GATE CSE NOTES

by

Joyoshish Saha

Basic Structure of Computers.

* Computer types :

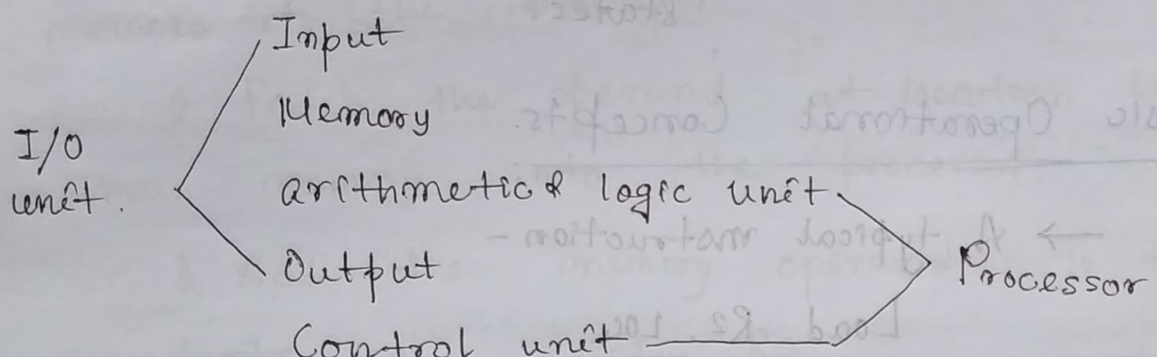
i) Embedded computers.

ii) Personal computers
- Desktop Computers
- Workstation Computers
- Portable Computers.

iii) Servers & Enterprise systems

iv) Supercomputers & Grid Computers.

* Functional units :



→ Input unit : Accepts coded information.
eg. keyboard, touchpad, mouse, joystick, microphone.

→ Memory unit : Stores programs & data.

Primary/main memory - Operates at electronic speed. Programs must be stored on this memory while they are being executed.

Cache memory - Used to hold sections of a program that are currently being executed, along with any associated data.

Secondary memory – Stores large amount of data, eg. CD, DVD, flash memory devices.

→ Arithmetic & Logic unit : Executes any arithmetic or logic operation.

→ Output unit : Sends processed results to the outside world.
eg. graphic display, printer.

→ Control unit : Sends control signals to other units & senses their states.

* Basic Operational Concepts.

→ A typical instruction –

Load R2, LOC.

This instruction reads the contents of memory location whose address is LOC & loads them into processor register R2. Original contents of LOC are preserved, whereas those of R2 are overwritten.

→ Different registers in processors. –

i) Instruction register (IR) - holds the instruction that's currently being executed.

ii) Program counter (PC) - Contains the memory address of the next instruction to be fetched.

→ An instruction consists of 2 parts — operation code & operands.

OPCODE	OPERANDS.

Operands are stored in memory. Individual instruction is brought from memory to the processor. Then the processor performs specified operation.

eg. ADD LOC R0

Steps to execute the operation:

1. Fetch the instruction from main memory into the processor.
2. Fetch the operand at location LOC from main memory into the processor.
3. Add the memory operand to the contents of register R0.
4. Store result in R0.

Same instruction can be realized using 2 instructions as:

Load LOC, R1
Add R1, R0.

* Main Parts of Processor.

→ Contains ALU, control circuitry & many registers.

→ Processor contains n general purpose registers R_0 through R_{n-1}.

→ IR, PC : IR holds the instruction that is currently being executed. PC contains the memory-address of the next-instruction to be fetched & executed.

During the execution of an instruction, the contents of PC are updated to point to next instruction.

→ The control-unit generates the timing signals that determine when a given action is to take place.

→ MAR (Memory Address Register).

Holds the address of the memory location to be addressed/accessed.

→ MDR (Memory Data Register)

Contains the data to be written into or read out of the addressed location.

→ MAR & MDR facilitates the communication with memory.

* Steps to execute an instruction

1. The address of first instruction gets loaded into PC.
2. Contents of PC (i.e. address) are transferred to the MAR & control-unit issues 'read' signal to memory.
3. After certain amount of elapsed time, the first instruction is read out of memory & placed into MDR.
4. Contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.
5. To fetch an operand, its address is placed into MAR & control unit (CU) addresses

Read signal. The operand is transferred from memory onto MDR & then to ALU.

6. Likewise required number of operands is fetched onto processor.

7. Finally ALU performs the desired operation.

8. If the result of this operation is to be stored in the memory, then the result is sent to MDR.

9. Address of the location where the result is to be stored is sent to the MAR & a Write cycle is initiated.

10. At some point during execution, contents of PC are incremented to point to next instruction in the program.

* Bus Structure: Bus is a group of lines that serves as a connecting path for several devices.

→ Single bus structure: Only 2 units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

Advantages - low cost, flexibility for attaching peripheral devices.

→ Multiple bus structure: Contain multiple buses to achieve more concurrency in operations. Two or more transfers can be carried out at the same time.

Advantage - Better performance

Disadvantage - Increased cost.

→ Buffer registers prevent a high speed processor from being locked to a slow I/O device during data transfers.

* Processor Clock :

Processor circuits are controlled by a timing signal called clock. The clock defines regular time intervals called clock cycles. To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps such that each step can be completed in one clock cycle.

Let P = length of one clock cycle

R = Clock rate,

$R = 1/P$ | R in cycles per second (Hz)

* Basic Performance Equation.

Let T = Processor time required to execute a program

N = Actual no. of instruction executions

S = Average no. of basic steps needed to execute one machine instruction

R = Clock rate

then $$T = \frac{N \times S}{R} = NSP$$

→ To achieve high performance, value of T must be reduced, N and S should be reduced & R should be increased

Value of N is reduced if source program is compiled into fewer machine instructions.

Value of S is reduced if instructions have a smaller number of basic steps to perform.

Value of R can be increased by using higher frequency clock.

* Pipelining: Technique by which overlapping of the execution of successive instructions occures so that execution proceeds at the rate of one instruction completed in each clock cycle.

→ Effective value of S in a pipelined processor is close to 1 even though the number of basic steps per instructions may be considerably larger.

→ Pipelining increases the rate of executing instructions significantly & causes the effective value of S to approach 1.

* Superscalar Execution:

A higher degree of concurrency of program execution can be achieved if multiple instruction pipelines are implemented in the processor. Multiple functional units are used, creating parallel paths through which different instructions can be executed in parallel. This mode of operation is called Superscalar Execution.

→ If it can be sustained for a long time during program execution, effective value of S can be reduced to less than one.

* Increasing clock-rate (R).

i) Improving the IC technology to make basic logic faster which reduces the time needed to complete a basic step.

ii) Reducing the amount of processing done in one basic step.

* Istruction Sets

Two cases :

1. Simple Instructions → Large value for N and Small value for S

2. Complex Instructions → lower value of N and large value of S.

→ Complex instructions combined with pipelining would achieve the best performance.

- RISC (Reduced Instruction Set Computers)

  Contain processors with simple instructions.

- CISC (Complex Instruction Set Computers)

  Contain processors with more complex instructions.

- Total number of clock cycles needed to execute a program is dependent not only on the choice of instructions, but also on the order in which they appear in program.

* Performance Measurement :

SPEC (System Performance Evaluation Corporation) selects & publishes representative applications for different application domains, together with benchmark test results.

$$\text{SPEC rating} = \frac{\text{Running time on reference computer}}{\text{Running time on computer under test}}$$

SPEC rating of 50 means that the computer under test is 50 times as fast as the reference computer.

$$\text{overall SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i \right)^{1/n}$$

n is the number of programs in the SPEC suite.

* Multiprocessor & Multicomputer.

→ Multiprocessor systems - In a system there are more the one processors to execute a number of different tasks in parallel or to execute subtasks of a single large task in parallel.

(Shared-memory Multiprocessor System).

→ Multicomputer System - Interconnected group of complete computers to achieve high total computational power.

(Message-passing Multicomputers).

* Generation of Computers.

1. First Generation:

Period : 1946 - 1959

Technology used : Vacuum Tube.

Programming Language used : Machine language

Memory used :

Primary - Magnetic core memory

Secondary - Magnetic drum, magnetic tape.

I/O Device - Punch card as i/p, printing device as o/p.

Use - Simple math calculation.

Description - Unreliable, very costly, generated lot of heat, need of AC, consumed lot of electricity.

eg. ENIAC, EDVAC, UNIVAC, IBM-701, IBM-650.

2. Second Generation:

Period: 1959 - 1965

Technology used: Transistor.

Operation speed: Microsecond range

Programming language: Assembly language

Memory used:

Primary - Magnetic core memory

Secondary - Magnetic drum, magnetic tape.

I/O: Punch card as i/p, printer as o/p.

Use: Complex scientific calculations.

Description: Reliable compared to first gen, smaller in size, generated less heat, consumed less electricity, very costly, AC needed

eg. IBM 1620, IBM 7094, CDC 1604, CDC 3600, UNIVAC 1108, LEO MARK III.

→ Advantages over vacuum tube of transistors.

i) One transistor could replace one thousand vacuum tubes.

ii) Size of a transistor is $1/200^{th}$ times of a vacuum tube.

iii) Power requirement of a transistor is $1/20^{th}$ times of a vacuum tube.

iv) Transistors are more reliable.

3. Third Generation:

Period: 1965 – 1971

Technology used: IC.

Operating speed: Nanosecond range

Programming Language: HLL (fortran, COBOL, PASCAL, C etc)

Memory:

Primary - Semiconductor memory (Si)

Secondary - Magnetic tape, magnetic disk like floppy disk, hard disk.

I/O: Keyboard as i/p & monitor as o/p

Use: Managing population census, bank, insurance company etc.

Description: More reliable, smaller size, generated less heat, costly, AC needed, consumed lesser electricity.

eg. IBM-360 series, Honeywell – 6000 series, PDP (Personal Data Processor), IBM-370/168, TDC-~~168~~.316, ICL-900 series.

→ IC & its types: (Silicon chip, fabrication)

i) SSI (1-20 components)	(< 100)
ii) MSI (21 – 100 ")	(< 500)
iii) LSI (101 - 1000 ")	(500 – 3,00,000)
iv) VLSI (1001 – 10 000 ")	(> 3 00000)
v) ULSI (More than 10000)	(> 15,00,000)

4. Fourth Generation :

Period : 1971 – 1980

Technology used : VLSI (or Microprocessor).

Operating speed : Pico second range

Programming Language : 4GL , HLL (High level language).

Memory :

Primary - Semi-conductor memory

Secondarey - Magnetic tape, magnetic disk, optical memory (CD/DVD)

flash memory (pen drive, memory card).

I/O : Mouse, touch screen, scanner, LCD, LED, color printer.

Use : All kinds.

Description : Very cheap, small size, pipeline processing, NO AC, network field development, power requirement very less, heat generation reduced.

eg. IBM desktop, HP laptop, Acer notebook, Mac book etc.

5. Fifth Generation : (1980 – onwards)

Technology to be used : Bio-chip

Operating speed : Femto second range

Programming language : Natural language.

Description : i) Will have AI. ii) Based on KIPS (Knowledge based Information

Processing System). iii) Will have parallel processing in full fledge.

→ Different Registers:

1. Memory buffer register (MBR).

Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.

2. Memory address register (MAR)

Specifies the address in memory of the word to be written from or read into the MBR.

3. Instruction Register (IR)

Contains the 8-bit opcode instruction being executed.

4. Instruction buffer register (IBR)

Employed to hold temporarily the right-hand instruction from a word in memory.

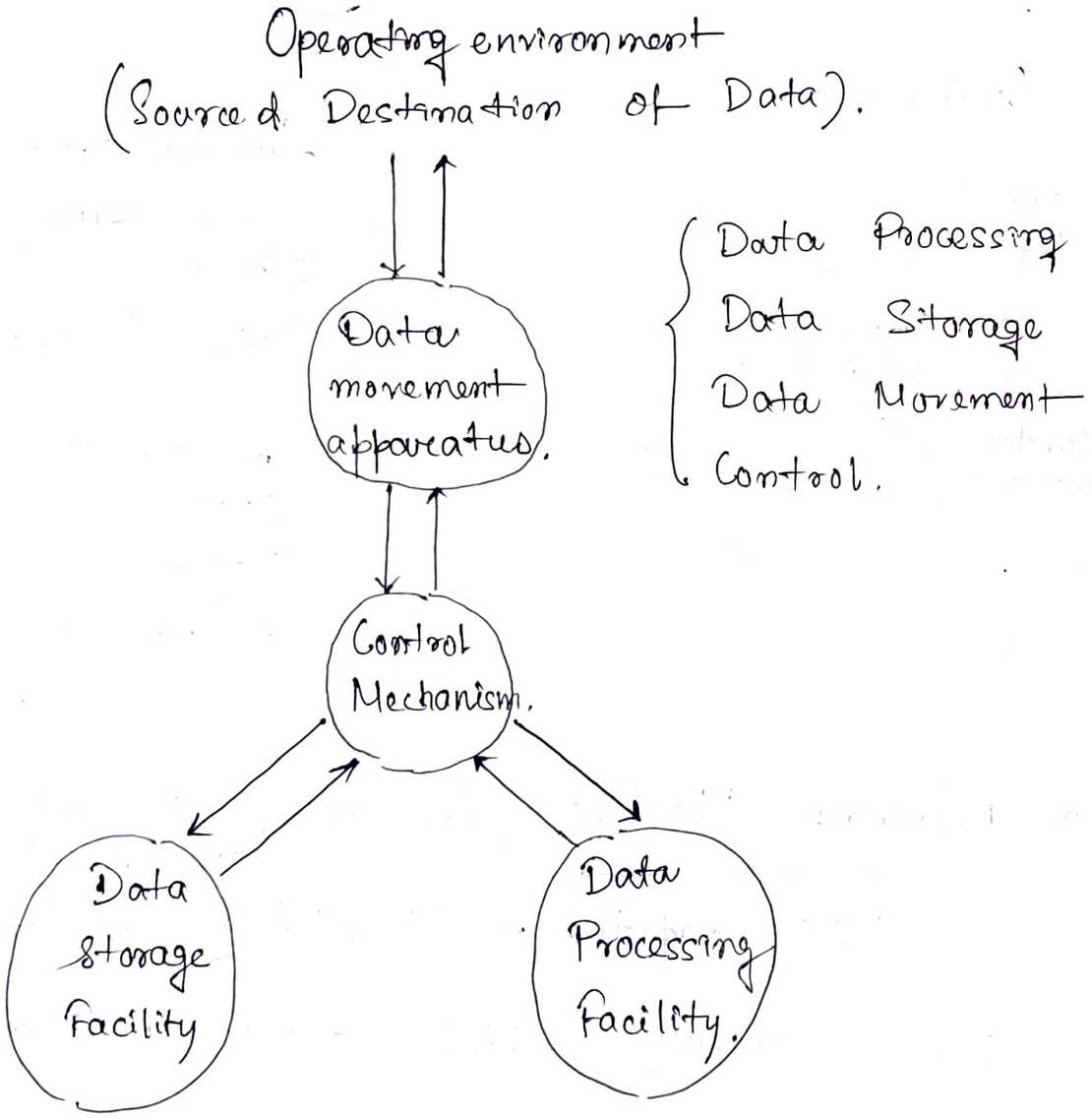
5. Program Counter (PC)

Contains the address of the next instruction-pair to be fetched from memory.

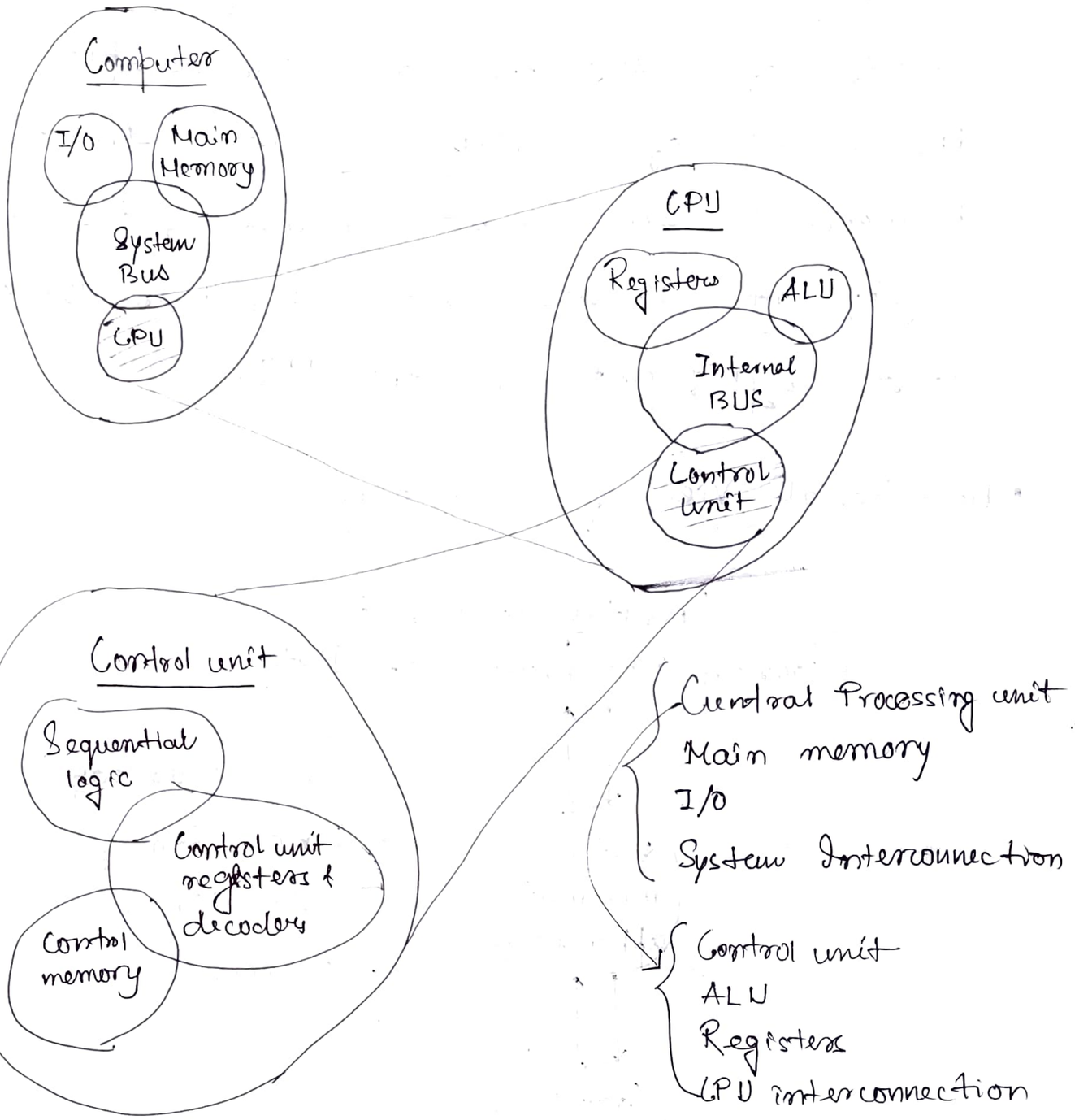
6) Accumulator (AC) & Multiplier Quotient (MQ)

Employed to hold temporarily operands & results of ALU operations. For example, the result of multiplying two 40-bit numbers is an 80-bit number; the most significant 40 bits are stored in the AC & least significant in the MQ.

- Functional View of Computer.



- Computer's Top-level Structure.



- Von Neumann Machine [Institute of Advanced Study, Princeton]

IAS Computer - Stored program Computer.

→ A main memory which stores both data & instructions.

→ An ALU capable of operating on binary data.

→ A control unit, which interprets the instructions in memory & causes them to be executed.

→ I/O equipment operated by control unit.

[Stallings 8e, 19 – 20]

→ Instruction Execution Rate.

Clock frequency = f $\qquad \tau = 1/f$.

Clock cycle time = τ.

Instruction count → I_c, Number of machine instructions executed for that program until it runs to completion or for some defined time interval.

Average cycles per instruction, CPI →

On any given processor, number of clock cycles required varies for different types of instructions.

Let CPI_i be the number of cycles required for instruction type i & I_i be the no. of executed instruction of type i for a given program. We can calculate overall CPI,

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{I_c}$$

Processor time T needed to execute a given program,

$$\begin{cases} T = I_c \times CPI \times \tau. \\ T = I_c \times [p + (m \times k)] \times \tau. \end{cases}$$

$p \rightarrow$ no. of processor cycles needed to decode & execute the instruction

$m \rightarrow$ no. of memory references needed

$k \rightarrow$ ratio between memory cycle time & processor cycle time.

Five performance factors (I_c, p, m, k, τ) are influenced by -

design of the instruction set (ISA),

compiler technology (how efficient the compiler is in producing an efficient machine language program from a high-level language)

processor implementation,

cache & memory hierarchy.

MIPS rate (millions of instructions per second)

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

MFLOPS rate (millions of floating point operations in a program/s)

$$\text{MFLOPS rate} = \frac{\text{No. of executed f-p operations in a program}}{\text{Execution time} \times 10^6}$$

→ SPEC benchmarks ↻

→ Amdahl's Law. Speedup using N processors

$$\text{Speedup} = \frac{\text{time to execute a program on single processor}}{\text{time to execute program on N parallel processors}}$$

$$= \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + f/N} = \frac{1}{(1-f) + \frac{f}{SU_f}}$$

fraction $(1-f)$ of the execution time involves code that is inherently serial & a fraction f that involves code that is infinitely parallelizable with no scheduling overhead. SU_f → speedup after enhancement

→ When f is small, use of parallel processors has little effect.

→ As N approaches infinity, speedup is bound by $1/1-f$, so that there are diminishing returns for using more processors.

→ **Amdahl's Law**: Formula that gives theoretical speedup in latency of the execution of a task at a fixed workload that can be expected of a system whose resources are improved. (often used in parallel computing to predict the theoretical speedup when using multiple processors).

(gfg)

- Speedup ~ Ratio of performance for the entire task using the enhancement & performance without using the enhancement.

$$\text{speedup} = \frac{P_e}{P_w} \text{ or } \frac{E_w}{E_e}$$

E - execution time.

- Fraction enhanced ~ Fraction of the computation time in the original computer that can be converted to take advantage of the enhancement. (always less than 1).

- Speedup enhanced ~ Improvement gained by the enhanced execution mode, i.e. how much faster the task would run of the enhanced mode were used. (always greater than 1).

→ Overall speedup = $\frac{\text{Old execution time}}{\text{New execution time.}}$

$$= \frac{1}{(1 - \text{frac. enhanced}) + \frac{\text{frac. enhanced}}{\text{speedup enhanced.}}}$$

→ Proof: Speedup be S, old execution time T, new T'. Exe^n time that is taken by portion A (that will be enhanced) is t, exe^n time that is taken by portion A (after enhancing) is t', exe^n time that is taken by portion that won't be enhanced is t_n, frac. enhanced is f', speedup enhanced is S'.

NOW, $S = T/T'$

$T = t_n + t$.

$T' = t_n + t'$

$$f' = \frac{t}{T} = \frac{t}{t + t_n}$$

$$1 - f' = 1 - \frac{t}{t + t_n} = \frac{t_n}{t + t_n}$$

$$s' = \frac{t}{t'}$$

$$t' = t/s' = \frac{T f'}{s'} = \frac{(t_n + t) f'}{s'} \Rightarrow \frac{t'}{t_n + t} = \frac{f'}{s'}$$

$$S = \frac{T}{T'} = \frac{t_n + t}{t_n + t'}$$

$$S = \frac{1}{(1 - f') + \frac{f'}{s'}}$$

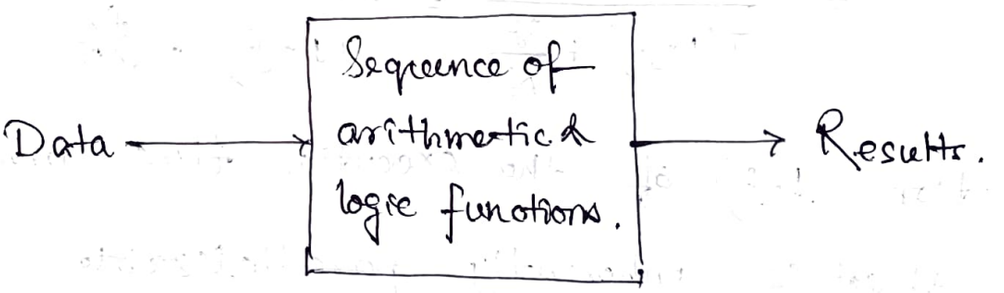
→ Von Neumann Architecture.

→ Data & instructions are stored in a single read – write memory.

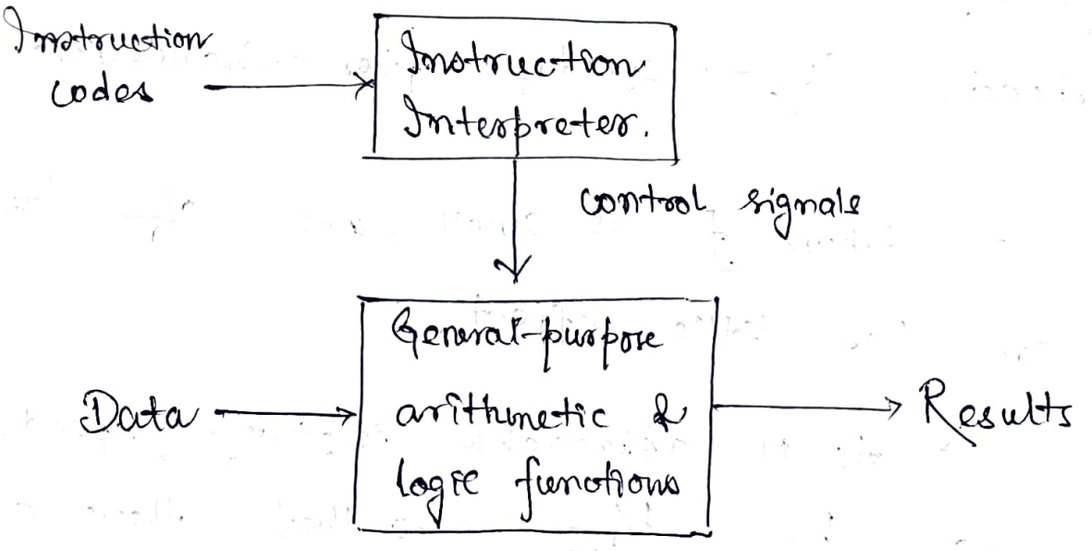
→ Contents of this memory are addressable by location, without regard to the type of data contained there.

→ Execution occurs in a sequential fashion from one instruction to the next.

