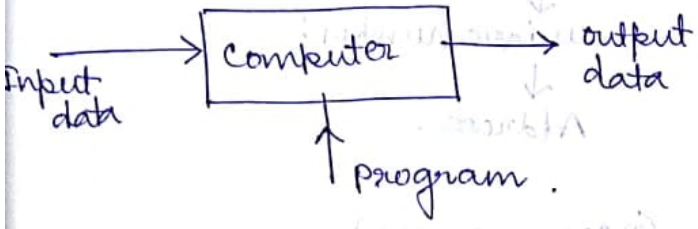
Computer fundamentals

Computer:- It is a computational device used to process the data under the control of a program. By executing the program, data will be processed in the computer system. Therefore computer functionality is program execution.



Program:- It is a sequence of instructions along with a data.

Program — Instruction
data.

Instruction:- It is a binary sequence designed inside the processor to perform some task.

Binary sequence — bind with — operation

Data:- It is a binary sequence which is associated with value based on the data format.

Binary sequence — bind with — value.

Functionality of computer:-

Computer functionality is program execution. Let us consider computer components as a reference to justify the computer functionality.

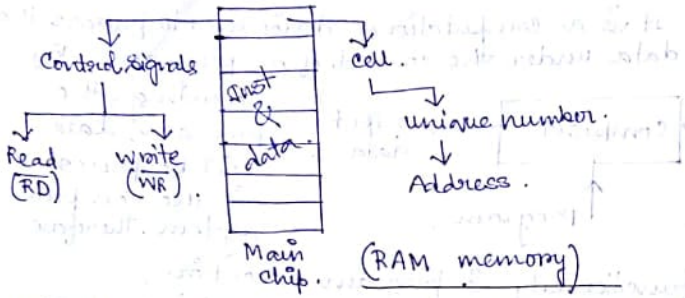
1. CPU (Processing)
2. Memory (storage)
3. IO (external communication).

Memory organisation:-

→ Memory is a storage element in the computer, so program is always stored in the memory.

→ Memory chip is divided into equal parts called as cells. Each cell is identified with unique number called as address.

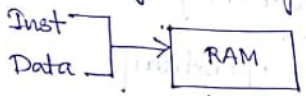
→ Memory chip recognises the control signals to indicate the type of operation.



(RAM memory)

Note :- Processors architecture is divided into 2 types based on the memory name as

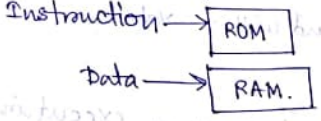
(i) Single memory architecture (Von-Neuman arch)



Ex:- up

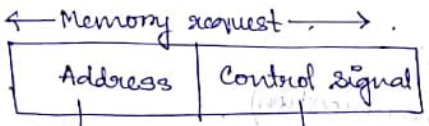
stored prog concept.

(ii) ~~No~~ Multi memory architecture : [Harvard arch]



Ex:- u controller.

→ CPU generates the memory request to access the program from memory chip i.e



Memory chip configuration :-

Number of address bits required to access the memory always depends on a memory capacity. Memory chip config is represented as 64KX8, 64K X 16, 64K X32, ...

64KX8 mem chip. data cell size (No of bits stored in cell).

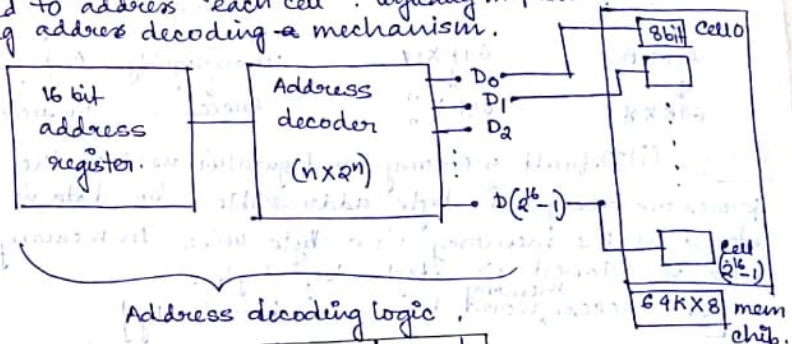
cells in memory chip 64K cells = $2^6 \times 2^{10} = 2^{16}$ cells.

16 bit address.

addr range / address space / mem map of a memory chip.

$\begin{pmatrix} 0000\ 0000\ 0000\ 0000 \\ 1111\ 1111\ 1111\ 1111 \end{pmatrix}_2$ address space.

→ each cell is not given 16 bit address line. Decoder is used to address each cell. logically implemented using address decoding mechanism.



up	Bit	Nibble	Byte	Word
8bit up	1	4	8	8
16bit up	1	4	8	16
nbit up	1	4	8	n

Byte is always having unique meaning. word is having multiple meaning hence ambiguity & unsuccessful.

Byte addressibe vs word addressible memories :-

→ Based on the cell size, memory chip configuration is divided into two types.

i) Byte addressible memory
ii) Word addressible memory.

→ when the memory cell size is 8 bit, then the corresponding address space is called as byte address

Ex: 4X8 : 2 bit address cell holds 8 bit info.
64X8 : 6 bit " " " 8 " "
256X8 : 8 bit " " " 8 " "
2^nX8 : n bit " " " 8 " "

Byte address. (cell size must be 8 bit)

→ when the memory cell size depends on word length of a processor then the corresponding address space is called as word address.

4X8 : 2bit address cell holds 1 word info.
64X8 : 6 bit address
256X8 :
2^nX8 :

word address

Cell size must be word size.

→ In the processor binary data is referred with a following referential elements.

~~Nip~~ ~~Bit~~ Nibble Byte Word

eg. Byte → 64KB → 64KXB → 64KX8.

Word → 64KW → 64KXW → 64KX?.

Byte → univalue meaning → unambiguity → success.

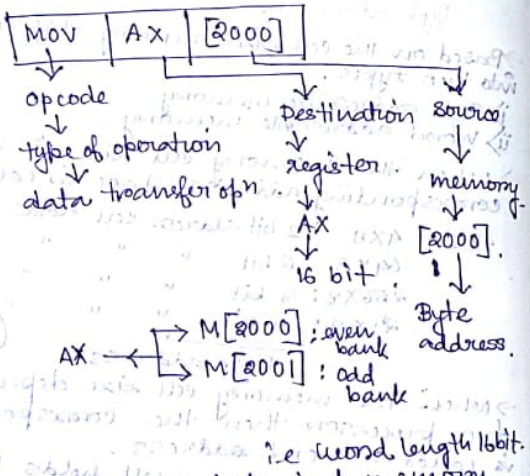
Word → multiple meaning → Ambiguity → Unsuccess.

Note:- (i) Default memory configuration used in the processors design is byte addressible. So data is stored in the memory in a byte wise. In memory data is stored in byte by byte.

[CPU access memory word by word .. parallelly]

(ii) In the processors operations are performed on a word format so when the word length of a processor is >8 bits then multiple call accessing is required in parallel to access the data from memory in a word wise.

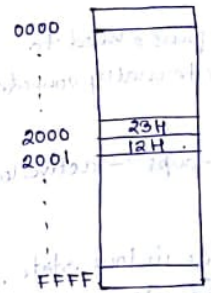
Ex:- In 8086 µp instructions.



i.e word length 16bit.

[In 8086 register contain 16 bit data, but memory addressing is byte address type. Hence 2 memory locations will be accessed parallelly. Interfacing is done to access 2 8 bit call parallelly to get 16 bit word.]

→ During the execution of above instruction, 2 possible outcomes are generated due to lack of order of a data storage in the memory i.e



① If [2000] contain LB then,
AX: 1223H.

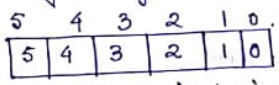
② If [2000] contain HB
AX: 2312 H.

→ In a 32 bit processor word length = 32 hence 4 cells needs to be accessed parallelly hence 2 possible outcomes are possible; Higher order to lower order or lower to higher order.

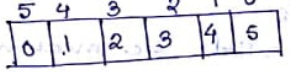
→ To handle the above problem, memory address interpretation mechanism (endian – ness mechanism) are used. These mechanisms shows the order of data storage in the memory. These are 2 kinds.

(i) Little endian
(ii) Big endian

Little endian mechanism shows the order of data storage as lower address contains lower byte & higher address contain higher byte.

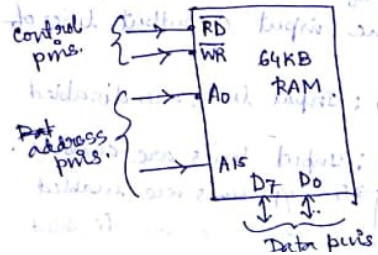


→ In big endian techniave lower address contain higher byte and vice versa. ~~So address at byte~~



Note:-
Default address interpretation mechanism in the processors design is little endian so the above instruction generates the following o/p.

AX = 1223 H.



Address pin, data pin and control pins are available in RAM hence processor should contain same kind of hardware pins for interfacing with memory.

Pin structure :-

Processor consists of a set of hardware pins, used to perform the various operations on a externally connected components i.e memory & I/O.

Hardware pins are categorized into 3 groups :- active low, active high, time multiplexed pin. ~~AT~~

Active low pin :-

This pin is enabled when clock pulse is in low state. Denoted as pinname. eg: $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ etc.

Active high pin :-

This pin is enabled when the clock pulse is in high state. Denoted as pinname eg: ALE, INTR, HOLD, HLDA etc.

Time multiplexed pin :-

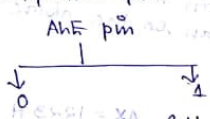
This pin is used to carry the multiple meanings but not at the same time. In the processors design data pins are time multiplexed with address pins so advantage is reduces the number of hardware pins.

eg: In 8085 $AD_0 - AD_7$ bits.

Address — data.

In 8086 $AD_0 - AD_{15}$.

ALE pin is used to differentiate the meaning of a time multiplexed pin.

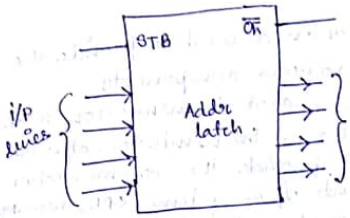


{time multiplexed pins carries data} {time multiplexed pin carries the address}.

Address latch :- It is a temporary storage comp used to hold the address. It consists of 2 control pins (STB & $\overline{OE}$) used to control the input & output lines of the latch respectively

STB (Strobe pin) → 0 : Input lines are disabled
→ 1 : Input lines are enabled.

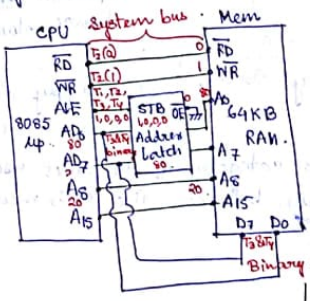
$\overline{OE}$ (output enable pin) → 0 : o/p lines are enabled
→ 1 : o/p lines are disabled.



7.4.16

Memory Interfacing

This concept is used to integrate the CPU and memory into a one component by providing the mapping between CPU pins & memory pins. let us consider 8085 processor as a reference CPU to analyse memory interfacing. It is a 8 bit microprocessor supports 64KB physical memory.



STB line 1
i/p lines of latch enabled.

Machine instr : LDA [2080]

Acc ← M[2080]

- CPU generates the mem readreset

2080 H	$\overline{RD}$

Machine cycle
{# CPU cycles required to complete the mem / IO opn}
⇒ [T_1. T_2 T_3 T_4].

Actions

① Send the addr
T_1 : ALE = 1
8085 ⇒ $AD_7 - AD_0$
8085 ⇒ $A_{15} - A_8$.

② Send the 'CS'
T_2 : ALE = 0
$\overline{RD}$ = 0
$\overline{WR}$ = 1

③ Read the data
T_3 & T_4 : ALE = 0
{mem delay} $D_7 - D_0$ ⇒ $AD_7 - AD_0$.

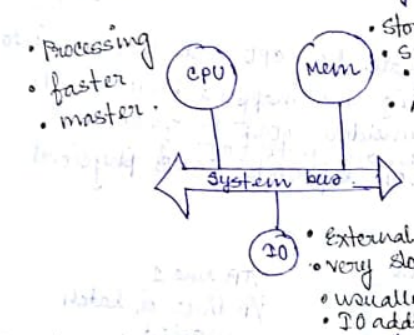
BUS :-

(i) BUS is a communication channel used to provide the communication between various components.

→ Characteristic of a bus is shared transmission media

→ limitation of a bus is only one transmission at a time

→ A bus which is used to provide the communication between major components of a system (CPU, memory and I/O) is called as system bus.



IO devices are not always slave. Power supply is acting like master.

CS → Control signal.

