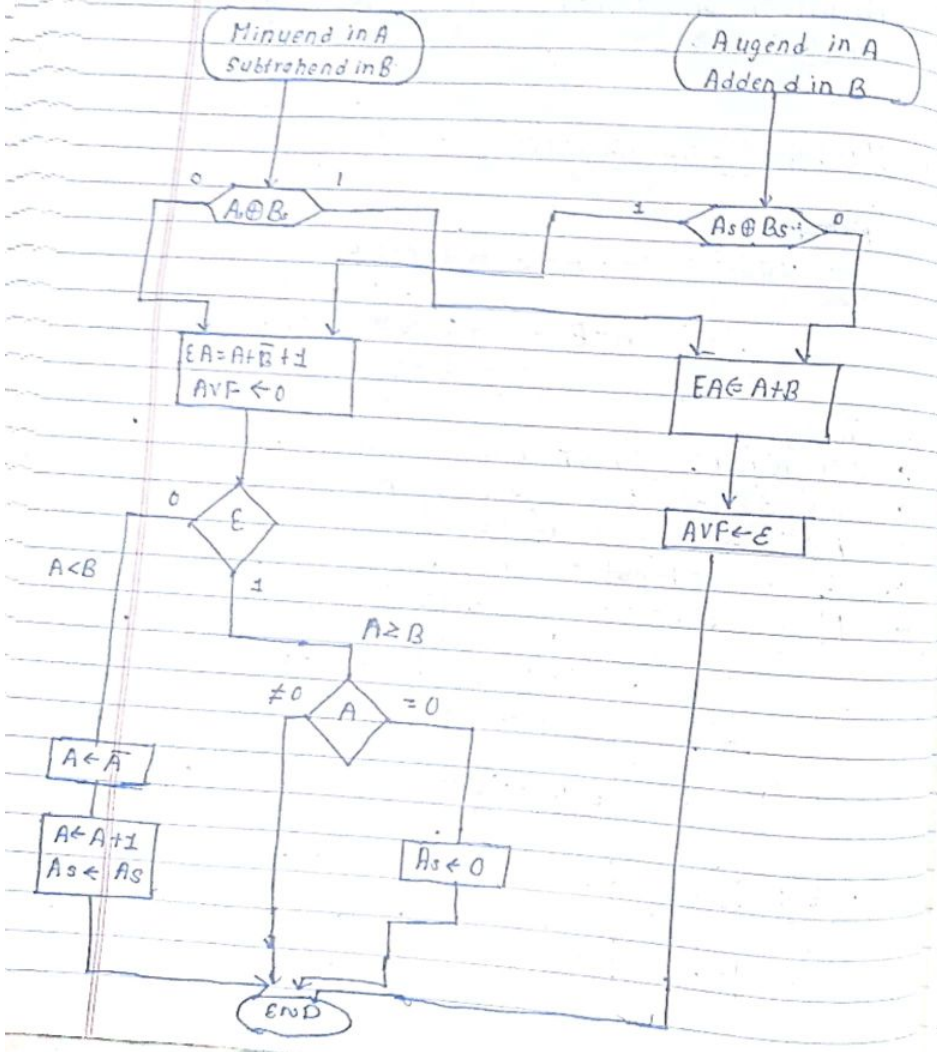
A - B = A + 2's complement of B
= A + $\bar{B}$ + 1

Thus,

→ We require adder, complementer.
→ As, Bs for the sign of A and B.
→ AVF for the overflow
→ E for output carry,
→ M for changing mode (addition or subtraction)
when M=1, complement of B is done.
when, M=0, B is transfered to adder



flowchart



Algorithm

① Input A and B in Signed Magnitude representation

② As and Bs store the sign.

③ peform As ⊕ Bs

for addition → if As⊕Bs = 1 in minuend and subtrahend
As⊕Bs = 0 in augent and addent
go to step ④

for subtraction → if As⊕Bs = 0 in minuend and subtrahend
As⊕Bs = 1 in augend and addent
go to step ⑤

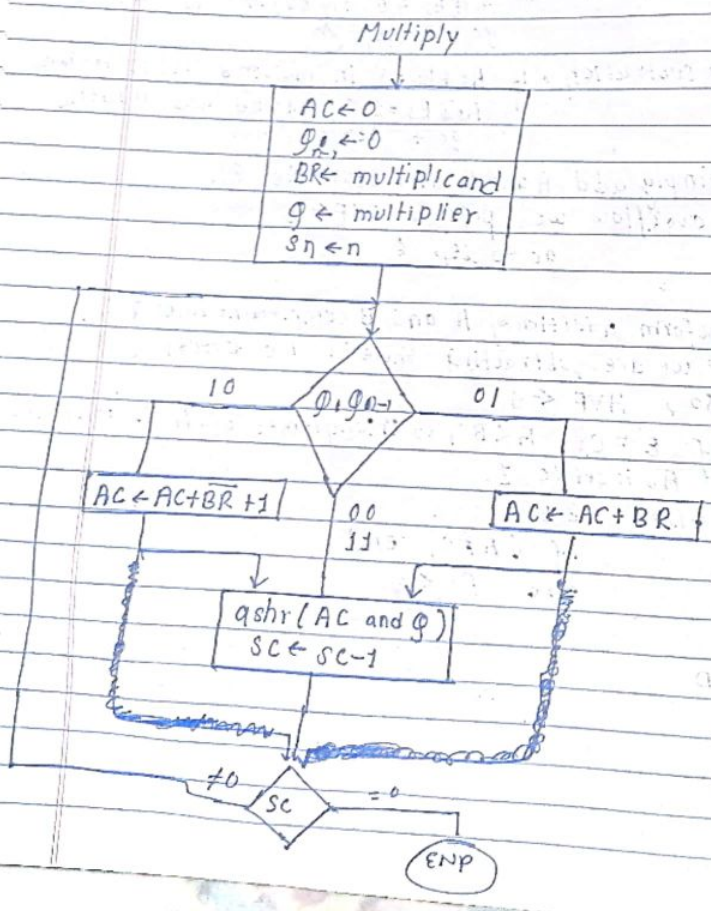
④ We simply add A and B and store to EA
If overflow we peform AVF ← E
go to step ⑥

⑤ We peform addition of A and B complement and 1
since we are subtracting there is no overflow.
so, AVF ← 0
If E = 0, A < B, so it's complemet of A is done and
As is set to 1.
else A ≥ B
if (A ≠ 0) end
else As ← 0

⑥ END

Booth Algorithm

→ Booth Algorithm gives the procedure for multiplying binary integer in signed 2's Complement representation.



Multiplication is peformed using Booth Algorithm.
Here, say B x Q
We initilize value of AC ← 0,
n represent no bits to be taken
B is multiplicant
Q is multiplier
Sn is for sequence counter
Let Q_{n+1} be 0 initially

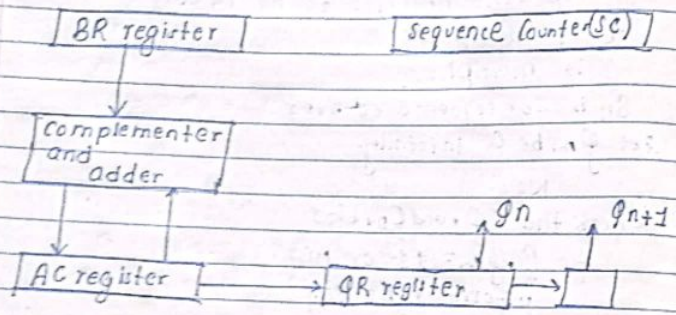
Now,
check the Q_n and Q_{n+1} bits
If $Q_n Q_{n+1}$ = 11 or 00
peform right shift
else if $Q_n Q_{n+1}$ = 10
peform 2's complement of B and add to AC
else $Q_n Q_{n+1}$ = 01
peform addition of AC and B

decrease the SC.

If (SC ≠ 0)
then go the check $Q_1 Q_0$

end

Hardware for booth algorithm



Q.N1 20 X -13

soln

Here,

Initially,

AC = 000000 $S_n = n = 5$

M = 20 = 0 10100 (no of bits in q)

q = -13 = 10011

$q_{n-1} = 0$

$\bar{M}+1$ = 101100

Now,

AC	q	q_{n-1}	S_n	Remark
000000	10011	0	5	Initial
101100	10011	0	5	AC ← AC + $\bar{M}$ + 1
110110	01001	1	4	qshr and S_{n-1}
111011	00100	1	3	ashr
001111	00100	1	3	AC ← AC + M
000111	10010	0	2	ashr
000011	11001	0	1	ashr
101111	11001	0	1	AC ← AC + $\bar{M}$ + 1
110111	11100	1	0	ashr

Here,

ACq = 110111111100

2's complement = 001000000011 + 1 = 001000000100

= -(260)

Q N2 // -4 X -3

soln

Here,

Initially,

Let,

AC = 0000

M = -4 = 0100 + 1 = 1011
+1
1100

$\bar{M}+1$ = 0100

Q = -3 = 1101 , $Q_{n-1}=0$

Sn = no of bits in Q = 1101 = 4

Now,

AC	Q	Q_{n-1}	Sn :	Remark
0000	1101	0	4	initial
0100	1101	0	4	AC ← AC+$\bar{M}$+1
0010	0110	1	3	ashr :
1110	0110	1	3	AC ← AC+M
1111	0011	0	2	ashr
0011	0011	0	2	AC ← AC+$\bar{M}$+1
0001	1001	1	1	ashr
0000	1100	1	0	ashr

Here,

LSB = (0000 1100)
↑ is 0 , so, no need to take 2's complement

so, 1 1 0 0
8 4 2 1

= 8+4 = 12 is answer

Example - 15/4 using restoring division algorithm

Soln

Here, $M = 4 = 0\ 0100$

$Q = 15 = 1111$

$\bar{M}+1 = 11100$

A	Q	Sn	Remark
00000	1111	4	initial
00001	111*	4	shL
11101	111*	4	$A \leftarrow A+\bar{M}+1$
00001	1110	3	$A \leftarrow A+M$, $Sn-1$, $Q_0=$
00011	110*	3	shL
11111	110*	3	$A \leftarrow A+\bar{M}+1$
00011	1100	2	$A \leftarrow A+M$, $Sn \leftarrow 1$, $Q_0=$
00111	100*	2	shL
00011	100*	2	$A \leftarrow A+\bar{M}+1$
00011	1001	1	$Q_0 \leftarrow 1$, $Sn-1$
00111	001*	1	shL
00001	001*	1	$A \leftarrow A+\bar{M}+1$
[redacted]	0011	0	[redacted] $Sn-1$, Q_0
00011			