→ Programming in hardware vs software.

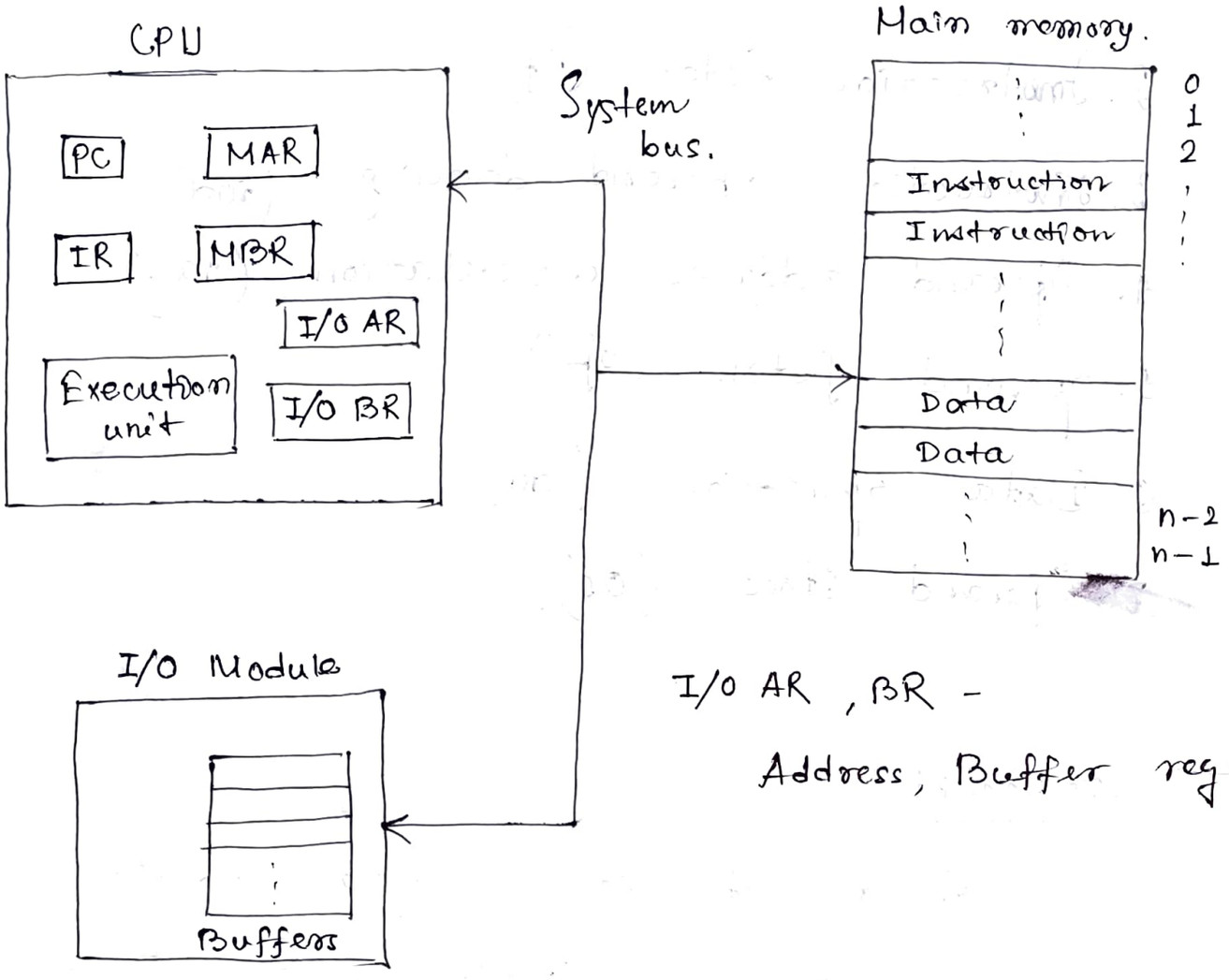


Programming in hardware.



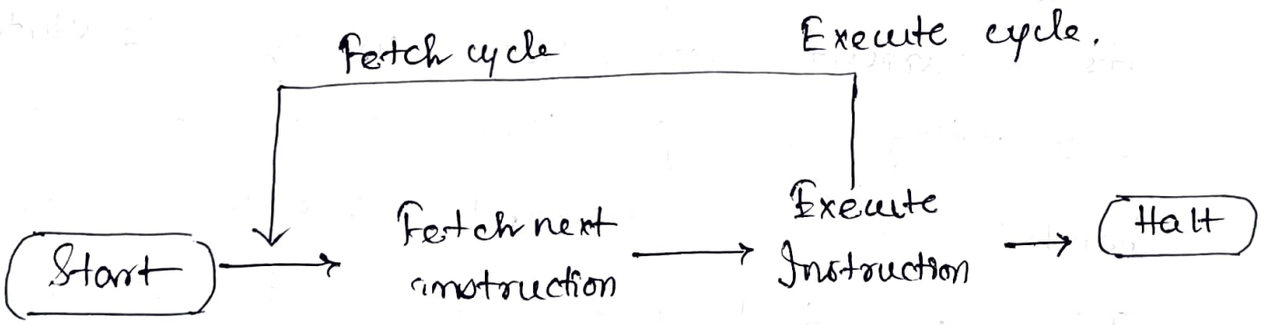
Programming in software.

→ Computer Components.



I/O AR, BR - Address, Buffer register

→ Basic Instruction Cycle.



→ Four kinds of instruction stimulated action -

Processor - memory transfer,

Processor - I/O transfer

Data processing,

Control.

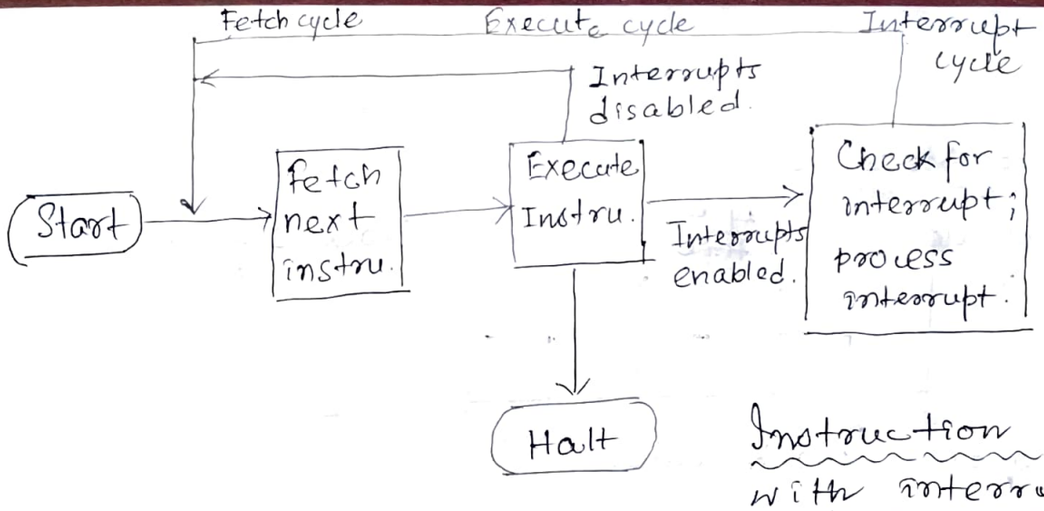
→ States for any given instruction cycle.

1. Instruction address calculation (iac)
2. Instruction fetch (if).
3. Instruction operand decoding (iod)
4. Operand address calculation (oac)
5. operand fetch (of)
6. Data operation (do)
7. Operand store (os).

→ Interrupt :

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. The processor responds by suspending its current activities saving its state & executing a function called an interrupt handler or an interrupt service routine (ISR) to deal with an event. Interruption is temporary & after the interrupt handler finishes, the processor resumes normal activities

Act of initiating a hardware interrupt is referred to as an interrupt request (IRQ).



Instruction cycle with interrupts.

Classes of Interrupts ~

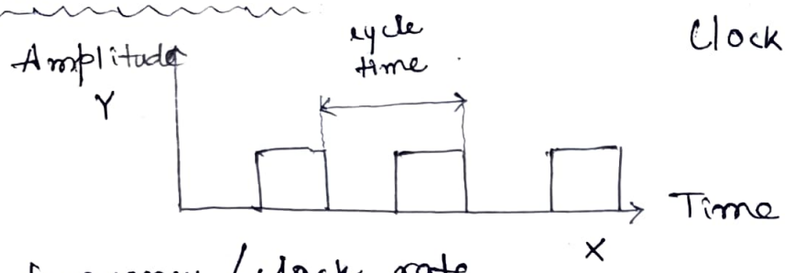
Program ~ Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.

Timer ~ Generated by a timer within the processor.

I/O ~ Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

Hardware failure ~ Generated by a failure such as power failure or memory parity error.

- **Performance Measures:**



f - frequency / clock rate.

CC - cycle count

CT - cycle time = time period = $1/f$.

CPI - cycles per instruction.

IC - instruction count.

MFLOPS - millions floating point operations / second

MIPS - millions instructions per second

→ Execution time = CC × CT = IC × CPI × CT = IC × $\frac{CPI}{f}$

→ CPI = $\frac{\text{Total CPU clock cycles for the program}}{\text{Instruction Count.}}$

= $\frac{\sum_{i=1}^{n} CPI_i \times I_i}{IC.}$ | $IC = \sum_{i=1}^{n} I_i$

→ MFLOPS = $\frac{\text{No. of floating point operation in program}}{\text{Execution time} \times 10^6}$

→ MIPS = $\frac{IC}{\text{Execution time} \times 10^6} = \frac{f}{CPI \times 10^6}$

Q. CPU with 200 MHz frequency executing a benchmark program - calculate CPI & MIPS.

Instⁿ category	percentage occurrence	No. of cycles/ instⁿ
ALU	38	1
Load-store	15	3
Branch	42	4
Others	5	5

→ CPI = $\frac{\sum_{i=1}^{n} CPI_i \times I_i}{\text{Instⁿ count}}$

= $\frac{38 \times 1 + 15 \times 3 + 42 \times 4 + 5 \times 5}{100}$

= 2.76

→ MIPS = $\frac{\text{Clock rate}}{CPI \times 10^6}$

= $\frac{200 \times 10^6}{2.76 \times 10^6}$

= 70.24

Machine Instructions & Programs.

* Instruction Set Architecture (ISA):

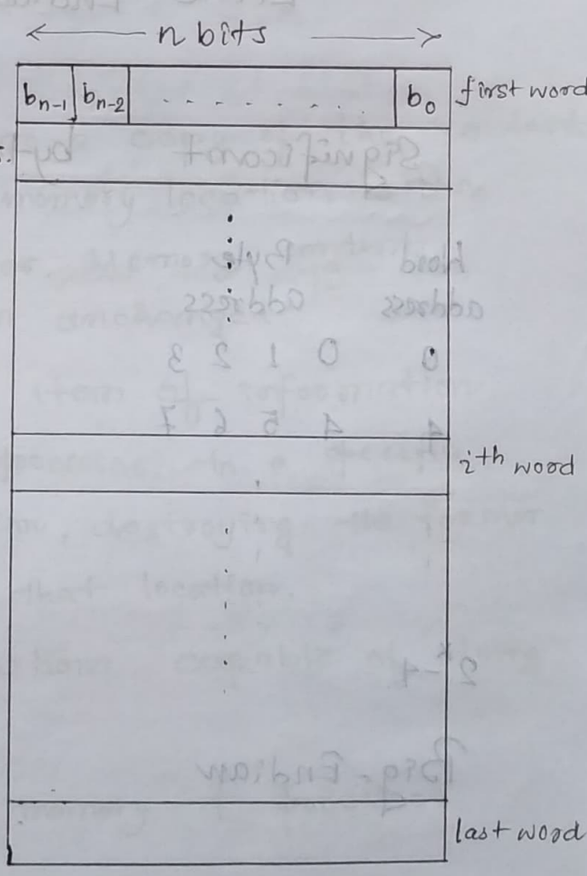
A complete instruction set of a processor specifying instructions, addressing methods used for the access of data operands & the processor registers available for use by the instructions.

* Memory Locations & Addresses.

→ Memory is organised so that a group of n bits can be stored or retrieved in a single, basic operations.

→ Each group of n bits is referred to as a word of information & n is called the word length.

→ Modern computers have word lengths 16 to 64 bits.



→ A K bit address generates 2^K addressable locations $[0 \text{ to } 2^K - 1]$

→ 1M – 2^{20}

1G – 2^{30}

1K – 2^{10}

1T – 2^{40}.

→ 2^{32} – 4G locations.

* Byte-addressability:

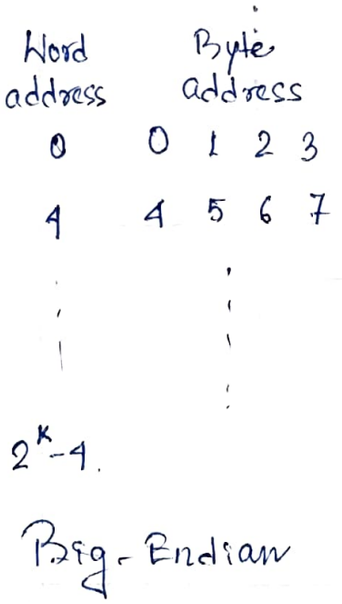
→ Every byte has its own unique address & can be accessed.

→ Byte locations have addresses 0, 1, 2, ... Thus, if the word length of machine is 32 bits, successive words are located at addresses 0, 4, 8, ... with each word consisting of 4 bytes.

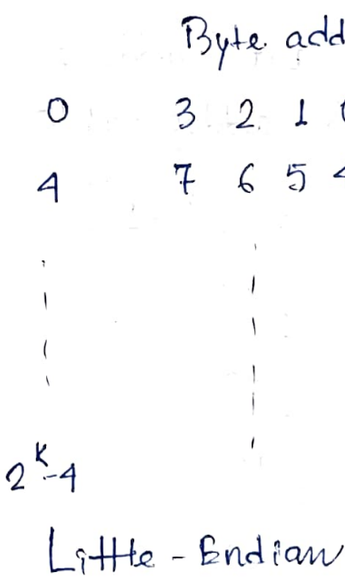
* Big-Endian & Little-Endian Assignments.

Big-Endian ~ Lower byte addresses are used for the more significant bytes.

Little-Endian ~ Lower byte addresses are used for the less significant bytes of the word.



Big-Endian



Little-Endian

* Word Alignment :

→ Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word.

→ eg. If word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4.

For a word length of 64, aligned words begin at 0, 8, 16, ...

→ There are 2 ways to indicate the length of the string. A special control character with the meaning "end of string" can be used as the last character in the string, or a separate memory word location or processor register can contain a number indicating the length of the string in bytes.

→ Memory operations —

Load/Read/Fetch → Transfers a copy of the contents of a memory location to the processor. Memory contents remain unchanged.

Store/Write → Transfers an item of information from the processor to a specific memory location, destroying the former contents of that location.

→ A computer must have instructions capable of doing 4 types of operations —

i) Data transfer between memory & processor registers.
ii) Arithmetic & logic operations on data
iii) Program sequencing & control
iv) I/O transfers.

* Register-transfer Notation (RTN):

$R_1 \leftarrow [LOC]$

contents of memory location LOC are transferred into processor register R_1.

$R_3 \leftarrow [R_1] + [R_2]$

Adds the contents of registers R_1 & R_2 and then places their sum into R_3.

RHS of an RTN always denotes a value & LHS name of location, where the value is to be placed.

* Assembly-Language Notation

```
MOVE  LOC, R1
```

Transfer data from LOC to R1.

```
ADD  R1, R2, R3.
```

Add R1 & R2's contents & place sum to R3.

* Elements of instruction:

Operation code ~ specifies operation to be performed (eg. add, move), specified by bin code.

source operand reference - may involve one or more operands as source.

result operand reference - may involve one or more results from operation.

next instⁿ reference - tells the CPU where to fetch the next instⁿ from after execution of instⁿ is complete.

- Source & result operands can be in one of these areas —
  main or virtual memory,
  CPU register, I/O device.

* Steps involved in instruction execution :

i) Instruc^{n} address calcul^{n} (iac) – Determine the address of the next ins^{n} to be executed.

ii) Ins^{n} fetch (if) – read ins^{n} from its memory loc^{n} into the processor.

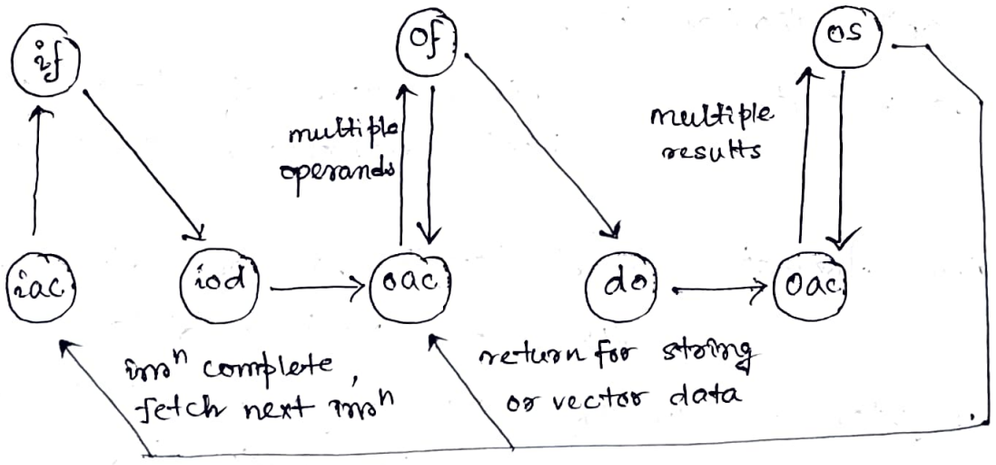
iii) Ins^{n} op.^{n} decoding (iod) – analyse ins^{n} to determine type of op^{n} to be performed & operands to be used

iv) Operand address calculation (oac) – determine the address of operand to be used.

v) Operand fetch (of) – fetch the operand from memory or read it in from I/O.

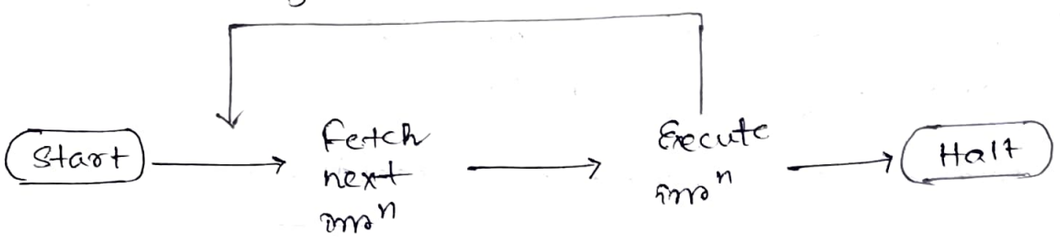
vi) Data operation (do) – perform the operation indicated on the ins^{n}.

vii) Operand store (os) – Write the result into memory or out to I/O.



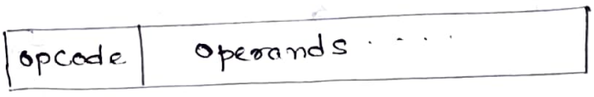
States on the upper part involve exchange between the processor & either memory or an I/O module.
States on the lower part involve only internal processor operations.

Basic instⁿ cycle –



- Each instⁿ represented as a sequence of bits; having been divided into different fields.

For a 16 bit CPU, where 4 bits are used to provide operation code, we have $2^4 = 16$ different set of instⁿs.



- Opcodes represented as mnemonics.
- Instⁿ types – data processing, data storage, data movement, control.
- In practice, most instⁿs have one, two or three operands addresses, with the address of the next instⁿ being implicit (obtained from PC).

* Instⁿ set design: Important design issues –

i) operation repertoire – how many, which opⁿs to provide & how complex opⁿs should be.

ii) data types – various types of data upon which operations are performed.

iii) instⁿ format – instⁿ length, no. of addresses, size of various fields.

iv) registers – no. of CPU registers that can be referenced by instructions.

v) addressing – addressing modes for operands.

⇝ Machine ins^{n}s are commands or programs written in machine code of a machine that it can recognize & execute.

The collection of machine ins^{n}s in main memory is called a machine language program.

⇝ Different types of instructions ~

a) Data transfer - move, load, exchange, input, output.

mnemonics - MOV, IN, OUT, LEA (load eff. address), LDS (load pointer using data segment), LES (load pointer using extra segment), PUSH, POP, XCHG, XLAT (translate byte using look-up table).

b) Arithmetic instructions - add, subtract, increment, decrement, convert, compare.

mnemonics - ADD, SUB, ADC, SBB, INC, DEC, NEG, CMP, MUL, DIV, IMUL, IDIV (integer), CBW (convert byte to word), CWD (convert word to double), AAA (ASCII adjust for add), AAS, AAM, AAD, DAA (decimal adjust For addition), DAS.

c) Logic instructions - AND, OR, shift, rotate, test.

mnemonics - NOT, AND, OR, XOR, TEST, SHL, SHR, SAL, SAR (arithmetic), ROL, ROR, RCL, RCR (through carry byte).

d) String manipulation - Load, store, move, compare.

mnemonics - MOVS, MOVSB, MOVSW, CMPS, SCANS, LODS, STOS

e) Control Transfer - conditional, unconditional.

JMP - unconditional jump

JNZ - jump till the counter value decreases to zero.

f) Control of loop : LOOP (loop unconditional), LOOPE (loop if equal), LOOPNE (loop if not equal), JCXZ (jump if CX = 0), CALL, RET (return from procedure), INT (interrupt), INTO (interrupt if overflow), IRET (return from interrupt).

g) Processor control : flag manipulation.

STC (set), CLC (clear), CMC (complement carry flag) STD (set dstn flag), CLD (clear dstn flag), STI, CLI (Clear interrupt enable flag), PUSHF, POPF.

~> Difference of CALL & JUMP instn ~

JUMP	CALL
i) Jump to the destination & execution carries on from there without bothering to come back later to the instn after the JUMP.	i) We jump to the subroutine pushing current instn pointer to the stack & execution carries on from there till the Return (RET) is executed in the subroutine.
ii) Program control is transferred to a memory location that is in the main program.	ii) Transferred to a mem. loc^n that is not a part of main program.
iii) Immediate addressing mode.	iii) Immediate + register indirect.
iv) Initialisation of SP is not mandatory.	iv) Mandatory.
v) Value of PC is not transferred to stack.	v) Transferred
vi) After jump, there is no RET instn.	vi) After CALL, there's a RET instn.
vii) Value of SP doesnot change.	vii) Decremented by 2.
viii) 10 T states required. to execute this instn.	viii) 18 T.
ix) 3 machine cycles.	ix) 5 machine cycles.

* Types of operands ~

addresses, numbers, characters, logical data.

* Types of operations —

data transfer, arithmetic, logical, conversion, I/O, system control, transfer control.

- Transfer of Control :

i) Branch instⁿ : Also called as Jump instⁿ, has one of its operands as the address of the next instruction to be executed.

2 types of branching — conditional, unconditional.

BRP	X	Branch to location X if result is	+ve
BRN	X	" " "	-ve
BRZ	X	" " "	zero
BRO	X	" " "	overflow occurs.

Another way, 3 address instⁿ.

BRE R1, R2, X Branch to X if [R1] = [R2]

ii) Skip instⁿ : One instⁿ to be skipped.

eg. ISZ R1 If result of the increment of value of R1 is zero, skip the next instⁿ.

iii) Procedure Call instⁿ : Procedure - a self contained program incorporated into a large program & can be invoked at any time of other program execution. Processor returns to the original program after executing procedure.

2 basic instⁿs —

a) a call instⁿ that branches from the present locⁿ to the procedure,

b) return instⁿ that returns from the procedure to the place from which it was called.

Both a & b are branch instⁿ.

A procedure can be called from more than one loc^{n}. A procedure call can appear in a procedure. Each procedure call is matched by a return in the called program.

Places for storing return address –
register, start of procedure, stack top

* Instn types examples.

i) Data transfer : MOVE, LOAD, STORE, PUSH, POP, XCHG (swap contents of source & destination), CLEAR (Reset destn with all 0), SET (set destn with all 1).

ii) Arithmetic : ADD, ADC (add with carry), SUB, SUBB (subtract with borrow), MUL, DIV, NEG, INC, DEC, SHIFT A (Shift arithmetic)

iii) Logical : NOT, OR, AND, XOR, SHIFT, ROT (rotate, shift with wrap-around), TEST

iv) Control transfer : JUMP, JUMPIF, JUMPSUB, RET (return), INT (interrupt), IRET (Interrupt return), LOOP

v) I/O instn : IN (read data from specified input port), OUT (write data into an output port), TEST I/O (read status from I/O subsystem & set condition flags), START I/O (inform I/O processor to start the i/o program), HALT I/O (inform I/O processor to abort the i/o program).

vi) String manipulation : MOVS, LODS, CMPS (compare), STOS, SCAS (scan)

vii) Translate : XLAT (translate by table lookup), PACK (convert unpacked dec. no to packed), UNPACK

viii) Transfer control : HLT (halt, stop instⁿ cycle), STI (EI) - (set interrupt), CLI (DI) - (clear interrupt), WAIT (freeze instⁿ cycle), NOOP (no operaⁿ), ESC (escape), LOCK (reserve the bus till the next instⁿ is executed), CMC (complement carry flag), CLC (clear carry flag), STC (set carry flag).

* 0-3 operand instⁿs :

i) 3 address instⁿ format : reference to 3 operands. uncommon because long, need multiple words in memory, multiple operand fetch cycle required. No. of instⁿs less.

Syntax - Opcode desⁿ S1, S2
(S2 : source)

ADD R1, 3030, #5.

ii) 2 address instⁿ format : One of the operands corresponds to both source & result, smaller length - more temp. storage.

Syntax -
Opcode source/desⁿ source.

eg. ADD R1, 3030.

iii) 1 address instⁿ - First operand - implicitly accumulator, all operations b/w AC & operand. no. of instⁿs high.

Syntax - opcode source/desⁿ

eg. ADD 3030.