→ System bus consists of 3 category of the lines used to perform the communication between the CPU, memory and I/O. name as

1. Address lines.
2. Data lines.
3. Control lines.

Address lines :- These lines are used to carry the address to memory & I/O. so address lines are unidirectional. Based on the address lines, we can determine the capacity of memory system.

ex :- In 8085 $AD_0 - AD_7$, $A_8 - A_{15}$. Total 16 address lines we have. 2^{16} cells of memory

= 64 KB memory. 64K memory cell.

ex :- In 8086

$AD_0 - AD_{15}$ $A_{16} - A_{19}$

↓

20 address lines

↳ 2^{20} cells → 1K × 1K cells → 1M cells → 1MB.

Data lines :-

These lines are used to carry the binary sequence between the CPU, memory & I/O. So data lines are bi-directional. Based on the data lines we can determine the word length of a processor. Based on word length we can determine the performance of a processor.

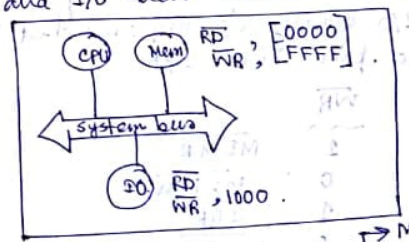
ex: In 8085 $AD_0 - AD_7$ so word length is 8 bit. so operations are performed on a 8-bit data format.

Similarly in 8086 $AD_0 - AD_{15}$ so word length is equal to 16 bit so operations are performed on 16 bit data format.

Control lines :-

These lines are used to carry the control signals & timing signals. These are unidirectional. Control signals indicates the type of operation and timing signals are used to synchronise the memory and I/O operations with a processor clock.

Note when there is a common bus, common control signal and common address space is used between memory and I/O then there is a possibility of ambiguity



CPU generates the mem request

1000H | RD

1000 → A's

RD → C's

→ Mem Controller. 1000 – Valid, RD – Valid } Mem operation

→ IO controller. 1000 – Valid, RD – Valid } IO opn

→ To handle the above problem bus configurations are used in the computer design.

BUS configuration

	IOP	Isolated I/O	MEM-Map
COMMON BUS	×	✓	✓
COMMON cs (control signal)	✓	×	✓
COMMON Address space.	✓	✓	×

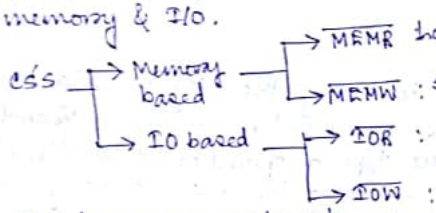
→ There are 3 different bus config used in the system design ma name as i) IOP [Input – output – processor]

ii) Isolated – IO [IO – mapped IO]

iii) Memory mapped IO.

DOP – This configuration uses the common control signals and common address space but different buses for both memory and I/O. Additional control mechanism is required to manage I/O bus and memory bus so it is expensive. Therefore it is suitable in the high performanced processors design.

Isolated I/O – This configuration uses the common bus and common address space but different control signals for both memory & I/O.



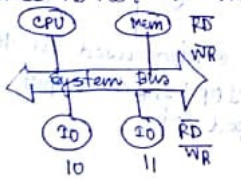
when the processor supports In out instructions then we can conclude that isolated I/O config used in the system design. 8085 and 8086 both ~~of are~~ the processors ~~is~~ use the isolated I/O config.

→ In 8085, isolated I/O config is implemented using the IO/$\overline{M}$ hardware pin. i.e ~~IO/M~~

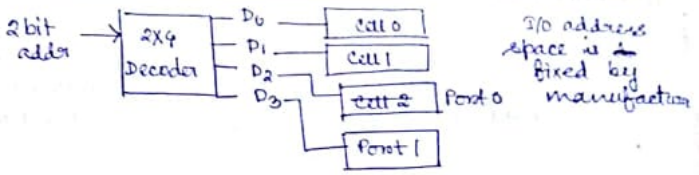
IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	
0	0	1	: $\overline{MEMR}$
0	1	0	: $\overline{MEMW}$
1	0	1	: $\overline{IOR}$
1	1	0	: $\overline{IOW}$

Memory mapped I/O :-

This config uses the common bus and common control sig but unique address space. Unique address space is maintained by taking some of the addresses from memory space and assign them to the I/O ports. Disable the same address in the memory space. In this config I/O address become the part of a memory address space so limitation is the complete memory space is not in use.



00
01
10
11
X
X
These are disabled for memory & reserved for I/O.



(5) Consider a hypothetical processors which supports ~~256 M~~ 256 M word memory. Processor uses the memory mapped I/O configuration. In which when the 3 MSB bits of address is 1 then assign them to I/O ports. How many ~~as~~ I/O addresses and memory address are possible in the processors?

Sol MM size = 256 M word.

= 256 M X W

= 2^{28} cells.

↓

28 bit address.

Addr :-

A_{27}	A_{26}	A_{25}	A_{24} A_{23} ..... A_0
0	0	0	25 bits.
0	0	1	
0	1	0	
0	1	1	Memory.
0	0	0	
1	0	1	
1	1	0	
1	1	1	I/O.

IO addresses = 1 * 2^{25}.

1st 3bits, remaining 25 bits.

(111)(000...)25 zeros

↓

(111)(111....)25 1's.

3bit 25 bit.

Memory address = 7 * 2^{25}.

Total space = IO space + Mem space.

= $(1 * 2^{25}) + (7 * 2^{25})$.

= $(1+7) * 2^{25}$

= $8 * 2^{25}$

= 2^{28}.

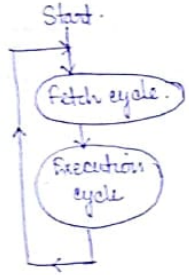
Instruction cycle :-

→ This concept describes the execution sequence of a program. It consists of ~~2~~ 2 subcycles name as fetch cycle and execution cycle.

→ Sequence diagram is

P.T.O

~~HALT instruction~~



Note → HALT instruction is used as a machine control instruction, to indicate the end point of a program.

→ This instruction invokes the unconditional jump with starting address ~~as a target address~~ of HALT as a target address.

HALT: I4: Jmp I4.

Prog: 1000: I1
1001: I2
1002: I3
1003: I4 : (HALT)
1004: I5
1005: I6

→ HALT instruction working nature is interpreted in 2 perspectives. User point of view HALT indicates end of the program execution. CPU point of view HALT means execute the same instruction upto an infinite number of times.

PC: 1000 [I1] ⇒ executed
: 1001 [I2] ⇒ executed.
: 1002 [I3] ⇒ executed
: 1003 [HALT] ⇒ execution ⇒ [JMP | 1003] (target address)
unconditional jump
~~1004~~ 1003 [Halt] ⇒ [JMP : 1003] PC : ~~1004~~ [1003].
~~1004~~ 1003.
PC: ~~1004~~ : 1003.

Fetch Cycle.

→ Objective of a fetch cycle is instruction fetch. In this process CPU reads the instruction from the memory based on a PC.

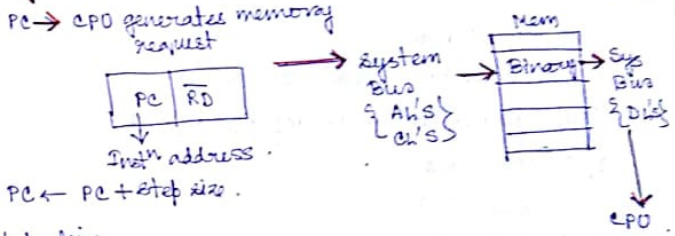
→ PC is a mandatory register in the CPU design used to hold the instruction address. So CPU executes the program based on a program counter.

→ Starting address of a program is supplied by the programmer along with an executable command. and the next instruction address is calculated by incrementing the PC with a word length of the processor. Finally at the end of fetch cycle, PC holds the address of a next instruction during the execution of current instruction.

Instⁿ fetch

- t = Starting address of a prog. ; executable statement

t → trace
execute prog instn by instrn.
a → go. AD execution executes at a time upto halt



Dh → data line
Ah → Address line
Ch → Control line.

(5) Consider a 32-bit hypothetical processor used to execute the following programme segment.

Instruction	Size (words)
I1	2
I2	1
I3	1
I4	2
I5	1
I6	2
I7.	1

(a). Assume that program is stored in the memory with a starting address of 1000 decimal onwards. what would be the value present in PC during execution of a I6 instruction.

I1: 1000–1007
I2: 1008–1011
I3: 1012–1015
I4: 1016–1023
I5: 1024–1027
I6: 1028–1031
I7: 1032–1035

By default memory config is byte addressible storage.
word size = 32 bit.
4 bytes hence 1 word instruction takes 4 cells of memory.

valid PC.

PC contains 1032.

(b) assume memory is word addressible and the program is stored in memory with starting add of 732 decimal onwards. what could be the value present in PC during execution of I6.

I1	2	I1:	732–733.
I2	1	I2:	734
I3	1	I3:	735
I4	2	I4:	736–737
I5	1	I5:	738
I6	2	I6:	739–740
I7.	1.	I7:	741 ← valid PC.

742.

Word addressible means each cell contain 1 word ie 32 bits.

∴ PC holds 741.

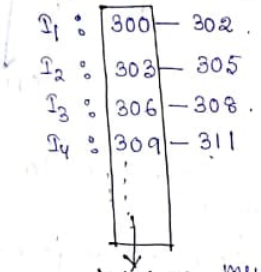
(8) A CPU has 24 bit instructions. Program is loaded in the memory with a starting address of 300 decimal onwards. Which one of the following is a valid program counter?

a) 400 b) 500 c) 600 d) 700

24 bit = 3 bytes

Every instruction needs 3 cells space.

Byte Addressible memory by default.



Valid PC. multiplicative factor of 3.
Hence 600 becomes valid PC.

Execution Cycle :-

Objective of an execution cycle is processing of a currently fetched instruction. Opcode is required to process the instruction.

→ Opcode is a part of instruction, specified by the instruction format.

→ Instruction format describes the layout design (internal structure of an instruction)

Note :- There is no common instruction format used in the CPU design so let us consider accumulator CPU as a reference model to take the instruction format.

In this CPU, first ~~operator~~ operand is always required in accumulator & 2nd operand is present either in register or in memory. After the data manipulation result is always placed in accumulator.

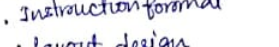
→ In the processor design only 1 accumulator is present so it become a default location so this address is not required in the instruction.

→ Compatible instruction format of an accumulator CPU is

execution cycle

- Opcode
- Instruction format
- layout design
- Acc – CPU

Only 1 memory register address is needed in acc computer, to manipulate the data.



: 1- address instruction format.

Type of operation

Data transfer operation : Source (memory / Acc) Destination (Acc / Mem).

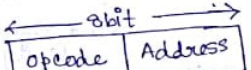
Data manipulation operation : Source1 (Acc) Source2 (Reg/mem) Dest^n (Acc).

Transfer of control op^n : Target address (memory address).

Instruction execution sequence :-

To analyze the execution flow of an instruction, let us consider accumulator machine as a reference CPU. It supports 8-bit instruction with 8 different operations so the associated instruction format is.

① 8bit



log$_2$ 8 = 3bit 5bit.

Instruction format is predefined in instruction register.

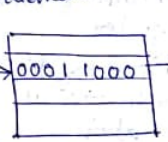
PC gives instruction address hence instr fetch.

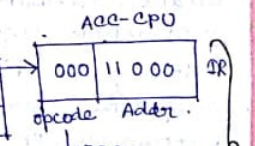
IR gives data address hence operand fetch.

→ = Starting address of a prog. : executable statement

PC → CPU generates mem read

PC ← PC + step size.

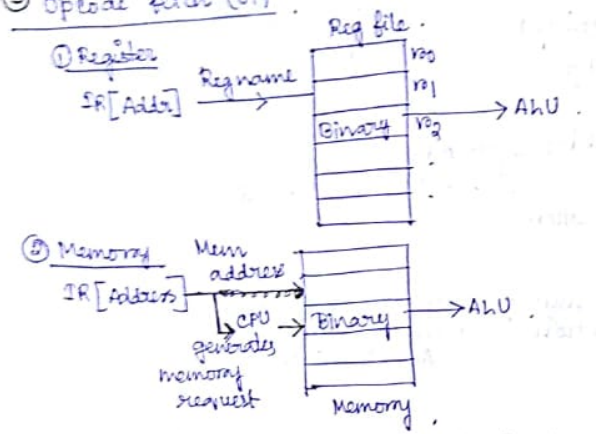




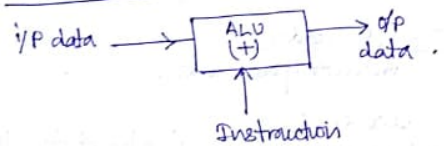
② Design time binding.

encoded binary	Decoded signal	Meaning
000	D_0	+
001	D_1	*
010	D_2	-
111	D_7	/

③ Opcode fetch (OF)



④ Process data (PD) & write Back (WB)



⑤ ex: 8-bit ALU

Instruction: ADD R0

↓

Acc ← Acc + R0.

Acc: 10001100

R0: 11001110

① 01011010

↓

extra register (PSW).

→ CPU reads the instruction from memory based on a PC called as instruction fetch.