Here,

Quotient = Q = 0011 = 3

Reminder = A = 0011 = 3

Soln - 11/3 using restoring division

$M = 00011$

$Q = 1011$

$1+\bar{M} = 11100+1 = 11101$

$A = 00000$

A	Q	Sn	Remark
00000	1011	4	initial
00001	011_	4	shL
11110	011_	4	$A \leftarrow A+\bar{M}+1$
00001	0110	3	$Q_0 \leftarrow 0$
00010	110_	3	shL
11111	110_	3	$A \leftarrow A+\bar{M}+1$
00010	1100	2	$Q_0 \leftarrow 0$
00101	100_	2	shL
00010	100_	2	$A \leftarrow A+\bar{M}+1$
00010	1001	1	$Q_0 \leftarrow 1$
00101	001_	1	shL
00010	001_	1	$A \leftarrow A+\bar{M}+1$
00010	0011	0	$Q_0 \leftarrow 1$
Reminder	quotient		

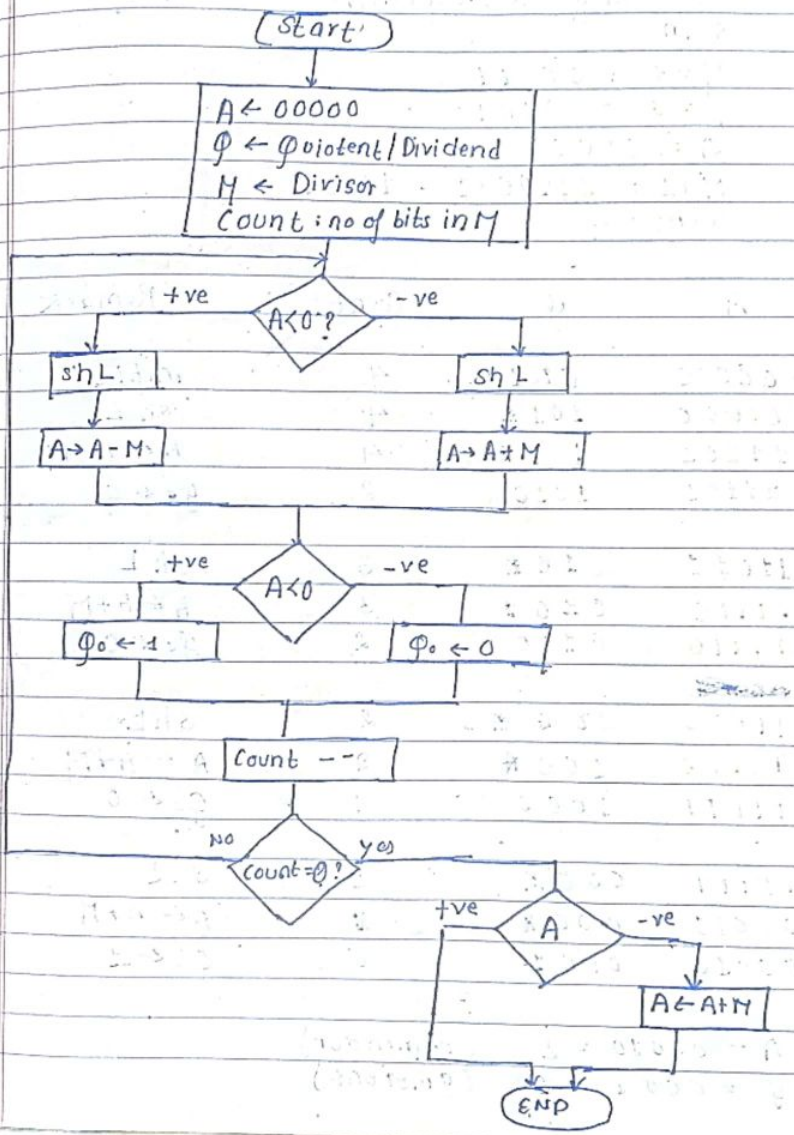
Q.N3 5/9

Here,

M = 0 0011
Q = 0101
Sη = 4
A = 00000

A	Q	Sη	Remark
00000	0101	4	

Non restoring division



qn- 5/3 using no-restoring

soln

M = 3 = 00011

Q = 5 = 0101

A = 00000

$\overline{M}+1$ = 11100+1 = 11101

Count = 4

A	Q	Count	Remark
00000	0101	4	initial
00000	101*	4	sh L.
11101	101*	4	A ← A−M
11101	1010	3	$Q_0 \leftarrow 0$
11011	010*	3	sh L
11110	010*	3	A ← A+M
11110	0100	2	$Q_0 \leftarrow 0$
11100	100*	2	sh L
11111	1000	2	A ← A+M
11111	1000	1	$Q_0 \leftarrow 0$
11111	000*	1	sh L
00010	000*	1	A ← A+M
00010	0001	0	$Q_0 \leftarrow 1$

A = 00010 = 2 (reminder)

Q = 0001 = 1 (quotient)

Unit8: Input Output Organization

Input Output Interface

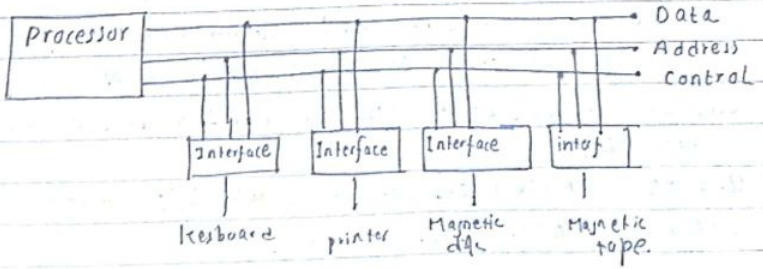
→ It is a type of interface which provides a method for transferring information between internal storage and external I/O device is called Input Output Interface.

Why we need I/O (device) Interface ?

- Helps in synchronization mechanism (
- Speed up the data transfer between components.
- Used in conversion of signal values.

I/O Bus and Interface Method

- I/O bus consist of data line, address line, control line.
- Different kinds of device can be employed in any general purpose computer.
- I/O bus can be defined as a pathway used for input & output device to communicate.



Types of I/O commands

- Status command (check various status condition)
- Data input command (tells interface to take the data for peripheral)
- Data output command (read the data from bus and save in interface buffer)

I/O vs Memory Bus

I/O bus	Memory Bus
→ used in transfering information between CPU and I/O device	→ used in tranfering information between main memory and CPU

3 ways bus can communicate between memory and I/O interface.

- Uses two seperate bus, one to communicate with memory and other to communicate with
- Uses one common bus for memory and I/O but seperate control line.
- Uses one common bus for memos and I/O with common control line for both.

Isolated I/O	Memory Mapped I/O
→ Address bus and data bus are common but seperate control line for I/O and Memory	→ Address, data and control bus are common for I/O device and memory
→ different address space are used for computer memory and I/O device.	→ Same address space are used by I/O device and computer memory
→ Special command are used in Isolated I/O such as MOV, IN, OUT instruction	→ There are no such special command in memory mapped I/O. only MOV instruction.
→ Seperate read/write control line for I/O and memory transfer	→ Single read/write control line for I/O and memory transfer.

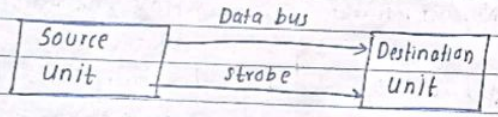
Asynchronous Data transfer

→ The transmisson of data sent in form of byte or character is called asynchronous data transfer.

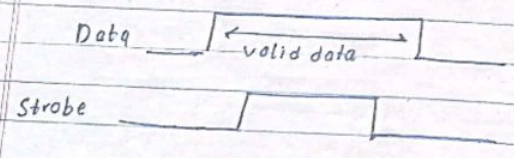
→ Discontinous and inconsistant timed transfer of data is called asynchronous data transfer. ↳ byte

Strobe

→ The method in asynchronous data transfer used for / that uses single control line for timming each transfer.



(a) block diagram



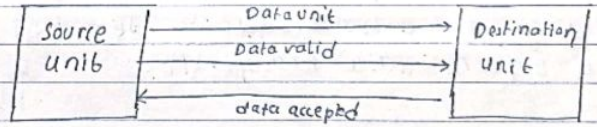
(b) Timming diagram

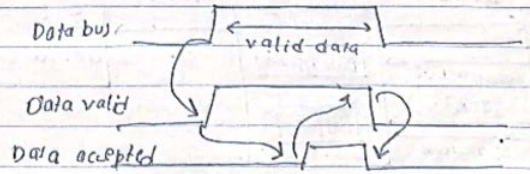
- Strobe may be activated by source unit or destination unit.
- The data bus carries binary information from source to data unit.
- The strobe is a single line that validate the transfered binary information.

Handshaking

→ In strobe, there was no any information about, the data transfer was succes or not. So, to over come this problem, we use handshake method.

→ Here, the problem is solved by introducing 2nd control signal which provides / relay information on transfer of data.





Mode of Transfer

1. Programmed I/O
2. Interrupt-Initiated I/O → Priority Interrupt
   - Daisy Chain Priority
   - Parallel Interrupt Priority
3. DMA

1. Programmed I/O

→ Programmed I/O operation are the result of I/O instruction written in computer program.

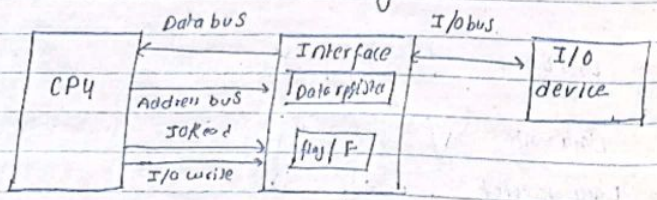
→ Each data transfer is initiated by I/O instruction.

→ CPU constantly monitor flags to let Interface inform the CPU when it is ready to transfer data.

→ CPU runs program peridically, check the status of I/O device one by one.

→ If any device has its state set, the cpu peform data transfer it.

→ Works on principle of polling

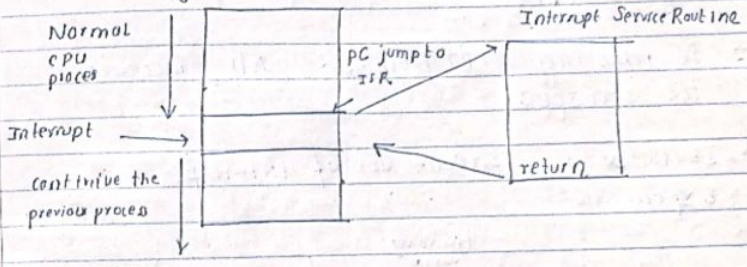


If flag is set, then data transfer takes place through C.P.U.

2. Interrupt-Initiated I/O

→ While CPU is running a programe, it doesn't check the flag constantly.

→ When the flag is set, the CPU deviates from current process and takes care of input or output transfer immediately.



→ CPU responds the interrupt signal by storing return address into memory stack and then control branch service routine that process I/O transfer.

→ Then, it resume the previous process with the help of information stored in stack.

