DATE:
PAGE:

Bhalodia Shruti

Computer System Architecture

Sem - IV

DATE:
PAGE:

→ Computer Architecture / Computer Organization

↓ what to do

(ex. For building, civil architec plans where should be living room, kitchen)

↓ How to operate

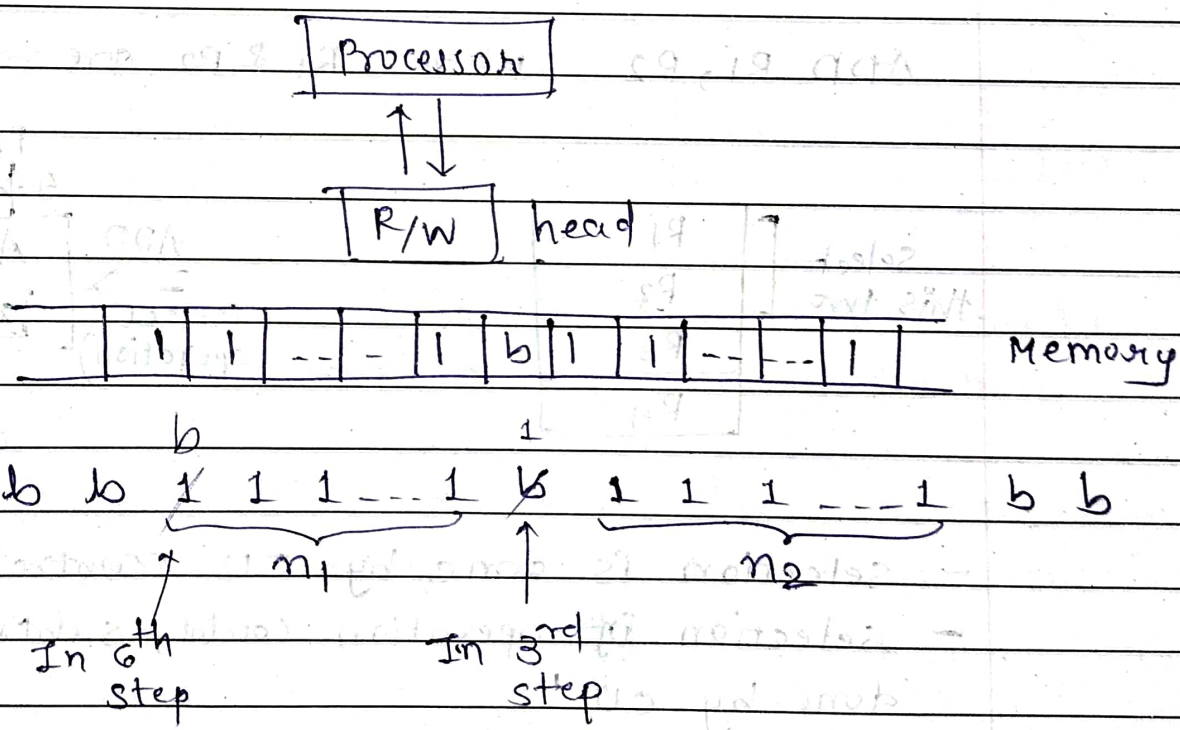
(ex. control signals).

- Actual implementation of various functional units required to build compt.

→ Three notions of Reasonable computer

1. computers should provide the answers whose programs are given.
2. It should operate in a finite speed
3. It can't store answers to all possible problems.

→

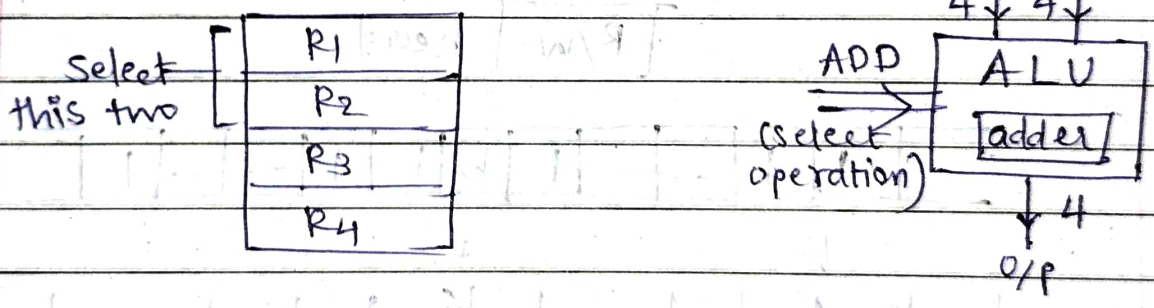


	Present state	I/P	R/W	next state
1.	S_0	b	R	S_1
2.	S_1	1	R	S_1
3.	S_1	b	1	S_2 (left shift)
4.	S_2	1	L	S_2
5.	S_2	b	R	S_3
6.	S_3	1	b	S_3
7.	S_3	b	Hold	S_3

we get continuous 1111111111 which is the sum of n1 & n2.