→ IR is a mandatory register in the CPU design used to hold the currently fetched instruction to decode because instruction format is predefined in the instruction register.

→ During the decoding process of an instruction the respective hardware is enabled in the system to perform the operation.

→ Based on the type of a CPU and the associated addressing modes of an instruction, data will be accessed either from register or from memory called as operand fetch.

→ Input data is processed based on the enabled hardware called as process data. later result is placed into a destination called as write back.

→ In the system design, extra storage is provided to hold the status of an instruction during the execution called as PSW or flag register.

PSW, Addressing mode, Instruction set] → Refer the Microprocessor

→ Instruction cycle along with interrupts:-

Interrupt is a signal which is generated by the low speed components. This concept describes the execution sequence of a program along with interrupts.

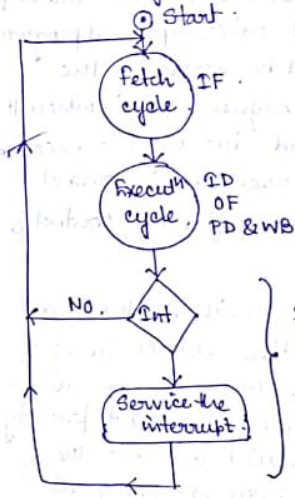
It consists of 3 subcycles

① → Fetch cycle

② → Execution cycle

③ → Interrupt cycle.

Sequence diagram is



→ Sequence diagram states that, CPU will respond to interrupt only after the completion of a current instruction execution.

→ During the program execution, after completion of every instruction, CPU reads the status of interrupt pins to detect the interrupt.

ex:- INTR pin

INTR pin
- 0 (NO): INT is not present ⇒ go to the next "instruction fetch" from the user prog
- 1 (yes): INT is present ⇒ goto int. service process

→ When the CPU detect the interrupt, then it saves the user prog context (PC, PSW & some important register content) into a stack. later transfer the control to interrupt vector table (IVT).

→ IVT is a part of the memory where the different interrupt subprogram's information is present.

→ Interrupt subprograms are ~~subprogrammed with~~ preferred with Vector address & end with IRET/RETI instruction.

→ During execution of interrupt subprograms, when the CPU encounters the IRET instruction, then it invokes POP operation to restore the program context into the registers. Therefore, after servicing the interrupt, control will be transferred to a user program.

Note: Objective of an interrupt cycle is interrupt subprogram initialisation. In this process, PC value is saved into a stack, later vectored address is loading into a PC.

→ There are 2 approaches present to service the interrupt
- (i) Single interrupt processing approach.
- (ii) Nested interrupt " "

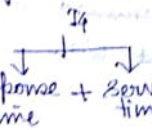
In the first approach, CPU disables the further interrupts during the execution of a current interrupt subprogram.
In the second approach CPU will be servicing the interrupts based on the priority assignment i.e when the high priority interrupt is occured during the execution of current interrupt then it saves ~~the~~ correct interrupt status into stack and transfers the control to high priority interrupt.

(8) Consider a hypothetical processor which supports 4 interrupts I_1, I_2, I_3, I_4 with respective service times of 20, 30, 15, 10 ns. Response time of an interrupt is 5 ns. I_1 has the highest priority & I_4 has the least priority. What is the range of time reqruried to service the I_4 interrupt when the interrupts may or may not occur simultaneously.

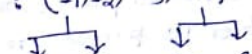
Delay between request line and acknowledgement line is response time.

a) minimum time reqruried for I_4 {without simultaneous occurance}

Only one INT at a time



Max time time reqruried for I_4 : (I_1, I_2, I_3, I_4).
{with simultaneous occurance}.



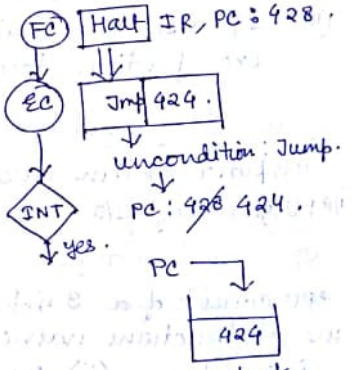
{15 ns to 95 ns}. : 95 ns.

(8) Consider a 32-bit hypothetical processor used to execute the following program segment. Program is stored in the memory with a starting addr 400 decimal onwards. If an interrupt occurs during execution of a HLT instruction, what could be the written address pushed into the stack?.

Inst	Size in words
I_1	2
I_2	1
I_3	1
I_4	2
I_5 (Halt)	1

Prog
400 — 407.
408 — 411.
412 — 415
416 — 423.
424 — 427.
428 — valid PC.

32-bit → 4 byte.
Stack content will be 424.



Only JMP instruction changes the PC.
If there was ADD instruction then the stack content would be 428. But in case of HLT unconditional JMP to 424 address location. Hence while executing interrupts HLT location will be stored in stack.

***

RISC (Vs) CISC:-

Characteristic of RISC:- (Reduced Instruction Set Comp).

1. It supports more number of registers.
2. It " less " " addressing modes.
3. " " fixed length instructions.
4. " " one instruction per cycle (CPI = 1).
5. " " successful pipeline (CPI = 1).
6. " " smaller instruction set.
7. It is a supercomputer.
8. It is used in real time application.
9. It is a high expensive processor.
10. It uses hardwired control unit.

example :- Motorolla processor
Power pc processor
ARM " (Advanced RISC machine)

Characteristics of CISC (Complex Instruction set comp)

① It supports less no of registers.
② " " more no of addressing mode.
③ " " Variable length instruction
④ " " no one instruction per cycle (CPI ≠ 1)
⑤ " " unsuccessful pipeline (CPI ≠ 1)
⑥ " " larger instruction set.
⑦ It is a general purpose computer.
⑧ It is used in general purpose applications.
⑨ It is a less expensive processor.
⑩ It uses the microprogrammed control unit
ex:- pentium processor.

Components of computer system :-

Computer system consists of 3 fundamental comp viz
i) CPU, ii) memory, iii) I/O.

CPU organisation :-

CPU consists of 3 internal components used to execute the instructions name as
(i) Register (ii) ALU (iii) Control unit.

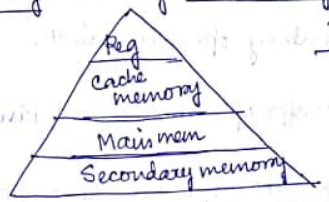
Register :- It is a temporary storage component used to hold the frequently referred data.

→ In the computer system, program is always present in a memory so frequent communication takes place between CPU & memory during program execution.

→ Memory is a slow component so processor performance degrades if there is no proper balancing. Therefore memory hirerchy design is used as a synchronisation mechanism in computer design to minimise the speed gap between CPU & memory.

→ This design organises system supported memories into 4 levels i.e.

Memory Hirerchy Hierarchy design.



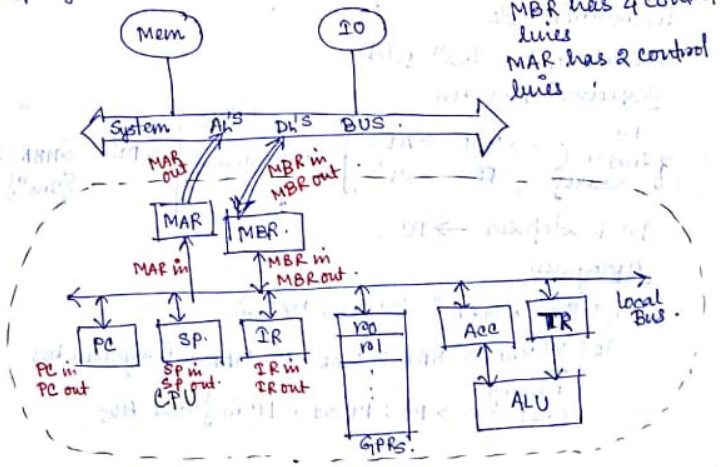
Bottom to top approach
① Capacity ↓
② Access time ↓ (AT)
③ performance ↑
④ cost/bit ↑.
⑤ expensive memory.

Performance $\propto \frac{1}{P}$ $\frac{1}{ET}$ ET ∝ AT ET → execution time.

$P \propto \frac{1}{AT}$

→ In the computer system, instruction is always present in the memory. During it's execution, data becomes most important element therefore keep the smallest storage component (register) in the CPU used to hold the frequently referred data. Therefore every processor contain a minimum set of mandatory registers. They are

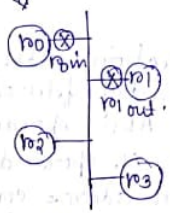
① PC : Holds the instruction address
② IR : Holds the currently fetched instruction
③ Acc : Holds the 1st ALU operand & ALU o/p
④ TR : Holds the 2nd ALU operand
⑤ MAR : " " mem address connected to address line of system bus.
⑥ MDR/MBR : Holds a memory content connected to data lines of system bus.
⑦ GPR : General purpose register, holds the frequently referred data.
⑧ PSW : holds status of an inst^n.
⑨ SP : " addr the top of stack address.



Micro-operation :-

→ Micro-operation is a elementary operation in the hardware.
→ Register to register transfer operation is one kind of microoperation.
→ Micro operation on average taking 1 cycle to complete the execution (when the microoperation contain memory reference then it takes more than 1 cycle to complete the execution).
→ Control signals are required to execute the micro operation.

Machine Inst: MOV R0, R1
⇓
Reg. transfer language: R0 ← R1.
⇓
Hardware design:



⇓
Mopn: T1: R0 in, R1 out.
control signals.

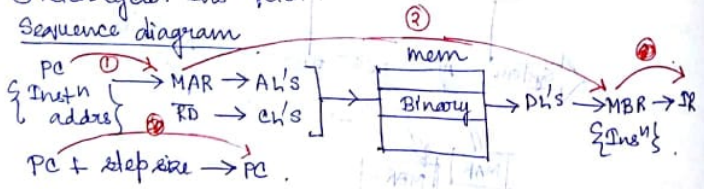
Microprogram :-

The sequence of a microoperation to perform some task is called as microprogram.

Instruction cycle

① Fetch cycle: Instn fetch.

Sequence diagram



Uprogram

T1: PC → MAR; PCout, MAR in.
T2: M[MAR] → MBR; MAR out, MBR in { System bus.
memory
PC+1 → PC; PCout, PC in { local Bus.

T3: MBR → IR; MBRout, IR in

In T2 PC needs to be incremented else in T3 MBR out will be in system bus so if PC out in is incremented in T3 PCout comes to system bus with MBRout. Hence there will be collision. PC cannot be increased in T4 but then 1 cycle will be wasted.

② Execution cycle :-

(a) ID stage: Enables the H/w to perform the operation.

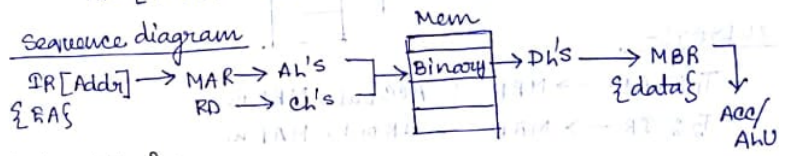
(b) OF stage: operand fetch. AM's are required to fetch the data.

(i) Direct AM — Addressing modes.

Instn: | Opcode | Addr | → EA.

Ex: Load [2000]; Acc ← M[2000]
ADD [2000]; Acc ← Acc + M[2000].

Sequence diagram



Uprogram :-

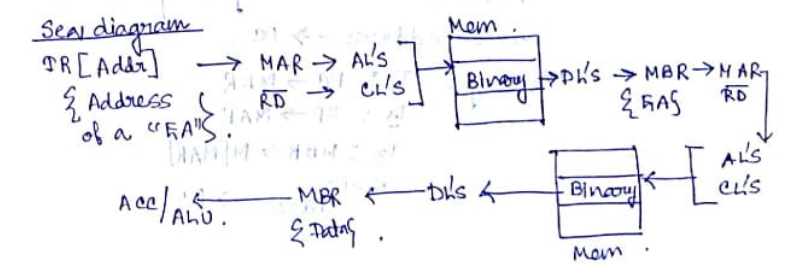
T1: IR[Addr] → MAR; IR out, MAR in
T2: M[MAR] → MBR; MAR out; MBR in } Sys Bus.
mem
T3: MBR → Acc/ALU; MBR out; Acc in / ALU in.

(ii) Indirect AM.

Instruction: | Opcode | Address | → Addr of a 'EA' → effective address

Ex: LOAD @ 2000; Acc ← M[[2000]].
ADD @ 2000; Acc ← Acc + M[[2000]].

Seq diagram



µprog

T_1: IR [Addr] → MAR

T_2: M[MAR] → MBR

T_3: MBR → MAR

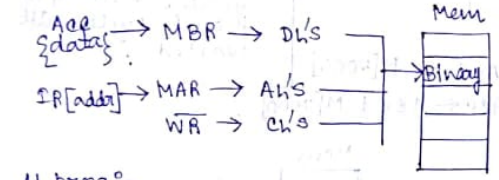
T_4: M[MAR] → MBR

T_5: MBR → ACC/ALU.

