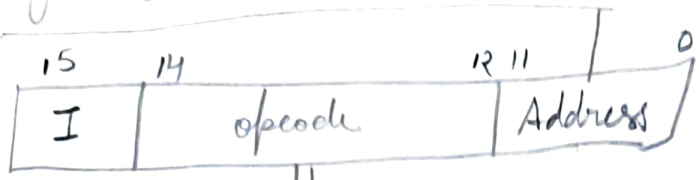
Memory Reference Instruction



000
110 → Decoder output $D_0 - D_6$

effective add during T_2 at $I=0$

during T_3 at $I=1$

→ execution during T_4.

Note: AC can't recieve data from Commonbus diratly.

1. AND to AC

$$AC \leftarrow AC \wedge M[AR]$$

$$\downarrow$$

$$AC \leftarrow AC \wedge DR$$

$D_0 T_4 : DR \leftarrow M[AR]$

$D_0 T_5 : AC \leftarrow AC \wedge DR \; ; \; SC \leftarrow 0$

2. ADD to AC

$$AC + M[AR]$$

$D_1 T_4 : DR \leftarrow M[AR]$

$D_1 T_5 : AC \leftarrow AC + DR \; ; \; SC \leftarrow 0 \; ; \; E \leftarrow Cout$

In Basic compu...

LDA

$AC \leftarrow M[AR]$

D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR$; $SC \leftarrow 0$

4. STA [Reverse]

D_3T_4 : $AC \rightarrow M[AR]$; $SC \leftarrow 0$

5. BUN : Branch Unconditionally

Main Program

PC → Statement 1

→ Statement 2

→ Bun [N1]

Statement n (end)

New Program

PC → N1: Statement 1

Statement 2

Statement n (end)

D_4T_4 : $PC \leftarrow AR(N1)$; $SC \leftarrow 0$

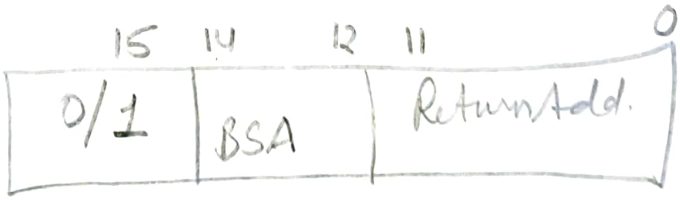
BSA : Branch & Save Return Address

Assembly

```
main Program
    1.  S1
    2.  S2
        :
    20  S20 → BSA
RA ⇒ 21.  S21 → Next inst.

    HLT

subprogram
start

    BUN
    HLT
```