→ Instruction set architecture

ADD R1, R2 where R1 & R2 are registers



- selection is done by CU (control unit)
- selection of operation (add, subtract...) is done by CU.
- select which operation is done (R/W)

ex. $R_1 \leftarrow R_1 + R_2$

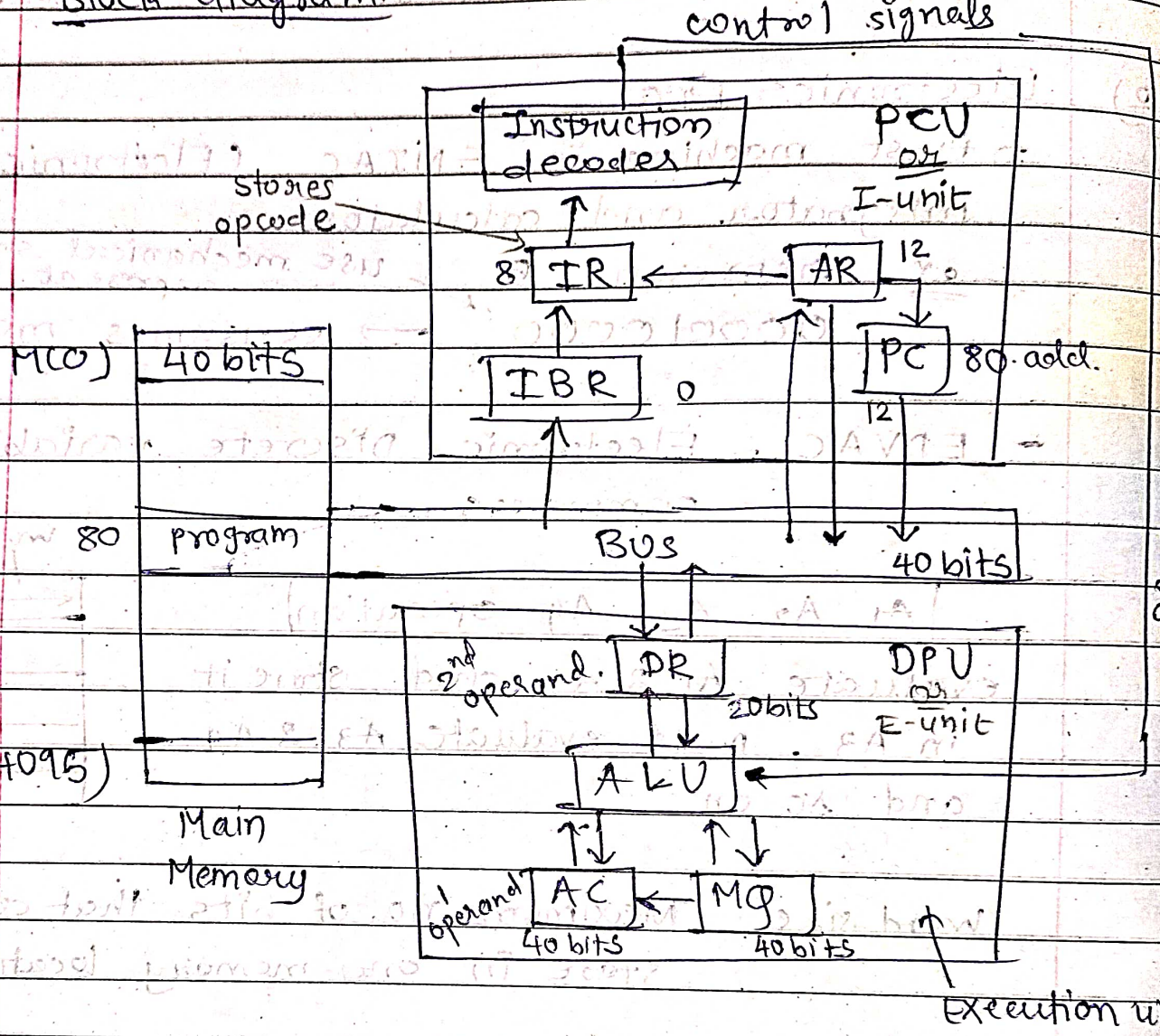
Read from R1 & R2, write on R1

⇒ OPcode - Operation code

(for every operation there is code associated with it)

3) TAS Machine:

→ Block diagram:



→ size of instruction = 20 bits

→ There are three types of bus
(1) Address bus - processor to memory
(2) data bus - memory to processor
(3) Control bus - R/w operation

→ Programs are stored into the memory. (In sequence). There might be 100 of programs into memory.