Direct Memory Access (DMA)

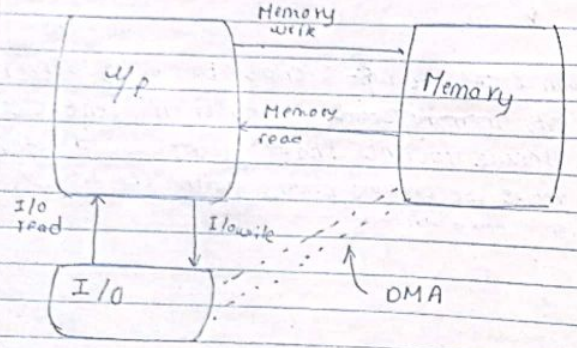
→ The process of transfer of data and instruction directly between the I/O device and memory is called direct memory access.

→ The transfer between of data, between storage speed divide and main memory are often limit by the CPU.

→ So removing CPU from the path, increases the overspeed

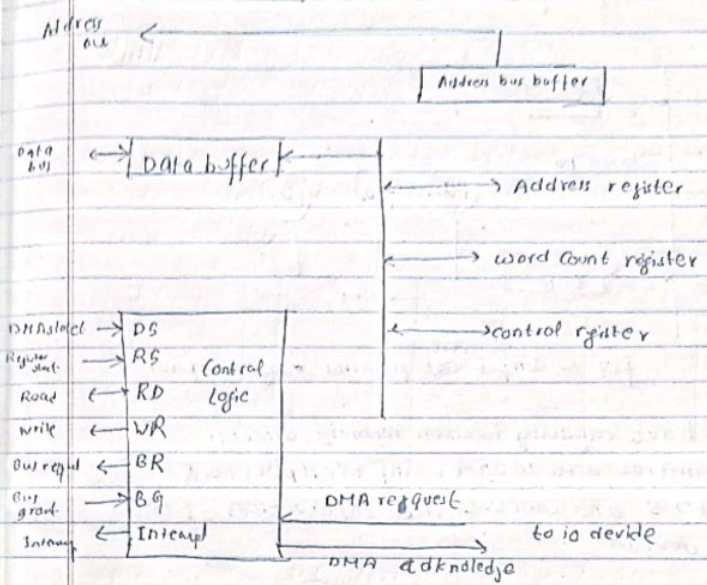
→ It helps in efficient use of interrupt

→ Expensive



→ In DMA transfer, CPU is ideal and has no control of memory bus.

DMA controller



Input-Output processor (IOP)

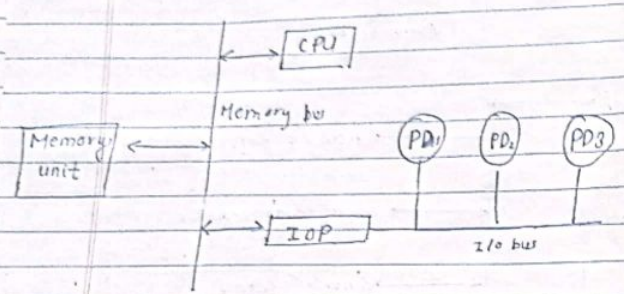


fig :- Input-Out processor of Computer

→ I/OP have capacity to access memory directly.
→ It control and manges input/output task
→ It fetchs and execute the instruction from I/O device.
→ I/O processor provides a path for transfer of data between pheripheral device and memory.

IOP vs DMA

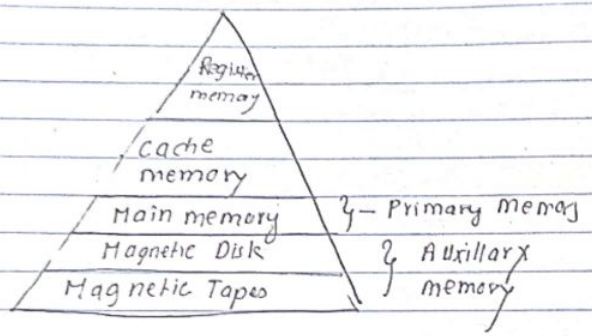
	I/O Processor	Direct Memory Access
→	Data transfer speed is slower	→ Data speed rate is higher
→	Less expensive	→ more expensive
→	Extra hardware are not required	→ extra hardware (DMA controller) are required
→	used in smaller data transfer	→ used in large data transfer

Unit : 9 Memory Organization

- Memory Hierarchy
- Main Memory
- RAM
- ROM
- Memory address Map
- Memory Connection to CPU
- Auxiliary memory
- Hardware Organization
- Match logic
- Read operation
- Write operation
- Locality of Reference
- Hit & Miss ratio
- Mapping
- Write Policies

Memory Hierarcy

→ It can be defined as the method of arranging different kind of storage present on the computing device based on speed of access.



Main memory → RAM, ROM

Key characteristics of memory system

- location
- capacity
- Unit of transfer
- Access method
- Performance
- organization
- physical characteristics.

RAM and ROM chip

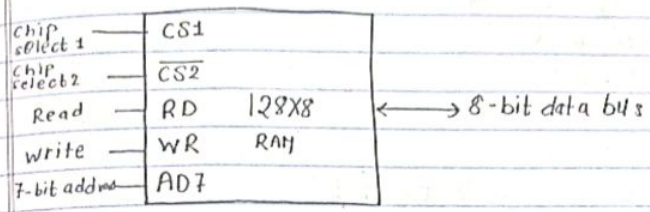


fig :- RAM Chip

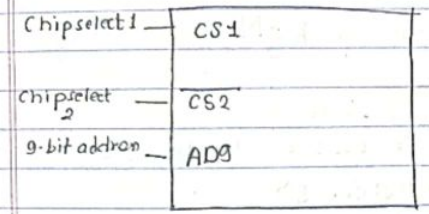


fig : Typical ROM chip

Auxiliary Memory

→ Secondary memory
→ Magnetic Disk
→ Magnetic tape

Memory Connection to CPU

→ RAM and ROM are connected to CPU through data and address buss.

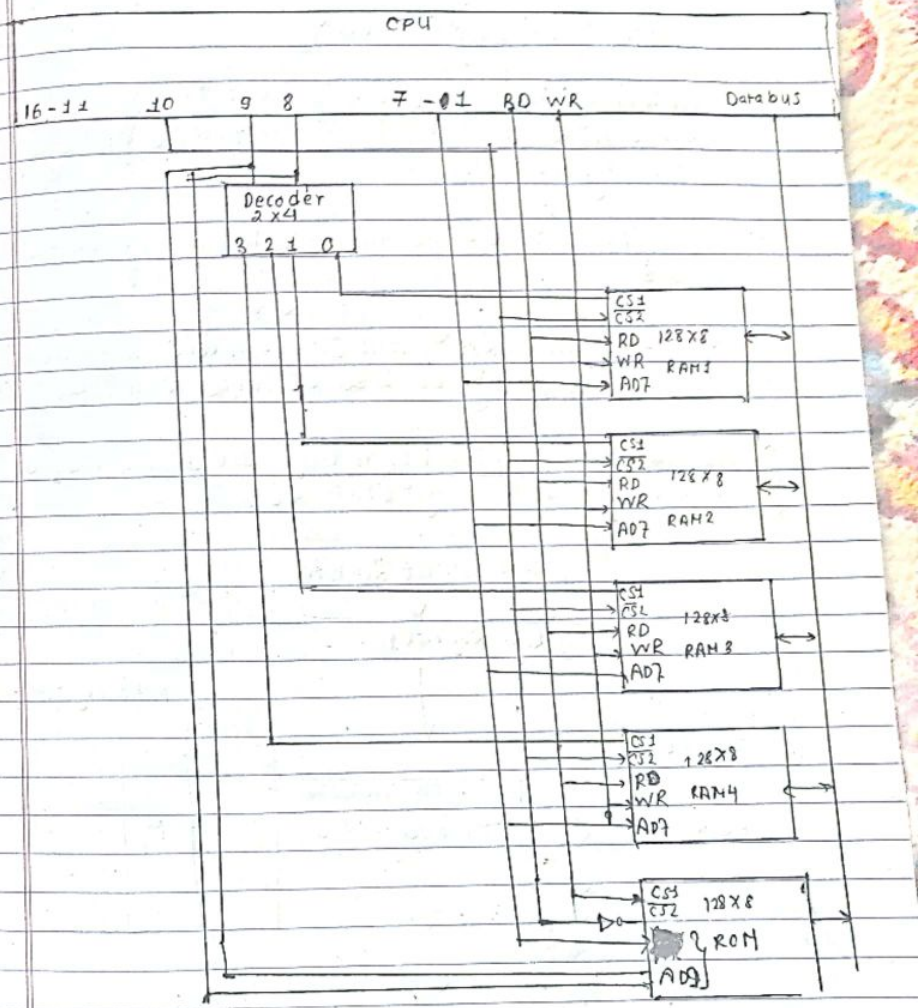
→ We have used 512 bytes of RAM and 512 bytes of ROM.

→ RAM recieves low-order bits of address bus (AD0-AD7)

→ ROM recieves ~~high-order of bits of address bus (AD8 - AD9)~~ (AD0 - AD9) order bits.

→ RD and WR output of CPU is taken as input for RAM and ROM

- AD 10 is used for selecting between RAM and ROM.



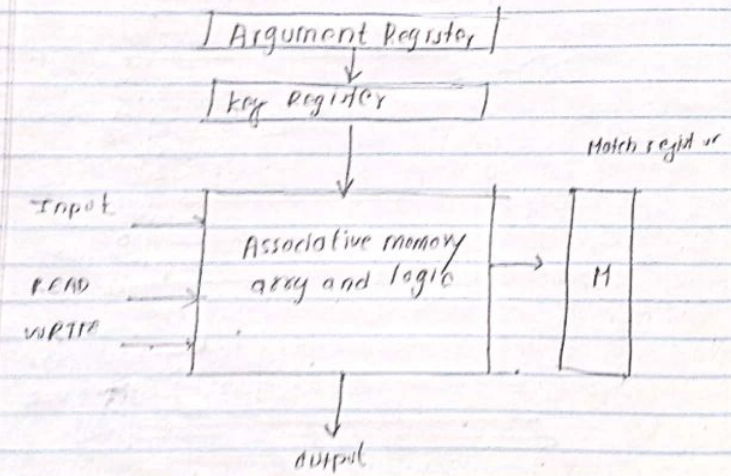
Associative memory / CAM

→ A memory unit assessed by content is called associative memory or content addressable memory (CAM)

→ Used / help to do parallel searches by data allocation.

→ Expensive than RAM, since each cell must have storage capacity as well as logic circuit.

→ So, Associative memory is only used, where search time is very critical.



Cache Memory

→ It can be defined as the chip that makes retrieving data from computer's memory more efficient.