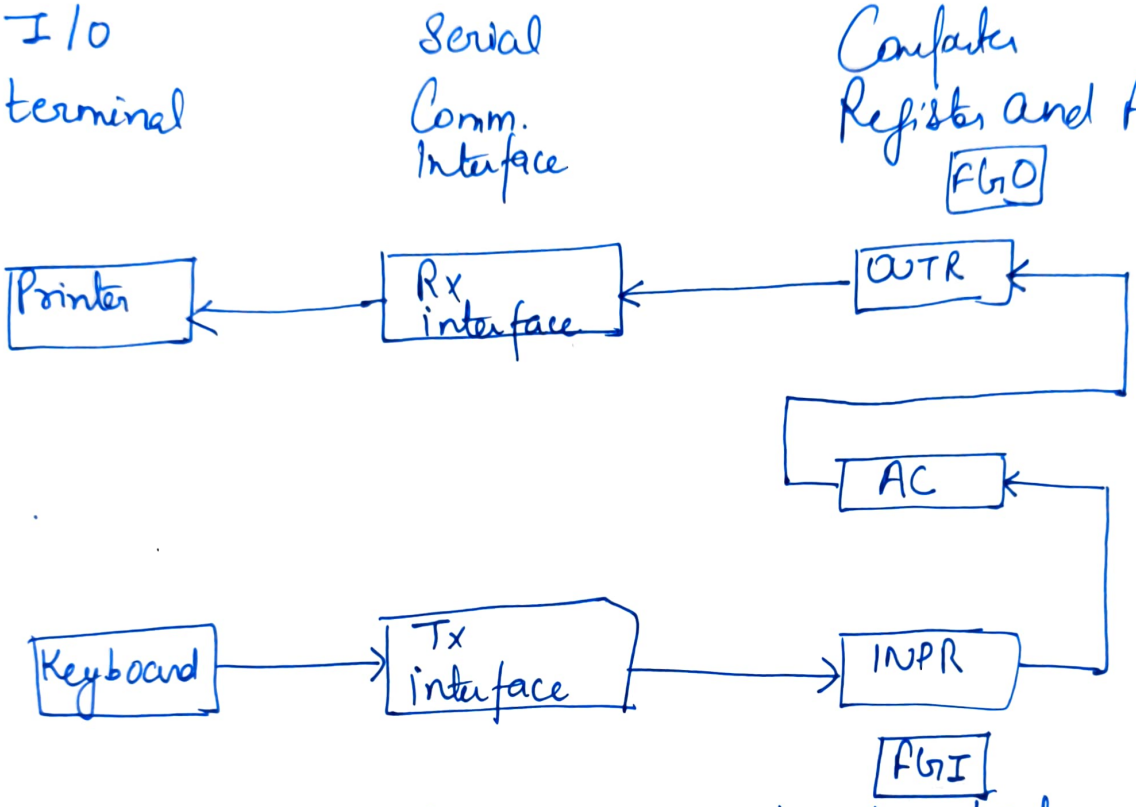
C program

```
main
{
 1. S1
    S2
    :
20. S20 → function
RA⇒21. S21 → Next inst.
}

function ( )
{

Ret ;
}
```



Input-Output & Interrupt

Input-Output Configuration



Terminal: Sends or Recieve serial information. Each unit of information has 8-bits of an alphanumeric code.

Keyboard: The serial information from Keyboard is shifted into INPR.

Printer: The serial information for the printer is stored in the OUTR.

INPR & OUTR: interact with peripheral thr. Tx or Rx interface serially. and with AC in parallel

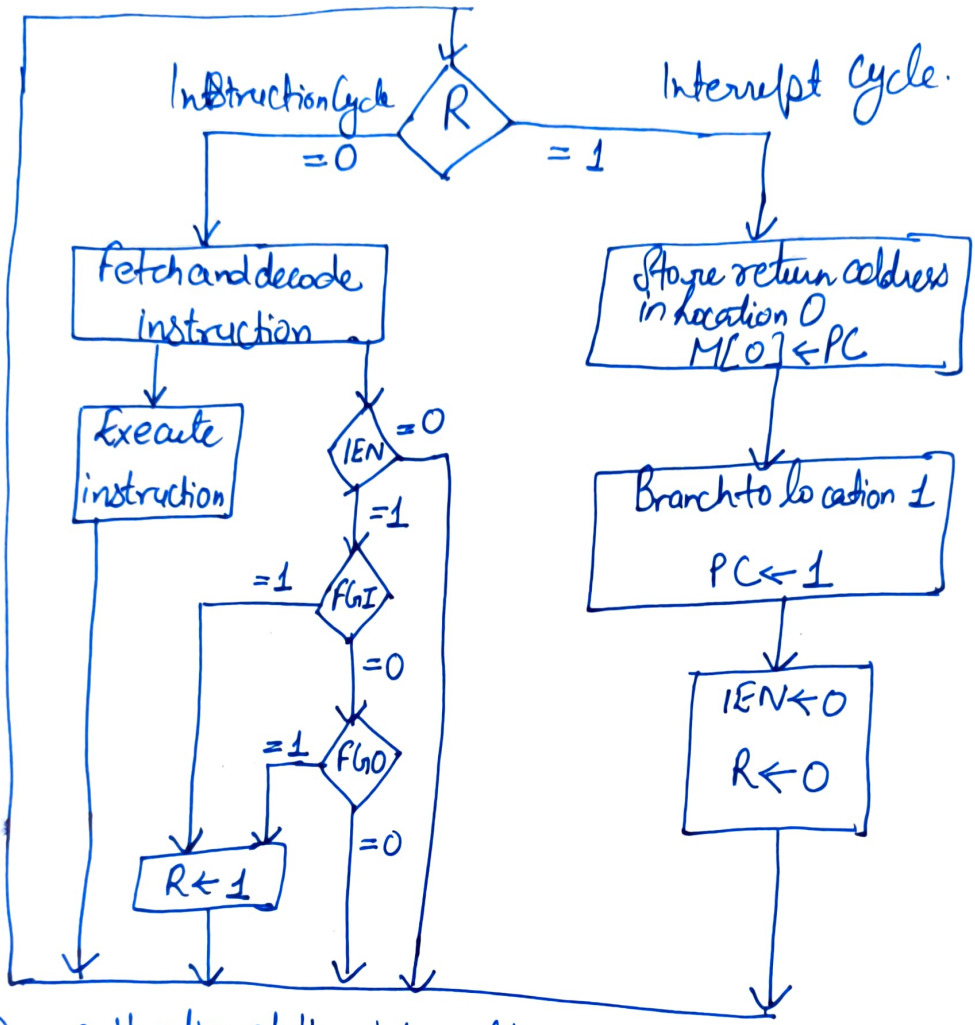
Tx: Recieves information from keyboard & transmit to INPR.