④ PD & WB stage: operand store.

Direct AM:-

ex:- store [2000] ; M[2000] ← ACC.



µprog:-

T_1: ACC → MBR ; ACC out, MBR in

T_2: IR → MAR ; IR out, MAR in.

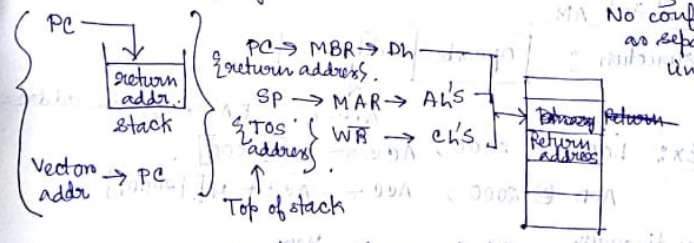
T_3: MBR → M[MAR] ; MBR out, MAR out.

mem

{ both will be out as AI's are connected to MAR & data lines connected to MBR } No conflict as separate lines.

⑤ Interrupt cycle:-

Interrupt subprog initialisation.



Vector address → PC.

µprog: T_1: PC → MBR

T_2: SP → MAR

T_3: MBR → M[MAR]

Vector address → PC.

Control unit design:-

→ Control signals are directly executed on a base hardware so hardware generates the desired output.

→ control unit generates the sequence of control signals.

→ control signals are implemented in the control unit.

→ control signals are required to execute the micro-operation.

→ Micro operation is an elementary operation in the hardware.

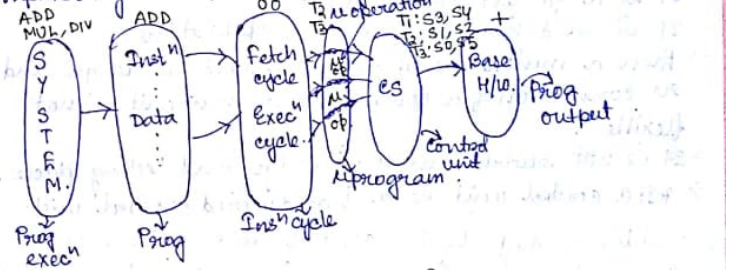
→ Microprogram is a sequence of a micro operations.

→ Sub cycles of an instruction cycle invokes their microprograms.

→ Instruction cycle consists of 2 cycles, fetch cycle and execution cycle.

→ Program execution sequence is described by the instruction cycle.

→ Computer system functionality is a program execution.



Pre-requirement of the CU design is:-

1) How many control lines are recognised by the hardware.

2) How many instructions are implemented in the hardware.

3) How many micro operations are required for each instruction.

4) What are the control signals required for each micro operation for each instruction.

ex:- ① # CS's in the H/W. {S0, S1, S2, S3, S4, S5}

② # instruct^n " " " : {I_1, I_2, I_3}

ADD (00) MUL (01) DIV (10).

③ # µop^n/Inst^n : {T_1, T_2, T_3}.

P.T.O

④

μop^n/Inst^n	I_1	I_2	I_3
T_1	S_3, S_4	S_1, S_0	S_1, S_3
T_2	S_1, S_2	S_2, S_4	S_2, S_4.
T_3	S_0, S_5	S_3, S_5	S_0, S_5

⇒ CU ⇒ CPU ⇒ Comp ⇒ Deploys.

Control signal data base is implemented into a control unit by using the following approaches

1) hardwired approach
2) Micro programmed approach.

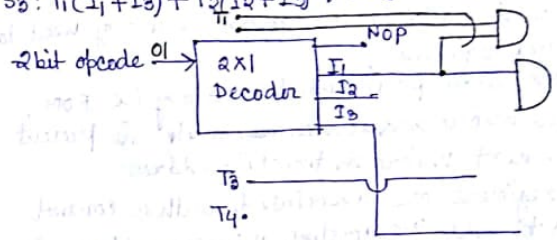
Hardwired control unit :-

→ In this design control signal is expressed in a sum of product expression format. This control expression is directly realised by the independent hardware.
→ It is a fastest control unit.
→ It is used in the real time applications.
→ Even a minor modification requires re-design and re connection of a control signal hence it is not flexible.
→ It is not suitable in the design and testing places.
→ RISC control unit is a hardwired control unit.

Consider a hypothetical control unit which supports 4 control signals (S_0, S_1, S_2, S_3) and 3 instructions (I_1, I_2, I_3). Each instruction takes 4 μoperations (T_1, T_2, T_3, T_4) to complete the execution. Following table shows the control signals required for each μoperation, for each instruction. Get the control expression for S_0, S_1, S_2, S_3 signals.

μop^n/Inst^n	I_1	I_2	I_3
T_1	S_0, S_2, S_3	S_1, S_0, S_2	$S_1, S_2 S_3$
T_2	S_0, S_1, S_2	S_1, S_2, S_3.	S_2, S_1, S_3.
T_3	S_1, S_0, S_3	S_1, S_3	S_2, S_0.
T_4	S_0, S_1, S_2	S_1, S_2	$S_2, S_0 S_3$.

S_0 : $T_1(I_1+I_2) + T_2 I_1 + T_3(I_1+I_3) + T_4(I_1+I_3)$.
S_1 : $T_2(I_2+I_3) + T_2(I_1+I_2+I_3) + T_3(I_1+I_2) + T_4(I_1+I_3)$.
S_2 : $T_1+T_2+T_3(I_3)+T_4$.
S_3 : $T_1(I_1+I_3) + T_2(I_2+I_3) + T_3(I_1+I_2) + T_4(I_3)$.



For S0 signal → o/p of OR gate is S_0 signal.
T_1 is req in all T_1, T_2, T_3, T_4.
T_1 is 1 on all $T_1 = 1$, $T_2 = 1$, $T_3 = 1$, $T_4 = 1$ for $I_1 T_1$, $I_1 T_2$, $I_1 T_3$, $I_1 T_4$.
1 → OR o/p is 1 in all the μ operation T_1, T_2, T_3, T_4.

For I_2 it is req in only in T_1 $I_1 T_1 \rightarrow 1$
but for T_2, T_3, T_4 μoperation NO ($T_2 I_2$) or ($T_3 I_2$) or ($T_4 I_2$)

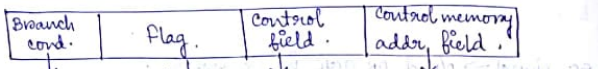
∴ o/p of OR gate is only high for T_1 μop only.

Micro programmed CU :-

ROM contains microprogram. User program is available in RAM.

→ In this design, control unit is programmed with the microprograms.
→ This control unit contains the control memory used to store the microprograms.
→ Control memory is a permanent memory i.e ROM.
→ In this design micro sequencer ~~are~~ unit is present to generate the next micro instruction address.
→ CAR & CDR registers are associated with a control memory used to hold the control memory address & control memory content respectively.
→ In the control memory, micro-instruction is stored in a following format.

← μ instruction/ control word format →



- Branch cond. → one of a system supported branch condition. → $\log_2$ (# BC's)
  Ex: 4 BC's

Name	Binary
C	00
NC	01
Z	10
NZ	11

- Flag. → one of a system supported flags. → $\log_2$ (# flags)
  Ex: 4 flags

Name	Binary
Carry	00
Zero	01
Sign	10
overflow	11

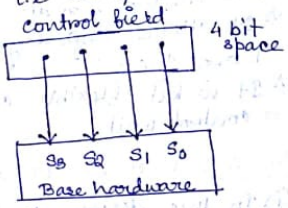
- Control field → Control signals.
  - Decoded binary. → 1 bit/control signals → Horizontal μ instruⁿ.
  - Encoded binary. → $\log_2 N$ bits for N c/s → Vertical μ instruⁿ
- Control memory addr field → Next μ instruction address → $\log_2$ CM size.

SOP, decoded, encoded formatted signals ← control signals.
Based on the way of representing control signals, micro instruction is divided into 2 kinds —
i) Horizontal micro instruction [Decoded form of CS]
ii) Vertical micro instruction [encoded form of CS].

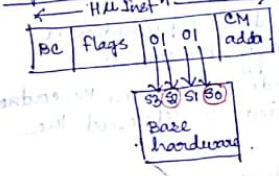
Based on the kind of micro instruction programmed in the control memory, control unit is divided into two types.
1. Horizontal microprogrammed control unit.
2. Vertical microprogrammed control unit.

Horizontal programming :-

CS's in the H/W : {S0, S1, S2, S3}.
Decoded form of a control signal i.e 1 bit/control signal.
{0 — disable, 1 — enable}



Horizontal μ instruction design with {S0, S2} signals.



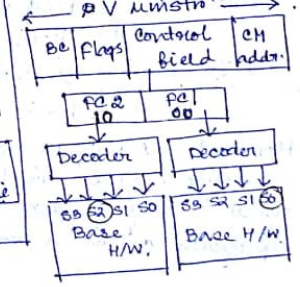
Vertical Programming :-

CS's in the H/W : {S0, S1, S2, S3}.
- encoded form of a control signal : $\log_2$ bit
  i.e $\log_2 N$ bits.
  ⇒ 2 bit.

00	S_0
01	S_1
10	S_2
11	S_3

Control field → 2-bit → decoder → S3 S2 S1 S0 Base Hardware

Vertical μ Instⁿ design with {S0, S2} signals
{function codes are used in the control field to enable more than 1 signal at a time}



Horizontal microprogrammed CU :-

(i) In this unit, control signal is expressed in a decoded binary format i.e 1bit/cs.
ex:- when the hardwire contain 100 control lines then 100 bits are required to represent 100 signals.
(ii) It supports longer control word.
(iii) There is no need of external decoders to generate the control signals hence it is faster than vertical.
(iv) It allows high degree of parallelism i.e none or more than 1.
(v) It is bit flexible compared with hardwire control unit.

Vertical microprogrammed CU :-

(i) In this design, control signal is expressed in a encoded binary format i.e $\log_2 N$ bit format.
ex:- when the CPU hardwire contain 100 control lines then 7-bit code is required to represent the control signal.
(ii) It supports shorter control word.
(iii) There is a need of external decoders to generate the control signals so it is slower than horizontal.
(iv) It allows low degree of parallelism i.e none or 1.
(v) It allows easy implementation of new instructions.
(vi) So it is more flexible.

Note :- Ascending order of a control unit design
(a) Interms of speed is;
Vertical < Horizontal < hardwired.
(b) In terms of flexibility is
hardwired < horizontal < vertical.

Horizontal μprogrammed control unit is not in the practice because directly working with decoding signal is very complex task therefore default μprogram CU is a vertical CU.

3/4/16

(i) HCW

BC	Flag	Control field	CM addr.
$\log_2 16$ = 4 bits		1 bit/CS = 48 bit	$\log_2 1024$ = 10 bits

HCW size = 62 bits
HCM size = 1024 CW
$= \frac{1024 \times 62}{8}$ bytes.

(ii) Vertical programming

VCW

BC	Flag	Control field	CM addr.
$\log_2 16$ = 4 bits	$\log_2 48$ = 6 bits		$\log_2 1024$ = 10 bits

NCW size = 20 bits
NCM size = 1024 CW.
$= 1024 \times 20$ bits
$= \frac{1024 \times 20}{8}$ bytes.

5. A CU supports 6 groups of mutually exclusive control signals are

Group	G_1	G_2	G_3	G_4	G_5	G_6
CS	1	3	5	8	20	30

How many bits are saved using vertical over horizontal programming.

~~How many bits as~~

Ans→

BC	Flags	Control field	CM address

G_1 G_2 G_3 G_4 G_5 G_6.
H V. H V.

Max bits req (Horizontal) = 1+3+5+8+20+30 = 67 bits.
Min bits req = $\log_2 1 + \log_2 3 + \log_2 5 + \log_2 8 + \log_2 20 + \log_2 30$.
$= 1+2+3+3+5+5 = 19$ bits.
∴ No of bits saved = 67 − 19 = 48 bits.

(8) A microprogrammed CU supports 256 control signals Micro instruction is designed such that 3 signals are active. what is the size of control field in micro instruction?
Ans:- default μprogrammed CU ≡ vertical.
CSs = 256
bits per CS = $\log_2 256 = 8$ bits.

Horizontal microprogrammed CU :-

(i) In this unit, control signal is expressed in a decoded binary format i.e 1 bit/cs.
ex:- when the hardwire contain 100 control lines then 100 bits are required to represent 100 signals.
(ii) It supports longer control word.
(iii) There is no need of external decoders to generate the control signals hence it is faster than vertical.
(iv) It allows high degree of parallelism i.e none or more than 1.
(v) It is bit flexible compared with hardwire control unit.

Vertical microprogrammed CU :-

(i) In this design, control signal is expressed in a encoded binary format i.e $\log_2 n$ bit format.
ex:- when the CPU hardwire contain 100 control lines then 7 bit code is required to represent the control signal.
(ii) It supports shorter control word.
(iii) There is a need of external decoders to generate the control signals so it is slower than horizontal.
(iv) It allows low degree of parallelism i.e none or 1.
(v) It allows easy implementation of new instruction.
(vi) so it is more flexible.