→ It acts as temporary storage area.
→ It is integrated directly into CPU chip
→ It increase efficiency / speed, since it is physically close to processor.
→ Less storage
→ smaller size

• Hit and miss ration

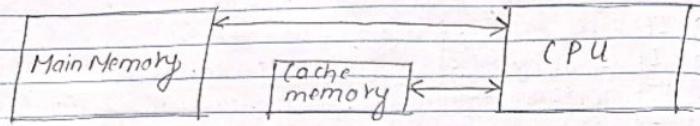
Cache hit refer to the situation where the cache is able to successfully retrieve data and content that was saved to it.

Cache miss refer to the situation where cache is unsuccessful to retrive data.

Hit ratio → The ratio of the number of hits divided by total CPU references to memory is hit ratio.

Mapping

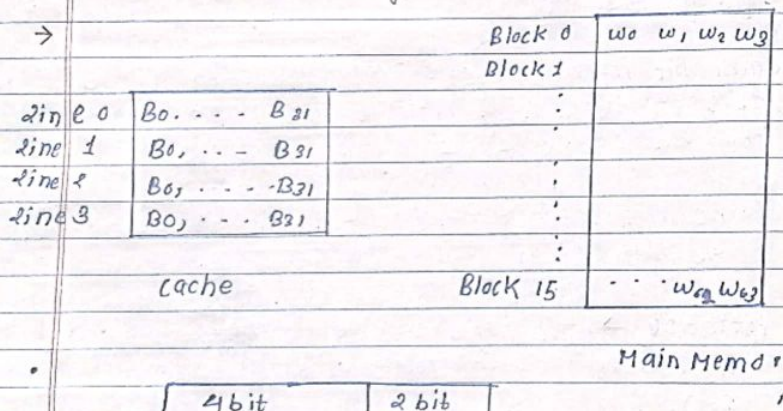
→ The transformation of data from main memory to cache memory is knowns as mapping.

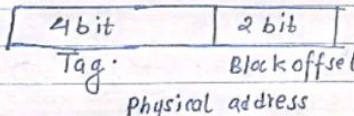


Types of mapping

Associative Mapping

→





→ Associative mapping is a technique or procedure used to assign each memory block in main memory to a freely available cache line.

- Main memory have 64 words (consider)
- Let assume Block size is 4 word

Thus,

no of blocks in main memory = $64/4 = 16$

say, we have cache of 16 word

- line size = block size = 4 word

Thus,

no of lines in cache = $16/4 = 4$

The physical address required to address the 64 word is given by

$= \log_2 64$

$= \log_2 2^6$

$= 6$ bit

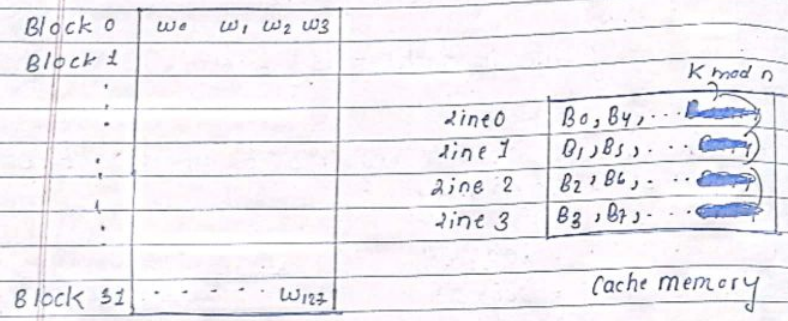
→ Here, 2 least significant bit is used for addressing block offset of main memory and,

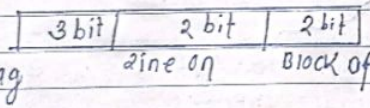
→ 4 most significant bit is used for addressing block of main memory.

→ Here, any block can reside on any cache line without any restriction.

Direct Mapping.

→ Direct Mapping is a technique or procedure used to assaign each memory block in main memory to particular cache line.





physical address

→ Consider, Main memory has 128 word. bay, block size is 4 word.
Then,
No of block in Main memory = $128/4$ = 32 (0, 1, 2 ... 31)

→ Consider, Cache memory has 16 word
Since, block size = line size = 4 word
Then,
No of line in cache = $16/4$ = 4 lines

→ Here, to address 128 word, we need physical address of
$= \log_2 128$
$= \log_2 2^7$
$= 7$ bit

→ Here, the block of main memory are assigned to particular lines in cache.
for eg:-
$B_0, B_4, B_8, B_{12}, \ldots B_{28}$ are assigned to line 0 in cache., this is done using K mod n.
K = block no, n = no of lines i.e 4.

→ Due to which there are more change / possibility of miss.

Set-Associative Mapping

→ Set-Associative mapping is the combination of direct and associative mapping.

Let say, we have k-ways set. Then

total no of set is given by = $\frac{\text{no of line}}{k}$.

for 2 way set associative,

no of set = $\frac{4}{2}$ = 2 say (S_0, S_1)

Block 1	$w_0\ w_1\ w_2\ w_3$
Block 2	
⋮	
Block 15	$\cdots\ w_{63}$

Line 0	$B_0, B_2, B_4 \cdots B_{30}$	S_0
Line 1	$B_0, B_2, B_4 \cdots B_{30}$	S_0
Line 2	$B_1, B_3, B_5 \cdots B_{31}$	S_1
Line 3	$B_1, B_3, B_5 \cdots B_{31}$	S_1

say, Memory word = 64

Block size = 4

No of block = 64/4 = 16 (0, 1, 2 ⋯ 15)

Cache word = 16

Line size = 4

No of line = 16/4 = 4

→ Here, we have mapped the cache using P.A

$= \log_2 64$

$= \log_2 16$

$= 6$ bit

3 bit	1 bit	2 bit
Tag	Set no	Block offset

Here, mapping is done using formula k mod set no.

In this case,

k = block no

s = set no

for example

for block no 4,

= 4 mod 2

= 0 (So, it lies in set S_0)

for block no 5

= 5 mod 2

= 1 (So, it lies in Set S_1)

Address space and Memory space

- The set of virtual address is called address space.
- The set of location, where address are store in main memory is called memory space.
- Address space is larger than memory space.
- Address space is generally generated by programs
- Memory space is the actual main memory location.

Example :-

Main-memory size = 32 k word (k = 1024)
= (32 x 1024) bits
= 32768 bits

Physical address to address 32768 bit = $\log_2 32768$
$= \log_2 2^{15}$
= 15 bits

Secondary memory space = 1024 k words
= 1024 x 1024 bits

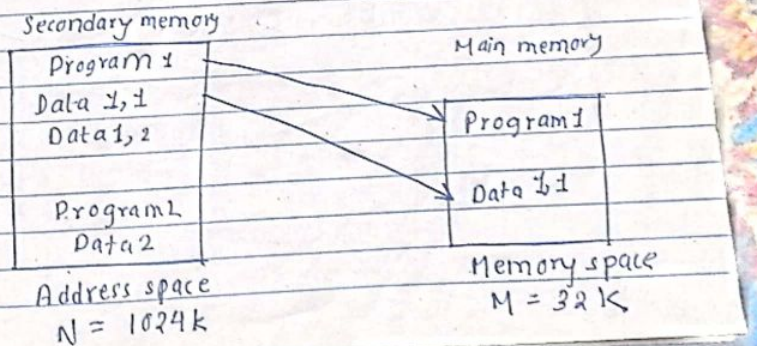
P.A = $\log_2 1024 \times 1024$
= $\log_2 2^{20}$
= 20 bits (Virtual address)

Here,
Secondary memory have capacity of storing 32 main memory.

ie :- $= \frac{1024}{32}$ (size of secondary memory / size of main memory)
= 32

Here,
N = 1024 k denoting address space

M = 32 k denoting memory space



Address Mapping using Pages

- Address mapping refer to process of determing a logical address using physical address and vice versa.

Page refer to the group of address space of same size.

Unit 5 : Central Progessing Unit

- Major Component of CPU
- CPU organization
- CPU Instruction
  - Instruction format
    - Instruction format
    - Addressing mode
    - Type of instruction
    - Program control
    - subroutine call
  - RISC and CISC
  - overlapping window

- Major Component of CPU

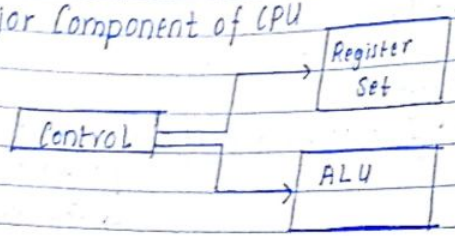
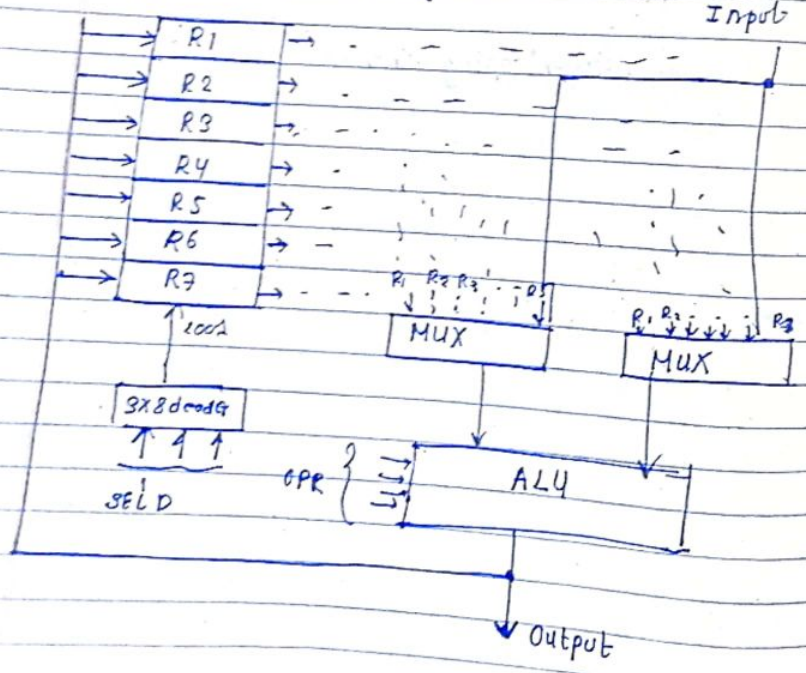


fig :- Major Component of CPU

- Register Organization



Stack Organization

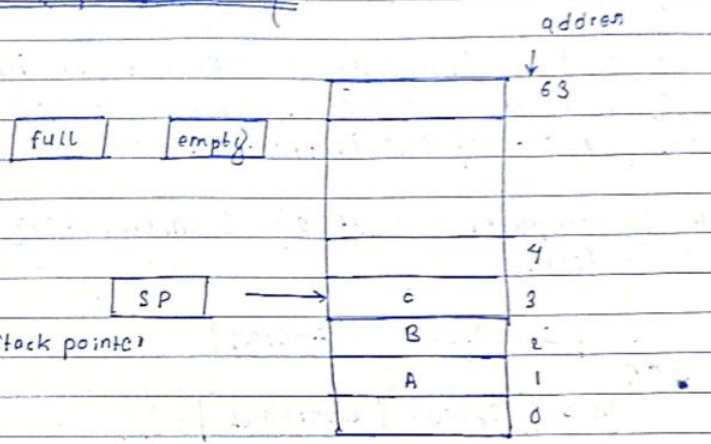


fig: Register stack

→ LIFO principle.
→ Stack is storage device.
- Two operation → insert (push)
  → delete (pop)

IF (FULL=0) (ie stack is not full)
new item can be pushed
If (empty = 1) (ie stack is empty)
no item can be poped

SP ← SP + 1 (increasing pack point)
M[SP] ← DR (write item on top of stack)
If (SP = 0) then (full ← 1)
Empty ← 0

Instruction format

→ It can be defined as a sequence of bits in machine instruction that defines the layout of the instruction.