Rx: Recieves information from OUTR & " " Printer. Serially.

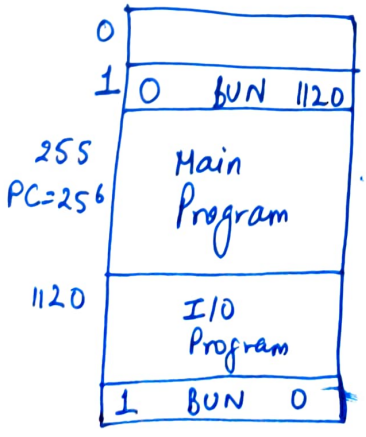
INPR: of 8 Bits and holds an alphanumeric input info

FGI: Input flag. is a 1-Bit FF.

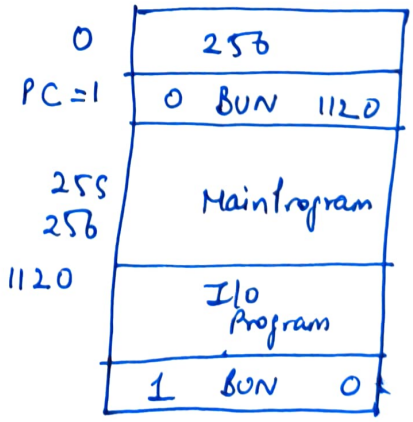
Flowchart for Interrupt Cycle.



Demonstration of the interrupt cycle.



(a) before interrupt



After interrupt cycle.

Input - Output Instruction

$D_7 I T_3 = p$; p: $SC \leftarrow 0$

INP	pB_{11}	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}	$OUTR \leftarrow AC(0\text{-}7)$, $FGI \leftarrow 0$
SKI	pB_9	if $(FGI = 1)$ then $(PC \leftarrow PC+1)$
SKO	pB_8	if $(FGO = 1)$ then $(PC \leftarrow PC+1)$
ION	pB_7	$IEN \leftarrow 1$
IOF	pB_6	$IEN \leftarrow 0$

Instruction format

Common field

1. opcode field
2. An address field
3. A mode field – specify the way the operand or effective address is determined.