→ | 8 bits | 12 bits | ← 20 bits. (1 instruction)
opcode — add. of memory

→ Total size of memory $= 2^{12} = 4K$ (0 to 4095)

→ Memory size = 8 bits
8 different memory location.
so $2^3 = 8$, [3] bits are required

8 bits

0	M(0) 000
1	001
2	⋮
⋮	⋮
7	M(7) 111

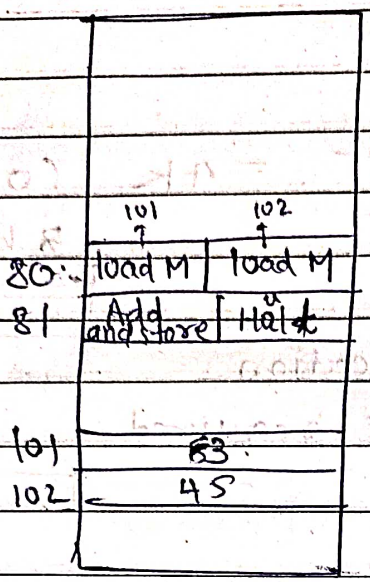
→ At 1 memory location, we can store 2 instruction.

→ IR → 8 bits
AR → 12 bits
PC → increment counter in memory
(If we are at M(0) → it will incremented by one. (M(1)))

eg Addition of data at M(101) + M(102) and result should be stored at M(103)

M(103) ← M(101) + M(102)
↖ Instruction and data is of 40 bits.

① AC := M(101) [load the content into AC]
② AC := AC + M(102) [Add the content of 102 to the AC]
③ M(103) := AC [store the content of AC in M(103)]



(1) LOAD M(101) (AC = M(101)
(2) ADD M(102) (∵ AC = AC + M(10
(3) M(103) = AC (STORE)
(4) Hault.

→ Data transfer Instruction

1. AC = MQ
Move the data from MQ to AC.

2. AC = M(X) (load instⁿ)
Move the data at x into AC
copy

3. M(X) = AC (store instⁿ)

4. ~~MQ~~ MQ = M(X)

5. AC = − M(X)

6. AC = |M(X)|

7. AC = − |M(X)|

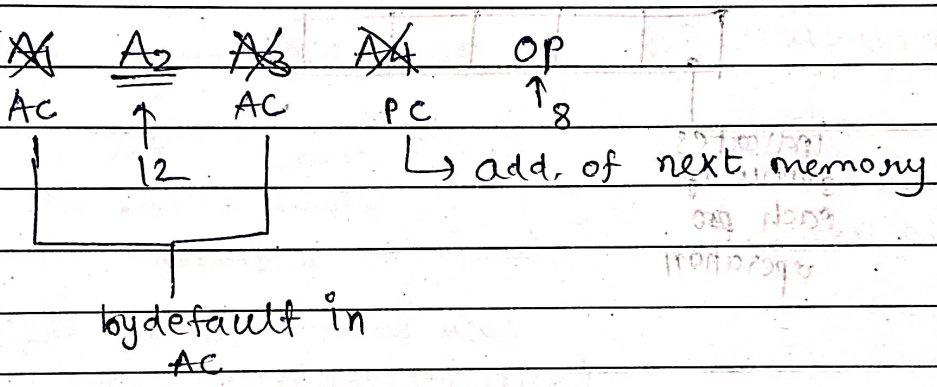
→ For every loc opcode would be provided which is of 8 bit.

DATE:
PAGE:

→ Data Processing Instⁿ.

1. ADD M(X) AC = AC + M(X)
   └ one operand is in M(X) and other is by default in AC. and after performing add. result is by default in AC
2. AC = AC + |M(X)|
3. AC = AC − M(X)
4. AC = AC − |M(X)|
5. AC · M(Q) = M(Q) × M(X)
6. M(Q) · AC = AC / M(X)
7. AC = AC × 2 (left shift)
8. AC = AC/2 (right shift)

→



→ n bit no. × n bit no = 2n bit no.

→ Program control Instⁿ.

1. go to M(X, 0 : 19) (0 to 19) 20 bits
2. go to M(X, 20 : 39)
3. If AC ≥ 0 then go to M(X, 0 : 19)
4. If AC ≥ 0 then go to M(X, 20 : 39)
5. M(X, 8 : 19) = AC(28 : 39)
   ↳ contains address
6. M(X, 28 : 39) = AC(28 : 39)

DATE:
PAGE:

Drawbacks of 1st gen. comp.

IAS comp.

1) Text processing was not included.
2) I/o operations can't done.
3) Index Registers was not provided
4) Structural programming can't done.
5) No. of Registers were very less.

Not in 1st generation comp.