→ Basic computer consist of 3-instruction code format

I	Opcode	Address

or,

Mode	opcode	Operand

→ It helps to represent instruction.

→ It is classified into 3 fields

- Address field → It specific the address of memory or register.
- opcode → It specifices the operation, which we want to peform.

Mode → It specifies how the operand is determined

Address field in Computer Organization

1) Single Accumulator organization

→ No need to specify the accumulator.

→ Operation automatically be peformed on it without defining accumulator.

for example:- CMA;

It complement content of accumulator.

ADD A ;

It peform AC → AC + A

2) General puropose Register Organization

→ It is the organization where crossponding instruction contains multiple address (register or operand)

for example

ADD R1, R2, R3

Here,

$R_1 \leftarrow R_1 + R_2 + R_3$ (R_1 is source)

MOV R_1, R_2 ($R_1 \leftarrow R_2$)

3. Stack Organization

→ According to address field, the stack organization have 5 address instruction

- 3 address instruction
- 2 address instruction
- 1 address instruction
- 0 address instruction
- RISC instruction

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

1. 3 address instruction

→ The crosspond instruction consist of three address that may be register or some memory address.

```
ADD  R1, A, B      R1 ← M[A] + M[B]
ADD  R2, C, D      R2 ← M[C] + M[D]
MUL  X, R1, R2     M[X] ← R1 * R2
```

2. 2 address instruction (Mov)

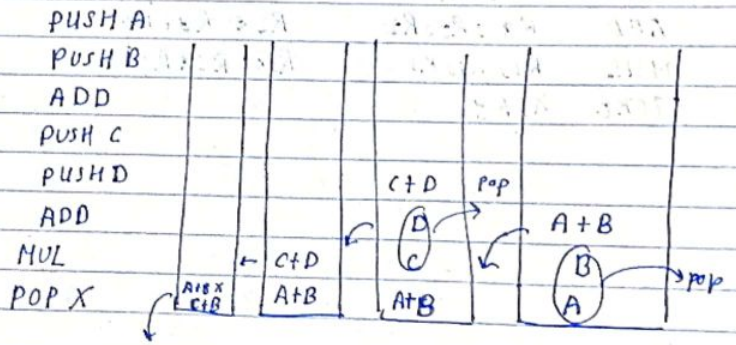
```
MOV  R1, A         R1 ← M[A]
ADD  R1, B         R1 ← R1 + M[B]
MOV  R2, C         R2 ← M[C]
ADD  R2, D         R2 ← R1 + M[D]
ADD  R1, R2        R1 ← R1 * R2
MOV  X, R1         M[X] ← R1
```

2. One address Instruction (load, store)

```
LOAD  A      AC ← M[A]
ADD   B      AC ← AC + B
STORE T      M[Temp] ← AC
LOAD  C      AC ← M[C]
ADD   D      AC ← AC + M[D]
MUL   T      AC ← AC x M[T]
STORE X      M[X] ← AC
```

4. Zero Address Instruction

→ We use stack operation such as push, pop



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

5. RISC Instructure

→ Reduce Instruction Set Computer instruction where, it is restricted or denied to use load and store when communicating between memory and CPU.

→ All the instruction should be executed with in register without referring to memory

```
LOAD   R1, A      R1 ← M[A]
LOAD   R2, B      R2 ← M[B]
LOAD   R3, C      R3 ← M[C]
LOAD   R4, B      R4 ← M[B]
ADD    R1, R1, R2 R1 ← R1 + R2
ADD    R3, R3, R4 R3 ← R3 + R4
MUL    R1, R1, R3 R1 ← R1 * R3
STORE  X, R1
```

Addressing Mode

→ Addressing mode can be defined as the method of specifying the location of operand.

→ It is used by microprocesor to deliver the instruction to mechine.

→ It specifies how to calculate effective memory address of operand.

Addressing

Types of ~~Implied~~ Mode

1. Implied Mode :- Operand in implicidely available. (CMA) (CLA)
   eg :- CMA, CLA
2. Immediade Mode :- operand is specified in instruction itself.
   eg :- JMP 3000, MVI B 05
3. Register Mode :- Operand are register that is present in CPU.
   eg :- ADD B, INRD
4. Register Indirect Addressing Mode :- selected register contains the address of operand
   eg :- MOV A, M
5. Direct Addressing Mode :-
   → Effective address = Address part of instruction.
6. Indirect Addressing Mode :-
   → Effective address = address part of instruction + content of CPU register
   of address contain effetade.

7. Relative Address Mode:-

* Effective address = PC content + Address part of Inst

8. Index Addressing Mode :-

* Effective address = Index register Content + Address part of Instruction

9. Base Register Addressing Mode :-

* Effective address = base register content + Address part of instruction

PC = 200

R1 = 400

XR = 100 ← Base Index

AC

200	Load to AC | Mode
201	Address = 500
202	Next instruction
399	450
400	700
500	800
600	900
702	325
800	300

	Addressing Mode	Effective Address	AC Content
*	Direct	500 (500→800)	800
*	Indirect	800 (500→100→200)	300
*	Immidate	201	500
	RegisterAddress	—	400
	Register Indirect	400	700
	Auto Increment	400	700
	auto decrement	399	450
*	Relative	702	325
*	Base	600	900
*	Index	600	900

Data Transfer and Manipulation

1. Data transfer Instruction

- Load
- store
- MOV
- IN
- OUT
- PUSH
- POP

2. Data Manipulation Instruction

- Arithmetic Instruction
  - Incl
  - Dec
  - ADD
  - MUX
  - SUB
- Logical
  - Clear
  - AND
  - OR
  - EX-OR
- Shift Instruction
  - Shr
  - shL
  - ROR
  - RoL

3. Program Control

- JMP
- BR
- CALL
- RETURN

Subroutine Call and Return

→ A subroutine can be defined as the self contained sequence of instruction that peform a given compytational task

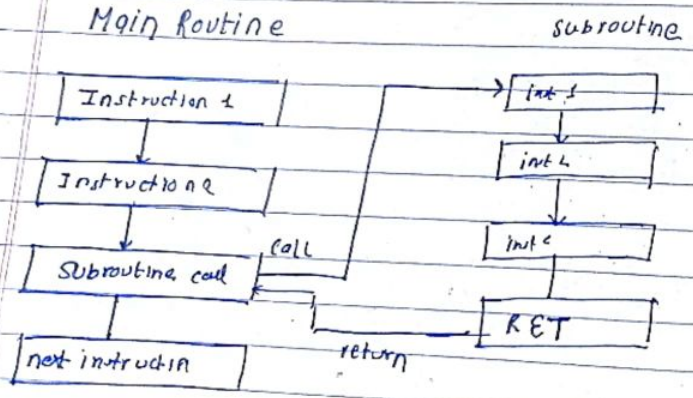
→ We use different instruction to transfer program control to subroutine.

like:-

CALL SUBROUTINE

JMP SUBROUTINE

BR(Branch) SUBROUTINE



Program Interrupt

→ Program Interrupt can be defined as the tranfer of program control from current programe to another service program as result of internal or external request.

→ The main difference between subroutine and program interrupt is that :-

- program interrupt is initited by extral or internal signal
- subroutine call is initited by execution of an instruction.

Type of Interrupt

1) External Interrupt → Due to I/O device
   eg: I/O device requesting tranfer of data.
2) Internal Interrupt → cause due to illegal or erroneous use of instruction.
   eg:- overflow, attempt to divided by zero
   Stackoverflow, invalid opcode
3) Software Interrupt → initiated by executing an instruction
   eg: Subroutine call

Complex Instruction Set Computer (CISC)

- Computer consisting of large number of instruction and addressing mode is known as CISC.
- Complex architecture
- High cost
- More powerful
- There are variation in length instruction format.
- Consist of several Instruction Cycle.
- Manipulation of data takes place directly in memory.
- It is Microprogrammed Control unit based
- Intel 8080, Motorola 6800 etc

example

MULT 2:2 3:3

or

ADD 2:2, 3:3

i.e. $x = x + y$

Reduce Instruction Set Computer (RISC)

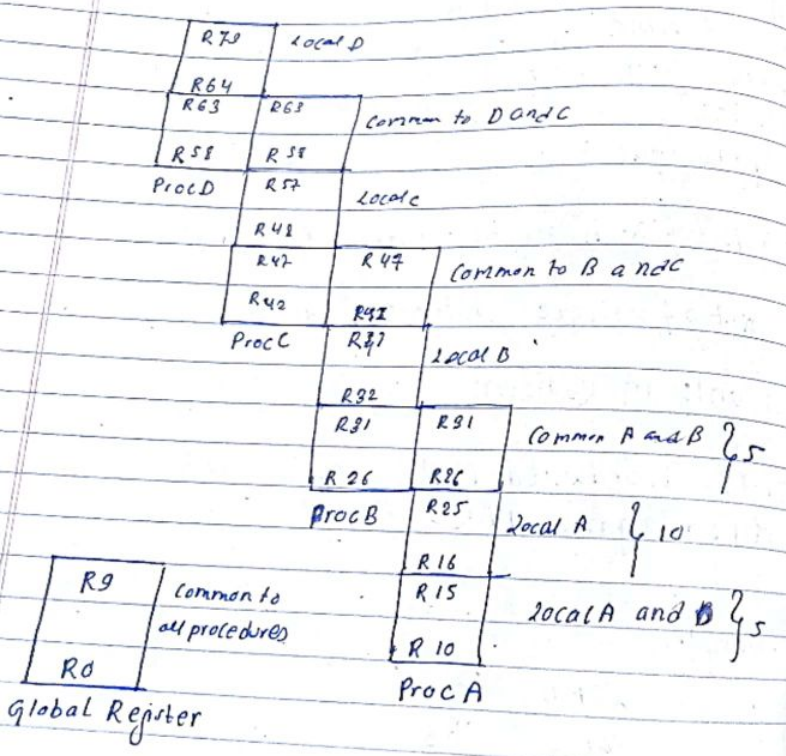
- Computer consisting of reduced/few size of instruction and few addressing mode is called RISC
- simple architecture
- Lower cost
- Best powerful
- The is fixed length Instruction format.
- Consist of single Instruction cycle.
- uses only in Register.
- It is hardwired Control unit based.
- MIPS, Fugaru, SPARC etc

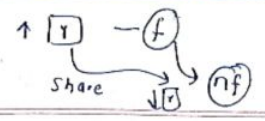
example

```
LOAD    R1, A
LOAD    R2, B
ADD     R1, R2
STORE   R1
```

$R_1 \leftarrow R_1 + R_2$

Register Overlapped Window





Here,
We use register Overlapped window in order to increase the peformance of the computer.