Note :- Ascending order of a control unit design
(a) Interms of speed is;
Vertical < Horizontal < hardwired.
(b) In terms of flexibility is
hardwired < horizontal < vertical.

Horizontal μprogrammed control unit is not in the practice because directly working with decoding signal is very complex task therefore default μprogram CU is a vertical CU.

2/4/16

(i) HCW

BC	Flag	Control field	CM addr
$\log_2 16$ = 4 bits		1 bit/cs = 48 bit	$\log_2 1024$ = 10 bits

HCW size = 62 bits
HCM size = 1024 cw
= $\frac{1024 \times 62}{8}$ bytes.

(ii) Vertical programming.

VCW

BC	Flag	Control field	CM addr.
$\log_2 16$ = 4 bits	$\log_2 48$ = 6 bit	$\log_2 1024$ = 10 bits	

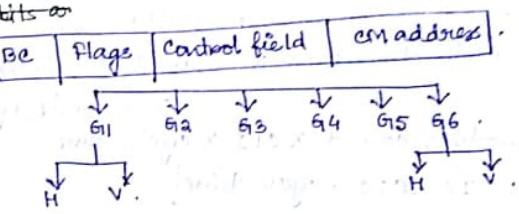
VCW size = 20 bits
VCM size = 1024 cw.
= 1024 × 20 bits
= $\frac{1024 \times 20}{8}$ bytes.

5. A CU supports 6 groups of mutually exclusive control signals are

Group	G_1	G_2	G_3	G_4	G_5	G_6
cs	1	3	5	8	20	30

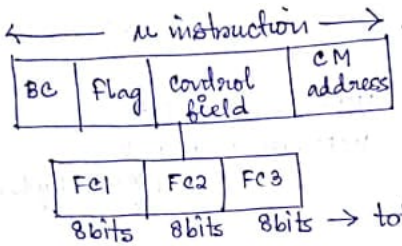
How many bits are saved using vertical over horizontal programming.

~~How many bits or~~
Ans→



Max bits req = 1+3+5+8+20+30 = 67 bits.
(Horizontal)
~~Max~~ bits req = $\log_2 1 + \log_2 3 + \log_2 5 + \log_2 8 + \log_2 20 + \log_2 30$.
= 1+2+3+3+5+5 = 19 bits.
∴ No of bits saved = 67 − 19 = 48 bits.

(9) A microprogrammed CU supports 256 control signals. Micro instruction is designed such that 3 signals are active. What is the size of control field in micro instruction?
Ans:- default μ programmed CU ≡ vertical.
cs = 256
bits per cs = $\log_2 256 = 8$ bits.



→ total 24 bits.

Performance evaluation of a processor.

Performance is an indirect measurement. So it is measured based on execution time. Amount of time required to complete program execution is called execution time.

System X — {Task 1} — ET$_x$ = 5ns

System Y — {Task 1} — ET$_y$ = 10ns.

→ Execution time is calculated based on processor clock that operated with a const freq.

→ CPU time = no. of seconds / program

= no. of instruction/prog × no. of cycles/instruction × seconds/cycle.

= instruction count × CPI × cycle time.

CPU time = IC × CPI × cycle time

→ Program is a combination of data transfer, data manipulation and transfer of control instruction. Different instr. consumes different cycles to complete execution. So prog execution time

CPU time = $\sum$ (IC$_i$ * CPI$_i$) cycle time

→ Speed up (S) factor is used to calculate the performance gain. It is a comparison factor used to compare diff CPU performance level to report the performance gain.

$$S = \frac{\text{Performance. X}}{\text{Performance Y}} = \frac{\frac{1}{ET_x}}{\frac{1}{ET_y}} = \frac{ET_y}{ET_x}.$$

S = n → some const.

→ System x will run n times faster than system Y.

(8) 2.3 GHz processor used to execute the following prog sequence.

Inst^n Type	Ins^n count	CPI.
Load	900	11
store	200	9
arithmatic	300	8
logical	150	6
Branch	50	4

a) Avg Inst^n Execution time of a prog

b) MIPS rate

c) Prog ET

a) Avg inst^n ET = $\frac{\text{Total no of \# cycles / prog}}{\text{total no of \# Ins^n / prog}}$

$$= \frac{(900\times11)+(200\times9)+(300\times8)+(150\times6)+(50\times4)}{300+200+300+150+50}$$

$$= \frac{8600}{1000} = 8.6 \text{ cycles}.$$

$$= 8.6 \times \frac{1}{2.3} \text{ ns}.$$

$$= 3.74 \text{ ns}$$

(b) MIPS rate
(Million instruction per second)

1 inst^n ——— 3.74 ns

? no of inst^n executed in 1 sec ⇒ $\frac{1 \text{ inst}^n}{3.74 \text{ ns}}$

$= \frac{10^9}{3.74}$ Instruction/sec

$= 0.2674 \times 10^9$

$= 267.4$ MIPS

1 Million = 10^6.

(c) Program execution time

$= \frac{\text{No of inst^n in the prog}}{\text{avg inst^n execution time}} = \frac{2}{}$

= (No of inst^n in the prog × avg inst^n ET)

= 1000 × 3.74 ns

= $10^3 * 3.74 \times 10^{-9}$ sec = 3.74 μsec.

(8) A 2.3 ns clock cycle processor consumes 8 cycles/load & store inst^n, 6 cycles for ALU inst^n and 4 cycles for branch inst^n. Relative freq of these inst^n are 40%, 40%, 20% respectively. Processor is enhanced then the clock cycle time is changed to 3.2 ns with an avg CPI of 1. How much performance

is improved in enhancement?

Here relative data given hence adding the values gives the average.

Execution time of old system

$ET_{old} = \sum (IC_i * cpi)$ cycle time

$= [(0.4 \times 8) + (0.4 \times 6) + (0.2 \times 4)]$ 2.3 ns.

$= 14.7$ ns.

$ET_{new} = \sum (IC_i * CPI_i)$ cycle time

$= [(0.4 \times 1) + (0.4 \times 1) + (0.2 \times 1)]$ 3.2 ns.

$= 3.2$ ns.

$S = \frac{ET_{old}}{ET_{new}} = \frac{14.7}{3.2} = 4.6.$

New system is running 4.6 times faster than old system

High performance CPU design

High performance processors exploits the concurrency. Concurrency means 2 or more events execution. Event may be a program or subprogram or instruction or stage of an instruction. Based on the event, the corresponding concurrency is named as program level or subprogram level or inter instruction level or intra instruction level of concurrency. Concurrency implies to parallelism or simultaneous or pipelining.

$(T_1\ T_2\ T_3\ T_4)$ $(T_5\ T_6\ T_7)$

Time period. period 2.

at same time instant — simultaneous operation.

~~not~~ same time period — parallel operation

→ Parallelism means two or more events execution in the same time period.

→ Simultaneous means two or more events execution in the same time instance

→ Pipelining means two or more events execution in an overlapped time span.

According to ~~three~~ Flynn's classification processor architecture is divided into 4 types

1) SISD – Single instruction stream and single data

2) SIMD — Single instruction stream and multiple data stream

MISD — Multiple " " " Single " "

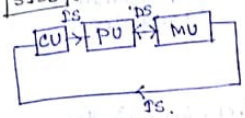
MIMD — " " " " Multiple " "

Short-cut.

CPU { CU: Control unit
PU: Processing unit (ALU)
MU: Memory unit

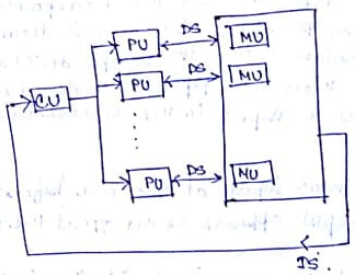
IS: Instruction stream
DS: Data Stream.

SISD :-



In this architecture, instruction & data both are present in the same memory. After the operation result is also present in the same memory as called as stored program concept. Von Neuman machine was designed based on the stored prog concept. This architecture is implemented in a uniprocessor system design.

SIMD. Single control unit, multiple processing unit.

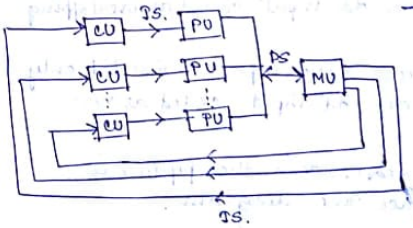


This architecture is implemented in array processor system design.
ex:- Staran Processor
PEPE "

(Concurrent operation can be performed. High performance computer) as different PU

$a_{11} * b_{11}$ can be $a_{12} * b_{12}$ done simultaneously

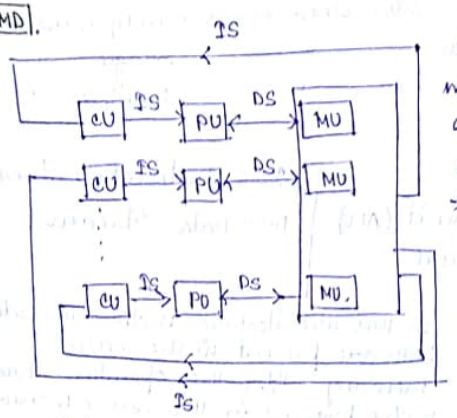
MISD.



→ In this architecture, physically multiprocessors are present but logically only one processor is in the use therefore this architecture is not yet implemented.

P.T.O

MIMD.



This architecture is implemented in a multi processor system design.

ex :- Cray processor
→ cyber processor.

Concurrency possible hence high performance computer

SISD is low performance processor.

Note :- Pipelining is an enhancement technique in the processor design used to improve the performance of a uni processor system.

PIPELINING

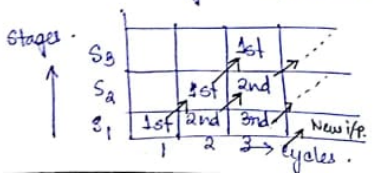
→ Function of a pipeline :- pipeline uses the decomposition technique. In this process, problem statement is decomposed into subproblems and assign them to independent hardware & later connect them in pipeline order i.e first unit op connected as a input to the second unit and so on.

Definition :- Accepting the new inputs at one end before the previously accepted input appears as an o/p at the other end.

Definition states that insert the new inputs into the pipeline before completion of old input. So that the new input is executing along with an old input called as overlapping execution.

In the non pipeline process new input is inserted only after the completion of an old input called as non overlapping execution.

→ Overlapping execution sequence in the pipeline is described using the space time diagram.



Successful characteristic of pipelining → in every new cycle new input is inserted into a pipeline therefore CPI = 1.

Design process :-

→ Pipeline has 2 ends name as input end and output end. Between these ends, multiple pipes are interconnected. Each pipe in the pipeline is called as stage or segment.

→ Interface register is used between the stages to hold the intermediate o/p. It is also named as buffer or latch or pipeline register.

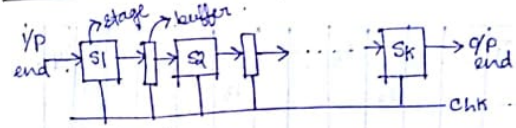
→ All the stages in the pipeline along with a interface register is controlled by the common clock so clock adjustment is very important factor.

Note :- ① When all the stages are perfectly balanced (uniform delay) then cycle time = stage delay.

② When pipeline stages are not perfectly balanced (non uniform delay, then cycle time = max (stage delay)

③ If buffer delay is included then ~~cycle time = max~~

cycle time = max(stage delay + Buffer delay)



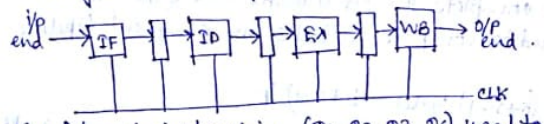
Instruction pipeline :-

→ The pipeline which is used to process the instruction is called as instruction pipeline.

→ Instruction execution process is decomposed into 4 sub operations

① Instruction fetch (IF) — Mem unit.
② Instruction decode (ID) — Decode unit.
③ Execute (EX) — ALU unit
④ Write back (WB) — register unit

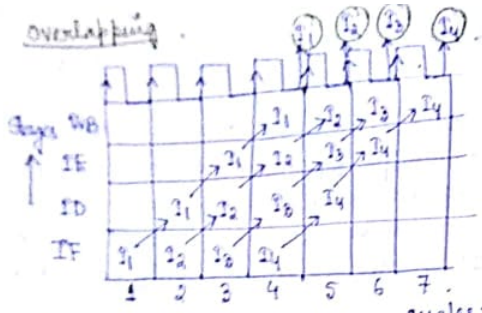
→ Sub operations are assigned to an independent hardware. Later connect them in a pipeline order becomes 4 stage instruction pipeline i.e



Consider 4 instructions (I1, I2, I3, I4) used to execute on a above pipeline. Overlapping execution sequence is

P.T.O

Overlapping.



7 cycles are needed.

Program CPI ≠ 1 due to limited I/P.