Critique

- In 2nd generation of computers, Transisters, Os, compiler, Virtual memo

→ It also uses flag registers

indicates status of each operation

→ To increase speed of processor
1) pipelining
2) Cache
3) Parallel processing

→

t = 1 2 3 4 5 6

IF1 IF2
(Fatch)
DI1 DI2
(Decoder)
FO1 FO2
(Fetch operand)
EX1 EX2
(Execution)
SR1 (store reg.)

pipelining

end of 1st inst

DATE:
PAGE:

2) Cache

cache memory < size of main memory

static RAM (costlier) — built using dynamic RAM

speed of processor > main memory speed

5) parallel processing

- no. of processor is placed in single IC.

Processor having two types of architecture

① Von Neuman

- Inst^n / Data placed in same memory

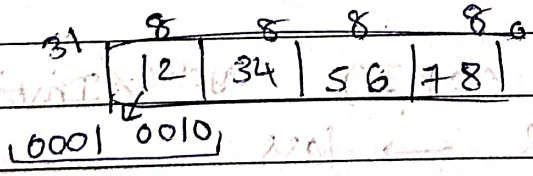
(all earlier systems uses Von Neuman structure)

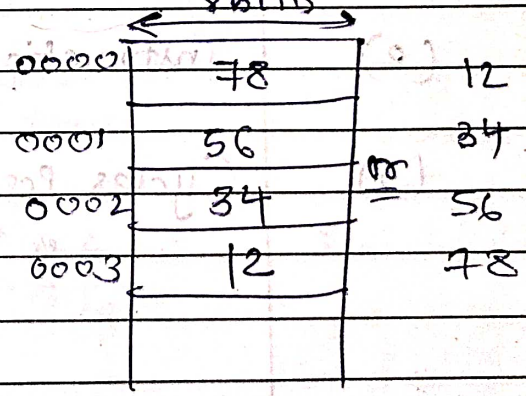
② Harward

- Inst^n / Data placed in seperate memory

- sections of code & data are at seperate address.

32 bit data



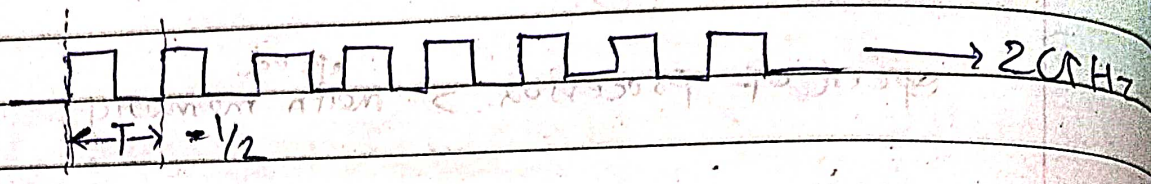


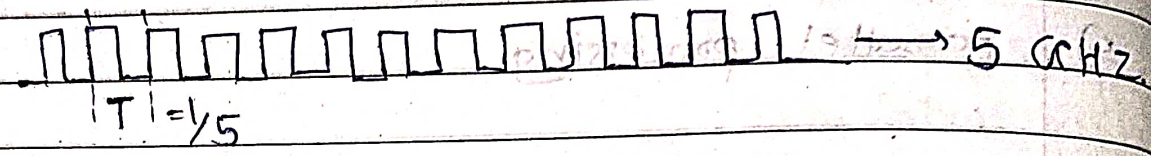
- little endian - LSB is placed is stored at the lower add. (0000) is known as little endian (Intel)

- Big endian - stored at the higher address (0003) (Motorola)

→ Factors affecting performance of a CPU.

(1) clock rate

(i) 

(ii) 

For performing one inst^n (i)

$$5T = 5 \times \frac{1}{2 \times 10^9}$$

↑ from previous $t=1$ to 5

(ii) $$5T = 5 \times \frac{1}{5 \times 10^9}$$

Therefore, (ii) is more faster

(2) Instruction count

(3) Cycles Per Inst^n (CPI) Cycles/Inst^n

ex AC+DR → less

AC+M(X) → more

DATE:
PAGE:

→ Execution time of program depends on

- VLSI
- Computer organization
- ISA (Instruction set Architecture)
- Compiler Technology
- size of the program

	Instruction count	Clock per Instruction (CPI)	Clock rate
VLSI	X	X	✓
Com or	X	✓	✓
ISA	✓	✓	✓
Com. Tech.	✓	✓	X
Program	✓	✓	✓

→ Execution Time = IC X CPI x Clock rate of a program

→ performance of CPU = $\frac{1}{\text{Execution time}}$

→ speedup = $\frac{\text{Performace of CPU}_A}{\text{Performance of CPU}_B}$

= $\frac{\text{Execution time of CPU}_A}{\text{Execution time of CPU}_B}$

Ex.1 A program is run in machine A, B, C. And execution time of it is given that 10, 25, 75 units.

Processor A is 2.5 times faster than B.
Processor A is 7.5 times faster than C.