There are 3 classes of register.

- Global register (G)
  - ↳ available to all function
- Window local register (L)
  - ↳ Variable local to function
- Window shared register (C)
  - ↳ permits data to be share

Here,
Window size $W = L + 2C + G$

Total no of register $= (L+C).W + G$

new
When function is called by a function, the higher numbered register of calling function window are shared with called function which consist of lower numbered register. This way the caller's high and called function's low register overlape.

Unit 4: Microprogrammed Control

* Hardwire Control Unit
* µp Control Unit

- Control word
- Micro program
- Control memory
- Address Register
- Sequencer

} Concept

* Addressing Sequencing

- Conditional Branching
- Mapping of instruction
- subroutine
- Microinstruction formal
- Symbolic Microinstruction.

* Design of Control Unit

- Decoding
- Microprogram Squence

Hardwired Control unit	Microprogram Control Unit
→ The control unit where the control signal are generated by hardware using conventional logic is called Hardwired Control unit	→ The control unit which stores or save the binary control values is called microprogram control unit.
→ Fast	→ slow
→ Cost more	→ cost less
→ It uses sequential circuit composed of logic gate	→ It uses microprogramming having micro instruction
→ Difficult to alter hardware for new instruction	→ Easy to alter instruction for new operation
→ instruction, which are less complex, they are implemented using Hardware ckt.	→ It can process complex instruction
→ Design are complicate → Decoding is complex	→ Design are systamatic → Decoding is easy.

Terms

1) Control Word
→ The word whose individual bits represent the various control signal is called control word.
→ Control variable is represented by control word string of 1's and 0's.

2) Microprogram
→ A microprogram can be defined as a sequence of microinstruction.
→ easy to alter
→ flexible
→ can solve complex instruction

3) Control Address Register
→ Control address register are those register which contains the address of next microinstruction.

4) Sequencer
→ It determine the next address from within control memory (where the micro instruction are stored.

Address Sequencing / Microprogram Sequencer

used in Microprogrammed Control unit

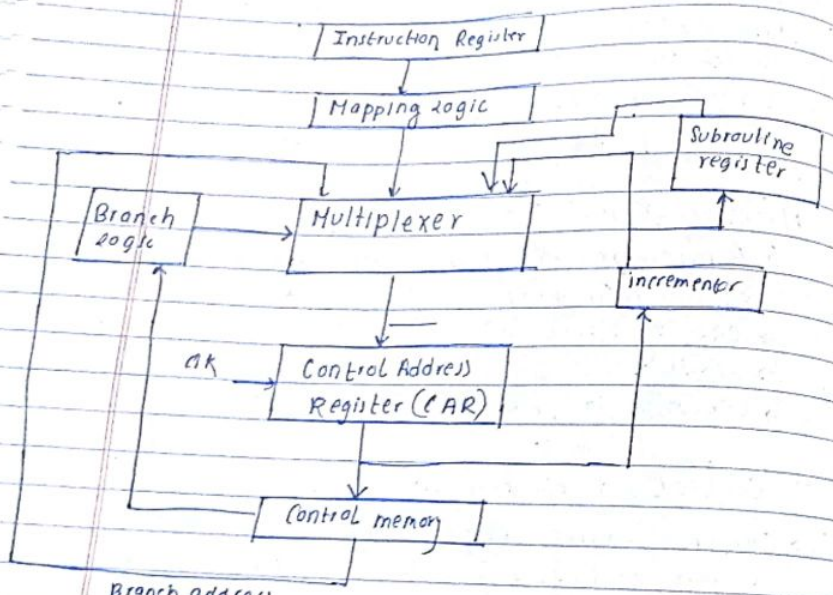


fig: Address Sequencing Selection

→ The Addressing sequencing provides a mapping from the instruction bit to control memory address.

→ The process of finding address of next micro-instruction to be executed is called address sequencing.

→ An Address Sequencer must have capability of finding address next micro instruction under following situation ~~such~~ such as

- Loop
- unconditional branch
- subroutine call
- in-line sequencing

* Incrementer → increment the content of Control Address Register by one to select next μ instruction in sequence.

* condition Branching
  → check zero, Overflow, Negative, Carry
  If condition is true, we peform branching
  else
  we increment Control address Memory

Mapping :- The process of generating the address from opcode is called mapping.

Mapping of Micro Instruction

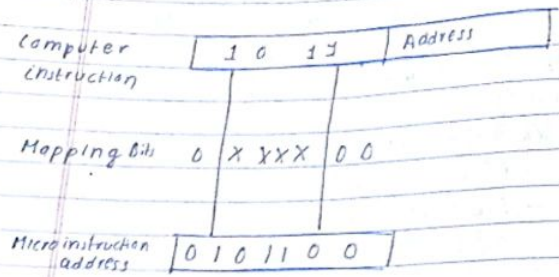


fig: Mapping from Instruction Code to micro instruction Address

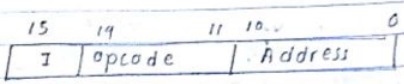
→ It has an operation code of 4 bit which can specify up to 16 distinct instruction.

→ The mapping procedure converts the 4 bit operation code to 7 bit address for control memory.

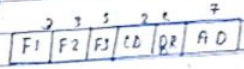
→ Mapping consist of 0 at MSB of all the address.

→ The mapping function is sometimes implemented through integrated circuit called PLD.

Microinstruction format



ADD	0000
BRANCH	0001
STORE	0010
EXCHANGE	0011



F1, F2, F3 : Micro operation field
CD : Condition for branching
BR : Branch field
AD : Address field

Symbolic Microinstruction

→ used to represent each field in micro instruction.

It is divided into five fields

→ Label → may be empty or may specify specific symbol

→ Micro operation : consist of 1, 2, 3 symbol seperated by comma

→ CD : {U, I, S, Z} any 1

→ BR {JMP, CALL, RET} any 1

→ AD {NEXT, EMPTY} any 1

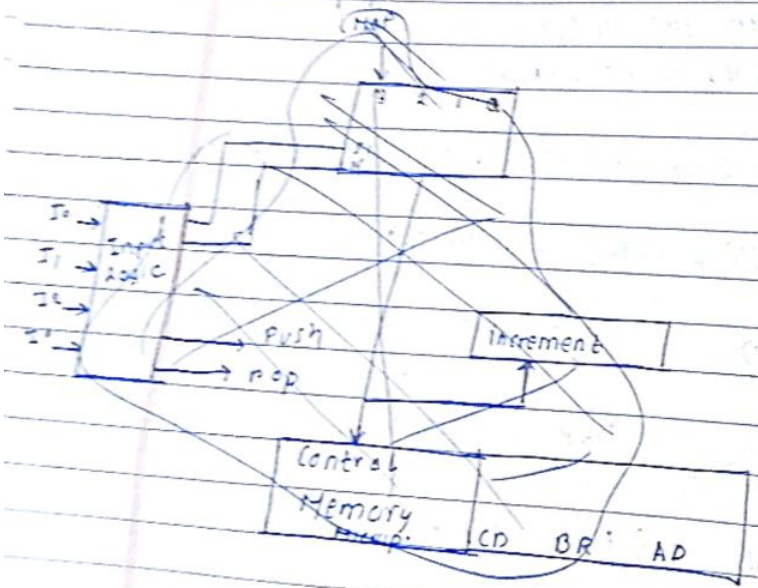
→ Symbols are used in symbolic instruction as in assembly language

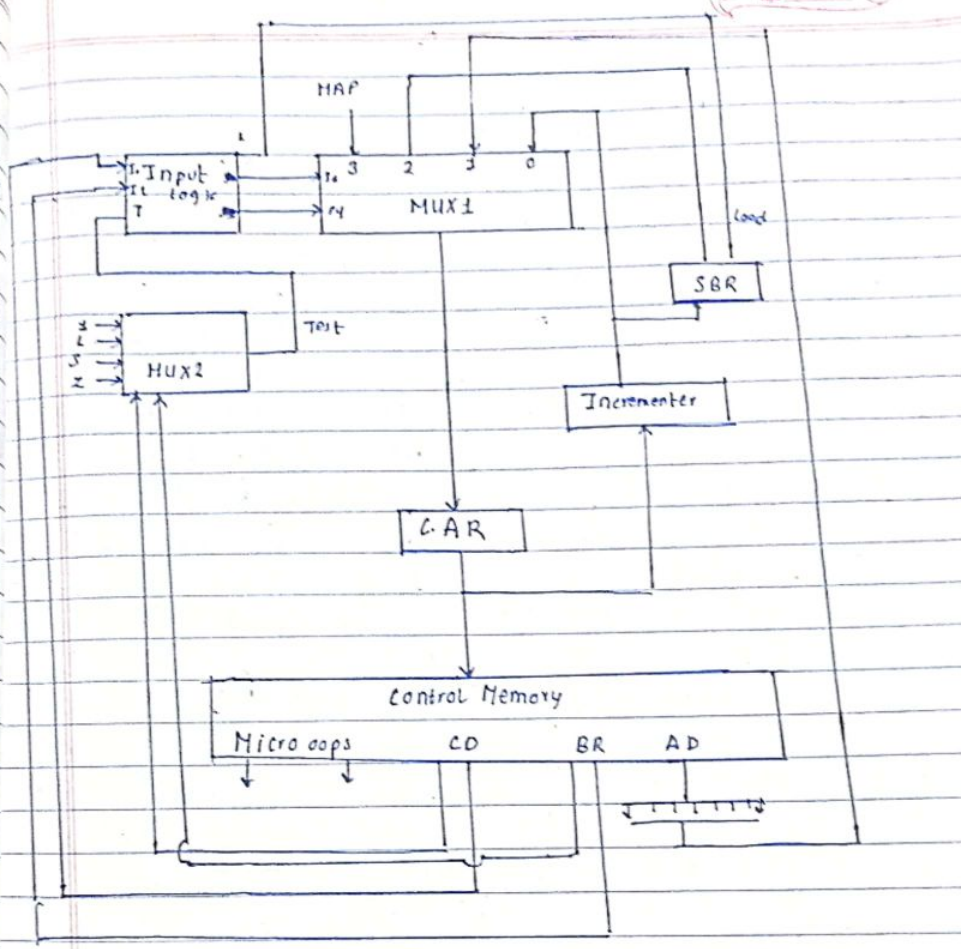
Micro program Sequencer

→ Used in Microprogram Control Unit.