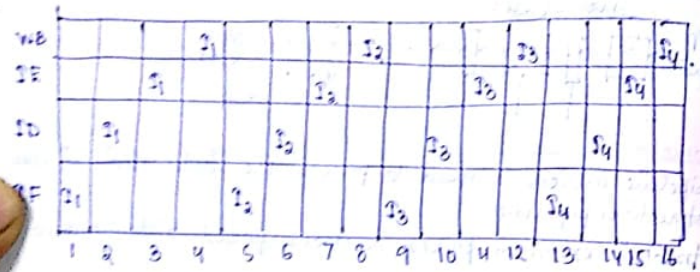
Pipeline CPI = 1

I_1 is using 4 clock cycles. I_2 ~~is~~ using 3 cycles of I_1 and 1 extra cycle is needed for I_2. Similarly I_3 and I_4 are also taking only 1 extra cycles each.

Hence total cycles = $4+(4-1)=7$.

If k cycles are needed for each instr^n and n instr^n are there then $k+n-1$ cycles will be needed.

Non overlapping.



Performance evaluation of pipeline:-

Consider k segment pipeline with cycle time t_p used to execute n tasks. The very first task is executed in the pipeline in a non overlapping method in k cycles.

The remaining $(n-1)$ tasks are emerged from the pipe at the rate of 1 task per cycle. So $(n-1)$ cycles are required to complete the $(n-1)$ tasks. Therefore total time required to complete n tasks in k segment pipeline is

$ET_{pipe} = (k+n-1)$ cycles $= (k+n-1)\,t_p$.

→ Consider non pipeline processor used to execute n tasks in which each task takes t_n time to complete.

Therefore total time required to complete n tasks in a non-pipeline process is

$$ET_{nonpipe} = n.t_n$$

→ Performance gain of a pipeline is $S = \frac{\text{Performance pipe}}{\text{Performance non pipe}}$

Speed up factor $S = \frac{\frac{1}{ET_{pipe}}}{\frac{1}{ET_{nonpipe}}} = \frac{ET_{nonpipe}}{ET_{pipe}}$

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

→ When the number of tasks increases then n becomes much larger than $k+n-1$ ~~(k-1)~~ then

$(k+n-1) \to n$

under this condition $S = \frac{t_n}{t_p}$ as $n \to \infty$.

t_n: one task $ET_{non pipe}$

t_p: cycle time of pipeline.

t_n: One task execution time in non pipeline.

→ When the pipeline stages are perfectly balanced then one task ET in the pipeline is also equal to the number of stages in the pipeline i.e $t_n = k$ cycles

$t_n = k\,t_p$.

$\Rightarrow S = \frac{k.t_p}{t_p} = k$ → number of stages in pipeline = pipeline depth.

$S = k$ 100% efficiency.

→ When the system is operating with 100% efficiency, then maximum speed up is possible that is also equal to a pipeline depth.

100% $\eta \Rightarrow S_{max}$.

$\eta \to S$.

$$\eta_{pipe} = \frac{S}{S_{max}} = \frac{S}{k}$$

~~(Throughput) TP pipe = total time taken to process the n tasks~~

(Throughput) TP pipe = $\frac{\text{\# tasks processed}}{\text{total time taken to process the tasks}}$

$$TP_{pipe} = \frac{n}{(k+n-1)T_p}$$ (Tasks/sec)

① Consider an instruction pipeline which has a speedup factor 10 while operating with 70% efficiency. What could be the no. of stages in the pipeline?

$S = 10$ $\eta = 70\%$ $\eta = \frac{S}{k} \Rightarrow 70\% = \frac{10}{k}$

$k = \frac{10}{0.7} = \frac{100}{7} = 14.2 = 15$

15 stages are present.

(8) Consider 2 pipelines A & B where A is having 8 stages of uniform delay of 2 ns. While B is having 5 stages with a respective delay of 2ns, 6ns, 4ns, 3ns & 1ns. How much time is saved when 100 tasks are pipelined using A instead of B?

$(ET)_A = (k+n-1)\,tp$.

$= (8+100-1) \times 2$ ns

$= (107 \times 2)$ ns

$= 214$ ns.

$(ET)_B = (k+n-1)\,tp$

$= (5+100-1) \times 6$ ns

$= 624$ ns.

$\therefore (ET)_B - (ET)_A = (642 - 214)$ ns.

$= 428 - 214$

$= (624 - 214) = 410$ ns.

(9) Consider 4 stage pipeline with a respective delay of 20 ns, 30 ns, 10 ns, 15 ns. What is the approx speedup and eff of pipeline when very large no of tasks are executed?

$S = \frac{tn}{tp}$

$= \frac{30.6}{15}$

$= 2.5$

$tp = 30$ ns.

$tn = (S1+S2+S3+S3)$ (time for one task to be performed, = total time of all the stages)

$= (20+30+10+15)$ ns

$= 75$ ns *

$\eta = \frac{S}{k} = \frac{2.5}{4} = 62.5\%$ $(\neq 100\%)$

If all the stages have same delay then $S = k \rightarrow 100\%\,\eta$

(8) Consider 4 stage pipeline with respective delay of 30ns, 40ns, 25ns, 20ns. Interface reg used between the stages have delay of 5ns what is performance gain of pipeline?

$S = \frac{tn}{tp}$; $n \rightarrow \infty$

$tp = \max$ (stage delay + buffer delay)

$= \max(35, 45, 30, 20)$

$= 45$ ns.

last stage doesnot need buffer so 20 taken.

* If it was a pipeline proces then $tn = (k+n-1)\,tp = 120$ ns.

But overall delay will be $120+30+30+30 = 210$ ns.

But in non-pipeline total time

$ET = 75 \times 4 = 300$ ns.

$tn = (S1+S2+S3+S4)$

$= 30+40+25+20$

$= 115$ ns

$S = \frac{115}{45} = 2.55$

last stage of p doesnot need buffer. $\therefore 0$

Buffers not required in non-pipeline. There is no need to store intermediate o/p as * after completion of one stage the other stage starts.

Note:- Interface register is not required in non-pipeline because data dependency condition doesnot occur.

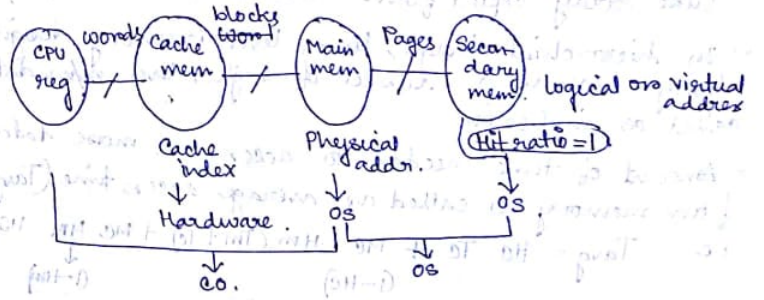
Memory Organisation

According to a memory hierarchy design system supported memory standard is as follows.

Mem Std's:

Level:	1	2	3	4
Name:	Register	Cache mem	Main mem	Secondary mem
Typical size:	<1KB	<16MB	<16GB	>100GB
Implementation:	Customised multipart	SRAM (flip flops)	DRAM (Capacitor)	Magnetic
Access time (ns):	(0.25-0.5)	(0.5-25)	(80-250)	50,00,000
Bandwidth (MB/sec):	(20000-1,00,000)	(5000-10,000)	1000-5000	20-150.
Managed by:	Compiler	H/W	OS	OS.
Backed by:	Cache mem	Main mem	Secondary mem.	Compact disc.

With reference to hierarchial design accessing sequence of a system supported memories is as follows so



→ CPU generated memory request is initially referred in the cache memory. When the data is in cache, then the operation becomes hit so the hit data is transferred to CPU in a form of words. Otherwise (data is not in the cache), the operation is miss so the reference is forwarded to main memory.

→ When the operation is hit in main memory, then the data is transferred to cache in a form of blocks later cache to CPU in a form of words. Otherwise the request will be forwarded to secondary mem.

→ Secondary mem is a final memory in the computer so operation is always hit in the secondary mem. Therefore hit data is transfered to main memory in a form of pages, main memory to cache in a form of blocks and cache to CPU in a form of words.

→ In the hierarchial design, CPU performs the read & write operations only on a reusable data space (cache memory). This principle is named as locality of reference. It is of two kinds.

(i) Temporal locality :- Means the same word in a same block is referenced by the CPU in a near future.

(ii) Spatial locality :- Means adjacent words in the same block is referenced by the CPU in a sequence (like array copied from main m/m to cache)

→ In hierarchial design, lower level memory design is always become a subset of higher level m/m data called as inclusion

→ Amount of time needed to access one more data from memory is called as average access time (Tavg).

i-e $T_{avg} = H_C T_C + M_C \cdot H_M (T_M + T_C) + M_C M_M H_S (T_S + T_M + T_C)$

$M_C = (1-H_C)$, $M_M = (1-H_M)$

$$T_{avg} = H_C T_C + (1-H_C) H_M (T_M + T_C) + (1-H_C)(1-H_M) H_S (T_S + T_M + T_C)$$

H_C → Hit cache
T_C → access time for data from cache.
M_C → Miss in cache $= (1-H_C)$
H_M → Hit into in main mem.
T_M → access time for data from cache.

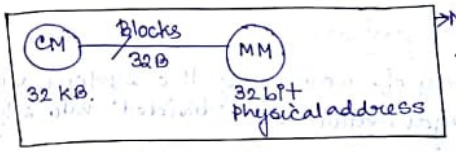
Main memory to cache memory organisation :—

→ In the hierarchial memory organisation, data is transferred from main memory to cache mem in form of blocks so both of the memories are organised into blocks based on block size.

$$\text{Number of blocks (lines) in CM} = \frac{\text{CM size}}{\text{block size}}$$

$$\text{Number of blocks (lines) in MM} = \frac{\text{MM size}}{\text{block size}}$$

Data moved b/w memories block by block.



→ Number of cache memory blocks $= \frac{32\,KB}{32B} = 1K = 2^{10}$.

→ Number of memory blocks in main memory $= \frac{2^{32}}{32} = 2^{27}$.

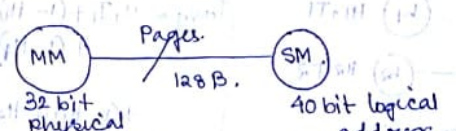
Cache memory contains 2^{10} block space. Main mem contains 2^{27} information i.e blocks. Block size is same

→ Secondary memory to main memory organisation :—

In the hierarchial design data is transferred from SM to MM in a form of pages so both of the memories are organised in form of of pages based on page size.

$$\#\text{ No of pages (frames) in MM} = \frac{\text{Main mem size}}{\text{page size}}$$

$$\#\text{ of pages in SM} = \frac{\text{SM size}}{\text{page size}}$$



$128 = 2^7$.

→ No. of the MM blocks (frames) $= \frac{2^{32}}{2^7} = 2^{25}$ frames.

→ No of pages in SM $= \frac{2^{40}}{2^7} = 2^{33}$.

Note :- Replacement algorithms are needed in the memory design used to replace the data when the memory is full. They are

1. FIFO →
2. LRU → least Recently used.
3. Optimal replacement.

→ In FIFO, replace the block or page which is having the longest time stamp.

→ In LRU, replace the block or page which is present in the memory longest time without reference.

→ In optimal replacement, replace the block or page which is not requried in future upto a long time. This algorithm is not in practice because references are not known in advance.

Types of memory organisation :-

Based on the way of accessing the system supports M/M, memory organisation is divided into 2 types

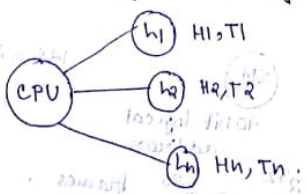
i) Simultaneous access
ii) Memory organisation
ii) Hierarchial access M/M org.

Simultaneous access M/M organisation :-

In this design CPU is directly connected to all the levels of memories.

Accessing sequence of this design is, when there is a miss in level 1 then CPU directly access the data from level 2 without copying into level 1. If not in level 2 then CPU directly access the data from level 3 without copying into level 2 & level 1 & so on.

→ In this organisation, data block needs not be required in the level 1 memory so locality of reference is not present.



$T_{avg} = H_1T_1 + (1-H_1)T_2H_2 + (1-H_2)(1-H_2)H_3T_3 + \dots + (1-H_1)(1-H_2)\dots(1-H_{n-1})H_nT_n$

$H_n = 1$

(a) In a 2 level memory organisation, level 1 memory is 8 times faster than level 2 & it's access time is 40ns less than the avg access time. Let level 1 access time is 20nsec. what is the hit ratio?

If 'B' contain hierarchy word or hierarchy meaning or cache memory word then use hierarchial organisation, else use simultaneous organisation. This 'B' belongs to simultaneous.

$h_1 \rightarrow T_1, H_1$ $\quad T_2 > T_1$.

$h_2 \rightarrow T_2, \boxed{H_2 = 1}$ $\quad S \propto \frac{1}{T}$.