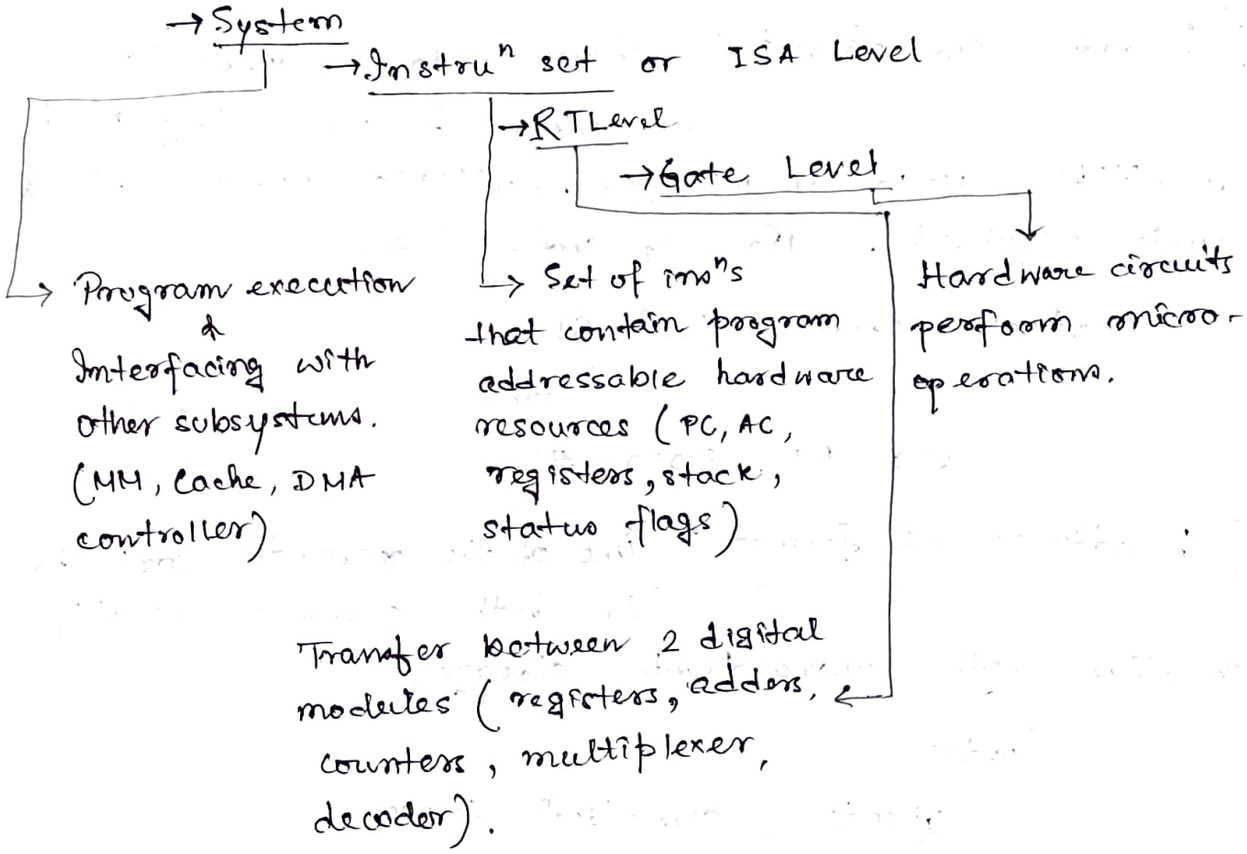
iv) 0 address instⁿ - Source, desⁿ both implicit & stored in stack ; absolute address of operand is held in a special register that points to top of stack, no. of instⁿs higher.

Syntax - opcode eg. ADD top 2 locⁿs of stack & writes back on top of stack.

eg. $Y = (A+B)/(C*D)$.

ADD Y, A, B	MOV Y, A	LOAD C
MUL Z, C, D.	ADD Y, B	MUL D
DIV Y, Y, Z	MOV Z, C	STORE Y
	MUL Z, D	LOAD A
3 ad. ins^n	DIV, Y, Z	ADD B
	2 ad. ins^n	DIV Y
		STORE Y.
		1 ad. ins^n.

* Processor - 4 levels :



* Datapath organisation :

Datapath includes - ALU, registers for temporary storage, digital circuits for executing different micro operations, internal path for ~~movement~~ movement of data between ALU & registers, driver circuits for transmitting signals to external ~~bus~~ units, receiver circuits for incoming signals.

- **Hardware - Software Interface :**

The status flags & control flags provide a means of comm^n b/w the hw & sw.

Status flags - Overflow, carry, sign flag.

Control flags - control processes operation.

Control + Status flags → flags register or PSW (Program status word)

TRAP flag - Informs processor to work in single step mode. Used for debugging. After each ins^n an internal interrupt.

INTERRUPT flag - honoring the hardware interrupt.

DIRECTION flag - used for string manipulation.

- Considerations for finalising instruction set :

i) Programming convenience
ii) Powerful addressing
iii) More no. of GPRs
iv) Target market segment
v) System performance.

- **Classification of ISA :**

i) Accumulator based CPU.
ii) Registers based CPU
iii) Stack based CPU.

- **Instruction Length :**

If the ins^n is too long,
i) Ins^{n}s occupy more memory space.
ii) Either the data bus width has to be large or ins^n fetch will take more time.

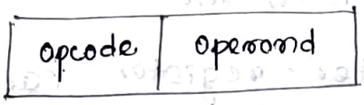
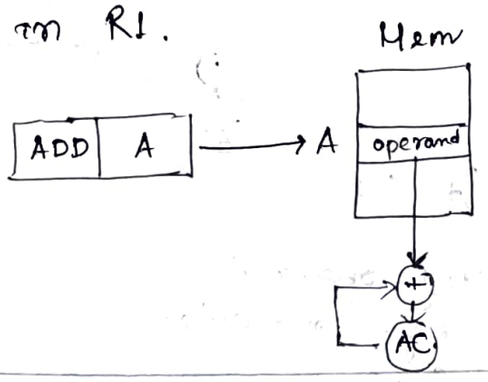
If the ins^n is too short,
i) There are too many ins^{n}s in the program.
ii) Program size increases.

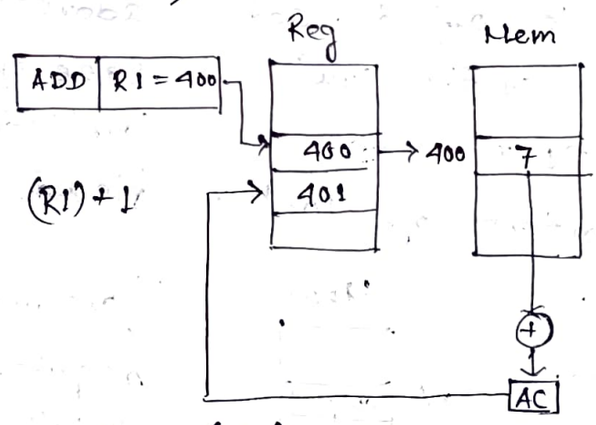
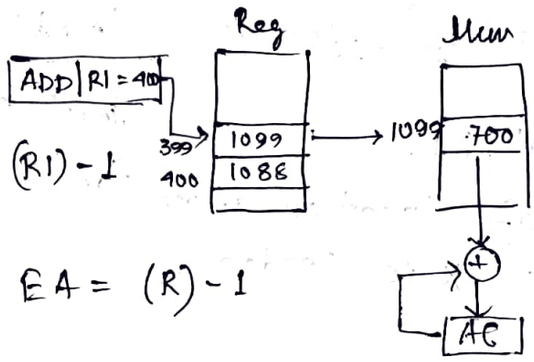
→ Variable length ins's give the programmers flexibility & save memory space.

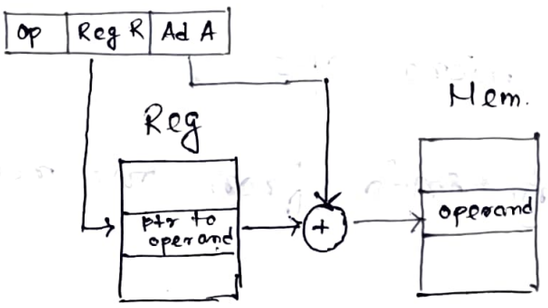
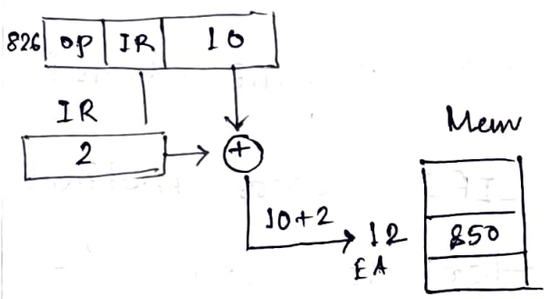
- Macro ~ A routine that can be invoked in any place in a program by just naming it. It is an independent subprogram with some parameter input whose value has to be supplied at the place of invoking the macro.

- Subroutine ~ A program to which the CPU temporarily branches to perform a special function.

* Addressing Modes.
  - Refers to the way in which the operand of an ins^n is specified.
  - Multiple addressing modes give flexibility to the programmer in writing efficient programs.
  - Addressing mode is indicated to the CU in any of the following ways -
    1. A separate mode field in the ins^n indicates the addressing mode used.
    2. The opcode itself explicitly specifies the mode.
  - Effective address / offset - offset determined by adding any comb^n of 3 address elements - displacement, base, index.
    - base: contents of base register (BX or BP)
    - index: content of index register (SI/DI)

Addressing Mode	Mechanism	Remarks.
i) Immediate (Symbol #)	Operand is the immediate value stored in the instruction itself. ADD #26, R1  	Operand is available in the CPU as soon as the ins^n fetch is over, hence ins^n cycle winds-off fast. Value of operand limited by the size of address field.
ii) Implied/ implicit/ inherent	Operand is specified implicitly in the ins^n itself. CMA - complement accumulator - operand specified in ins^n. RLC - rotate content of AC.	Zero address ins^{n}s are designed with this mode in stack organised computer.
iii) Direct/ Absolute (Symbol [])	Operand is in memory; operand's offset - address is given in the ins^n. LOAD R1, X loads contents of loc^n X in R1.  	No need for operand address calculation. Size of operand address limited by operand field. (one mem. reference required to access data)
iv) Indirect (symbol @ or ())	Address field of ins^n contains the address of effective address. MOVE (X), R1. contents of location whose address is X is loaded into R1.	Flexible for programming (implementing pointers) Ins^n cycle time increases for 2 memory accesses.

Addressing Modes	Mechanism	Remarks.
	Based on availability of EA, 2 kinds - register indirect - EA is in register, register name given in instⁿ memory indirect - EA is in memory, mem. address given in instⁿ.	reg. ind - one reg. ref, one mem. ref. mem. ind. - 2 mem. ref's
v) Register direct	Operand in register; register address given in instⁿ. ADD R1, R2 One reg. ref. required to access data.	Faster operand fetch without memory access; No. of registers is limited & hence effective utilisation is essential.
vi) Auto increment/ decrement	EA is in the register; after accessing operand contents of register are auto-incremented to point to the next memory locⁿ. (R+) EA = (R)  ADD (R2)+, R0 Before accessing operand contents of reg. are auto-decremented. (-R)  EA = (R) - 1	Can be used to implement as push, pop stack. Useful for LIFO data st.s. (One register ref, one mem. ref & one ALU opⁿ required to access data).

Addressing Modes	Mechanism	Remarks.
vii) Displacement based addressing (Comb^n of direct & reg. ind.)	Needs the ins^n to have 2 address fields - at least one of which is explicit & other indirect.  3 types – a) Relative ~ Content of PC is added to the address part of ins^n to obtain EA. EA = A + PC. b) Index register ~ Content of index reg. is added to direct address part. EA = A + Index  c) Base register ~ Content of base reg. is added to direct address part. EA = A + base	Relative – smaller no. of bits in address field. Base – operand ad. field is very short since it gives offset only; used for relocation of programs in the memory (Useful for multiprogramming systems).
viii) Stack addressing	Operands taken from the top of stack. ADD adds two popped operands of stack & push into stack. (Implied)	No operand field in the ins^n, hence shorter ins^n.

- Micro-operations.
    1. Register-transfer Micro-ops
    2. Arithmetic micro-ops
    3. Logic micro-ops
    4. Shift micro-ops

1. transfer binary info. from one reg. to another.
2. perform arith. ops on numeric data stored in registers.
3. perform bit manipulation ops on non-numeric data stored in registers.
4. perform shift ops on data stored in regs.

* Stack Based CPU Organisation:

→ Uses LIFO access method.

→ A register is used to store the address of top of the stack (Stack Pointer SP).

→ Operations done on the operands stored in the stack using Push & Pop.

Push	Pop
SP ← SP - 1	(mem. address) ← SP
SP ← (mem. address)	SP ← SP + 1

→ Instruction does not need any address field as the operands are on the stack.

→ Stack organised computers –

PDP-11, Intel's 8085, HP 3000.

→ Advantages :

i) Efficient computation of complex arithmetic expressions.

ii) Execution of ins^{n}s is fast because operand data are stored in consecutive memory locations.

iii) Length of address field is 0 here, so instruction size is short.

→ Disadvantage :

Size of the program increases.

→ Stack-based CPU organisation uses zero address instruction.

* General Register Based CPU Organisation

→ We use multiple general purpose registers on the CPU in this kind of organisation.

→ Two or three address fields in the instruction format.

eg. MUL R1, R2, R3

→ Use of a large number of registers results in short program with limited instructions.

→ Examples of this organisation -

IBM 360, PDP-11.

→ General register based CPU organisation has two types:

i) Register - memory reference architecture (CPU with less register) ~

Source 1 is always required in register & source 2 can be present either in memory or register. 2 address ins^{n} is the compatible format.

ii) Register - register reference architecture (CPU with more register) ~

ALU operations done only on a register data. So, operands are required on the register. After manipulation, result is also stored in the register.

3 address ins^{n} is the compatible format.

→ Advantages: i) Efficiency of CPU increases as there are large numbers of registers that are used.

ii) Less memory space is used to store the program since the ins^{n}s are written in compact way.

→ Disadvantages:

i) Care should be taken to avoid unnecessary usage of registers.

ii) Extra register cost.

* Single Accumulator based CPU organisation:

→ Accumulator register is used implicitly for processing all instns of a program & store the results on the accumulator.

→ Instn format has one address field. Address field has the address of the operand.

→ The first operand is always stored in the AC & the second operand is present either on registers or on the memory.

→ 2 types of operations are performed in this organisation ~

i) Data transfer:

LOAD X AC ← (X)

STORE Y: Y ← (AC)

ii) ALU operation:

MUL X AC ← (AC) * (X).

→ PDP-8 processor used this organisation.

→ Advantages ~

i) Short instn size. ii) Instn cycle takes less time. (as fetching takes m")

→ Disadvantages ~

i) Memory size increases for complex expressions because of many short instns, ii) exen time increases.

* Opcode size = $\log_2$ (Instⁿ set size)

↓

total no. of instⁿs defined on the processor

Q. G'16 Consider a processor with 64 registers and an instⁿ set of size twelve. Each instⁿ has 5 distinct fields, namely opcode, 2 source register identifiers, 1 destination register identifier & a twelve-bit immediate value. Each instⁿ must be stored in memory in a byte-aligned fashion. If a program has 100 instⁿs, the amount of memory (bytes) consumed by the ~~the~~ program text is —

100/200/400/500

→ Opcode size will be 4 bits.

[$2^4 = 16$, instⁿ set size = 12]

For identifying a register $\log_2 64 = 6$ bits are required.

Now, (3×6) = 18 bits required for register identifiers (2 src, 1 dsn).

Total bits for an instⁿ =

(4 + 18 + 12) = 34 bits

For 100 instⁿs, no. of bits = 3400 bits

= 425 bytes.

So, answer is 500 B (> 425 B).

Q'G'16 A processor has 40 distinct instⁿs & 24 general purpose registers. A 32-bit instⁿ word has an opcode, 2 register operands and an immediate operand. The number of bits available for the immediate operand field is ___

→ No. of bits for opcode = 6

$(2^6 = 64 > 40)$.

No. of bits for identifying a register = 5

$(2^5 = 32 > 24)$.

Total bits for opcode & 2 register operands

$= (6 + 2 \times 5) = 16$.

No. of bits for immediate operand $= (32 - 16)$

$= 16$ bits.

Q'G'14 A machine has a 32-bit architecture, with 1 word long instructions. It has 64 registers each of which is 32 bits long. It needs to support 45 instⁿs, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer, the maximum value of the immediate operand is ___.

→ 32 bit architecture implies,

1 word = 32 bits = instⁿ size.

No. of bits for identifying 1 register = 6 $[2^6 = 64]$

No. of bits for opcode = 6 $[2^6 > 45]$

Bits for immediate operand $= 32 - (6 + 2 \times 6)$

$= 14$ bits

Max value of the unsigned integer $= 2^{14} - 1$

$= 16383$.

Q G'18 A processor has 16 integer registers (R0, ... R15) & 64 floating point registers (F0, ..., F63). It uses a 2 byte instr format. There are 4 categories of instructions: T-1, T-2, T-3, T-4. T-1 category consists of 4 instructions, each with 3 integer register operands (3Rs). T-2 category consists of eight instrs each with 2 floating point register operands (2Fs). T-3 category consists of 14 instrs, each with one integer & one FP register operand (1R + 1F). T-4 category consists of N instrs, each with a floating point register operand (1F). Max value of N ___

→ 2 byte instr format,

∴ possible encodings = 2^{16}.

No. of bits for one integer register identifier = 4 ($2^4 = 16$)

No. of bits for one FP register identifier = 6 ($2^6 = 64$)

T-1 category -

No. of encodings = $4 \times 2^{(4 \times 3)} = 2^{14}$

T2 category -

No. of encodings = $8 \times 2^{(6 \times 2)} = 2^{15}$

T-3 category -

No. of encodings = $14 \times 2^{(6+4)} = 14336$

T-4 category -

$N \times 2^6$

Now,

$N \times 2^6 = 2^{16} - 2^{14} - 2^{15} - 14336 = 2048$

$\Rightarrow N = 32$ (Ans).

Q G'17

```
struct data {
    int marks[100];
    char grade;
    int cnumber;
};
struct data student;
```

Base address of student is available in register R1. the field student.grade can be accessed efficiently using

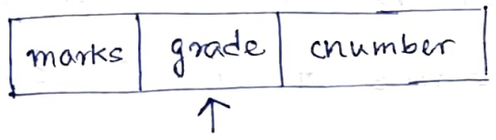
a) Post-increment addressing mode (R1)+

b) Pre-decrement addressing mode -(R1)

c) Register direct addressing mode R1

d) Index addressing mode, X(R1), where X is an offset represented in 2's complement 16-bit representation.

→



Ans. — (d). base address + index.

Q G'17 Consider a RISC machine where each instn is exactly 4 bytes long. Conditional & unconditional branch instructions use PC-relative addressing mode with offset specified in bytes to the target location of the branch instruction. Further the offset is always with respect to the address of the next instruction in the program sequence. Consider following instruction sequence -

instn No.	Instn
→ i :	add R2, R3, R4
i+1 :	sub R5, R6, R7
i+2 :	cmp R1, R9, R10
i+3 :	beq R1, offset (branch if equal)

If the target of the branch instn is i, then the decimal value of the offset is — (-16) (Ans)

```
1000 : i
1004 : i+1
1008 : i+2
1012 : i+3
(1016):
```

→ EA = PC + value

Q G'15 For computer based on three-address instⁿ formats, each address field can be used to specify which of the following:

(S1) : A memory operand

(S2) : A processor register

(S3) : An implied accumulator register.

a) Either S1 or S2 c) Only S2 & S3

b) Either S2 or S3 d) All of S1, S2, S3.

Q G'11 Consider a hypothetical processor with an instruction of type LW R1, 20(R2) which during execution reads a 32-bit word from memory & stores it in a 32-bit register R1. The effective address of the mem. locⁿ is obtained by the addition of a constant 20 & the contents of register R2. Which of the following best reflects the addressing mode implemented by this instⁿ for operand in memory?

a) Immediate c) Register indirect scaled

b) register d) Based indexed

Q G'08 Assume that EA = (X)+ is the effective address equal to the contents of location X, with X incremented by one word length after the effective address is calculated; EA = -(X) is the ea equal to the contents of the location X, with X decremented by one word length before the effective address is calculated; EA = (X)- is the ea equal to the contents of location X, with X decremented by one word length after the ea is calculated. The format of

the instruction is (opcode, src, dsn), which means dsn ← src op dsn. Using X as a stack pointer, which of the following can pop the top two elements from the stack, perform the addition & push result back to the stack.

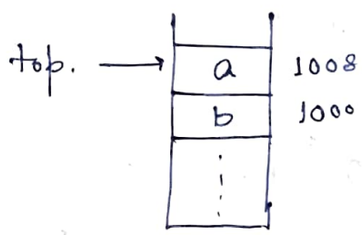
a) ADD (X)-, (X)

b) ADD (X), (X)-

c) ADD -(X), (X)+

d) ADD -(X), (X)

→ ADD [1008], [1008]



EA first - 1008

EA second - 1000

(X)-

[1008] + [1000]

↓

[1000] store.

Q G'06 The memory locations 1000, 1001, & 1020 have data values 18, 1 & 16 respectively before the following program is executed -

```
MOVI   Rs, 1          ;  Move immediate
LOAD   Rd, 1000(Rs)   ;  Load from memory
ADDI   Rd, 1000       ;  Add immediate
STOREI 0(Rd), 20      ;  Store immediate.
```

Which is true after it is executed?

a) mem. loc^n 1000 has 20

b) " " 1020 has 20

c) " " 1021 " 20

d) " " 1001 " 20.

1000 - 18

1001 - 1

1020 - 16

① [1] Rs ② Rd ← M[1000 + [Rs]]

M[1000 + 1]

Rd ← 1 [1] Rd

③ [1001] Rd ④ M[0 + [Rd]] ← 20

M[1001] ← 20

[1001 - 20]

Q G'06 Which is false about relative addressing mode?

a) It enables reduced mm size (True)

b) It allows indexing of array elements with same mm. (True)

c) It enables easy relocation of data (True)

d) It enables faster address calculations than absolute addressing. (PC + value)

Q G'05 Consider a 3 word machine mm.

** ADD A[R0], @B.

The first operand (dsn) A[R0] uses indexed addressing mode with R0 as the index register. The second operand (source) @B uses indirect addressing mode. A & B are memory addresses residing at the 2nd & the 3rd words, respectively. First word is the opcode, the index register designation, & the source & destination addressing modes. During execution of ADD mm, the two operands are added & stored on the destination (first operand). No. of memory cycles required during the execution cycle is -

a) 3

b) 4

c) 5

d) 6

2 + 3 + 1

- If — 1 memory reference (opcode)
- ID — 2 memory references (2 operands)
- OF — M[A + [R0]] 1 mr; @B 2 mr } 3 mr
- Operation on data — 1 ALU
- Write — 1 mr.

Input - Output Organisation.

* I/O Interface : Method that is used to transfer information between internal storage & external I/O devices.

There exists special hardware components between CPU & peripherals to supervise & synchronise all the i/p & o/p transfers that are called interface units.

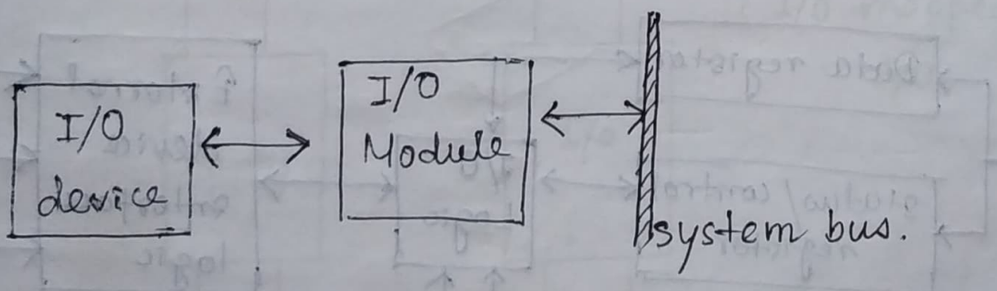
* Reasons for not connecting peripherals directly to the system bus :

i) It's impractical to include different logics for different peripherals on the processor.

ii) It's impractical to use high speed system bus to communicate directly with slow peripherals.

iii) Peripherals often use different data formats & word lengths than the computer.

Thus we use I/O module that interfaces to the system bus & controls one or more peripherals.



* Major functions of an I/O module :

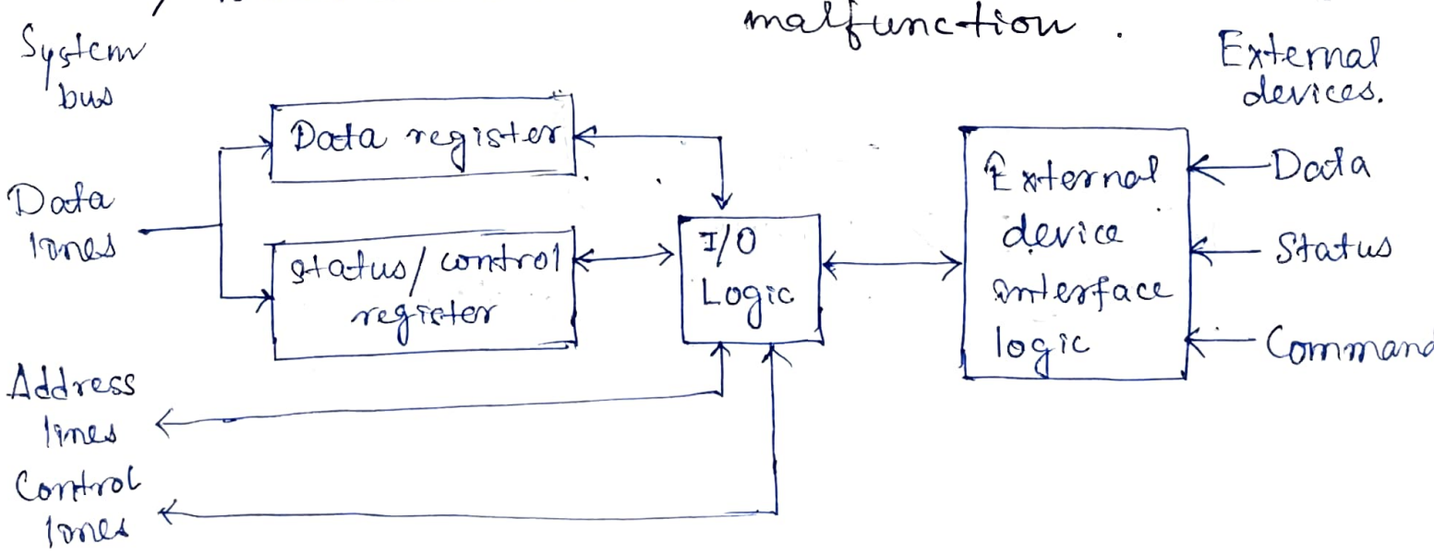
i) Control & timings : Coordinates the flow of traffic between internal & external resources.

ii) Processor communication : Communicate with processor during I/O operation. It involves command decoding, data exchange, states reporting (BUSY, READY), address recognition.

iii) Device communication : Involves command, status information and data exchange.

iv) Data buffering : Due to mismatch of the speed of CPU & peripheral devices. I/O module stores data in the data buffer to regulate data transfer at device's speed.

v) Error detection : Mechanical & electrical malfunction.