→ It generate the next micro instruction address and provide to Control Address Register.

→ Microprogram Sequencer can be defined as the subunit of Microprogram Control unit with generate the next micro instruction address and provide to Control Memory so that execution to that instruction can be performed.





Unit. 3 (Basic Computer Organization)

* Concepts
  - Instruction code
  - operation code
  - Concept of instruction format ✓
  - store program ✓

* Computer Register and Memory
  - List of register
  - Memory of Basic Computer
  - Common bus for basic computer

* Computer Instruction
  - Fetch, Decode
  - Type of Instruction
  - Memory Reference Instruction
  - I/O Instruction
  - I/O interrupt
  - Program Interrupt
  - Interrupt Cycle

* Flow chart of basic Computer

* Instruction code

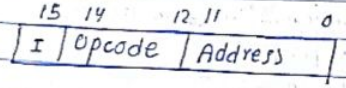
→ It is a group of bits that instruct computer to perform specific operation.

* Operation code

→ Operation code can be defined as there code ure to define operation such as add, subtract, multiply, shift etc.

* Instruction format

→



- Defines layout of the instruction.
- Consist of 3 format code
- represent instruction

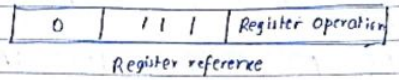
Classified in 3 fileds

Address field :- It specific the address of memory or register

Opcode :- Specifies the operation

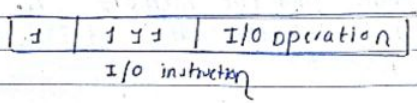
I :- Mode, specifies how operand is determined.

If opcode = 111 and I = 0



Register reference

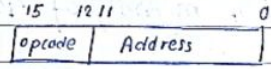
If opcode = 111 and I = 1



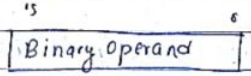
I/O instruction

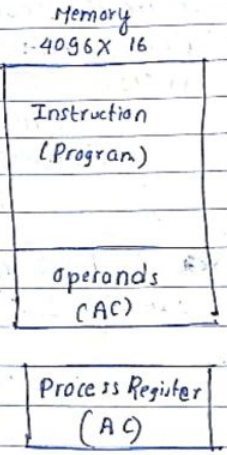
* Stored Program Organization

→



Instruction format





→ One processor register and instruction format of 2 part are required for organize a computer in simplest way.

→ Here, one part of Instruction format is opcode, which specifies the operation to be peformed with operands.

→ Next part is address part which specifies the address of address of operand.

→ This operand is read from the memory and operated together with the data stored in processor register.

→ Instruction are stored in one section of memory and data in another.

→ Instruction code is stored in 16-bit memory word.

→ Here, we have taken memory of 4096 words.

We need 12 bit to specify the address in memory.

$2^{12} = 4096$

We have now 4 bit left, which is used to specify one of 16 possible operation.

$2^4 = 16$

* Instruction Set Completeness

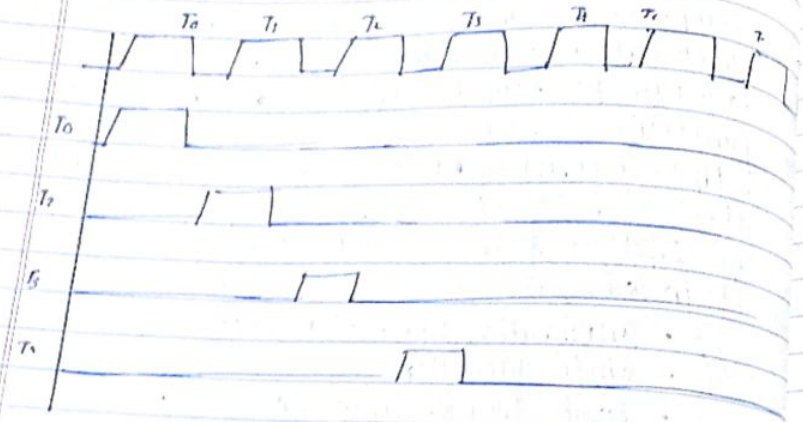
→ A computer should have a certain set of instruction which allows the user to construct machine language program to implement or evaluate function.

→ so, the user should have access to the complete instruction set.

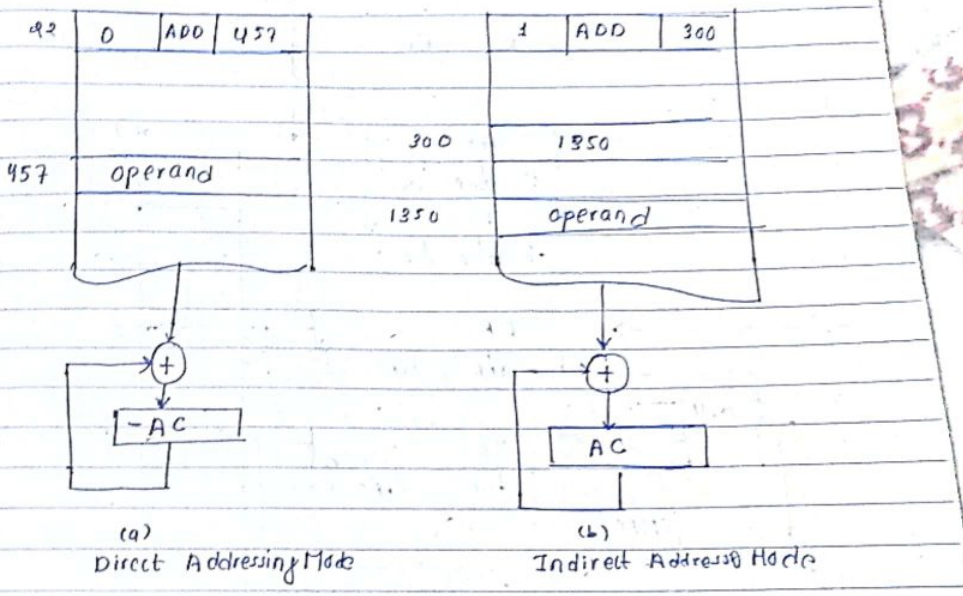
→ The Instruction set is said to be complete if it includes:-

- Arithmatic and logical instruction
- Shift instruction
- Input / Output instruction
- Instruction for moving information from one memory and register
- Program control instruction

Timing and Control



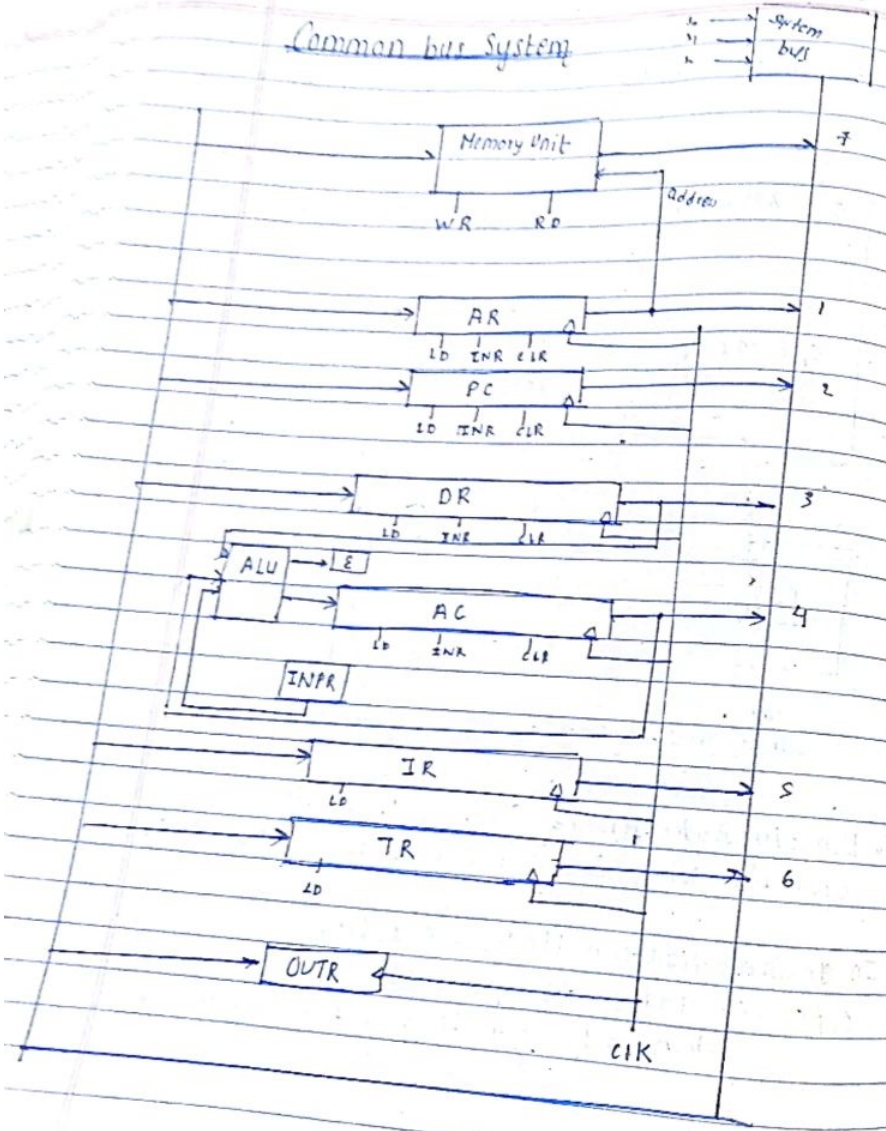
Direct and Indirect Addressing of Basic Computer



In Directing Addressing Mode, the address part holds the operand. When I=0, it indicate direct addressing mode

In, Indirect Addressing Mode, the address part holds the effective address of operand.
When I=1, it indicate indiref address mode.

Common bus System



Computer Register

→ Register can be defined as the high speed memory storing unit which is used by microprocessor during any operation.

Lets say a memory unit of 4096 words and each word contain 16 bit.