$S = \frac{T_2}{T_1} \Rightarrow 8 = \frac{T_2}{T_1} \rightarrow \boxed{T_2 = 8T_1}$

$T_1 = T_{avg} - 40$

$\boxed{T_{avg} = T_1 + 40}$

let $T_1 = 20nsec$.
$T_2 = 160$ ns
$T_{avg} = (20+40)ns = 60ns$.

$T_{avg} = H_1T_1 + (1-H_1)H_2T_2$

$60 = H_1 \times 20 + (1-H_1) \times 1 \times 160$

$\Rightarrow H_1 =$

(b) 3 level memory organisation has the following specifications.

level	Access time/word	Block size in words	Hit ratio
1	20ns	4	0.7
2	50ns	8	0.8
3	100 ns		1

If the referred block is not in h_1 then transfer the data from h_2 to h_1. If not in h_2 then transfer the data from h_3 to h_2 & h_2 to h_1. How long does it take to access the data?

Hierarchial.

× access time/block.

$T_1 = 20ns$

$T_2 = 200ns$

$T_3 = 800$ ns.

$T_{avg} = H_1T_1 + (1-H_1)H_2(T_2+T_1) + (1-H_1)(1-H_2)H_3(T_3+T_2+T_1)$.

$= 0.7 \times 20 + (0.3) \times 0.8\,(220) + (0.3)(0.2) \times 1\,(1020)$.

$= 14 +$

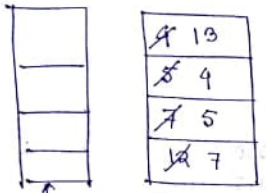
$= 128$.

(8) Consider 4 block cache (empty) with the following main memory block references. 4, 5, 7, 12, 4, 5, 13, 4, 5, 7.
Identify the hit ratio using
a) FIFO b) LRU c) optimal replacement.

Hit ratio = $\frac{\text{Number of hits}}{\text{Total no. of accesses}}$.

4, 5, 7, 12, 4, 5, 13, 4, 5, 7.

FIFO.



4 – M
5 – M
7 – M
12 – M
4 – Hit
5 – Hit
13 – M
4 – M
5 – M
7 – M

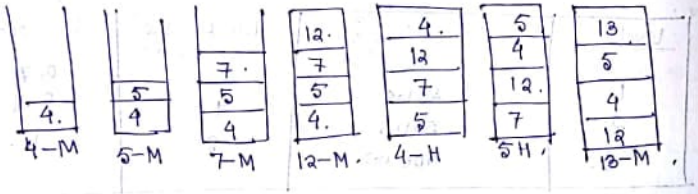
First in first out.
Here when miss of top element will be replaced then second and so on.

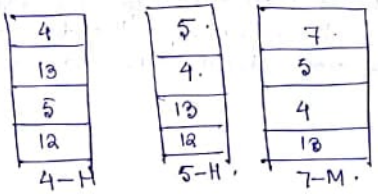
$H = \frac{2}{10}$

$H = 0.2$

Initially cache empty.

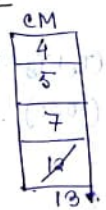
LRU.





always at the top data is stored. whatever is hit comes to top.

Optimal



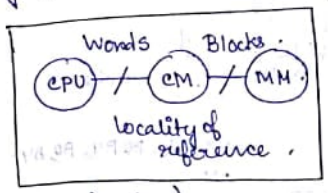
4 – M
5 – M
7 – M
12 – M
4 – Hit
5 – Hit
13 – Miss
4 – Hit
5 – Hit

7 – Hit
$H = \frac{5}{10}$
$= 0.5$

If 12 is also present in seq then also 12 will be deleted here as it is needed in long future.

Cache memory:-

Cache memory is a intermediate memory between the CPU & main memory. It is used to hold the image of main memory. So CPU will be accessing the main memory data from cache memory within a less amount of time. This principle is called as locality of reference.

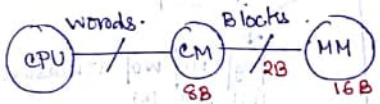


(main mem is slow)
Cache is fast

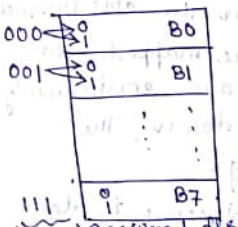
→ Designing elements of a cache memory is
① Memory organisation
② Mapping techniques
③ Replacement algorithms
④ Writing policies.

Memory organisation:-

In the hierarchial system, data is transferred from main memory to cache memory in a form of blocks so both of the memories are organised into a blocks based on the block size.



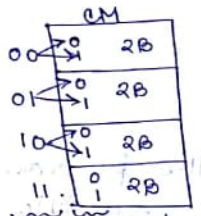
MM



4bit address.

Memory is in form of blocks.
4bit address interpretted into
3bit line offset & 1 word offset.

$\#\text{lines} = \frac{8}{2} = 4$.



Memory is in a form of ~~blocks words~~ blocks.
3 bit address interpreted into 2 bit line offset & 1 bit word offset.

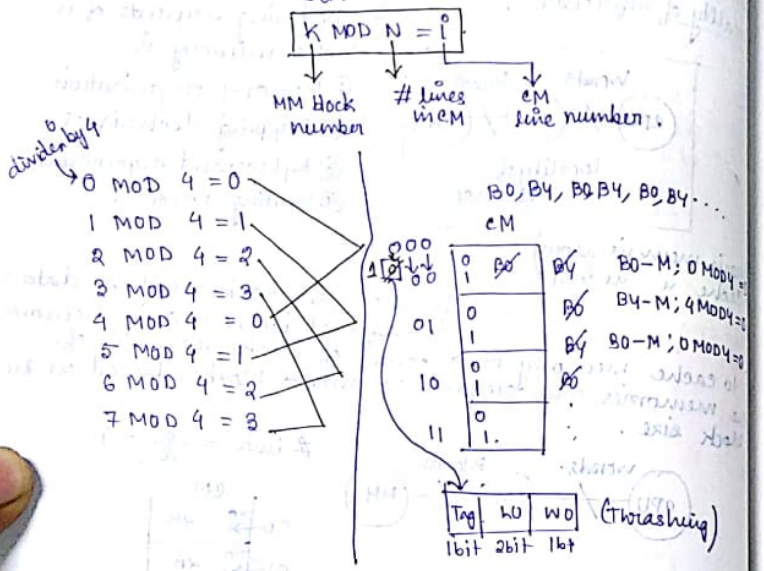
In CM.
Physically chip is having 8 cells but logically 4 blocks.
4 block data in CM

In MM
Physically MM chip is having 16 cells but logically 8 blocks.
logically cells are grouped into blocks.
8 block data in MM.

Mapping tech:-

The process of transferring the data from main memory to cache memory is called as mapping. There are 3 diff mapping tech used to map the data.

① **Direct mapping**:- Mapping fn is used to map the data.

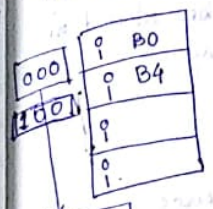


Note

During the program execution, when the CPU frequently refers the multiple blocks which are mapped into same cache line the cache blocks are continuously swapping. So hit ratio will be falling down. This phenomenon is known as thrashing.

→ In the direct cache design, explicit replacement algo are not required to replace the data because every main memory block is having fixed location in the cache memory & every cache line is holding only one block at a time.

② **Associative mapping**:- In this technique mapping fn is not used to map the data. So any blo main memory block can be mapped into any cash line in a sequence so cash memory line number is not in the use.



B0, B4, B7, . . .

B0 – M; Assoc-map to any lines.

B4 – M; " "

→ In this cache design, explicit replacement techniques are needed to replace the cache data when it is full.

→ Disadvantage of this mapping is comparison ckt becomes complex and expensive so hit time increases. Therefore alternative is required i.e set associative mapping.

Set associative mapping

In this cache design, lines are grouped into sets so that each set can be able to hold more than one block at a time to minimize the thrashing problem.

No of sets $S = \frac{N}{P\text{-way}}$

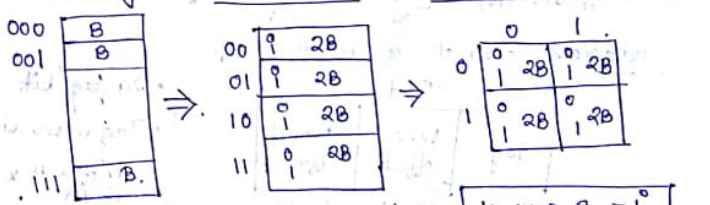
S → No. of lines in cache M/M

P → No of lines in the set.

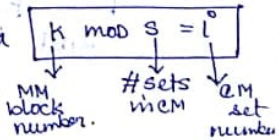
Ex:- 2 way set associative cache of 8B with a block size of 2B.

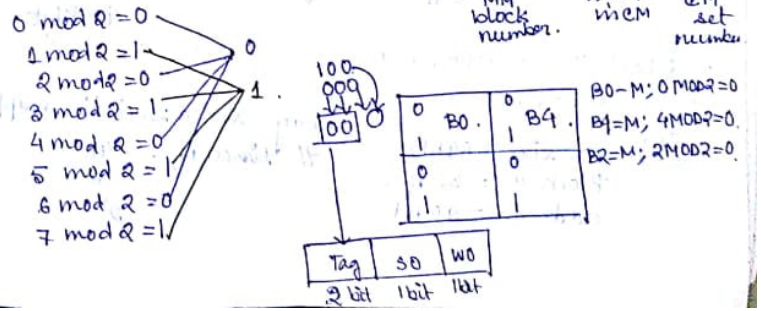
Number of lines (N) $= \frac{8}{2} = 4$.

No. of sets $S = \frac{4}{2} = 2$.



Mapping fn is used to map the data





[3 bit memory address is coming for B0. 1 bit is set address (here 0). Remaining 2 zeros will be stored in cache control reg and will be treated as tag.
No. of sets = 2
Hence for B0: 0 MOD 2 = 0 → means set 0.
B4: → 4 MOD 2 = 0 → " " 0.
We can store B0 and B4 in a sequence
when B2 ~~he~~ comes 2 MOD 2 = 0 but set 0 is full hence replacement needed. That will be done depending on 3 replacement tech FIFO/ LRU/...]

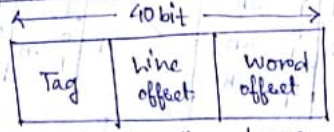
→ In this cache design explicit replacement tech are used to replace the data when the set is full.

(a) Consider a 64 KB direct mapped cache with a block size of 32B. CPU generates the 40 bit physical address. How many tag bits are required in the cache line and what is the tag memory size in the cache controller?.

CM size = 64KB
Block size = 32 B
address " = 40 B
Mapping tech = direct map.

$$\#\text{ lines}(N) = \frac{64K}{32} = \frac{2^{16}}{2^5} = 2^{11}$$

address format of direct map



• 24 tag bit.

• Tag mem size
= No of lines in CM × No of tag bits in line
= $2^{11} \times 24$ bits
= 48 Kbits.

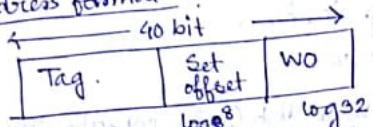
(b) Repeat the question with 8-way set associative cache design.

Sol: CM size = 64KB
Block " = 32 B
add " = 40 bit

Binary set associative mapping
$\#\text{ lines}(N) = \frac{64K}{32}$
$= 2^{11}$

No of sets = $\frac{N}{\text{p-way}} = \frac{2^{11}}{8} = 2^8$.

Address format



∴ 27 bits in tag.

Tag mem size
= # sets in CM × # lines in set × # tag bits in the line

Tag memory size = # sets in CM × # lines in the set × # tag bits in the line
$= 2^8 * 8 * 27$ bits
= 54K bits.

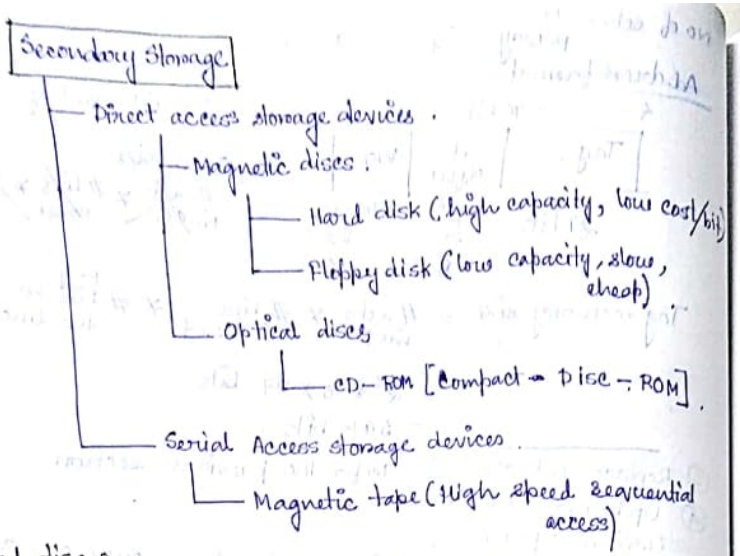
③ Replacement algo's :— refer the previous section
④ Updating techniques :— (writing policies).