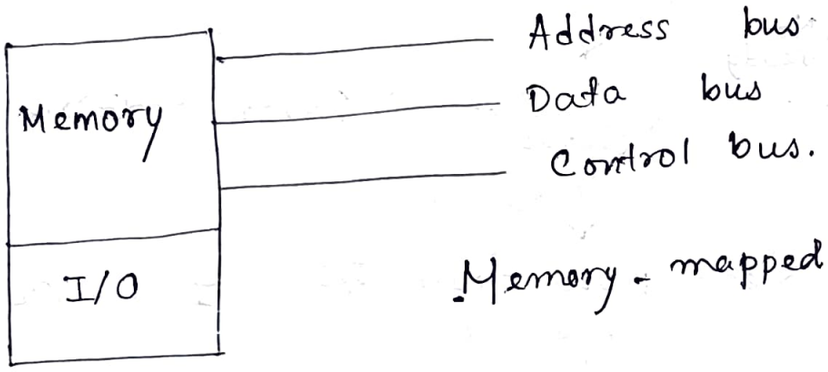


* Memory-mapped I/O : There is a single address space for memory locations and I/O devices. The processor treats the status & address register of I/O module as memory location. Processor & I/O are mapped using the memory address.

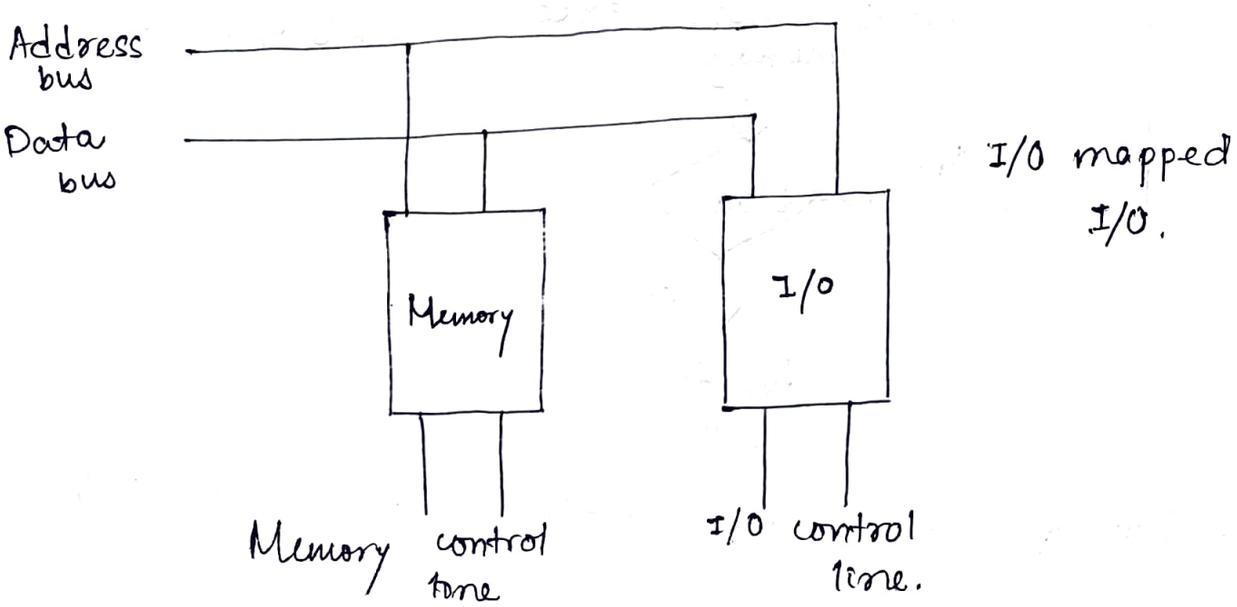
Available addresses for memory are less.

* Isolated or I/O mapped I/O :

Address space of memory & I/O is isolated. Memory & I/O has separate address space. All address can be used by the memory. I/O addresses are called ports. This is more efficient due to separate buses.



Memory-mapped

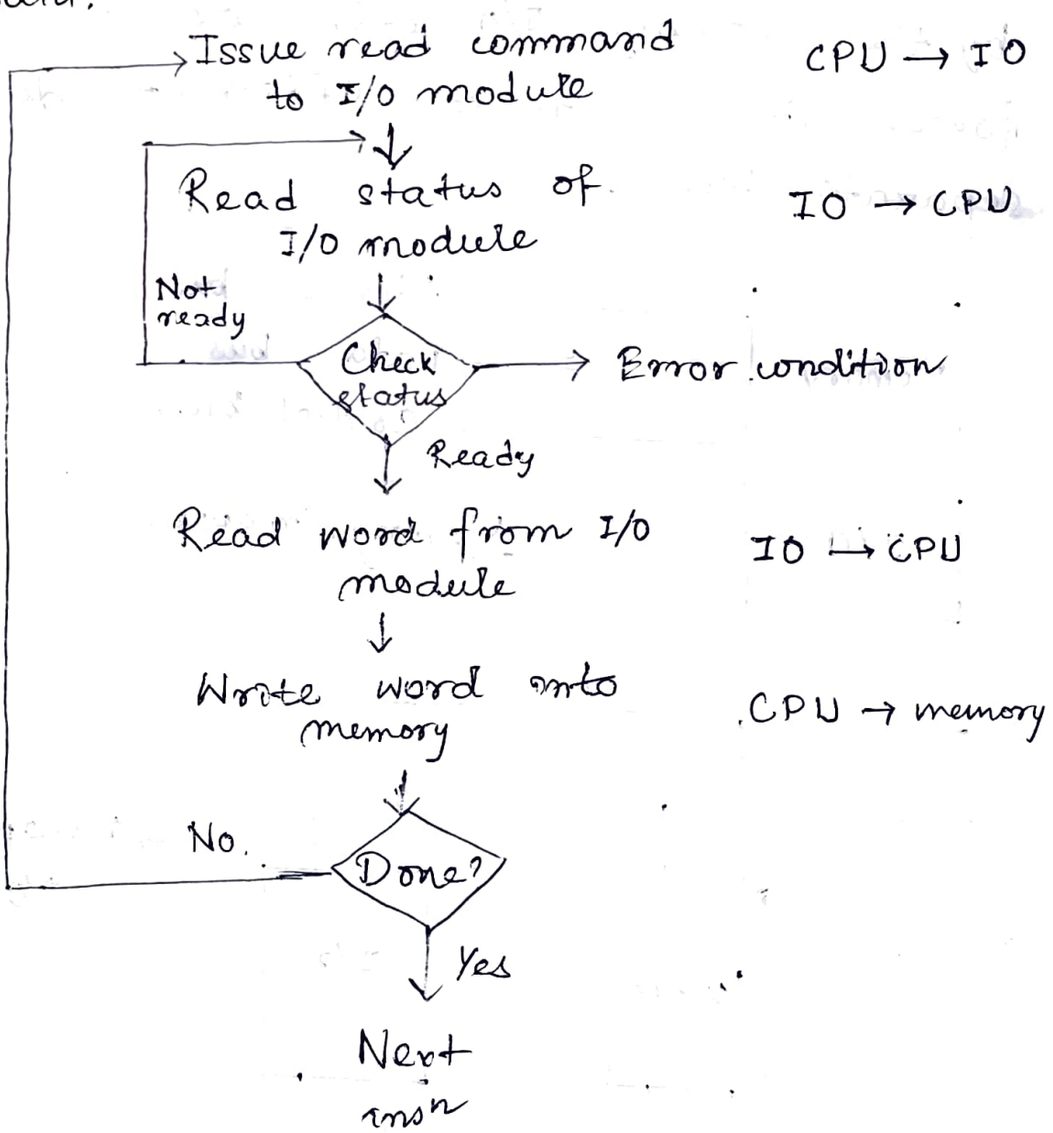


I/O mapped I/O.

* Mode of transfer : Data transfer between CPU and the I/O devices may be done in three ways -

1. Programmed I/O
2. Interrupt driven I/O
3. Direct Memory Access.

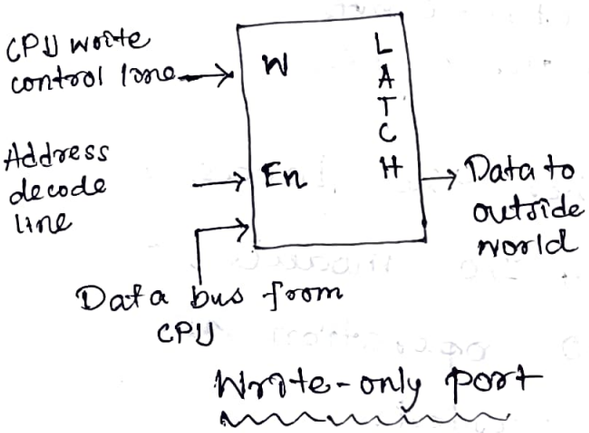
* Programmed I/O : The processor executes a program that gives its direct control of the I/O operation, including sensing device status, sending a read or write command & transferring the data.



Input of block of data. on Programmed I/O.

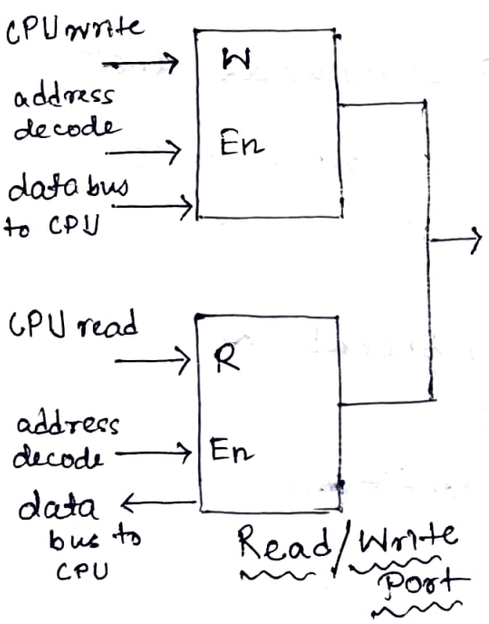
→ I/O Port : Device that looks like a memory cell but contains connections to the outside world. I/O port uses a latch. When the CPU writes to the address associated with the latch, the latch device captures the data & makes it available on a set of wires external to the CPU & memory system.

I/O ports can be read-only, write-only, or read/write.



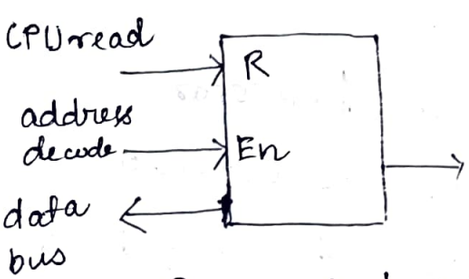
Write-only port

1) CPU places the address of the device on the I/O address bus & with the help of address decoder a signal is generated that will enable latch.
2) CPU indicates write operation by control line.
3) Data will be placed on the CPU bus, for onward transmission.



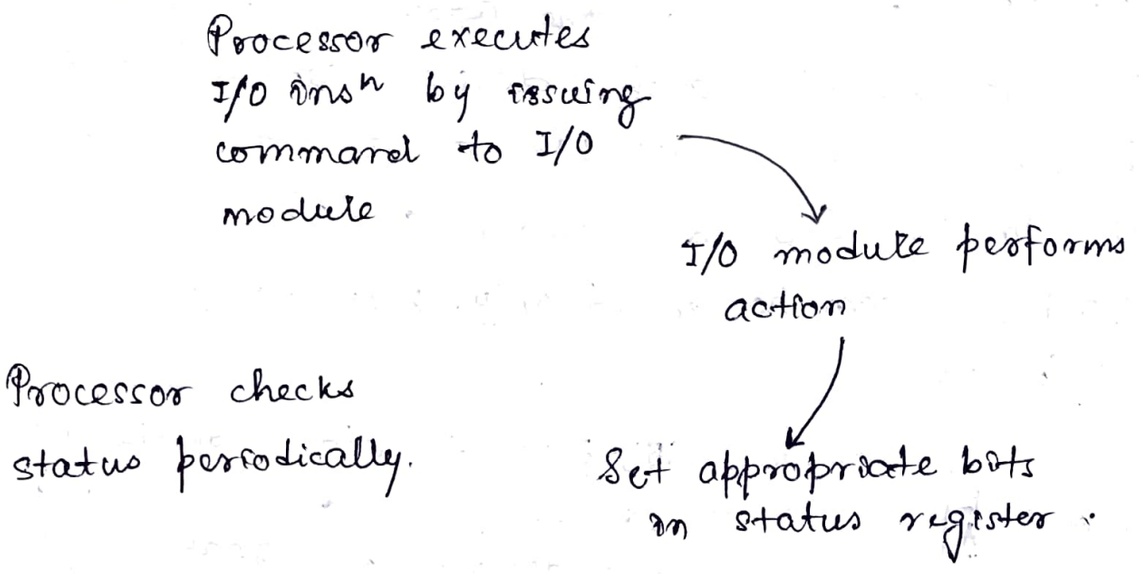
Read/Write Port

1) Read or write signal is generated using address decoding.
2) Data placed on latch (write) or transferred to CPU (read).



Read-only port

→ In programmed I/O, the data transfer between CPU & I/O device is carried out with the help of a software routine.



→ In programmed I/O, when the processor issues a command to an I/O module, it must wait until the I/O operation is complete. So, CPU time is wasted.

→ There are 4 types of I/O commands that an I/O module will receive when it is addressed by a processor –

a) Control : Activate a peripheral & instructing it.

b) Test : Testing status conditions.

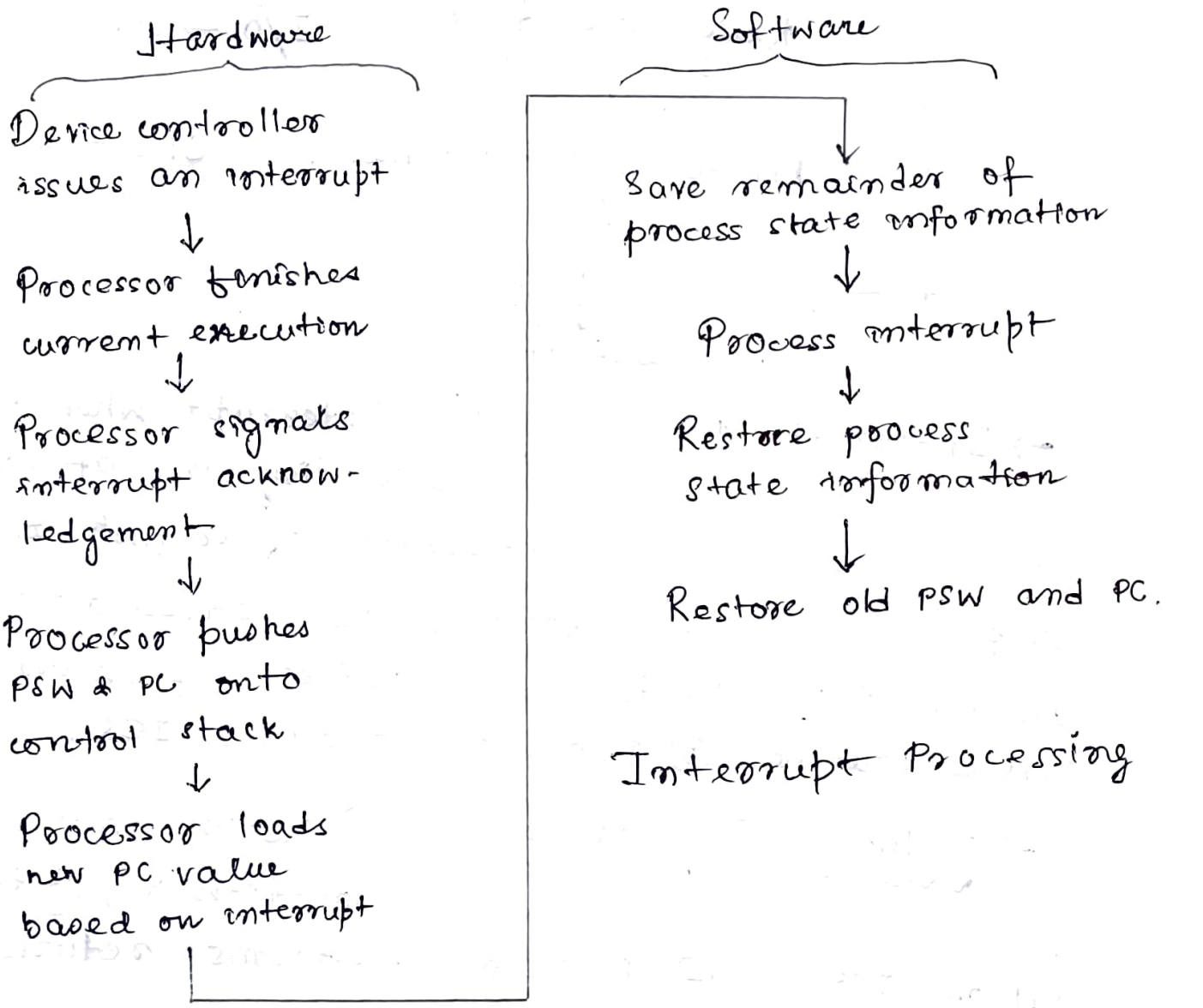
c) Read

d) Write.

→ In programmed I/O, we say processor polls the device as the processor repeatedly checks the status flag to achieve the required synchronisation between the processor & the I/O device.

* Interrupt - driven I/O :

Way of controlling input/output activity where a peripheral that needs to make or receive a data transfer sends a signal to the processor and this causes a program interrupt which is followed by a interrupt service routine performed by the processor.

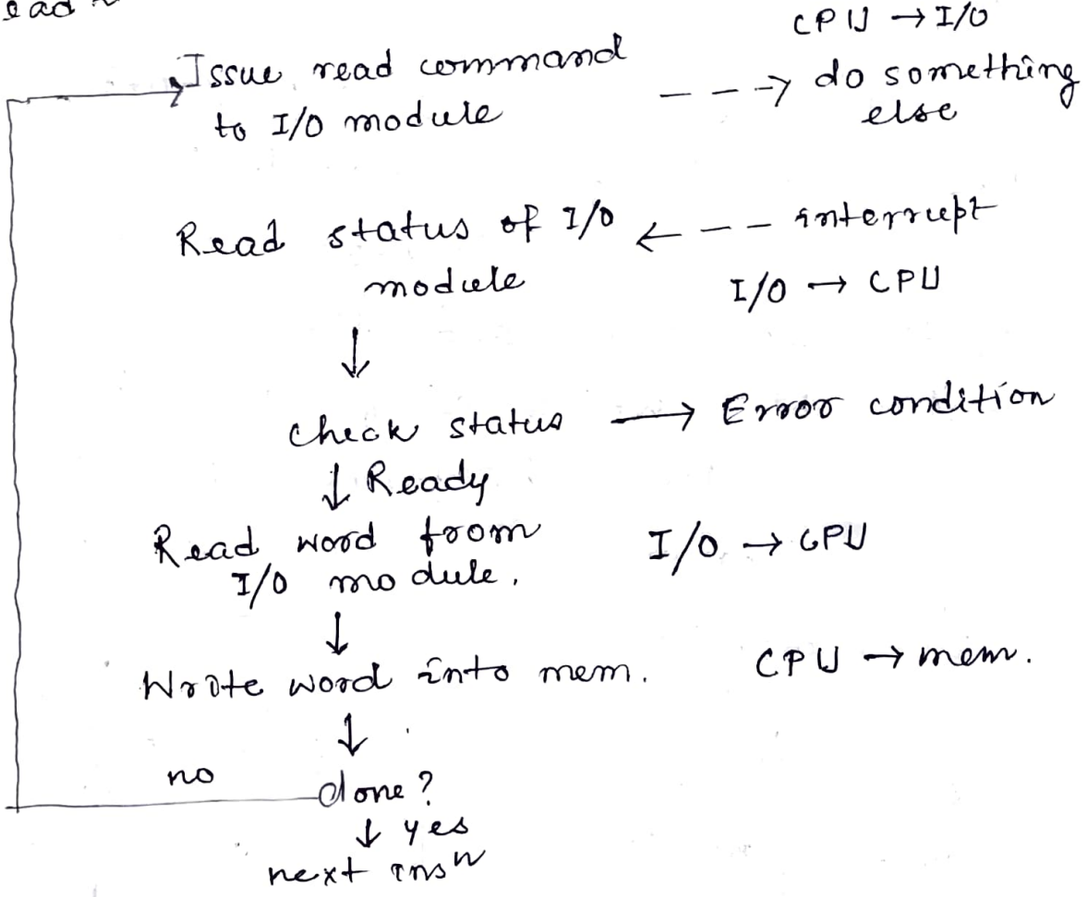


Interrupt Processing

→ Interrupt Latency ~ Delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.

To decrease interrupt latency, only processor status register's content & program counter are saved when interrupt is accepted.

→ Read ~



→ Interrupt Processing : ① Interrupt when processor executing instn of locn N. ② Processor services interrupt after instn completion. ③ Processor status word & PC saved onto the stack. ④ PC loaded with address of interrupt service routine. ⑤ Processor executes ISR.

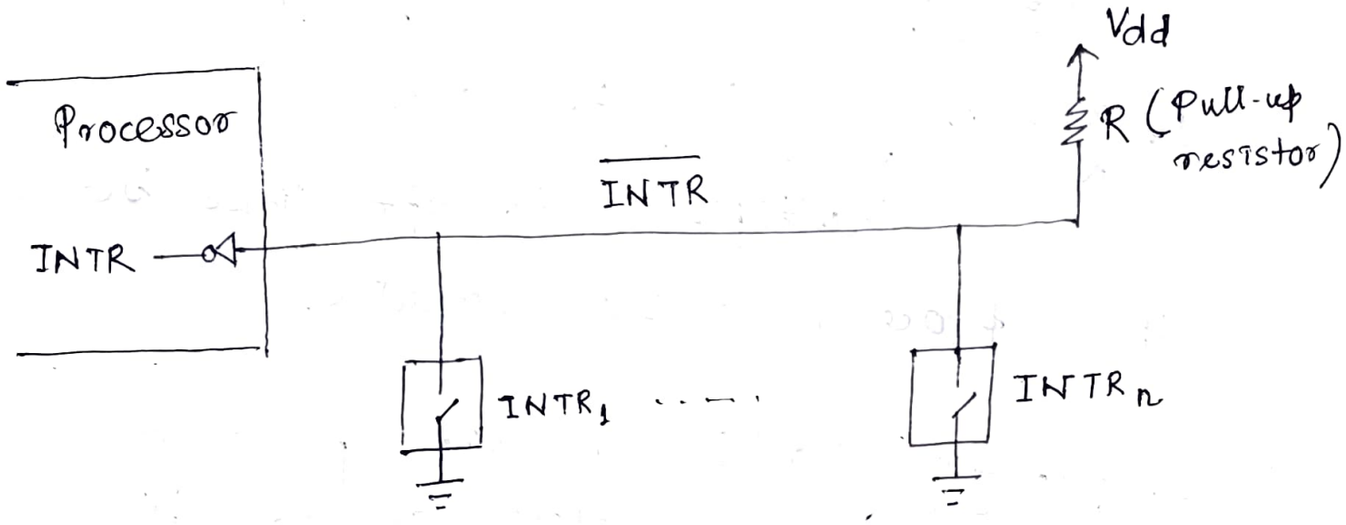
→ Return from interrupt : ① Return instn in location x, at the end of ISR. ② Processor performs return from ISR. ③ While returning, processor restores value of PC, PSW from control stack. ④ Processor starts executing user's interrupted program.

→ Interrupt handler : ① Save contents of all registers on the stack. ② Examination of status information relating to the I/O operation or other event that caused the interrupt. ③ After interrupt processing, saved register values are retrieved.

→ Interrupt Hardware :

A single interrupt-request line may be used to serve n devices. To request an interrupt, a device closes its switch. When all switches are open, voltage on line is V_{dd}. When a device requests an interrupt by closing its switch, voltage on the line drops to zero, causing the interrupt-request signal INTR to go to 1.

$$INTR = INTR_1 + \cdots + INTR_n$$



Pull-up resistor pulls the line voltage up to the high-voltage state when the switches are open.

→ Enabling - disabling interrupts :

It is important because interrupts can hamper normal flow or sequence of programs.

i) Using interrupt-disable instruction in the ISR as no interrupt will be allowed to provide interrupt request. Using interrupt-enable at the last of ISR (before 'return').

ii) Processor disabling interrupts before starting the execution of any ISR. One bit in the Processor Status register (Interrupt-enable bit) indicates whether interrupt is enabled.

iii) Processor having a special interrupt-request line for which the interrupt-handling circuit responds only to the leading age of the signal (edge-triggered).

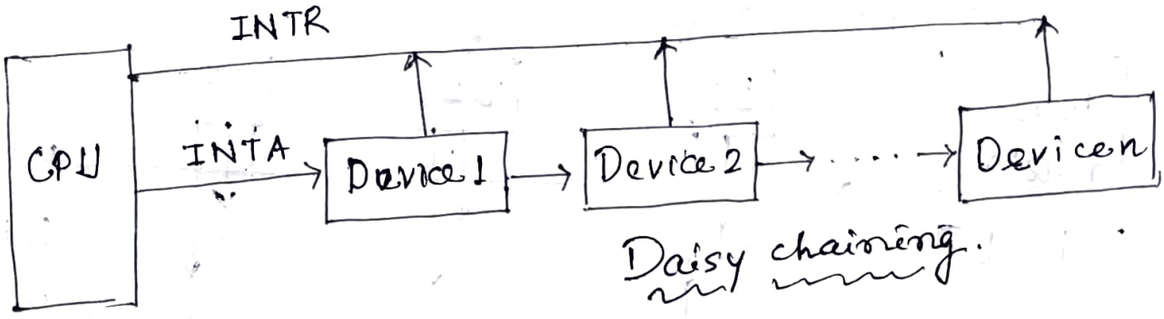
→ Device identification techniques.

1. Multiple Interrupt Lines:
Multiple interrupt lines between the processor & I/O modules.

2. Software Poll: When the processor detects an interrupt, it branches to an ISR whose job is to poll each I/O module to determine which module caused the interrupt.