Address field: The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of the type of CPU organizations

1. Single accumulator Organization.
2. General Register Organization
3. Stack organization

1. Accumulator type Organization :

all operations are performed with an implied accumulator register use 1 address field.

e.g. ADD X

↳ address of operand

AC ← AC + M[X]

↳ ACC. Register

2 General Register Organization.

* Uses 3 address field: 2 for operands + 1 for Result

e.g. ADD R_1, R_2, R_3

$R_1 \leftarrow R_2 + R_3$

* Uses 2 address fields as well: 2 for operands & One of them is used for Result storage left most.

e.g. ADD R_1, R_2

$R_1 \leftarrow R_1 + R_2$

* Uses 2 or 3 address field, where add. field may specify a processor Register or a memory word.

e.g. ADD R1, X

$R_1 \leftarrow R_1 + M[X]$

* 2 address field for data transfer

e.g. MOV R_1, R_2

Move or transfer

$R_1 \leftarrow R_2$.

Instruction FORMATs

⇒ Computer with multiple processor registers use the Move instruction with Mnemonic MOV to transfer instruction.

MOV R_1, R_2 [transfer type instruction needs two address fields

$R \leftarrow R_2$

→ General register type employ two or three address fields with processor register & memory word.

ADD R_1, X

$R_1 \leftarrow R_1 + M[X]$ has two addressing field

→ Stack organization used PUSH & POP instruction which require an address field

PUSH X

Types of address Instruction

① Three address Instruction
② Two Address Instruction
③ one Address Instruction
④ Zero Address Instruction

Assume operand are in memory addresses A, B, C, D
Result in memory X

① Three address Instruction

Assemble Langvag Program for $X = (A+B)*(C+D)$

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

→ advantage - short program but require toomany bits

② Two address Instruction

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

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

③ One Address Instruction Implied accumulator (AC) register

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

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

MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

(4) Zero Address Instruction

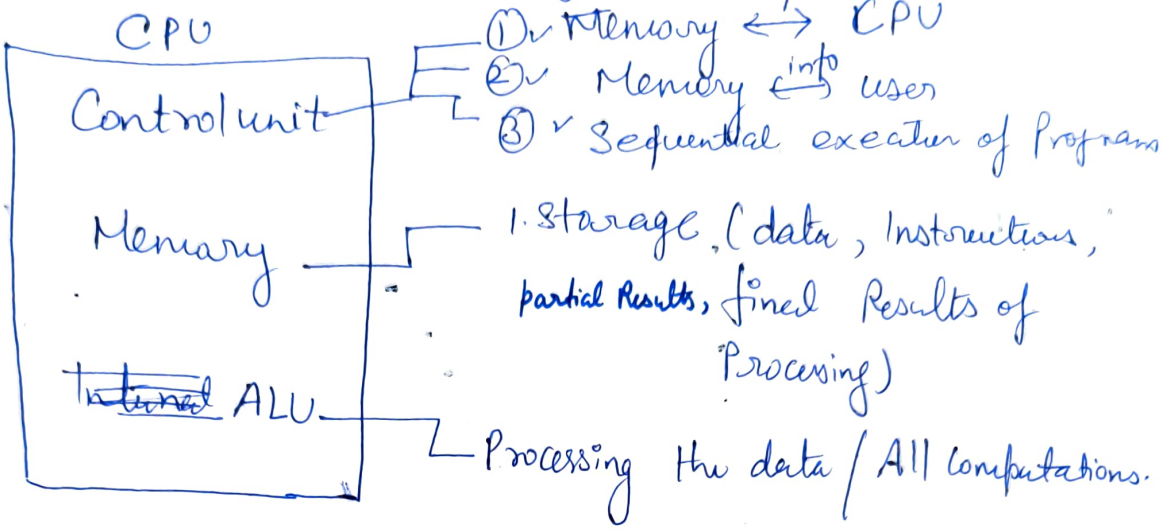
A stack-organized computer does not use an address field for the instruction ADD and MUL, The PUSH POP instructions, however, need an address field to specify the operand that communicates with stack. (TOS ← Stands for TOP of stack)