ICi ⇒ Instruction count of type i

ex.1

1	2	3	4	5	6
IF_1	IF_2	IF_3	IF_4		
	DI_1	DI_2			
		FO_1			
			EX_1		
				SR_1	

```
1  ADD AC, DR
2  JMP next   ← unconditional Jump
3  MUL AC, DR
4  next.
```

ex

CPIi ⇒ Cycles / Instruction

CPU clock cycles $= \sum_{i=1}^{n} ICi \times CPIi$

Inst^n count (IC) $= \sum_{i=1}^{n} ICi$

CPI for entire Inst^n set $= \dfrac{\sum_{i=1}^{n} ICi \times CPIi}{IC}$

DATE:
PAGE:

$$\text{CPI} = \sum_{i=1}^{n} \left(\frac{IC_i}{IC}\right) \times CPI_i$$

Exe Consider an implementation of ISA where inst^{n}s are classified into 4 diff. types with CPI values of 1, 2, 3, 4. Two code sequences have following inst^{n} count (IC)

	1 clock cycle ↓ IC_1	2 ↓ IC_2	3 ↓ IC_3	4 ↓ IC_4
code seq. 1	20	15	5	2
code seq. 2	10	12	10	4

Find which is more better ?

$$\sum_{i=1}^{4} IC_i \times CPI_i$$

$$\text{CPU}_1 \text{ cycles (clock)} = (20\times1) + (15\times2) + (5\times3) + (4\times2) = 73$$

$$\text{CPI for entire Inst}^{n} \text{ set} = 73 / \text{Instruction} = \frac{73}{42} = \boxed{1.74}$$

$(20+15+5+2)$

$$\text{CPU clock cycles} = (10\times1) + (12\times2) + (10\times3) + (4\times4) = 80$$

$$\text{CPI} = \frac{80}{36} = \boxed{2.22}$$

Therefore A is better

EX.3 Machine A executes programs with avg. CPI of 2.3, Machine B executes with same inst^n set but better compiler a same prog with 20% less inst^{n}s and CPI of 1.7 at 1.2 GHz. What should be the clock rate of A so that two machines are same performance

→ (Execution time)$_A$ = (Execution time)$_B$

$\therefore IC_A \times CPI_A \times \frac{1}{\text{clock rate}_A} = IC_B \times CPI_B \times \frac{1}{\text{clock rate}_B}$

$IC_A \times 2.3 \times \frac{1}{\text{clock rate}_A} = (IC_A - 0.2IC_A) \times 1.7 \times \frac{1}{1.2\text{ GHz}}$

$\therefore 2.3 \times \frac{1}{\text{clock rate}_A} = 0.8 \times 1.2\text{ GHz} \times 1.7$

$\therefore \frac{1}{\text{clock rate}_A} = 0.49$

$\therefore$ (clock rate)$_A$ = 2.04 GHz

clock rate$_A$ = 0.4928×10^{-9}

$= \frac{1}{0.4928 \times 10^{-9}}$

$= 2.04$ GHz

⇒ RISC / CISC

RISC
- Reduced Inst^n set computers
- pipelining is done easily
- Increase No. of Inst^n per program
- Decrease CPI and clock cycle time as inst^{n}s and Implem

CISC
- Complex Inst^n set Computers
- more no. of clock cycles
- fetching, executing is done in one cycle
- pipelining is difficult

From experiments,
RISC architecture gives better performance.

Ex. For implementation of RISC ISA machine,

Type	freq.	CPI
load	20%	4
Store	8%	3
ALU	60%	1
Branch	12%	2

$$\text{Overall CPI}_{Avg} = \sum_{i=1}^{n} \left(\frac{IC_i}{IC}\right) \times CPI_i$$

$$= (0.2 \times 4) + (0.08 \times 3) + (0.6 \times 1) + (0.12 \times 2)$$

$$= \boxed{1.88}$$

Ex Suppose that a program is running on a machine with a foll^{n} Instr^{n} types.

Type	CPI	freq
A	1.3	60%
B	2.2	10%
C	2.0	30%

The CPU designer gives two options, (i) first is reduce CPI of type A to 1.1 (ii) Reduce CPI of type B to 1.6.

→ CPI Avg. $= (1.1 \times 0.6) + (2.2 \times 0.1) + (2.0 \times 0.$
in option (i) $= \boxed{1.48}$

CPI Avg. $= (1.3 \times 0.6) + (1.6 \times 0.1) + (2.0 \times 0.3$
in option (ii) $= \boxed{0.84} \; \boxed{1.54}$

(i) is better than (ii) as $1.48 < 1.54$

→ MIPS (Millions of Inst^{n} per second).

$$MIPS = \frac{IC}{\text{execution time}} \times 10^{-6}$$

→ Performance measurement of
MFLOPS (Millions of floating point Inst^{n}s/se

→ MIPS depends upon: (factors)
1) Inst^{n} set.
2) MIPS varies from one program running on same processor to other program running on the same processor.
3) Higher MIPS may not mean better performan
(compiles should also be same)

→ MIPS is only valid if full^{2} cond^{n}s are satisfied
1) same program is used
2) same ISA
3) same compiler is used.