3. Daisy chaining: Common interrupt request line for all I/O modules. Interrupt-acknowledge (INTA) line is connected on a daisy chain fashion. When the processor senses an interrupt, it sends out an acknowledgement.

The requesting module typically responds by placing a word on the data lines. This word is ~~reffer~~ referred to as a vector & is either the address of the I/O module or some other unique identification. (Vectored interrupt).



Daisy chaining.

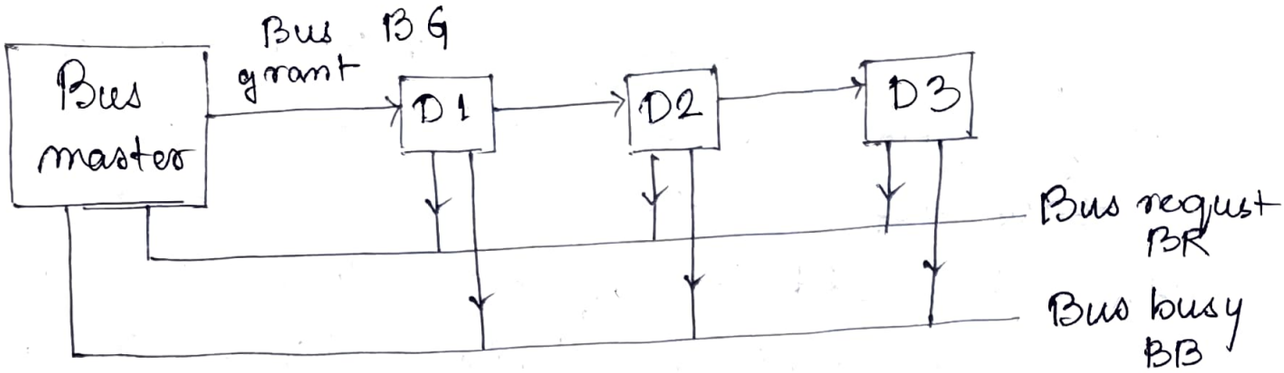
4) Bus Arbitration: I/O module must first gain control of the bus before it can raise the interrupt request lines. Arbitration refers to the process by which the current bus master (controller that has access to a bus at an instance) accesses & then leaves the control of the bus & passes it to the another bus requesting processing unit.

There are 2 approaches to bus arbitration-

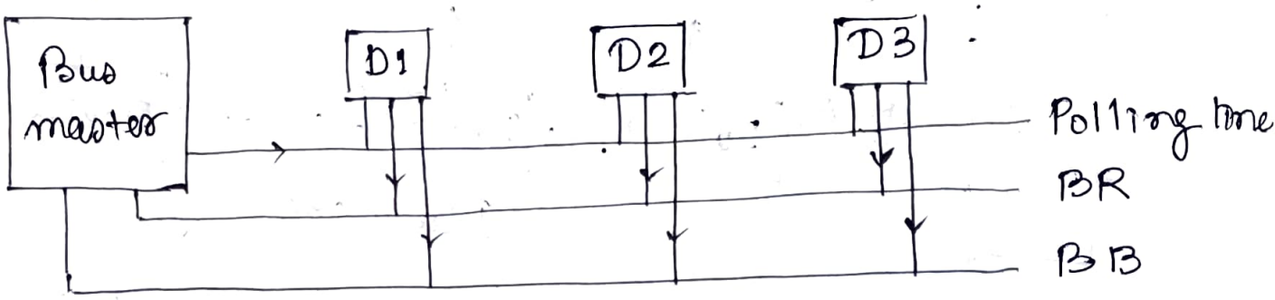
a) Centralised: A single bus arbiter performs required arbitration. Three different schemes that use this approach are -

i) Daisy Chaining:

If bus isn't busy, make bus request. Master activates bus grant. If device gets bus grant, mark bus busy.

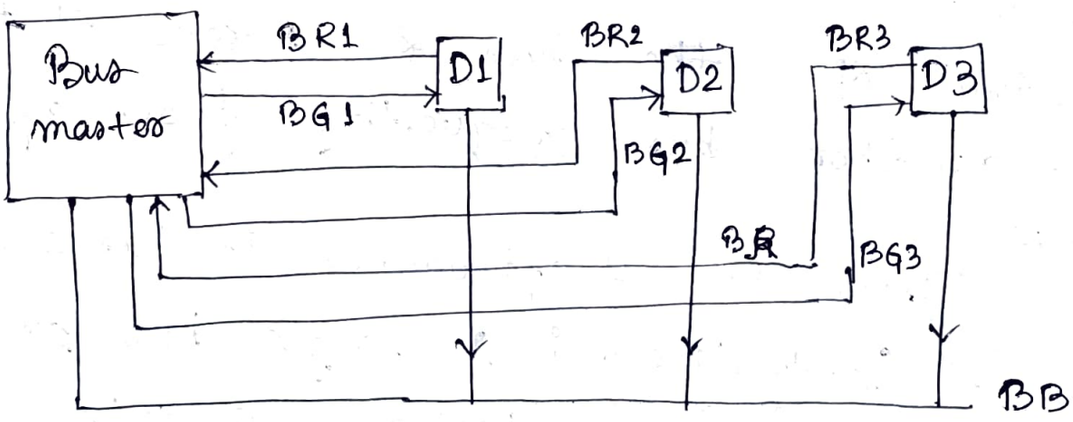


ii) Polling:



If bus isn't busy, make bus request. Master polls by placing device ID on polling lines. If device gets bus grant, mark bus busy.

iii) Independent request:



If bus isn't busy, make bus request. Master decides who to grant access & indicates grant line. If device gets bus grant, mark bus busy.

b) Distributed: All devices participate in the selection of the next bus master. Each device on the bus is assigned a 4-bit identification number. Devices assert the start-arbitration signal & place their 4 bit ID number on arbitration lines.

→ Handling Multiple Interrupts:

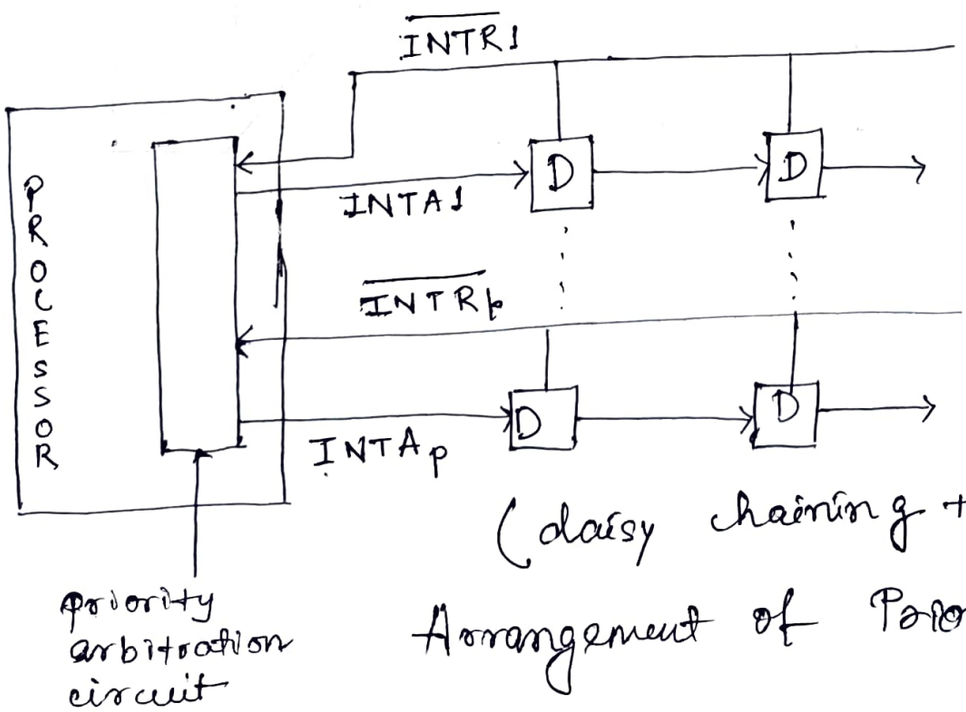
i) With multiple lines, the processor picks the interrupt line with highest priority.

ii) With software polling, the order in which modules are polled determines their priority.

iii) In case of daisy chaining, the priority of a module is determined by the position of the module on the daisy chain.

iv) In case of bus arbitration, centralised or distributed approach is taken.

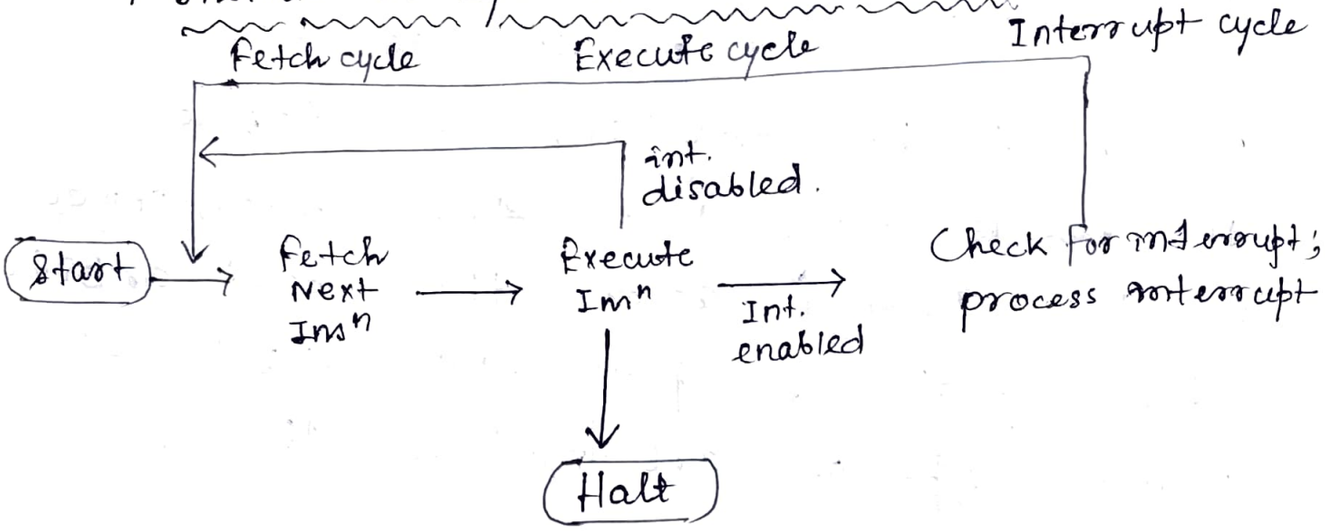
→ Combining identification techniques:



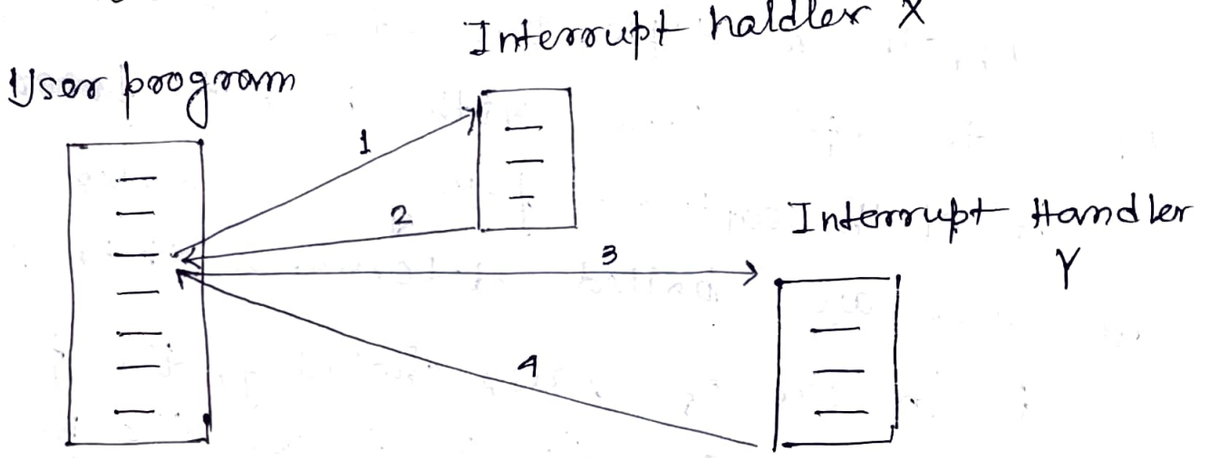
(daisy chaining + polling)

Arrangement of Priority groups.

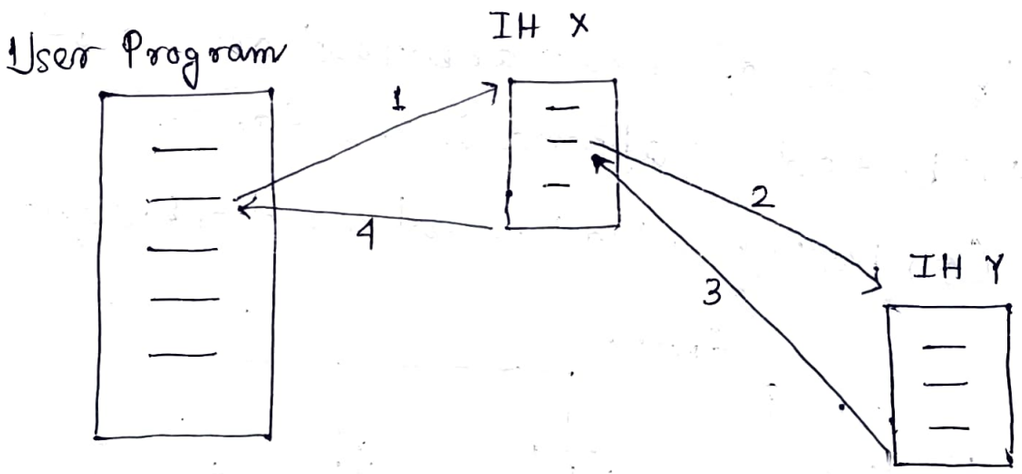
→ Instruction cycle with interrupts:



→ Sequential Interrupt Processing:



→ Nested Interrupt Processing:



↝ Exceptions: An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions during the execution of a program. (G Note).

I/O interrupt is an example of exception.

- Other types of exceptions - recovery from errors, debugging, privilege exception.
- Many source of errors in a processor like error in the data stored, error during the execution of instruction.
- Exception processing ~ Processor takes the same steps as in the case of I/O interrupt request. (Exception service routine)
- In case of I/O interrupt request, the processor usually completes the execution of an ins^n in progress before branching to interrupt-service routine. But for error (creating exception), execution halts for running ins^n & exception handling starts.
- Debugger uses exceptions to provide important features - trace, breakpoints.

Trace mode - Exception after execution of each ins^n. Debugging program is used as exception service routine.

Breakpoints - Exception occurs only at specific points selected by the user. Debugging program used as the exception service routine.

• Certain instructions can be executed only when the processor is in the supervisor mode. These are Privileged instructions. If an attempt to execute a privileged instruction in the user mode is made, a privilege exception occurs.

Privilege exception causes - processor to switch to the supervisor mode, execution of ESR.

• Return from exception (RFE) is a privileged trap instr.

• Exception can be classified as

i) Interrupt : due to I/O device

ii) Trap : Caused by the program making a syscall.

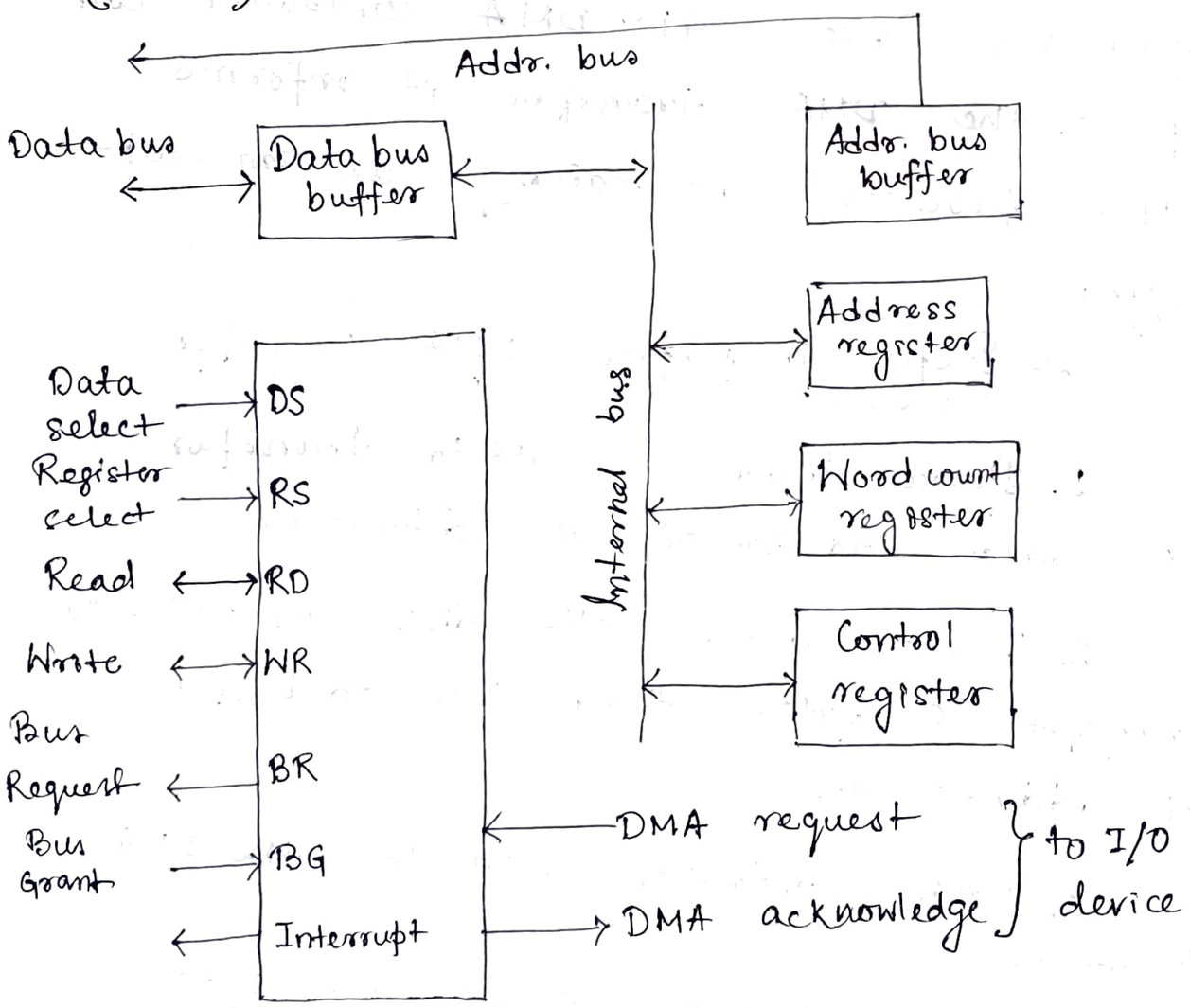
iii) Fault : Accidentally caused by program. (÷0, Null pointer).

RFE is a trap instr as it is causing a switch from kernel to user mode. RFE is a privileged instr as it can be executed only in kernel/supervisor mode. As soon as

RFE starts processing, all other traps/ints are disabled.

* Direct Memory Access : Method that allows an I/O device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

• Special control unit that perform data tranfer directly between I/O and memory, without continuous intervention by the processor, is the DMA controller (DMAC). This is a part of the I/O interface.



DMAC

• DMA controller gets four parameters - the target address, the source address, read or write, byte count.

• DMA controller performs functions that would be normally carried out by the processor ~ for each word, it provides the memory address & all the control signals. To transfer a block of data, it increments the memory addresses & keeps track of the number of transfers. Operation of DMA controller must be under the control of a program executed by the processor. Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

~> Steps to perform DMA transfer:

1. Device driver is told to transfer disk data to buffer at address x.

2. Device driver tells disk controller to transfer c bytes from disk to buffer at address x.

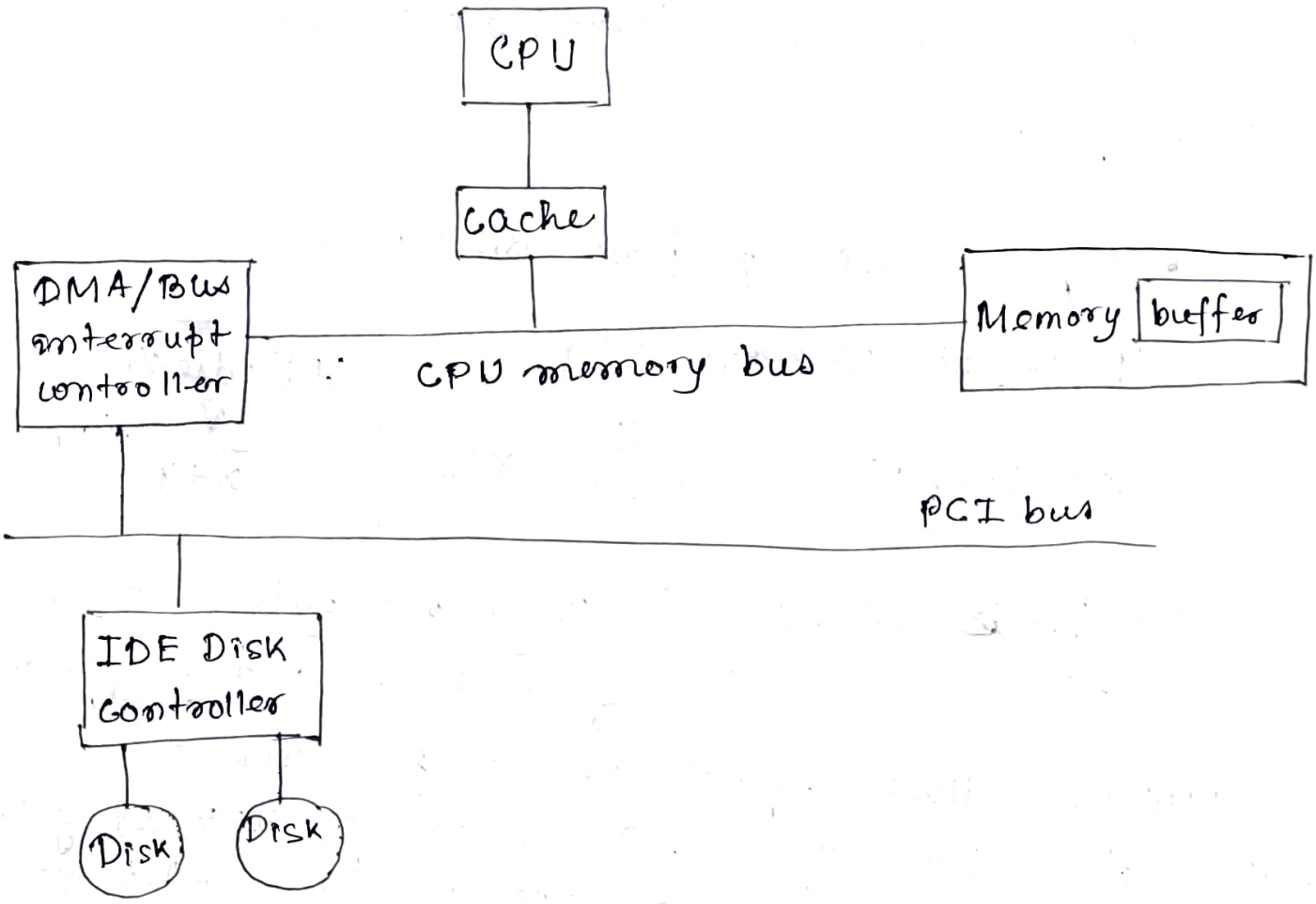
3. Disk controller initiates the DMA transfer.

4. Disk controller sends each byte to DMA controller.

5. DMA controller transfer bytes to buffer x, increasing memory address & decreasing

c until C = 0.

6. When C = 0, DMAC interrupts CPU to signal transfer completion.



→ We use CPU instead of DMA if a) device speed is fast, b) CPU has nothing to do, c) want to save money (by getting rid of DMA).

→ Types of DMA transfer using DMA Controller:

i) Burst Transfer: Processor is disconnected from system bus during DMA transfer. N number of machine cycles are adopted (N - no. of bytes to be transferred).

DMA sends HOLD signal to processor to request for system bus & waits for HLDA (acknowledge) signal.

After receiving HLDA signal, DMA gets control of system bus & transfer one byte. After this, it increments memory address, decrements counter & transfer next byte.

In this way, it transfer all bytes and then DMAC disables HOLD signal & enters into slave mode.

Total time taken to transfer N bytes =

Bus grant request time + N × (memory transfer time/byte) + Bus release control time

Burst mode is very fast data transfer technique.

- X μs → data preparation time
  Y μs → transfer time

$$\text{CPU busy \%} = \frac{X}{X+Y} \times 100$$

$$\text{CPU idle \%} = \frac{Y}{X+Y} \times 100.$$

ii) Cycle stealing : Only one byte is transferred at a time (slower).

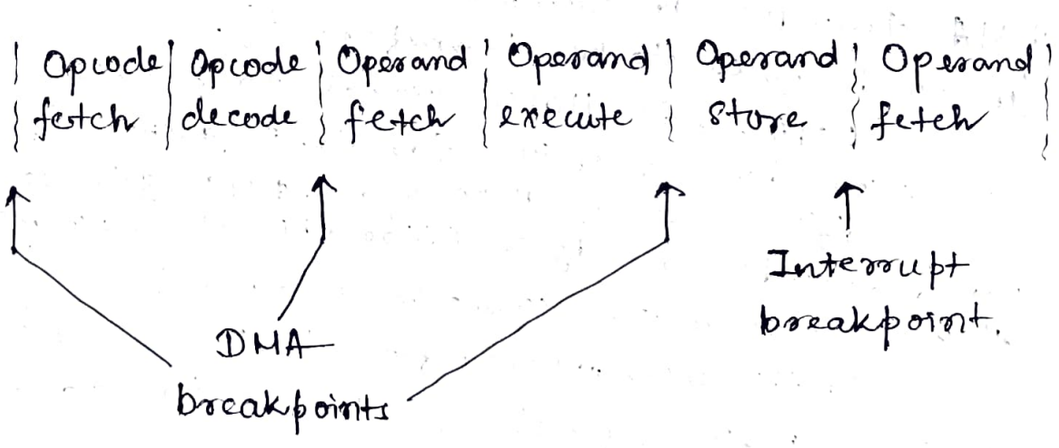
DMA sends HOLD signal to the processor & waits for HLDA signal. On receiving HLDA, it gains control on system bus & executes one DMA cycle.

After transferring one byte, it disables HOLD signal & enters into slave mode.