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

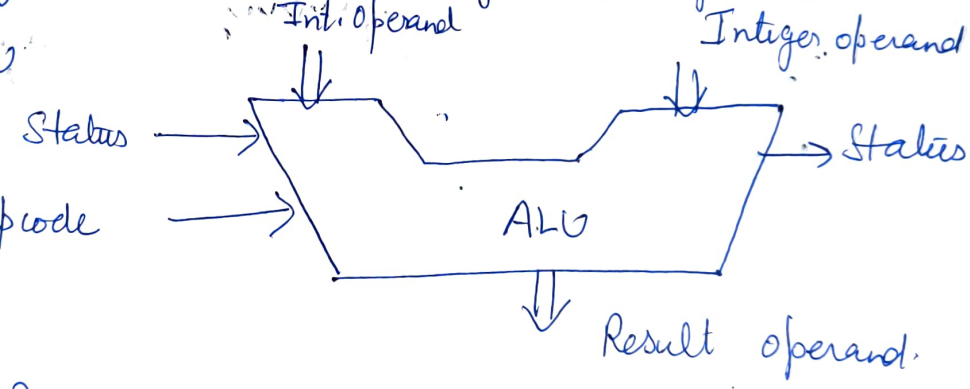
PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A + B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C + D)
MUL		TOS ← (A+B) * (C + D)
POP	X	M[X] ← TOS

RISC Reduced Instruction Set Computer

CPU (Central Processing unit).



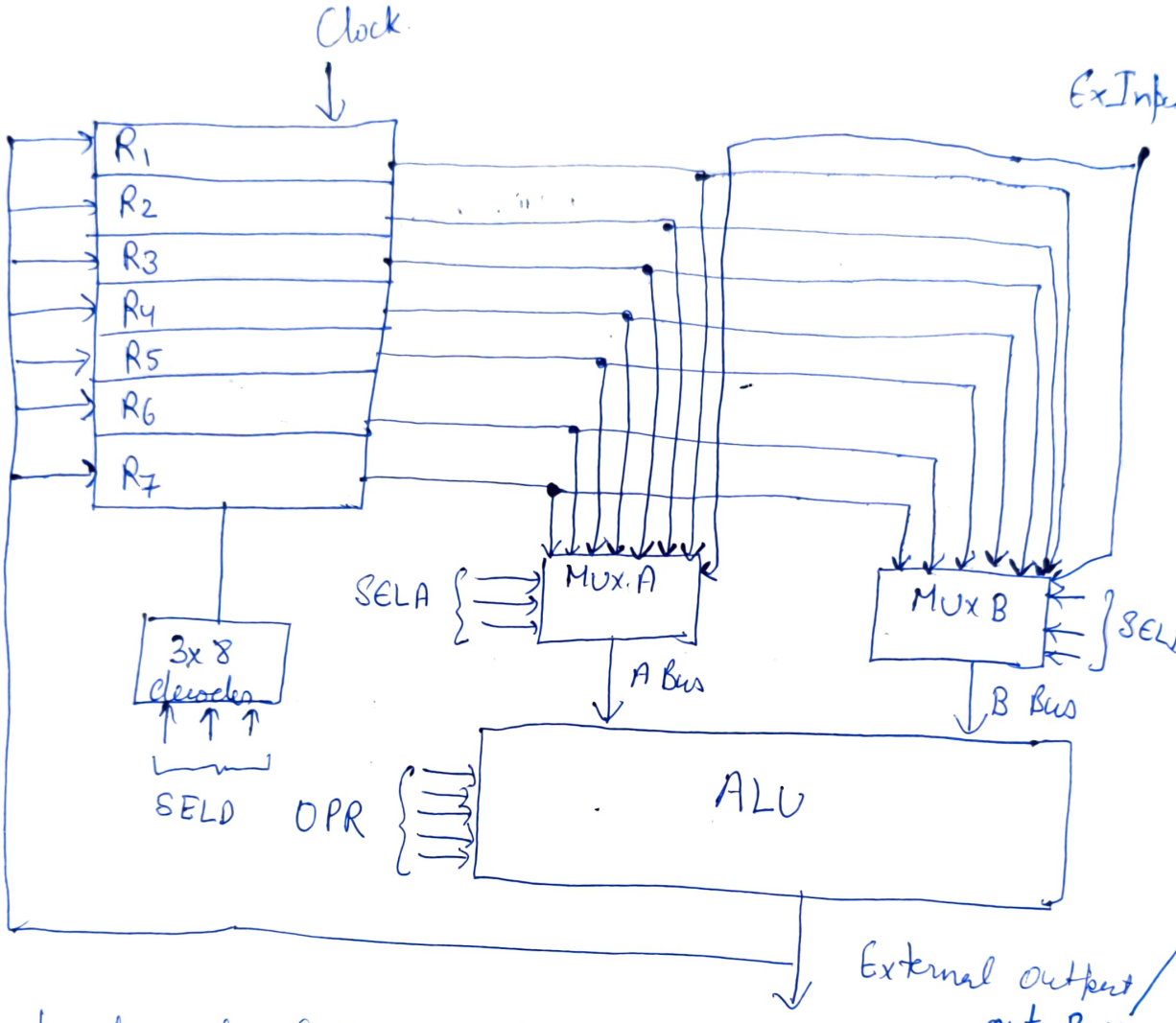
* Memory unit (Internal Registers, Flags (E)
* ALU