DATE:
PAGE:

Ex Consider a processor with three inst^n classes A, B, C with the corresponding CPIs are 1, 2, 3. The proc. runs at 1GHz. For a given prog. written in C two compilers produce the foll^g inst^n count

(in million)

	IC_A	IC_B	IC_C
Compiler 1.	7	2	1
compiler 2	12	1	1

Compute the MIPS rating and CPU time for two program versions

→ For compiler 1,

$$CPI_1 = \frac{(7M \times 1) + (2M \times 2) + (1M \times 3)}{(7+2+1)M}$$

$$\boxed{CPI_1 = 1.4}$$

unit of freq. = Hz = cycles/sec

unit of CPI = cycles/inst^n

$$(MIPS\ rating)_1 = \frac{CPU\ frequency}{CPI_1} = \frac{cycles/sec}{CPI_1}$$

$$= \frac{1 \times 10^9}{1.4} = \boxed{714.3\ MIPS}$$

→ For compiler 2,

$$CPI_2 = \frac{(12 \times 1) + (1 \times 2) + (1 \times 3)}{14} = \boxed{1.21}$$

$$(MIPS\ rating)_2 = \frac{1 \times 10^9}{1.21} = \boxed{826.45\ MIPS}$$

DATE:
PAGE:

→ CPU time using compiler 1 = IC × CPI × clock rate
$\times 10^6$
$= (7+2+1) \times 1.4 \times \frac{1}{10^9}$
$= \boxed{0.014 \text{ secs}}$

→ CPU time using compiler 2 $= (12+1+1) \times 1.21 \times \frac{1}{10^9}$
$= \boxed{0.016 \text{ secs}}$

By looking CPU time, compiler 1 is more faster

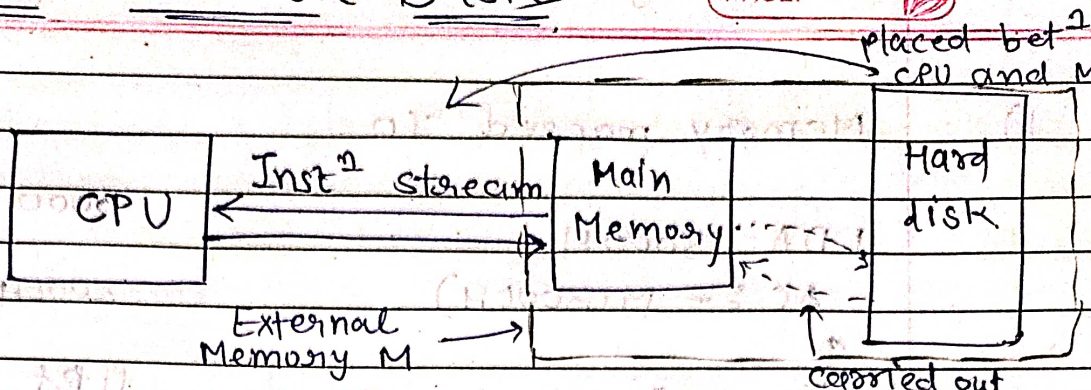
⟹ SPEC (Standard performance Evaluation corporation) - used for comparing two p

→ Execution time of prog = $\frac{\text{no. of seconds}}{\text{program}}$
$= \frac{IC}{prog.} \times \frac{\text{Clock cycle}}{Inst^n} \times \frac{sec}{cl}$

→ It contains set of Bench Mark programs.

3. Processor Basics

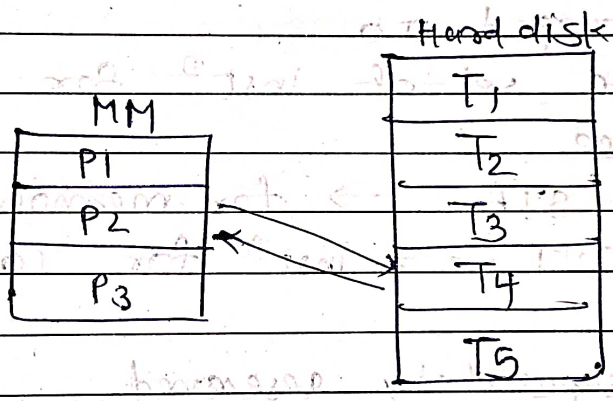
DATE:
PAGE:



→ Processor working in two modes
1.) User mode — written by users prog.
2) supervisor mode — written by OS prog.