Processor gains control of system bus & executes next machine cycle. When CPU is idle again, if count is not zero & data is available then the DMAC sends HOLD signal to the processor.

At DMA breakpoints, CPU is idle & DMAC uses cycle stealing then.

eg. ins^n cycle →



Total time taken to transfer N bytes =

$$N \times \left(\begin{array}{c} \text{Time for} \\ \text{bus grant} \end{array} + \begin{array}{c} \text{1 bus cycle} \\ \text{to transfer} \\ \text{data} \end{array} + \begin{array}{c} \text{bus release} \\ \text{time} \end{array} \right).$$

- In cycle stealing mode we always use pipelining concept that when one byte is getting transferred then device is preparing next byte.
- X μs → data preparation time
  Y μs → transfer time

$$\text{CPU busy \%} = \frac{X}{Y} \times 100\%$$

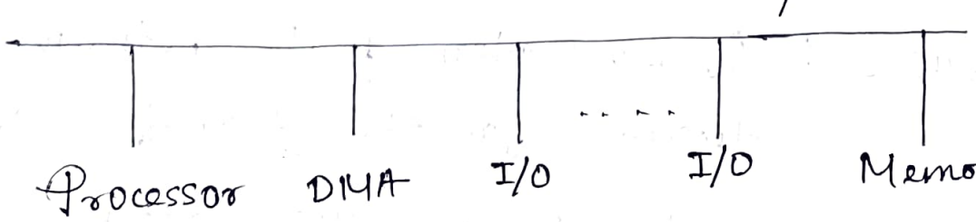
$$\text{CPU idle \%} = \frac{Y}{X} \times 100\%$$

iii) Interleaved Mode: DMAC takes over the system bus when the microprocessor is not using it. An alternate half cycle — half cycle DMA + half cycle processor.

→ DMA Mechanism can be configured in different ways —

i) Single-bus, detached DMA:

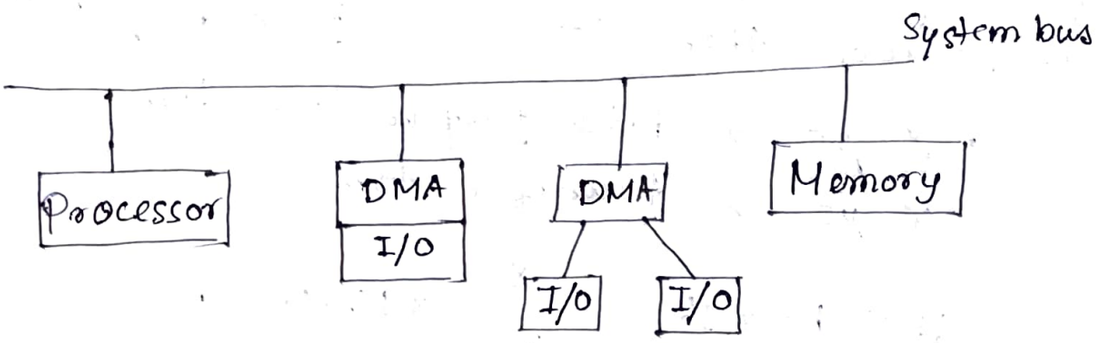
DMA acts as a surrogate processor. This configuration uses the bus twice (IO → DMA, DMA → Memory)



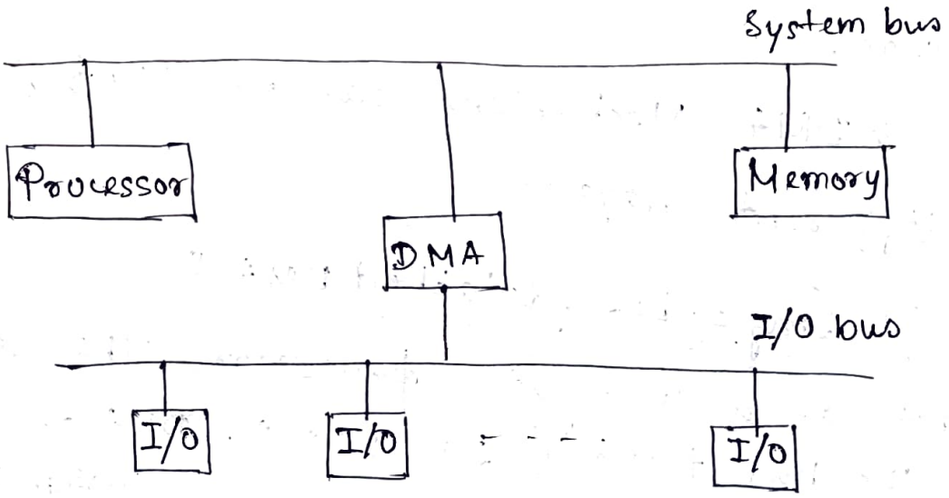
Transfer consumes two bus cycles.

ii) Single bus, Integrated DMA :

Integrated DMA & I/O. System bus is used only once, So, processor is suspended once. One bus cycle required.



iii) Separate I/O bus : DMA module is reduced to one. Transfer of data between I/O module & DMA module is carried out through the I/O bus. Processor suspended for one bus cycle. (DMA ↔ Memory transfer).



* Buses : Communication path between the devices for the transfer of data.

• The bus lines are grouped into three categories :

i) data line
ii) address line
iii) control line. (control & timing information)

Data bus provides a path for moving data among system modules. The width of the data bus is a key factor in determining overall system performance.

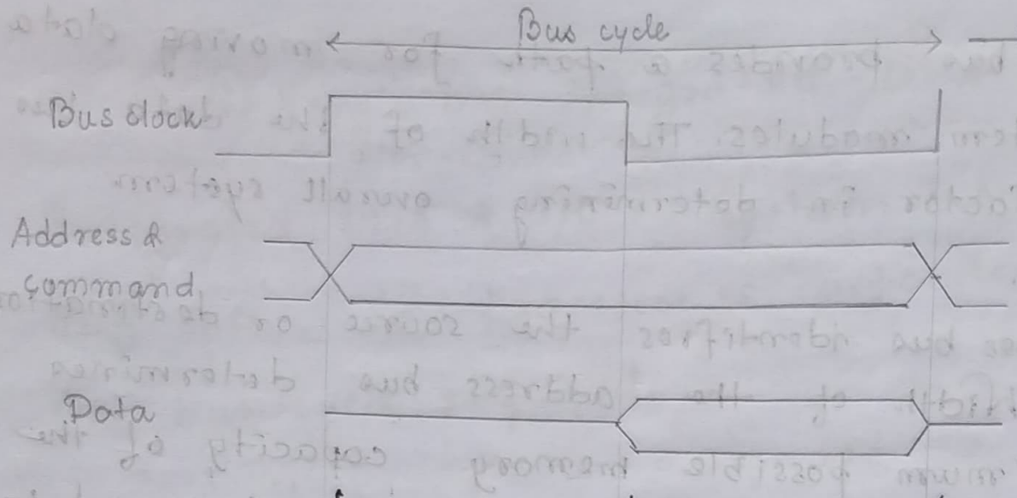
Address bus identifies the source or destination of data. Width of the address bus determines the maximum possible memory capacity of the system. The address lines are generally also used to select a I/O port. Higher order bits are used to select a particular module on the bus & the lower order bits select a memory location or I/O port within the module.

Control bus is used to control the access to & the use of the data & address lines. Control signals transmit both command & timing information among system modules. (Read/Wrote, transfer ACK, Bus request, Bus grant).

- A bus protocol is the set of rules that govern the behaviour of various devices connected to the bus, as to when to place information on the bus, when to assert control signals etc.
- Schemes for timing of data transfers over a bus can be classified as —
  synchronous, asynchronous.

→ Synchronous bus: All devices derive timing information from a common clock line. One data transfer takes place during one bus cycle.

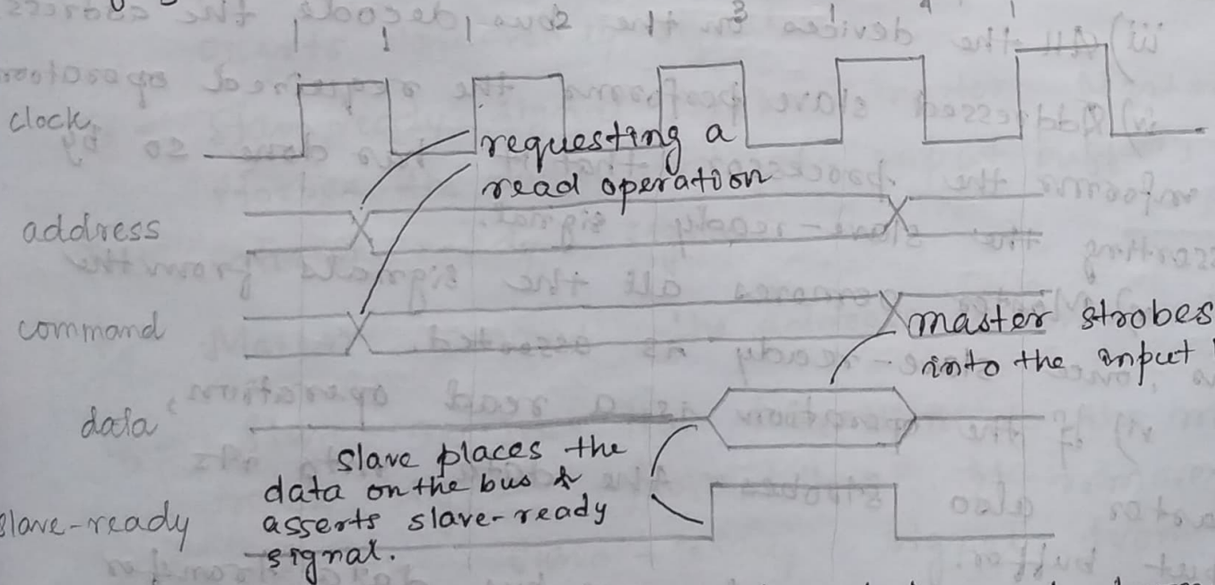
eg. Sequence of events during an input (read) operation.



- Once the master places the device address & command on the bus, its takes time for this information to propagate to the devices. This time depends on the physical & electrical characteristics of the bus.
- All the devices have to be given enough time to decode the address & control signals, so that the addressed slave can place data on the bus.
- Clock pulse width $t_1 - t_0$ depends on -
  i) Maximum propagation delay between two devices connected to the bus.
  ii) Time taken by device to decode the address & control signals so that the addressed device (slave) can respond at time t_1.
- At the end of the clock cycle at time t_2, the master strobes (to capture the values of the data at a given instant & store them into a buffer) the data on the data lines into its input buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.

width of the pulse $t_2 - t_1$ should be longer than max. propagation time on the bus plus the setup time of the input buffer register of the master.

- As signals take time to travel from one device to another, a given signal instruction is seen by different devices at different times. Although, we assume that the clock changes are seen at the same time by all devices on the bus.
- Most buses have control signals to represent a response from the slave. Control signal informs the master that the slave has recognized the address & is ready to participate in a data transfer. Control signal enables to adjust the duration of the data transfer based on the speed of the participating slaves.
- Clock period is chosen to accomodate longest delays on the bus & slowest device interface. This forces all devices to operate at the speed of the slowest device.
- Sometimes, a high-frequency bus clock is used where data transfer spans several clock cycles instead of just one clock cycle as in the normal case.



Slave-ready signal is an acknowledgement from the slave to the master, confirming the valid data have been sent.

If the addressed device does not respond at all, master waits for some predefined max. no. of clock cycles, then aborts the operation.

• Clock signal used on a computer bus is not necessarily the same as the processor clock.

↝ Asynchronous bus : In this scheme, data transfer on the bus is controlled by a handshake between the master & the slave.

• Common clock of the synchronous bus is replaced by 2 timing control lines – master-ready & slave-ready.

• Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.

Slave-ready signal is asserted by the slave in response to the master-ready & it indicates to the master that the slave is ready to participate in a data transfer.

• Data transfer using the handshake protocol :

i) Master places the address & command information on the bus.

ii) Master asserts the master-ready signal to indicate to the slaves that the address & command information have been placed on the bus.

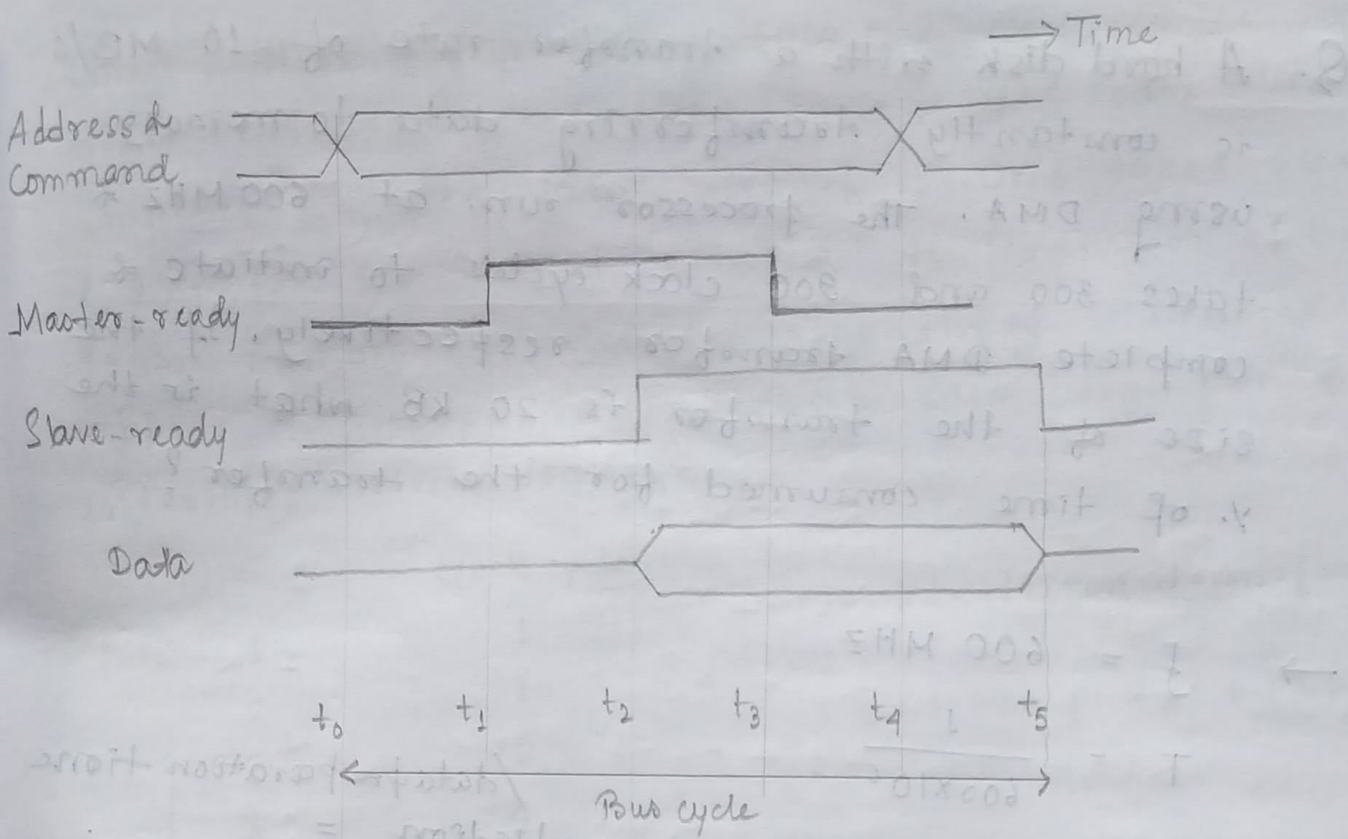
iii) All the devices on the bus decode the address.

iv) Addressed slave performs the required operation, & informs the processor that it has done so by asserting the slave-ready signal.

v) Master removes all the signals from the bus, once slave-ready is asserted.

vi) If the operation is a read operation, master also strobes the data into its input buffer.

• In case of asynchronous bus, data transfer does not need synchronisation between the sender & receiver. It can accomodate varying delays automatically, using the slave-ready signal.



Handshake control of data transfer during input.

t_0 - master places address & command information on the bus.

t_1 - Master asserts master-ready signal. The delay $t_1 - t_0$ is intended to allow for any skew that may occur on the bus. Skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different times.

t_2 - Addressed slave places data on the bus & asserts slave-ready signal.

t_3 - Slave ready signal arrives master. Master strobes the data into its input buffer, drops the master-ready signal.

t_4 - Master removes the address & command info.

t_5 - Slave receives the transition of the master-ready signal from 1 to 0. It removes the data & the slave-ready signal from the bus.

• A change of state in one signal is followed by a change on the other signal. Hence, it is called a full-handshake.

Q. A hard disk with a transfer rate of 10 MB/s is constantly transferring data to memory using DMA. The processor runs at 600 MHz & takes 300 and 900 clock cycles to initiate & complete DMA transfer respectively. If the size of the transfer is 20 KB, what is the % of time consumed for the transfer?

→ $f = 600\ MHz$

$$T = \frac{1}{600 \times 10^6}$$

/data preparation time

DMA initialisation & completion, =

$$(900 + 300) \times \frac{1}{600 \times 10^6} = 2\,\mu s$$

Disk transfer rate = 10 MB/s

Time to transfer 20 KB = $\frac{20 \times 10^3}{10 \times 10^6} = 2000\,\mu s$

% time consumed by

$$\frac{2}{2000} \times 100\% = 0.1\% \quad (Ans)$$

Q. GIII On a non-pipelined processor a program segment which is the part of ISR, is given to transfer 500 bytes from I/O device to memory

1 Initialise the address register.
2 Initialise the count to 500.
LOOP: 3 Load a byte from device.
4 Store in memory as address given in address register.
5 Increment the address register.
6 Decrement the count.
7 If count ≠ 0, goto Loop.

Load & store instructions take 2-clock cycles & other instructions 1 clock cycle. DMA can also implement the same transfer. DMA takes 20 ns & 2 ns for its initial setup & to transfer 1 byte from memory to device respectively. Calculate approx. speedup ratio between DMA transfer & interrupt driven transfer.

	Ins^n	Clock cycle
→	1	1
	2	1
500 times	3	2
	4	2
	5	1
	6	1
	7	1

Interrupt-driven transfer time =

$1 + 1 + 500\,(2+2+1+1)$ ns

$= 3502$ ns

DMA transfer time =

$20 + 500 \times 2$ ns

$= 1020$ ns.

Speedup ratio $= \frac{3502}{1020} = 3.4$. (Ans.)

Q G'05 A device with data transfer rate 10 KB/s is connected to a CPU. Data is transferred bytewise. Let the interrupt overhead be 4 μs. The byte transfer time between the device interface register & CPU or memory is negligible. What is the maximum performance gain of operating the device interrupt mode over operating it under programmed-control mode?

→ In programmed-I/O, the CPU issues a command & waits for I/O to complete. So, here, CPU will wait for 1 sec to transfer 10 KB of data.

Overhead in programmed I/O is 1 sec.

In interrupt mode, data is transferred word by word (Word size is 1 Byte)

So data overhead $= 4 \times 10^{-6} \times 10^4$ s. (for 10 KB)

Performance gain $= \frac{1}{4 \times 10^{-6} \times 10^4} = 25$. (Ans)

Q G'07 Starting address of the program is 100.

	Insⁿ	Operation	Insⁿ size (# words)	
	MOV R1, (3000)	R1 ← M[3000]	2	
	MOV R2, (R3)	R2 ← M[R3]	1	
Loop:	ADD R2, R1	R2 ← R2 + R1	1	
	MOV (R3), R2	M[R3] ← R2	1	
→	INC R3	R3 ← R3 + 1	1	← INTR
	DEC R1	R1 ← R1 - 1	1	
	BNZ LOOP	Branch on not zero	2	
	HALT.	Stop	1	

Assume memory is byte addressable & word size is 32 bits. If an interrupt occurs during execution of INC R3, then what return address will be pushed on the top of the stack ?

→ $1000 + (2+1+1+1) \times 32/8 = 1020$

$(1020 + 4) = 1024$ (Ans)

↳ As INC R3 will be executed fully, after which interrupt will be handled.

Q G'04 What is the bit rate of video terminal unit with 80 characters, 8 bits/char & sweep time of 100 μs (including 20 μs of retrace time).

→ Bit rate = $\frac{80 \times 8 \text{ bits}}{100\ \mu s} = 6.4$ Mbps.

Q G'16 The size of the data count register of a DMAC is 16 bits. The processor needs to transfer a file of 29154 KB from disk to main memory. The memory is byte addressable. The minimum number of times the DMAC needs to get control of the system bus from the processor to transfer the file is —

→ Size of data count register = 16 bits.

Data that can be transferred in one go =

2^{16} bytes = 64 KB

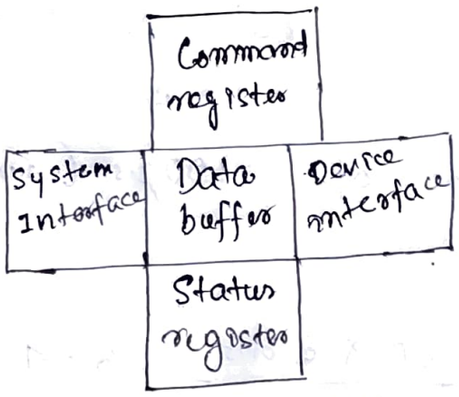
Ans is Ceil (29154/64) = 456. (Ans)

* Device controller / I/O controller / I/O interface :

I/O controller is a series of microchips (installed on the mother board) which help in the comm" of data between CPU & the peripheral devices.

Sometimes, an uncommon device requires its own controller that must be plugged into a connector on mother board.

→ Device driver : Software that acts as a translator between the I/O device & the OS. It communicates with the hardware by means of the comm" subsystem.



Device controller.

→ Functions ~

i) Control & timing
ii) Processor communication
iii) Device communication
iv) Error detection unit
v) Data buffering.

* Components to support a device :

i) Device controller , ii) Device driver,
iii) Device interface cable.

The Memory System.

* Characteristics of Memory Systems:

1. Location: (i) Internal memory (processor registers, control unit internal memory, main memory, cache etc.) (ii) External memory (Optical disks, magnetic disks, tapes etc.)

2. Capacity: Capacity of the memory.

3. Unit of transfer: Often the word length for internal memory.

   - Word ~ Natural unit of memory organisation.
   - Addressable units ~ 2^A (A – no. of bits of address)
   - For main memory, unit of transfer is the no. of bits read out or written into memory at a time. Unit of transfer need not equal a word or an addressable unit.

   For external memory data are transferred in blocks (collection of words).

4. Access method:

   - Sequential access ~ Memory organised into units of data, called records. Access made in linear sequence. Time to access an arbitary record is highly variable. eg. tape units.
   - Direct access ~ Individual blocks have a unique address based on physical location. Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting or waiting to reach the final destination. Access time is variable. eg. disk units.

   Both sequential & direct access use shared read-write mechanism.

- Random access ~ Individual addresses identify locations exactly. Access time is independent of location or previous access.
- Associative ~ Data is located by a comparison of a portion of the store with the word. Access time is independent of location or previous access and is constant.

5. Performance :

- Access time (latency) ~ Time between presenting the address and getting the valid data (eg. in memory, time between the Read & the MFC signal).
- Memory cycle time - Time required for the memory to recover before next access. It is latency + recovery time. It's concerned with the system bus, not the processor.
- Transfer rate ~ Rate at which data can be transferred into or out of a memory unit. For random access memory, it is 1/(cycle time).

$$TR = (\#\text{ of bits (bytes)}) \times \frac{1}{\text{cycle time}}$$

- For non-random access memory -

$$T_N = T_A + \frac{n}{R}$$

T_N - avg. time to read or write N bits

T_A - avg. access time

n - no. of bits

R - transfer rate (bps)

6. Physical types : Semiconductor (RAM), magnetic (Disk & tape), Optical (CD & DVD), magneto-optical etc.

7. Physical characteristics :

→ Volatibality ~

Volatile - Information decays naturally or is lost when electrical power is off. (RAM)

Non-volatile - Information once recorded remains without deterioration until deliberately changed; no electrical power needed to retain information. (ROM)

→ Erasability ~

Erasable, Non-erasable

→ Power-consumption / heat

8. Organization : Physical arrangement of bits to form words.

* The memory hierarchy :

There is a trade-off among the three key characteristics of memory - capacity, access time and cost.

Faster access time ⇒ greater cost per bit

Greater capacity ⇒ smaller cost per bit

Greater capacity ⇒ slower access time

A memory hierarchy is there to deal with the tradeoff. As one goes down the hierarchy, followings occur -

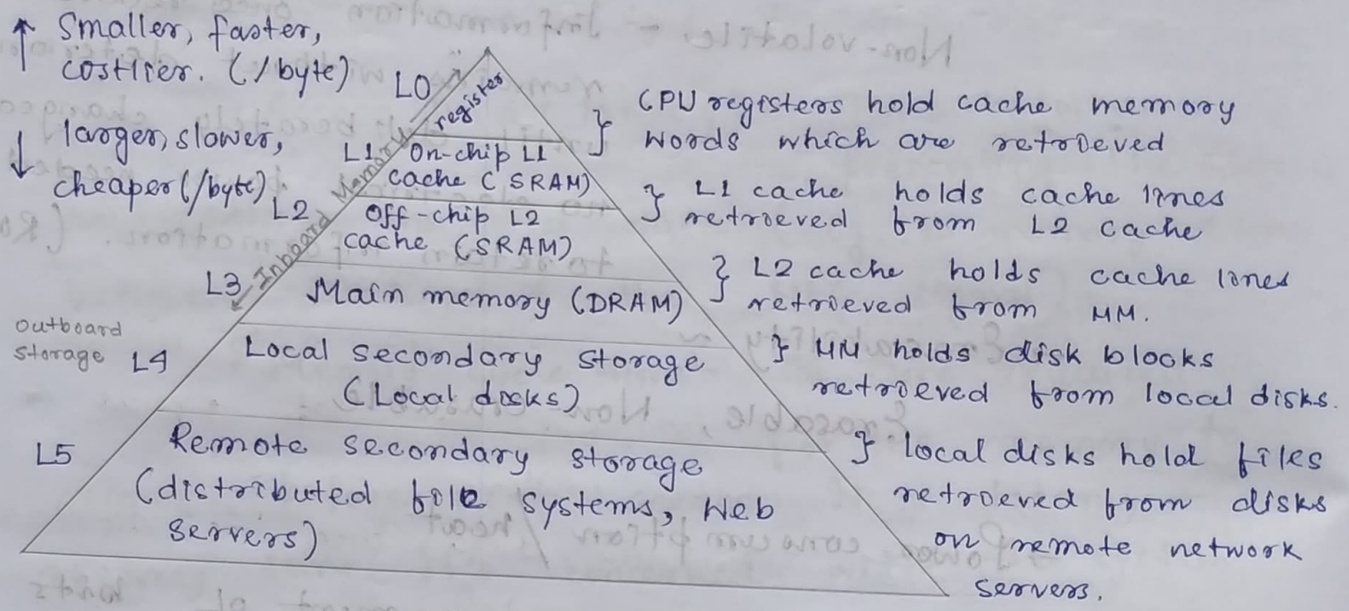
a) Decreasing cost/bit

b) Increasing capacity

c) Increasing access time

d) Decreasing frequency of access of the memory by the processor.