General-Register Organization.

Requirement: To store intermediate Results for fast processing.

7 Register Organization



Operation of Control unit

Example 1: $R1 \leftarrow R2 + R3$.

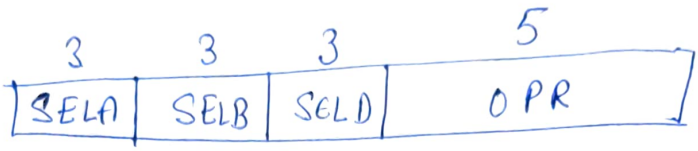
1] MUX A Selection (SELA): Bus A $\leftarrow R_2$

2] " B " (SELB): Bus B $\leftarrow R_3$

3] ALU operation Selector (OPR): ALU to ADD

4] Decoder destination selector (SELD): $R_1 \leftarrow$ Out Bus.

Control Word.



Encoding of Registers Selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

Examples of ALU Microoperations

μOperation	SELA	SELB	SELD	OPR
① $R_1 \leftarrow R_2 - R_3$	R_2	R_3	R_1	Sub
② $R_4 \leftarrow R_4$ OR R_5	R_4	R_5	R_4	OR
③ $R_6 \leftarrow R_6 + 1$	R_6	-	R_6	INCA
④ $R_7 \leftarrow R_1$	R_1	-	R_7	TSFA
⑤ Output $\leftarrow R_2$	R_2	-	None	TSFA
⑥ Output $\leftarrow$ input	Input	-	None	TSFA
⑦ $R_4 \leftarrow$ shl R_4	R_4	-	R_4	SHLA
⑧ $R_5 \leftarrow 0$	R_5	R_5	R_5	XOR

RISC and CISC

CISC	RISC
1. Complex Instruction Set Computer.	1. Reduced Instruction Set Computer.
2. Large no. of Instructions.	2. Less no. of Instructions
3. Variable Length Instruction format.	3. fixed length Instruction format
4. Large no. of addressing modes.	4. few no. of addressing modes
5 Cost is high	5 Less Cost
6. More Powerful.	6. Less Powerful
7. Several Cycle Instruction	7. Single Cycle Instruction
8. Manipulation directly into memory.	8. Only in Registers
9. Microprogrammed Control Unit.	9. Hardwired Control Unit.
10. Eg: Mainframe, Intel 8086	10. ARM : Advanced RISC Machine Supercomputer
11. Slow Processor.	11. Fast Processor.