- If resources are not enough there is switching bet^n these two modes. of processor

→



If MM is full and CPU requires prog. T4 then there is switching bet^n prog. of Hard disk and prog of Main Memory (which is not required for now).

This switching is bet^n user mode & supervisor Mode (carried out by OS.)

→ every IO device is accessed by some port address

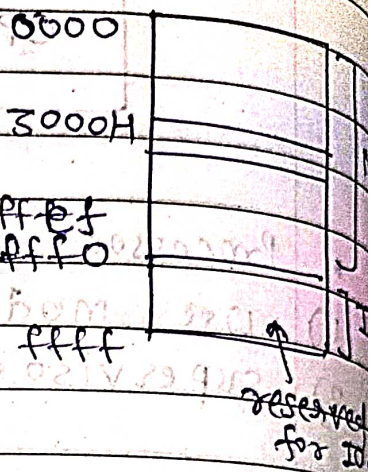
→ 2 types of IO inst^n to access the IO ports
1) Memory mapped IO
2) IO memory mapped

Programs are classified into two groups
① User programs / App^n programs
② OS programs

DATE:
PAGE:

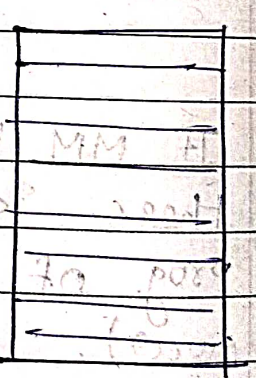
1) Memory mapped IO

LDA 3000H
AC := M(3000H)
↑
add. bet^n 0000 to ffef
therefore this is known as Memory mapped IO.
(same set of inst^{n}s are used).
LDA fff0 ← memory mapped IO devi

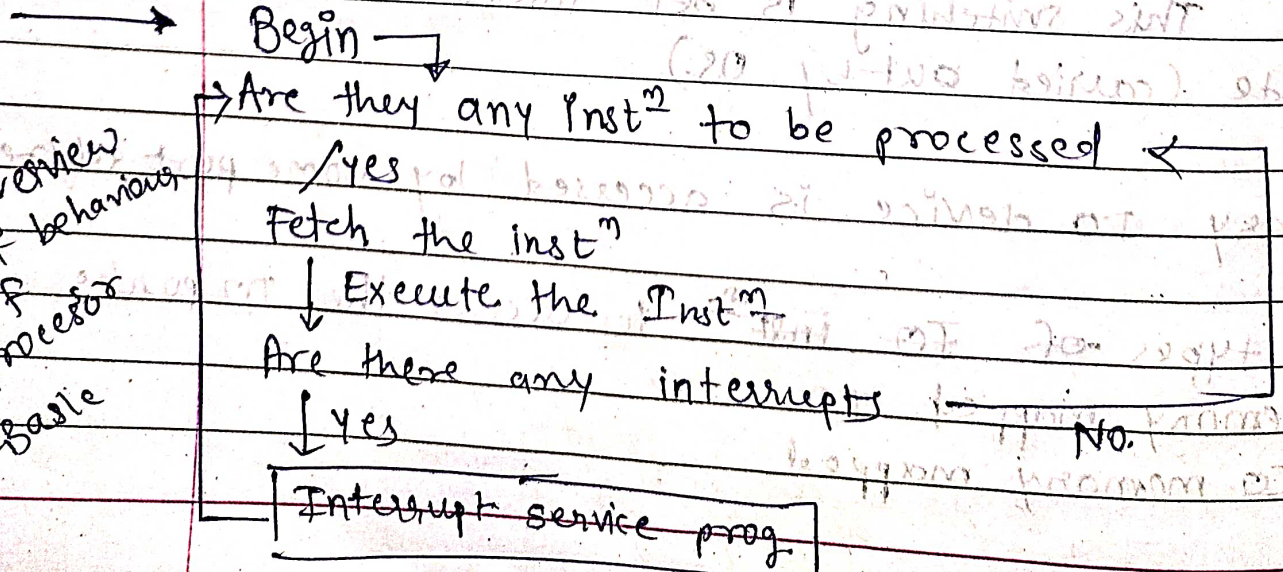


2) IO mapped IO
seperate set of inst^n for memory and IO device.
LDA / STA → for memory add.
IN / OUT → inst^n for IO device

No seperately reserved space based upon types of inst^n it will be differenciated