Key to success of this organisation is (d): decreasing frequency of access.



* Principle of Locality / Locality of reference:

States that programs access a relatively small portion of their address space at any instant of time. Two different types of Locality ~

i) Temporal Locality (Locality in time) ~

The principle stating that if data location is referenced then it will tend to be referenced again soon.

ii) Spatial locality (locality in space):

The principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

* Hit rate ~ The fraction of memory accesses found in a level of the memory hierarchy.

Miss rate ~ The fraction of memory accesses not found in a level of the memory hierarchy. (1 - hit rate).

Hit time ~ The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or miss.

Miss penalty ~ The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, & then pass the block to the requestor.

• If the hit rate is high enough, the memory hierarchy has an effective access time close to that of the highest (& fastest) level and a size equal to that of the lowest (& largest) level.

- h_1 – hit ratio at first level

  h_2 – hit ratio at second level.

Average memory access time (three level hierarchy)

$$t_{avg} = h_1 \times t_1 + (1-h_1)\left[t_1 + h_2 \times t_2 + (1-h_2)(t_2+t_3)\right]$$

$$t_{avg} = t_1 + (1-h_1)\left[t_2 + (1-h_2)t_3\right]$$

t_1, t_2, t_3 ~ access times at three levels.

- $t_{avg} = hC + (1-h)M$.

  h – hit rate

  C – time to access data from a level

  M – miss penalty for that level

- $t_{avg} = h_1 C_1 + (1-h_1) h_2 C_2 + (1-h_1)(1-h_2) M$

eg. h_1 – hit ratio in the L_1 cache

h_2 – hit ratio in the L_2 cache

C_1 – time to access information in the L1 cache

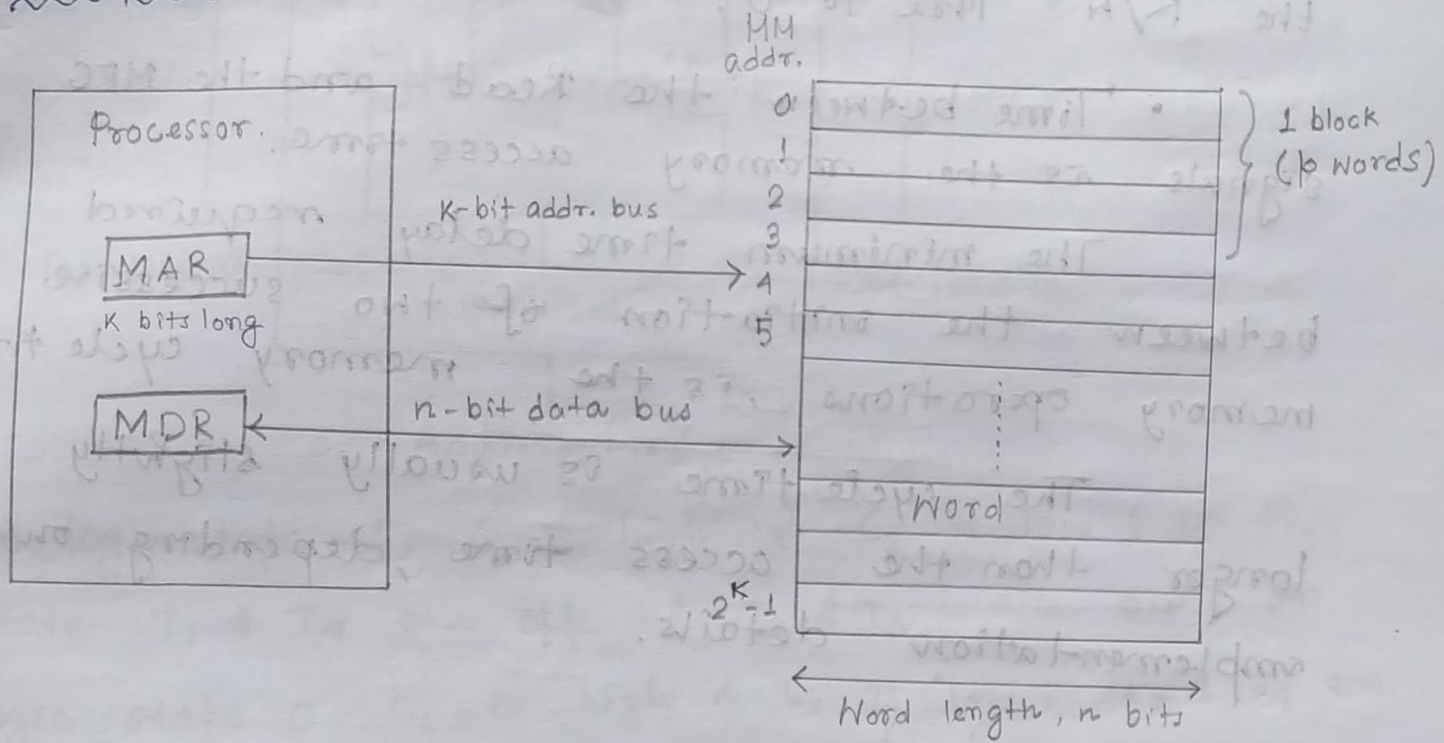
C_2 – time to access information in the L2 cache

M – time to access information from the main memory.

* → The maximum size of memory that can be used in any computer is determined by the addressing scheme.

→ The number of bits actually stored or retrieved in one memory access is the most common definition of the word length of a computer.

* Main memory basics :



Each memory cell consists of a word, which has an address numbered $0, 1, \ldots, 2^k-1$.

• If MAR is k-bits long & MDR is n bits long, then the memory unit may contain upto 2^k addressable locations. During a memory cycle, n bits of data are transferred between the memory and the processor. This transfer takes place over the processor bus; which has k address lines and n data lines.

• READ ~ The memory responds by placing the data from the addressed location onto the data lines & confirms the action by asserting the Memory function Completed (MFC) signal.

• Write ~ The processor writes data into a memory location by loading the address of this location into MAR & loading the data into MDR. It indicates that a write operation is involved by setting the R/$\bar{W}$ line to 0.

• Time between the Read and the MFC signals is the memory access time.

The minimum time delay required between the initiation of two successive memory operations, is the memory cycle time.

The cycle time is usually slightly longer than the access time, depending on the implementation details.

* Memory Technologies :

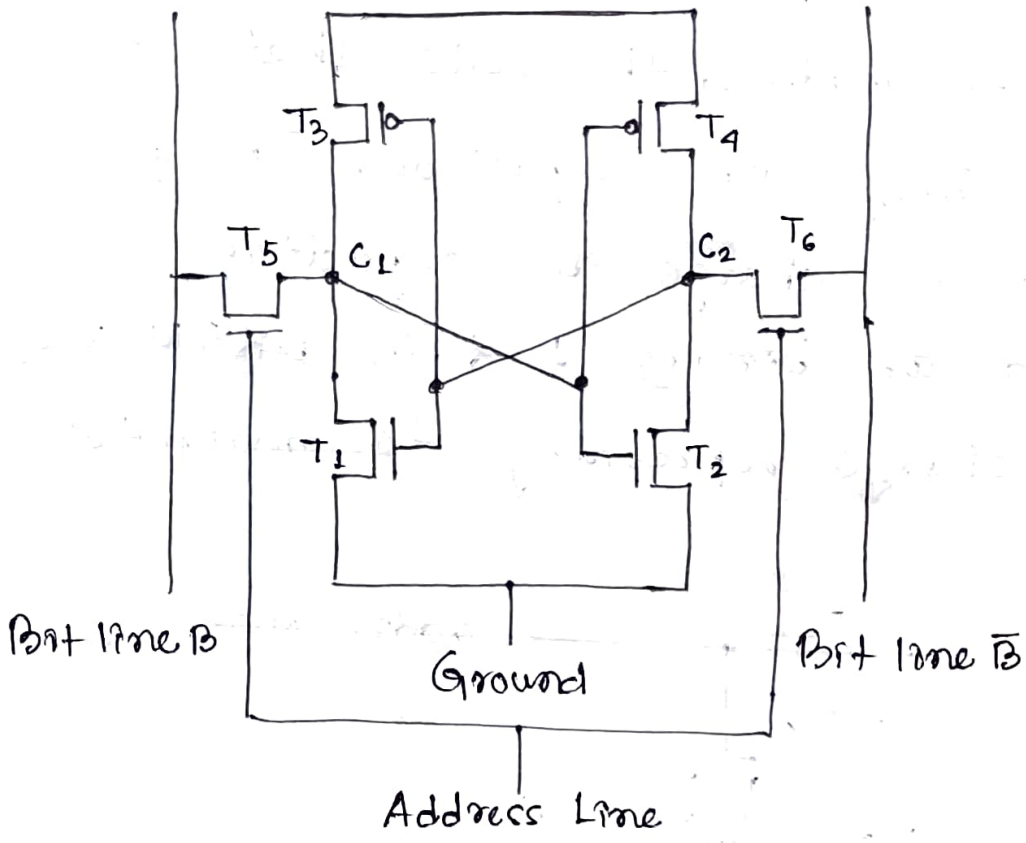
• Four primary memory technologies — SRAM, DRAM, Flash memory & magnetic disk.

Typical access time -

SRAM ≳ DRAM < Flash memory < Magnetic disk

• SRAM (Static RAM)

SRAMs dont need to refresh & so the access time is close to the cycle time. SRAMs need minimal power to retain the charge in stand by mode. SRAM is volatile. In a SRAM, binary values are stored using traditional flip-flop logic-gate configurations.



In logic state 1, C_1 is high & C_2 is low; in this state T_1 & T_4 are off & T_2 & T_3 are on. In logic state 0, C_2 is high & C_1 is low; T_1 & T_4 are on & T_2 & T_3 are off. Both states are stable as long as the dc. supply is applied. The address line is used to open or close a switch which is another transistor. Address line controls T_5 & T_6.

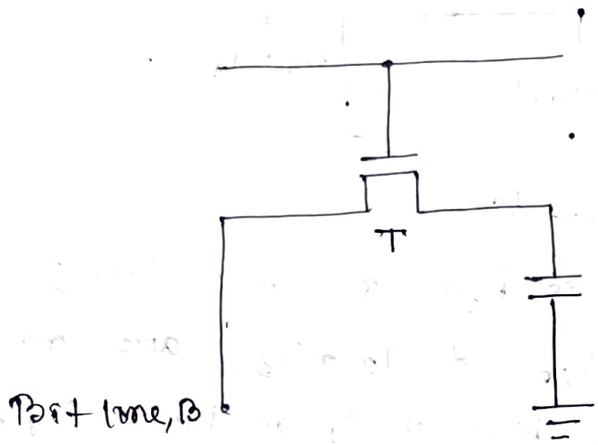
For a write operation, the desired bit value is applied to line B & its complement is applied to $\bar{B}$.

For a read operation, the bit value is read from the Bit line B. When a signal is applied to the address line, the signal of C_1 is available on the bit line.

- SRAM is used to build cache.

• DRAM (Dynamic RAM) :

DRAM is made with cells that store data as charge on capacitor. The presence or absence of charge in capacitor is interpreted as binary 1 & 0. DRAM requires periodic charge refreshing to maintain data.



For write operation, a voltage signal is applied to the bit line B, a high voltage represents 1 & a low voltage represents 0. A signal is then applied to the address line which will turn on the transistor allowing a charge to be transferred to the capacitor.

For read operation, when a signal is applied to the address line, the transistor turns on & the charge stored on the capacitor is fed out onto the bit line B.

• DDR SDRAM – Double data rate synchronous DRAM, data transferred on both rising & falling edge.

A DDR4 - 3200 DRAM can do 3200 million transfers per second which means it has 1600 MHz clock.

• Flash Memory : It is a type of electrically erasable programmable read-only memory. (EEPROM).

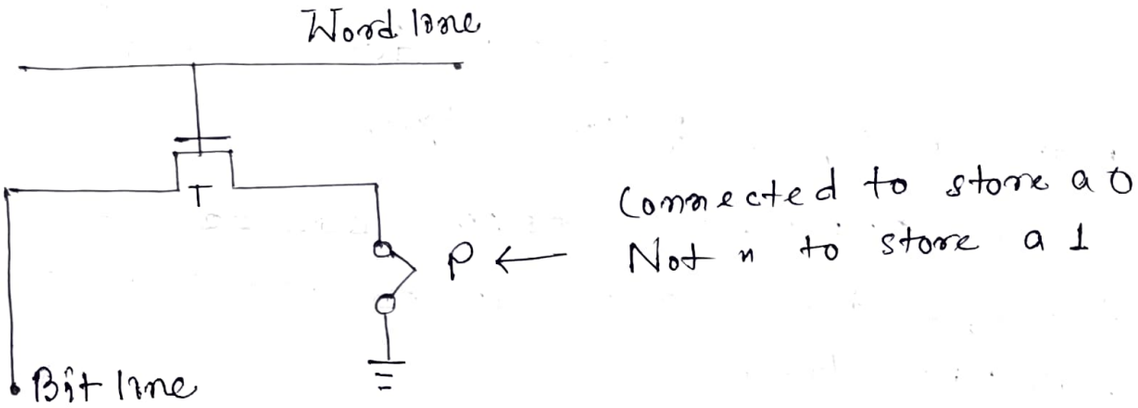
↝ Weare leveling

↝ In a flash device, it is possible to read the contents of a single cell but it is only possible to write an entire block of cells.

It is used in hand-held computers, cell phones, digital cameras.

• Disk memory : Magnetic hard disks consisting of a collection of platters. A read-write head used to data transfer.

• Read-only Memory (ROM) : Nonvolatile and contains permanent pattern of data that cannot be changed. ROM is extensively used in microprogramming. Contents of ROM can be read but a special writing process is needed to place the information into memory.



Different types of ROM ~

a) PROM (Programmable ROM) ~

Can be programmed once, inserting a fuse at point P.

b) EPROM (Erasable PROM) ~

Allows the stored data to be erased & new data to be loaded. Erasing can be done by exposing the chip to UV light. Entire contents are erased.

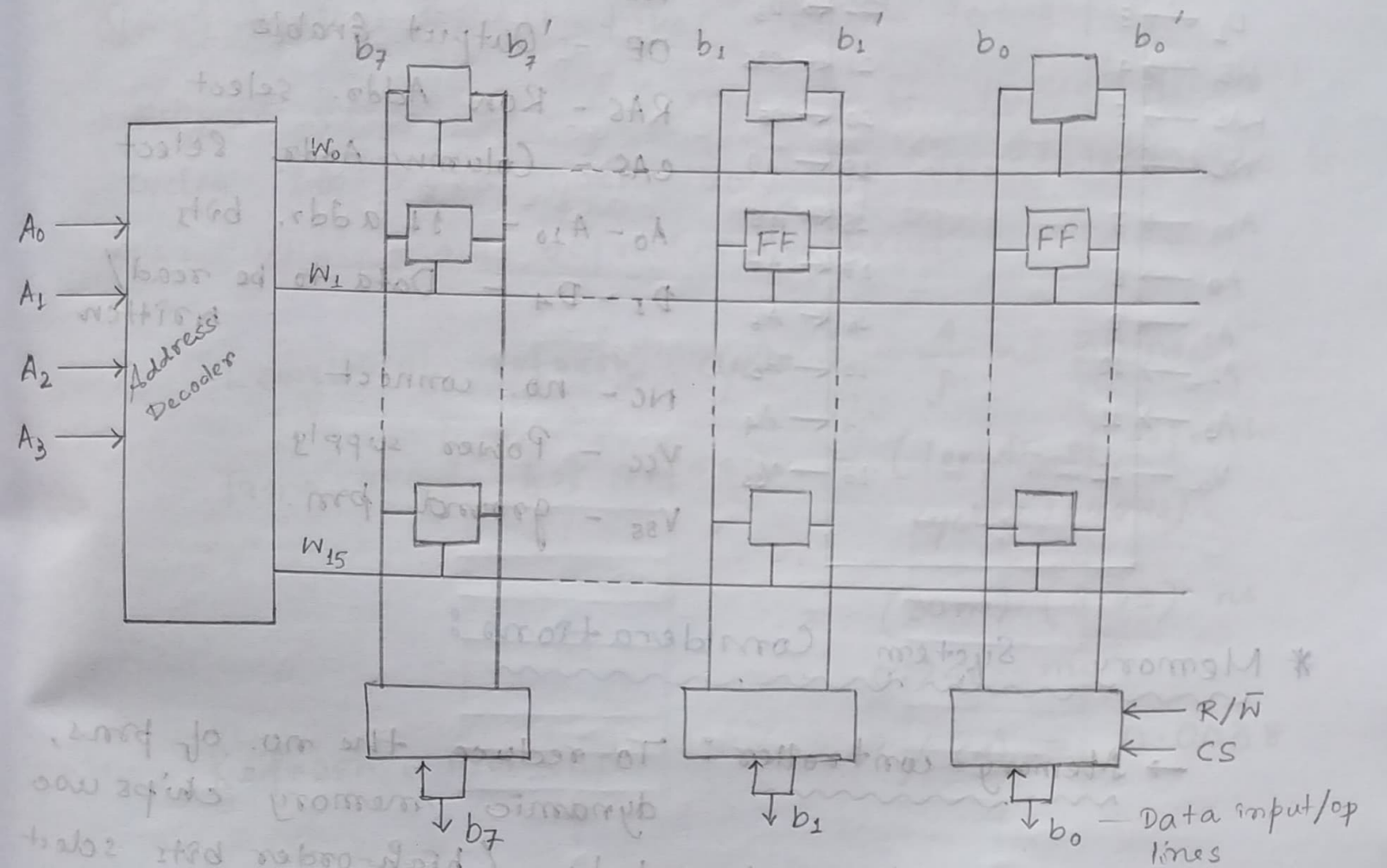
c) EEPROM (Electrically Erasable PROM) ~

Can be programmed & erased electrically. Possible to erase the cell contents selectively.

* Internal Organization of Memory Chips :

- Each row of cells constitutes a memory word and all cells of a row are connected to a common line referred to as the word line, which is driven by the address decoder on the chip. The cells in each column are connected to a Sense/Write circuit by two bit lines. The sense/write circuits are connected to the data input/output lines of the chip.

Two control lines R/$\bar{W}$ & CS (chip select) are provided. The R/$\bar{W}$ input specifies the required operation and the CS input selects a given chip in a multichip memory system.



Memory chip consisting 16 words of 8 bits each. (16 x 8 organisation)

No. of external lines =.

no. of addr. lines + no. of data lines + [2 lines for R/$\bar{W}$ & CS]

+[2 lines for power supply & ground]

eg. for above 16x8 organisation = 4+8+2+2

= 16

eg. For 1K memory (128 x 8) = 7+8+2+2

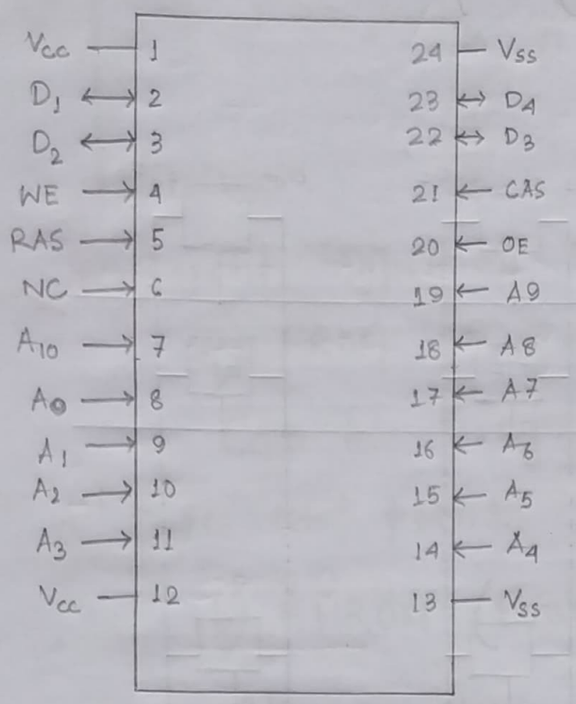
= 19.

For 1K memory (1024 x 1) = 10+1+2+2

= 15.

* Chip Logic :

16 Mbit Chip. (4M x 4).



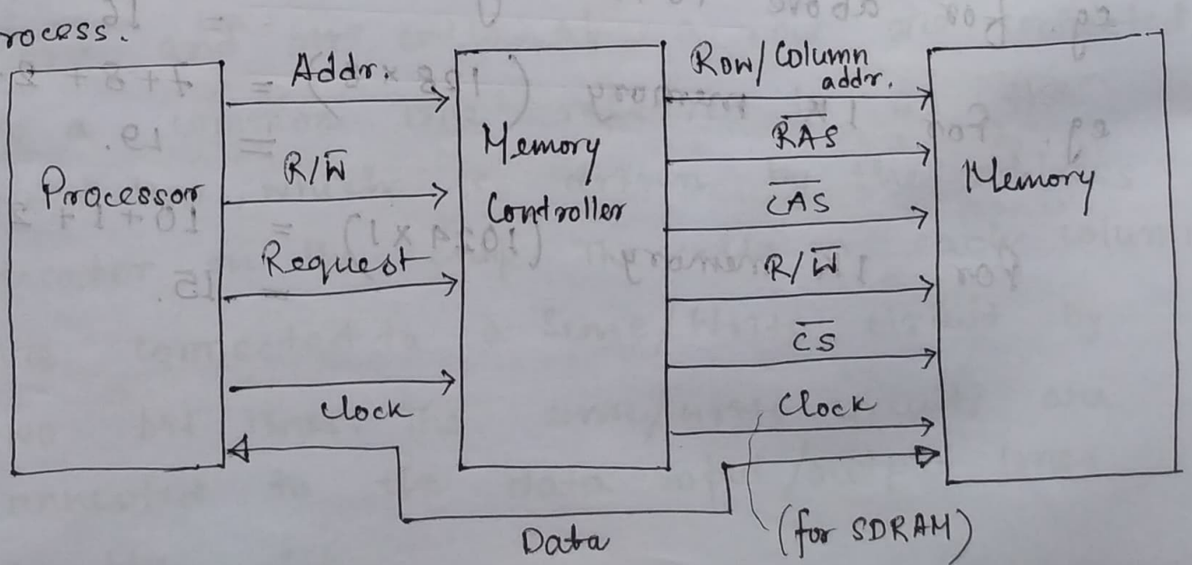
4M in 11 rows by 11 columns ($2^{22} = 4M$)

WE - Write Enable
OE - Output Enable
RAS - Row Addr. select
CAS - Column Addr. Select
$A_0 - A_{10}$ - 11 addr. bits
$D_1 - D_4$ - Data to be read/written

NC - no. connect
V_{CC} - Power supply
V_{SS} - ground pin.

* Memory System Considerations :

→ Memory controller : To reduce the no. of pins, dynamic memory chips use multiplexed addressed inputs. (high-order bits select row in array & latched using RAS signal ; low-order bits select column (provided on same addr. pins) & latched using CAS signal). Required multiplexing of addr bits is performed by a memory controller that's interpreted between the processor & the dynamic memory. For DRAM, memory controller has to provide information needed to control the refreshing process.



→ Refresh Overhead : Fraction of time memory spends on refresh.

$$\text{refresh overhead} = \frac{\text{time required for refresh, ms.}}{\text{refresh interval, ms.}}$$

Q.. A typical DRAM takes 64 ms to refresh. Suppose there are 8K rows & it takes 4 clock cycles to access each row. If the clock rate is 133 MHz, find refresh overhead.

→ Length of refresh cycle $= \frac{4}{f} = 30$ ns.

Time reqd. for refresh = (length of refresh cycles)(rows)

$= (30\text{m})(8192)$ ns

$= 0.246$ ms

Refresh overhead $= (0.246 / 64) = 0.0038$

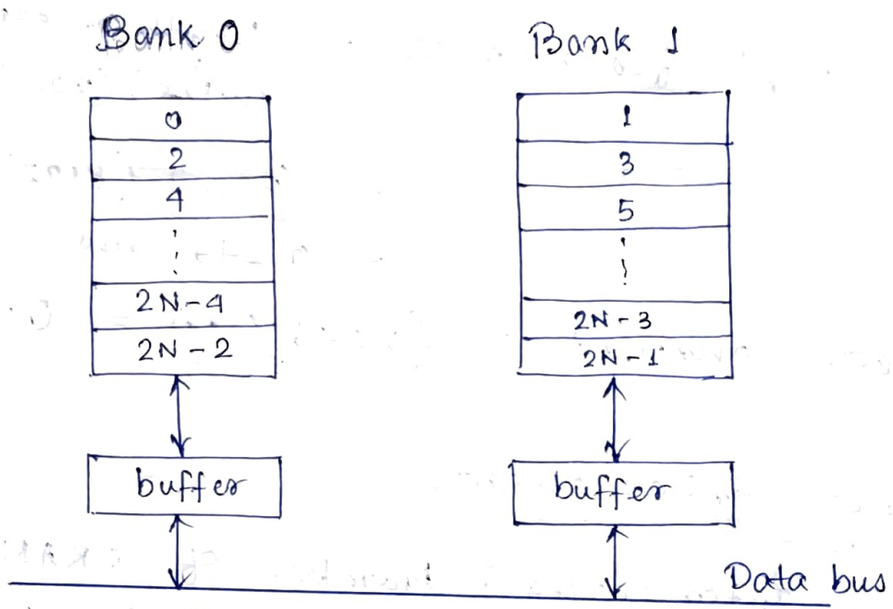
* Performance Enhancements :

1. Memory Interleaving : Number of DRAM chips form a bank, with a port for transfer of data to and from the processor or an intermediate cache. Multiple memory banks can be connected together to form an interleaved memory system. With K banks, an interleaved memory can service K requests simultaneously, increasing the peak data transfer rate by a factor of K. To implement the interleaved structure, there must be 2^m modules ; otherwise there will be gaps of non-existent locations on the memory addr. space.