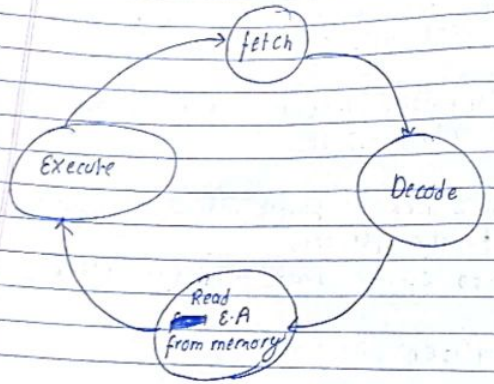
Here,
- 12 bit are used to specify address of operand
- 3 bit for the operation
- 1 bit to specify direct or indirect address

List of Register

- DR (Data register) → holds data - 16 bit
- AC (Accumulator) → all operation are peformed in AC - 16 bit
- IR (Instruction Register) → instruction read from memory are holded - 16 bit
- TR (Temp Register) → temporary hold of data during processing - 16 bit
- PC (Program counter) → holds the address of next instruction - 12 bit
- AR (address Register) → holds address of current instruction - 12 bit
- INPR → input
- OUTR → output

} 8 bit

Instruction Cycle

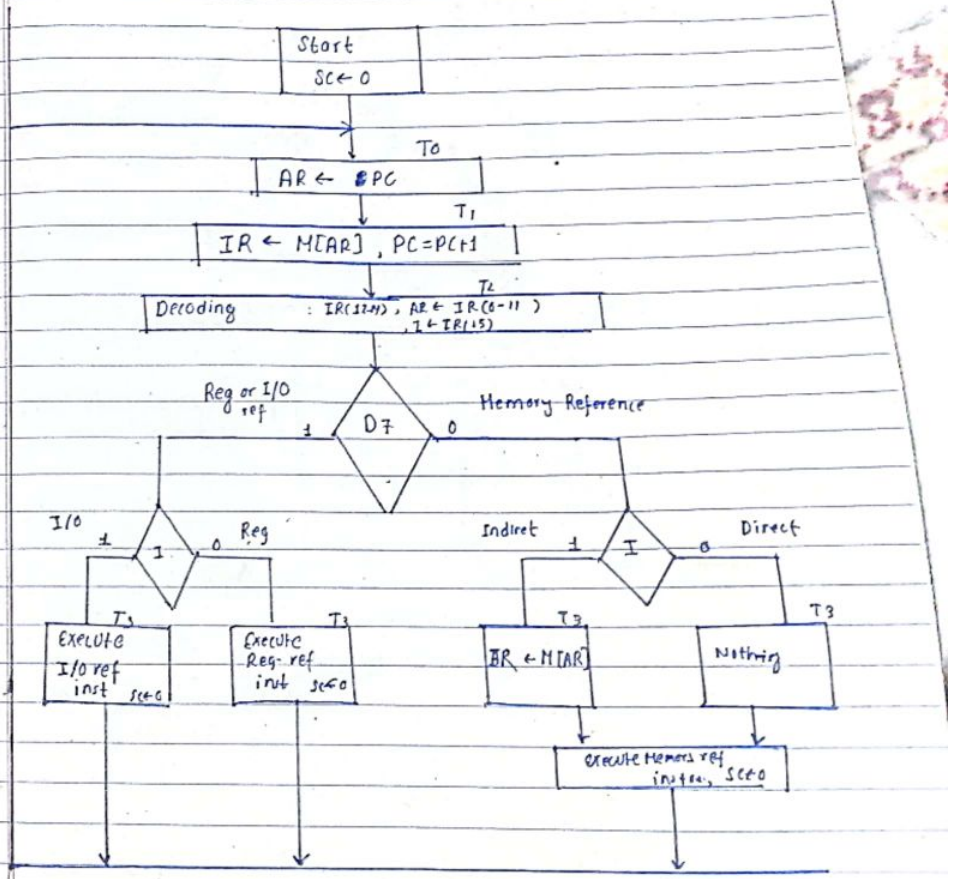


Fetch and Decode

→ T0 : AR ← PC
T1 : IR ← M[AR], PC ← PC+1
T2 : D0, . . . D7 ← Decode IR(12-14)

→ Initially, PC consist the address of 1st instruction
→ Then, SC is set to zero, (SC ← 0).
→ The value of PC is loaded to AR (address Register).
→ Now AR holds the address of 1st instruction
→ The data / memory value of AR is loaded to IR.

After every clock pulse, the value of Sequence Counter (SC) is incremented by 1, so the timing signal also go through sequence T0, T1, T2 . . . Tn.



$D_7 I T_3$: I/O-reference Instruction is Executed
$D_7 I' T_3$: Register reference Instruction is Executed
$D_7' I T_3$: $AR \leftarrow M[AR]$
$D_7' I' T_3$: Nothing

Register Reference Instruction

$D_7 I' T_3 = r$
$B_i = IR(0-11)$

→ INC	increase AC	
→ CMA	complement	
→ CLA	clear	
→ SPA rB_4	skip if positive	
→ SNA rB_2	skip if -ve	
→ SZA rB_2	skip if 0	
→ SZE rB_1	skip if $E=0$	
→ HLT rB_0	Halt computer	

Memory Reference Instruction

✓ → AND	D_0	$AR \leftarrow AC \wedge M[AR]$
✓ → ADD	D_1	$AC \leftarrow AC + M[AR]$
✓ → LDA	D_2	$AC \leftarrow M[AR]$
✓ → STA	D_3	$M[AR] \leftarrow AC$
→ BUN	D_4	$PC \leftarrow AR$
→ BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
↔ ISR	D_6	$M[AR] \leftarrow M[AR]+1$
		if $(M[AR]+1 = 0)$ then $PC \leftarrow PC+1$

* ADD to AC

$D_1 T_4$: $DR \leftarrow M[AR]$
$D_1 T_5$: $AC \leftarrow AC + DR$

* LDA

$D_2 T_4$: $AC \leftarrow M[AR]$

* BUN : Branch Unconditionally

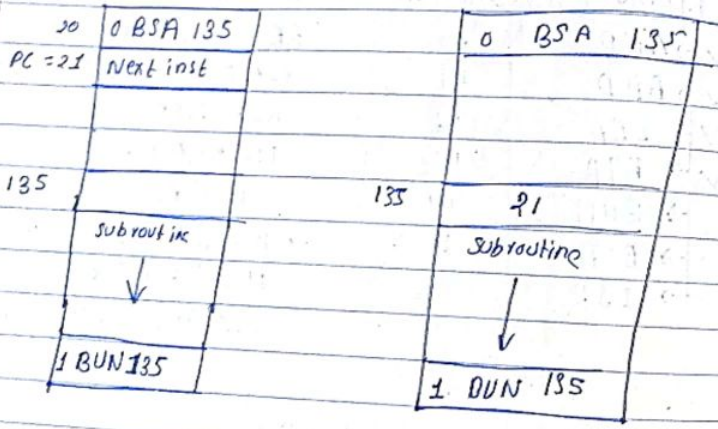
→ This instruction transfer the program to the instruction specified by the effective address.

D_4T_4 : $PC \leftarrow AR$, $SC \leftarrow 0$

and

* BSA (Branch, Save Return Address).

→ Used in subroutine.



At Time T_4

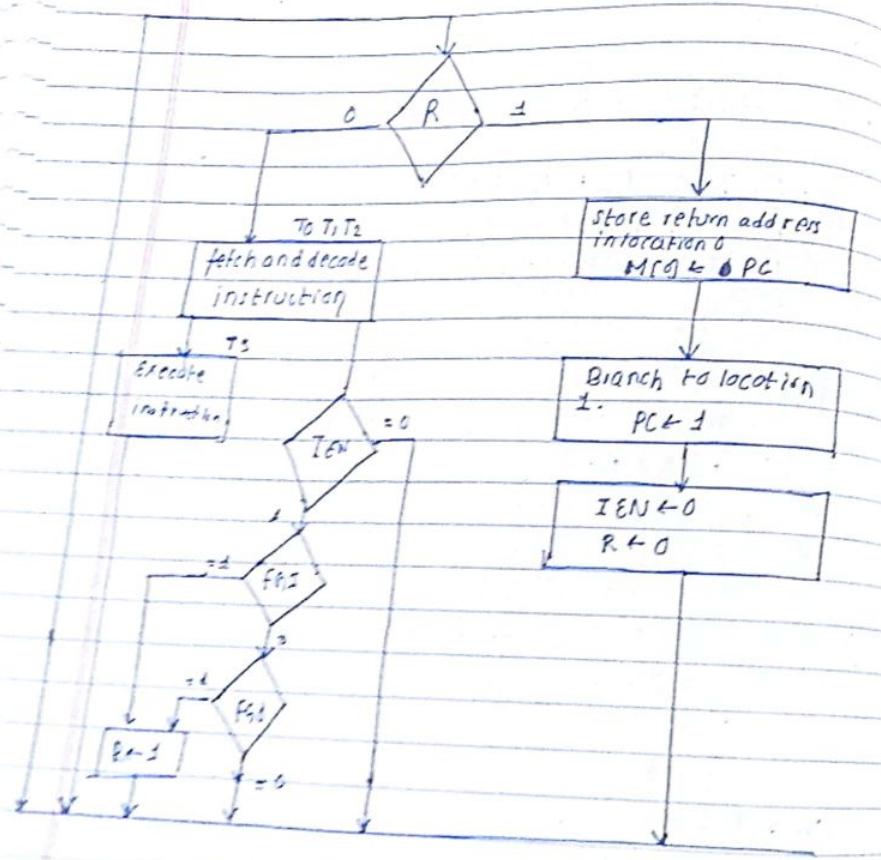
At time T_5

D_4T_4 : $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
D_5T_5 : $PC \leftarrow AR$, $SC \leftarrow 0$

* ISZ : Increment and Skip if zero.

D_6T_4 : $DR \leftarrow M[AR]$
D_6T_5 : $DR \leftarrow DR + 1$
D_6T_6 : $M[AR] \leftarrow DR$, if $(DR = 0)$ then $PC \leftarrow PC + 1$, $SC \leftarrow 0$

Interrupt Cycle



→ The flow chart gives the information who the interrupt is handled by the computer.

→ It consist of interrupt flipflop R.

When, R = 0,
computer goes through instruction cycle

When R = 1,
computer goes through interrupt cycle

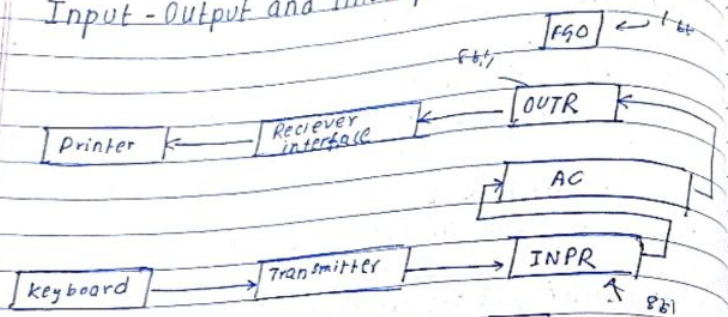
→ IEN (Interruple Enable flage) = 0,
then programmer don't want to use interrupt
so, control continius with next instruction.

when, IEN = 1,
it check for flag bit, if both (FGI and FGO are Zero)
neither Input or output register are ready.
If either of flage are set, then R is set to 1. (R ← 1)

A-B

A = t1 = Q...
B = t2 = Q...

Input - Output and Interrupt



Input output - Instruction	
INP	FGI ← 0 (initially)
OUT	FGI ← 1 (when new info is available at the input device)
SKI	When FGI = 0,
SKO	Info from INPR to OUTR thy PC
ION	
IOF	FGO ← 1 (initially) then OUTR ← AC then FGO ← 0.

flow chart of Basic Computer