Overview of behaviour of processor basic

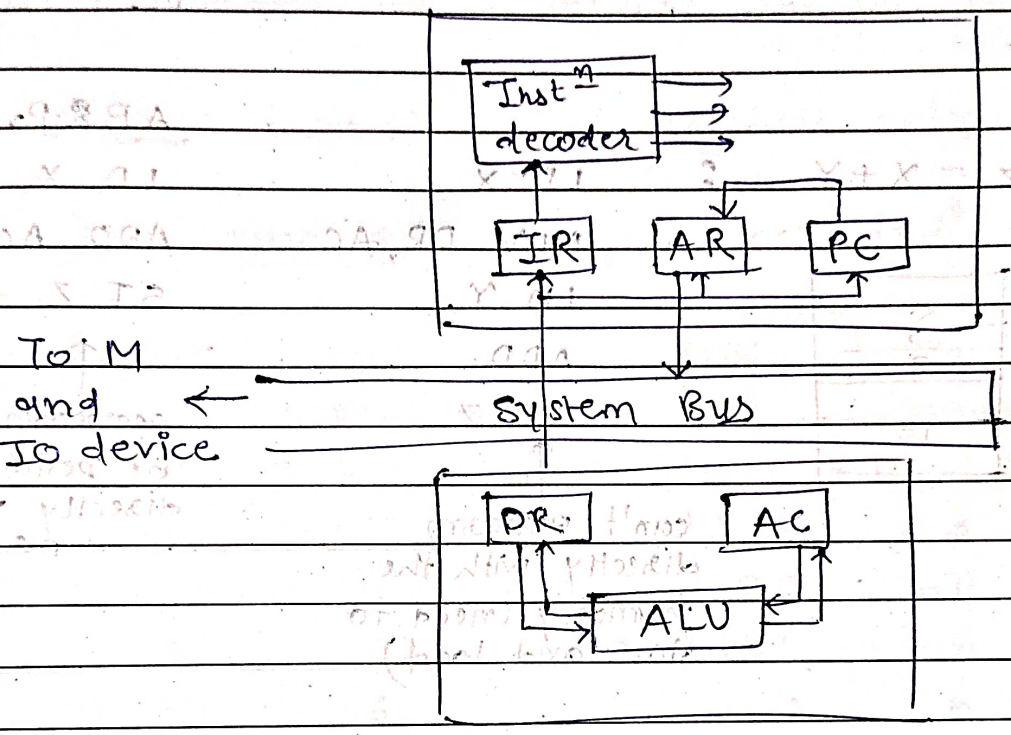


DATE:
PAGE:

→ Hypothetical System which is known as Accumulator based processor

↓

Block diagram:



- Data transfer Inst^n

LOAD AC = M(X) — LD X (add.)

STORE M(X) = AC — ST X (addr)

Move to Reg. DR = AC — MOVE DR, AC

AC = DR — MOVE AC, DC

] 1 bit Inst^n

- Data Processing Inst^n

ADD AC = AC + DR

SUB AC = AC − DR

AND AC = AC and DR

NOT AC = −AC

- Program Control : Branch PC = M(adr.)
  Branch zero If AC = 0 then PC = M(adr)

→ Z = X + Y :

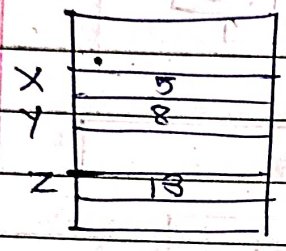
LD	X
MOV	DR, AC
LD	Y
ADD	
ST	Z

↑ can't perform directly with the memory (need to store and load)

ABB Proc.

LD	X
ADD	AC, M(Y)
ST	Z

↑ operation can be perform directly with the memory



→ Data Representation:

- Data can be either fixed point or floating point number. It also numeric or non-numeric no.
  ↑ fixed or floating ↑ using ASCII

- Any Binary no. can be represented by BCD or can be Binary format.

- Weight of 7 in BCD are changing

$2^3\ 2^2\ 2^1\ 2^0$ $2^3\ 2^2\ 2^1\ 2^0$
0 1 0 0 0 1 1 0

while in case of Binary, weight is continue

DATE:
PAGE:

- BCD is not preferable in addition of 2 BCD number because answer which we get can be wrong

→ signed number : (+ve) as well as (-ve)
MSB ~~repre~~ denotes the no. is +ve or -ve.

Onsigned number : +ve numbers only.

→ signed number : Range is bet^n $(-2^{4-1}-1)$ to $(2^{4-1}-1)$

signed Magnitude Rep.

0000 : +0	1000 : -0
0001 : +1	1001 : -1
0010 : +2	1010 : -2
0011 : +3	1011 : -3
0100 : +4	1100 : -4
0101 : +5	1101 : -5
0110 : +6	1110 : -6
0111 : +7	1111 : -7

Range = $-(2^{n-1}-1)$ to $(2^{n-1}-1)$

- For +ve numbers all rep.s remains same (signed magnitude Rep, 1's comp, 2's comp.)

• 1's complement :

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

2's complement :

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

- In signed magnitude and 1's comp. we have two diff. rep. for +0 or -0 so there is comparision bet^n them and we need extra bits (space) for it.
  While in 2's complement range is bet^n $+(2^{n-1}-1)$ to -2^{n-1} [-8 to +7].
  (+7) (-8)