→ CPU performs the read & write operations only on a cache memory or in the hierarchial memory system.
→ When the CPU performs write operation then only the cache block is updated. The same updation is not performed in the main memory so the same address contain diff values at diff places. This kind of inconsistency prob in memory is called cache coherance.
→ coherance causes the data loss. To handle this problem writing policies are used in the memory design. These are two types.
i) write through.
ii) write back.
→ In the write through protocall CPU performs the simultaneous write op^n in both cache mem & main mem, so no coherence present.
→ In the write back protocall each cache line maintains one extra bit name as update bit. This bit is set when the CPU updates the cache data. Therefore based on the status of this bit, CPU will write back the data into main memory before replacement. Therefore data is saved.

Secondary storage :—

Classification of a secondary storage components is as follows.

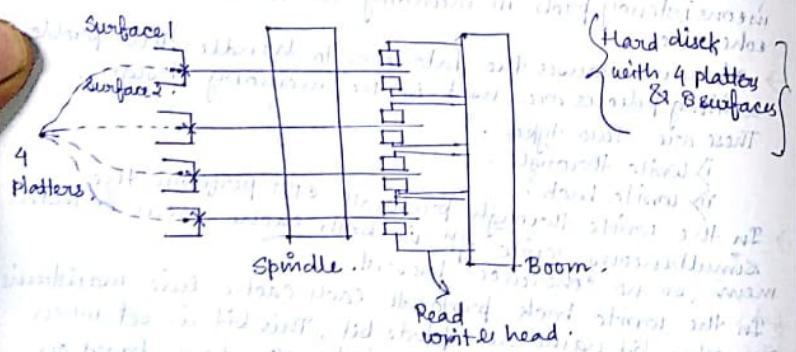
P.T.O



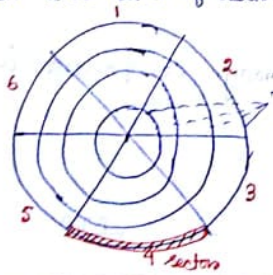
Hard disc :-

Hard disc contain a bunch of magnetic coated platters used to store the data. Each platter contains two surfaces. Every surface contain their own read write head.

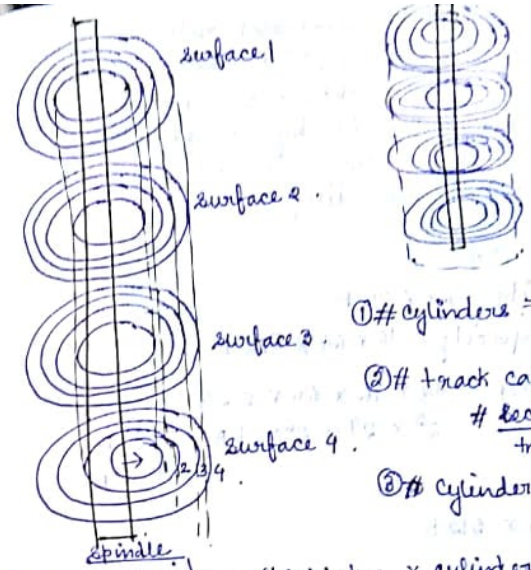
ex :-



→ Surface contains a set of concentric circles called as tracks. Each track divided into a set of sectors. Sector is a unit of data storage in the hard disc.



→ In the hard disc design, same track no in all the surfaces formed as a cylinder.



① # cylinders = # tracks in the surface

② # track capacity = $\frac{\#\text{ sectors}}{\text{track}} * \frac{\text{Bytes}}{\text{sector}}$.

③ # cylinder capacity = # surfaces (tracks in the disc) * capacity.

④ Disc capacity = # cylinders * cylinder capacity.

→ Access time of a data from hard disc is calculated using the following formula ie

$$T_{avg} = \text{Seek time} + \text{Average rotational latency} + \text{Transfer time} + \text{Overhead}$$

→ Seek time is amount of time required to move the read-write head to the desired track in the surface.

→ Rotational latency means amount of time required to adjust the read write head at the begining of a desired sector in the track. It can be measured in 3 cases.

i) Best case — In this case, rotational latency is zero because during the seek time itself read-write head is adjusted

ii) Worst case — In this case, one complete revolution is required to adjust the read write head.

iii) Average case — In this case half of a revolution is required to adjust the read-write head.

Transfer time means amount of time required to access the data from the hard disc.

Overhead means additional delays in the hard-disc operation.

(8) Consider a hard disc with a rotational rate of 7800 rpm. Seek time of a disc is 8.3 ms. Disc contain 8 platters. Each surface contain 32 tracks. Each track containg 64 sectors. Sectors holds the 512 bytes of data. Record size is 4KB. How much time is required to access the record from the hard-disc?

① # cylinders = 32.

② track capacity = 64 * 512 B

③ Cylinder capacity = 16 * 64 * 512 B.

④ Disc capacity = 32 * 16 * 64 * 512 B
$= 2^5 \times 2^4 \times 2^6 \times 2^9$ B $= 2^{24}$ B
= 16 MB.

Alternative
⇒ 16 * 32 * 64 * 512 B.
⇒ $2^4 \times 2^5 \times 2^6 \times 2^9$ B
⇒ 16 MB.

① Seek time = 8.3 ms.

② Rotational time: Depends on the rotational speed.

7800 rev ——— 60 sec
1 rev ——— $\frac{60}{7800}$ sec
1/2 rev ——— $\frac{30}{7800}$ sec. = 7.69 msec.

Average rotational latency = $\frac{1}{2}$ revolution time
$= \frac{7.69}{2}$ msec = 3.84 msec.

③ Transfer time: depends on rotational speed

record size = 4KB.
sector size = 512 B.
1 sector ——— 512 B.
? # sector ——— 4KB.

sector = $\frac{4KB}{512B} = \frac{4 \times 2^{10}}{2^9} = 2^3 = 8$.

1 revolution time ——— 1 track (64 sectors).
? time ——— 8 sectors
⇒ $\frac{7.69 \text{ ms} \times 8}{64}$

1 rev = 1 track data = 64 sectors
1/8 rev = 8 sectors.
1 rev = $\frac{60}{7800}$ sec
1/8 rev = $\frac{60}{8 \times 7800}$ sec = 0.96 ms.

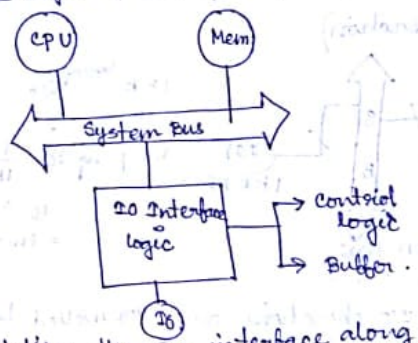
$T_{avg} = (8.3 + 3.84 + 0.96)$ ms = 13.1 ms.

I/O organisation:-

IO devices are electro magnetic component, CPU is electronic comp. So there is a difference in terms of operating modes, word formats, data transfer rate.

To synchronise the I/O speed with a processor speed, high speed interface logic (I/O interface or I/O Module) used as a synchronisation technique.

I/O interface takes the responsibility of I/O. So processor time is utilised for other useful task. In the system design all the I/O devices are connected to a system bus via I/O interface logic.



→ CPU initializes the IO interface along with a I/O command later busy with other useful task.
→ I/O interface control logic interprets the I/O command and enables the respective port for the operation.
→ Based on the speed of a device consumes the time to complete the operation. Later status of an operation is transferred to buffer.
→ When the buffer contain the status then ~~data~~ I/O interface generates the interrupt signal to CPU & waiting for acknowledgement value.
→ After receiving the acknowledgement signal the buffer content will be transferred to a CPU.
→ In this entire process CPU time is wasted by taking the data from the buffer. So buffer delay is very less. Speed gap is minimized.

I/O ~~ports~~ modes :- There are 3 diff I/O transfer modes present in the system design used to transfer the data from I/O to other components of a computer (CPU & main memory). They are

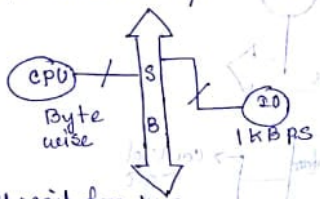
(i) Programmed - I/O
(ii) Interrupt - driven I/O
(iii) DMA.

Programmed I/O :- In this mode, all the I/O devices are directly connected to a system bus without using I/O interface logic. So CPU takes the responsibility of a I/O operation.

→ In this mode, CPU is waiting until the I/O opn is completed. So processors utilization is inefficient.

→ In this mode CPU time depends on as speed of the I/O device and the amount of data to be transferred.

→ This mode is used in the application where the I/O operations are most important than CPU opn. (like ATM transaction)



1KB taking 1 sec.
1B — ?.
ET prog. IO = 1B / 1KB sec
= 10^{-3} sec
= 1ms.

CPU will wait for 1ms.

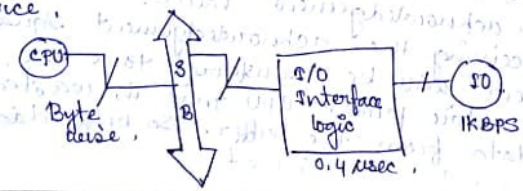
Interrupt - driven I/O :-

→ In this mode, I/O operations are controlled based on the interrupt signals.

→ All the I/O devices are connected to system bus via I/O interface logic. So I/O interface takes the responsibility of I/O operation. Therefore CPU time is utilised for other useful task.

→ In this mode, CPU utilisation is efficient.

→ In this mode, CPU time depends on the latency of an interface rather than the speed of an I/O device.

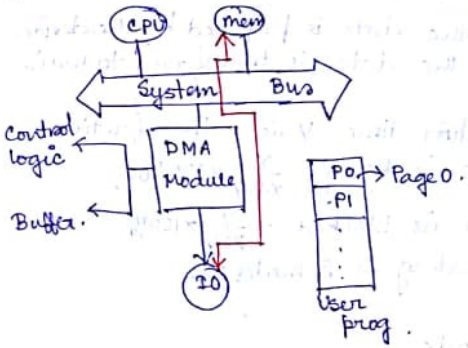


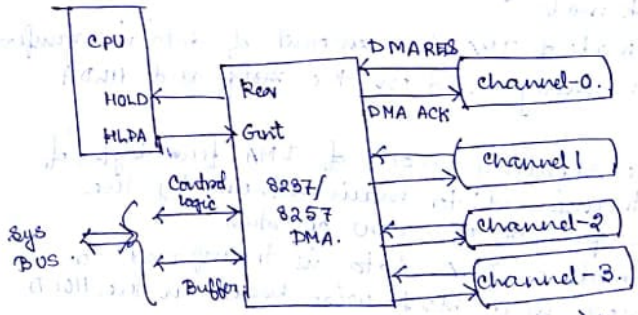
ET INT-IO = 0.4 usec. (CPU time depends on latency of an interface)

DMA :- In this mode, bulk amount of data is transferred from I/O to main memory without involvement of CPU. When the user program size is greater than the main memory size then virtual mem concept is used to increase the address space. This concept states that use the secondary storage component to store the user program.

→ Secondary storage is an electromagnetic storage so it is classified under I/O category.

→ Secondary storage is connected to a system bus via DMA module so during the execution of a program there is a need of transferring bulk amount of data from secondary storage to main memory via DMA module without involvement of a CPU.





CPU initialises the DMA module along with an I/O command. Later busy with other useful tasks. I/O command contain the information about port address, memory address, control signals and count value. DMA control Logic interprets the command & enables the operation.

- Based on the speed of a device consumes the time to prepare the data, later enables the DMA R&B signal.
- When the DMA module receives this signal then it enables the hold signal to gain the control of a system bus and waiting for HLDA signal.
- After receiving the HLDA signal, DMA module enables the DMA ACK line. When the device receives this signal, then it transfers the data to main memory via DMA module until the count becomes zero. After DMA operation, bus connection is re-established to CPU.
- In the DMA operation, CPU is present in 2 states:
  1) Busy state.
  2) Blocked state / HOLD state.
- CPU is busy until the data is prepared by the device.
- CPU is blocked until the data is transferred to main memory.
- let x is a preparation time, y is a transfer time. Then % time CPU is busy $= \frac{x}{x+y} \times 100\%$.

  % time CPU is blocked $= \frac{y}{x+y} \times 100\%$.
- DMA module is operating in 3 modes:—
  i) Burst mode
  ii) Cycle stealing mode
  iii) Block mode.

→ In burst mode of DMA, bulk amount of data is transfer to main memory based on the ~~hold~~ HOLD and HLDA signal.

→ In a cycle stealing mode of DMA, few bytes of data is transferred to main memory by the forcible suspension of a CPU operation.

→ In a block mode of DMA, data is transferred to main memory in a block wise based on the HOLD & HLDA signals.