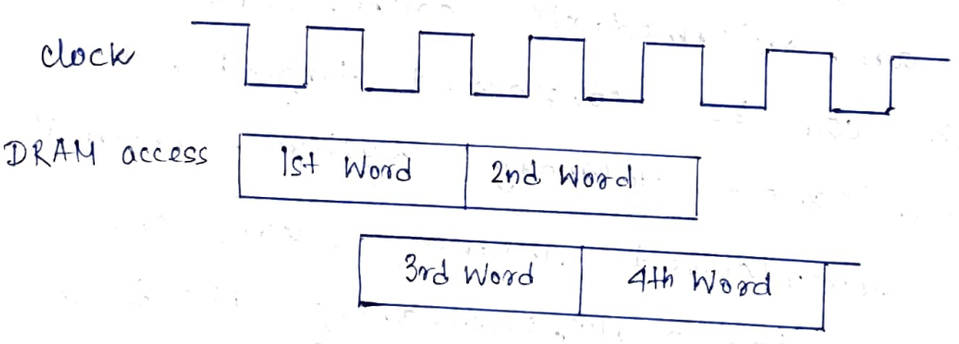
If the address length is $m+k$ bits, the upper (msbs) m bits select a word within a memory bank & the lower k bits select the given memory bank (/module).

The interleaved memory system is most effective when the no. of memory banks is equal to or an integer multiple of the no. of words in a cache line.

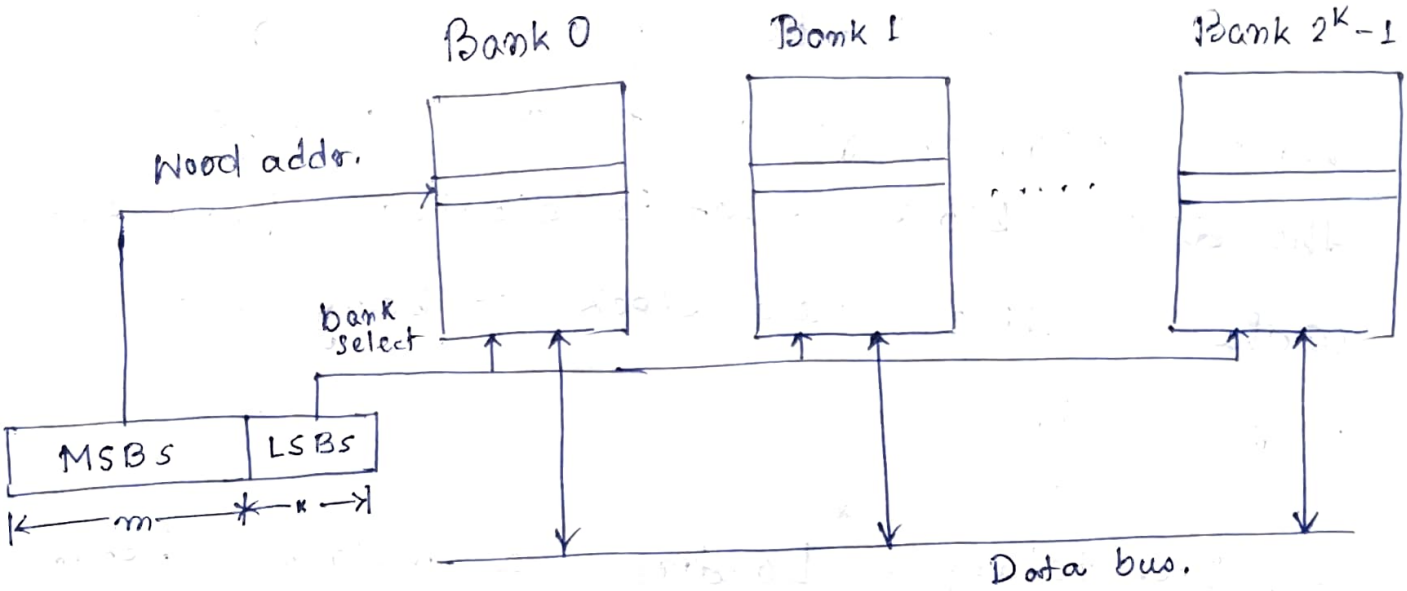
2-way interleaved memory ⟹



Interleaved Memory Organisation



Interleaved memory timing



2. Caches on the processor chip:

Using level 1 & level 2 cache.

L1 cache on processor chip & L2 cache outside.

Cache increases hit rate.

$$t_{avg} = h_1 C_1 + (1-h_1) C_2 h_2 + (1-h_1)(1-h_2) M.$$

3. Using Write-Buffer: When the write-through protocol (data simultaneously updated in the main memory along with cache) is used, each write operation results in writing a new value into the main memory. To improve performance (slowing down due to all write requests), a write buffer is included for temporary storage of write requests. Processor places each write request into this buffer & continues execution of the next instruction. Write requests stored in the buffer are sent to the memory whenever the memory is not responding to read requests. Always, read requests have more priority than write requests as the processor can't wait to read data from memory.

For a write-back protocol, a fast write buffer, for temporary storage of the dirty block that is ejected from the cache while new block is being read, is provided.

4. Prefetching: Loading a resource before it is required to decrease the time waiting for that resource.

eg. Web browser requesting copies of commonly accessed web pages.

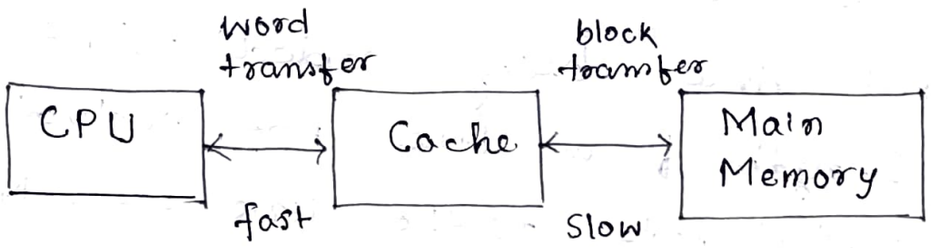
5. Lock-up free cache: A cache that can support multiple outstanding misses. A processor can access locked cache while a miss is being serviced. Since, cache can service one miss at a time so it has to keep track of miss by using special register to service multiple outstanding misses.

(While a prefetch is getting done, other accesses to the cache is stopped. It is called a lock for the cache while it services a miss.)

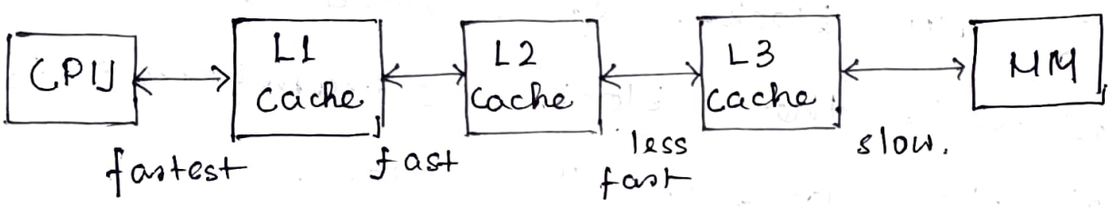
→ Memory Latency: Time to transfer a word of data to or from the memory.

Memory bandwidth: No. of bits or bytes that can be transferred in one second.

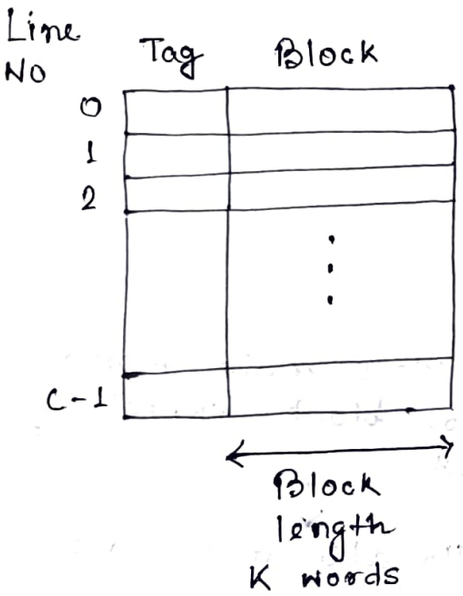
* Cache Memory: Intended to give memory speed approaching that of the fastest memories available and at the same time provide a large memory size. Cache contains a portion of main memory.



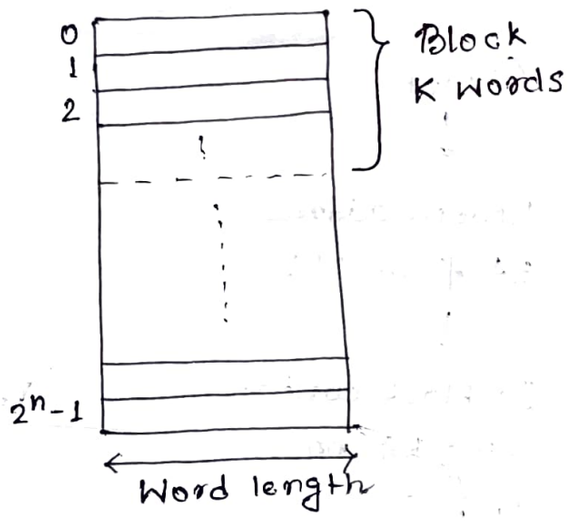
Single Cache



Three Level Cache Organization.



Cache



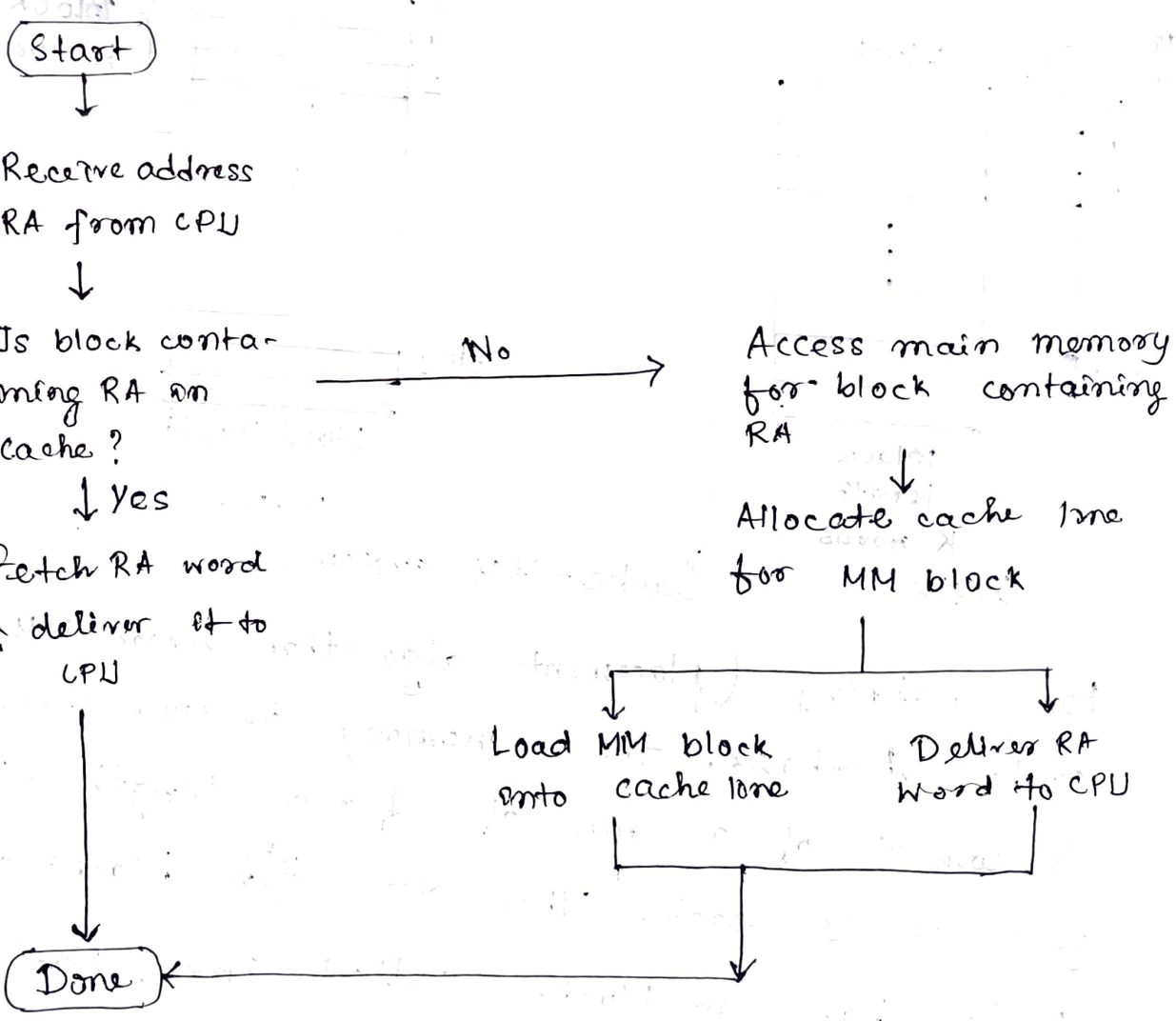
M.M.

Cache - M.M system.

For mapping (placement algorithm from MM to cache) purposes, the memory is considered to consist of a no. of fixed length blocks of K words each. There are $2^n/K$ blocks in MM. Cache has m blocks, called lines. Each line consists K words plus a tag of few bits.

Tag is a field that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word. Each line also includes control bits, such as a bit to indicate whether the line has been modified since being loaded into the cache. Length of a line, not including tag & control bits is the line size. No. of lines is considerably less than the no. of MM blocks. If a word in a MM block is read, that block is transferred to one of the lines in the cache.

↝ Cache READ



→ Impact of temporal locality :

$$t_{avg} = \frac{nt_c + t_m}{n} = t_c + \frac{t_m}{n}$$

$n \uparrow \Rightarrow t_{avg} \downarrow$

$t_m \rightarrow$ time to fetch MM block to cache

$t_c \rightarrow$ cache fetching time to processor

$n \rightarrow$ no. of repeated accesses

assumed that requested memory element has created a cache miss, thus leading to transfer of MM block in time t_m.

→ Impact of spatial locality :

$$t_{avg} = \frac{mt_c + t_m}{m} = t_c + \frac{t_m'}{m}$$

$m \uparrow \Rightarrow t_{avg} \downarrow$

assumed that the requested memory element has created a cache miss thus leading to the transfer of a MM block, consisting of m elements in time t_m.

→ Impact of combined temporal & spatial locality :

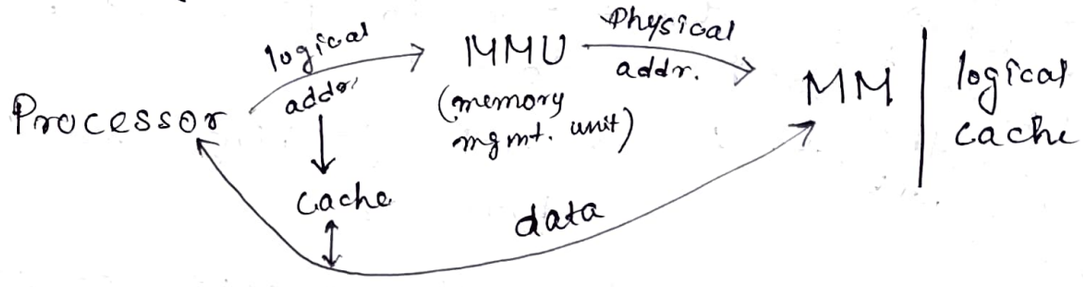
$$t_{avg} = \frac{\left(\frac{mt_c + t_m}{m}\right) + (n-1)t_c}{n} = \frac{t_m}{nm} + t_c.$$

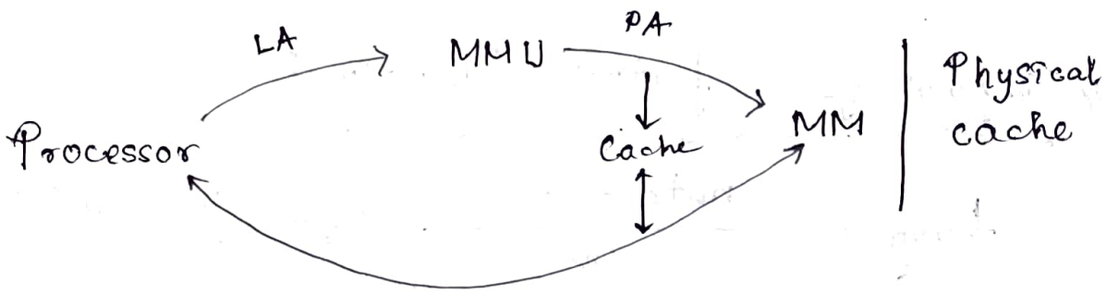
If $t_m = mt_c$, $t_{avg} = \frac{n+1}{n} t_c$.

As n increases, $t_{avg} \rightarrow t_c$.

[Assumed that the element requested created a cache miss leading to the transfer of block, consisting of m elements to the cache. Due to spatial locality all m elements were requested one at a time (mt_c). Following that therequested element was accessed $(n-1)$ times. (temporal Locality)].

→ Cache Addresses : A logical cache (virtual cache) stores data using virtual addresses. A physical cache stores data using main memory physical addresses.





→ Cache Size : Size should be small enough so that the overall average cost/bit is close to that of MM alone & large enough so that the overall average access time is close to that of the cache alone.

→ Cache Line Size : As the block size increase from very small to larger sizes, the hit ratio will at first increase because of the principle of locality. However the hit ratio will begin to decrease, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced.

• Larger blocks reduce the number of blocks that fit into a cache. Because, each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.

• As a block becomes larger, each additional word is farther from requested word & therefore less likely to be needed in the near future.

→ Unified Cache : Data and instructions are cached in the same cache.

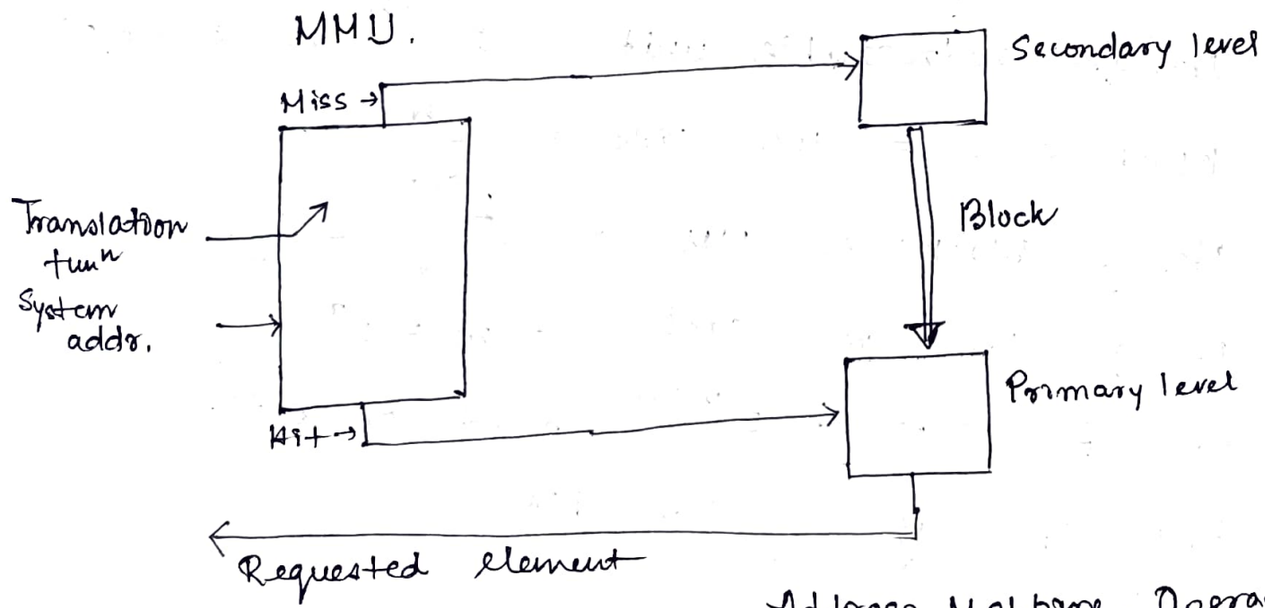
Advantages ~ Balances possible imbalance between amount of data & instructions in a program.

Split Cache : Separate caches for data and instructions.

Advantages ~ Eliminates contention between inst^n fetch & execute units. Supports pipelining & speculative execution.

→ Mapping Functions.

Algorithm for mapping main memory blocks into cache lines.



Address Mapping Operation.

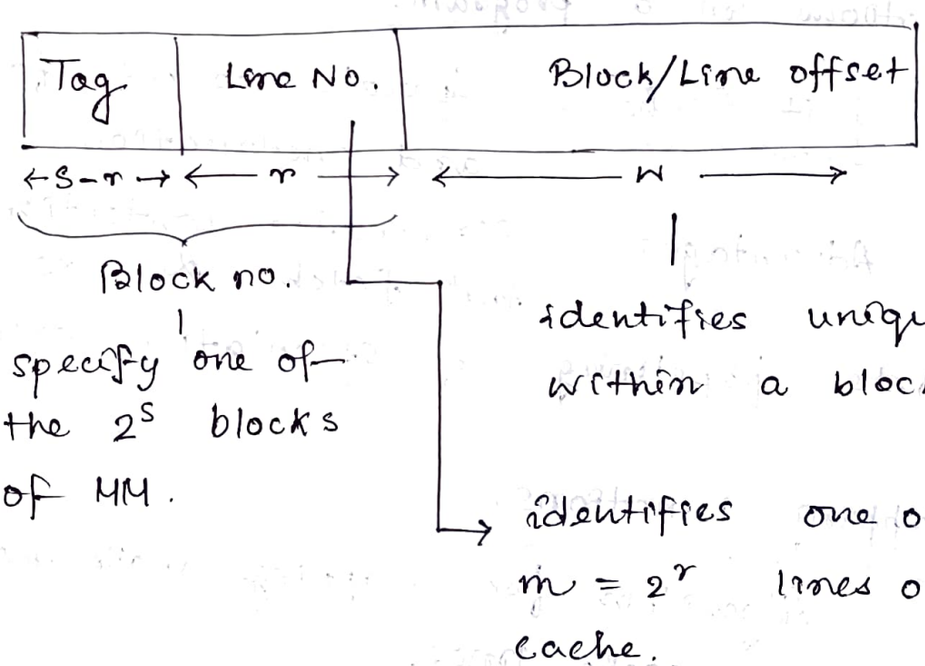
There are three different mapping techniques.

1. Direct Mapping: Maps each block of main memory into only one possible cache line. Expressed as.

$$\text{Cache line no.} = \left(\begin{matrix}\text{MM block}\\ \text{no.}\end{matrix}\right) \text{modulo} \left(\begin{matrix}\text{No. of}\\ \text{cache lines}\end{matrix}\right)$$

Direct mapping's performance is varied proportionately to the hit ratio.

~ Division of Physical address ~



Address length = $s + w$ bits

No. of addressable unit = 2^{s+w} words or bytes

Block size = line size = 2^w

No. of blocks in MM = $\frac{2^{s+w}}{s^w} = 2^s$.

No. of lines in cache = $m = 2^r$

Size of cache = 2^{r+w} words or bytes

Size of tag = $s - r$ bits

→ Working : After CPU generates a memory request ~

a) Line no. field of address is used to access the particular line of cache.

b) Tag field of CPU address is then compared with tag of cache line.

i) If 2 tags match, cache hit occurs and desired word found in cache.

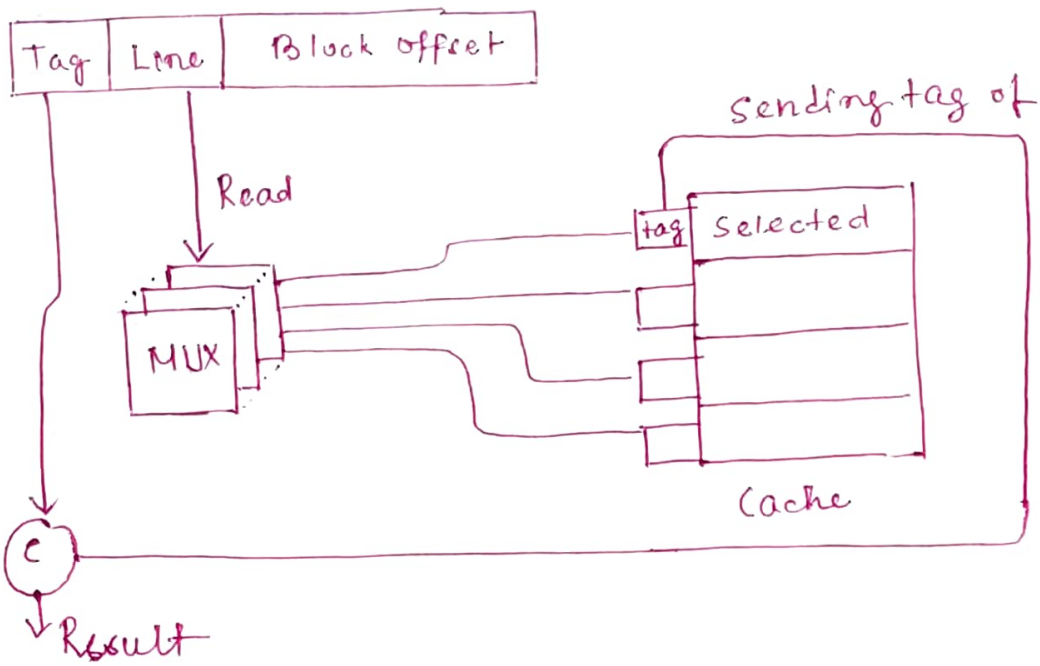
ii) If 2 tags don't match, cache miss occurs & required word has to be brought from main memory. It's then stored in cache together with the new tag replacing previous one.

Cache line.	MM blocks assigned.
0	0, m, 2m, ..., $2^s - m$
1	1, m+1, 2m+1, ..., $2^s - m + 1$
⋮	⋮
m-1	m-1, 2m-1, ..., $2^s - 1$.

→ Drawbacks : There's fixed cache location for any given block. If a program happens to reference words repeatedly from 2 different blocks that map into same line, then blocks will be continually swapped in cache & hit ratio will be low. It is known as thrashing.

Merits : Hardwire is simple. The cache controller quickly issues hit or miss information.

- Direct Mapping.



→ Hit latency = mux latency + comp. latency.

→ No. of mux required = no. of bits in one tag

→ Size of each mux = No. of lines in cache x 1

→ No. of comparators = 1

→ Size of comparator = no. of bits in tag.

eg. Consider a machine with byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct-mapped cache consisting of 32 lines is used.

a) How is a 16 bit memory address is divided ?

b) Into what line would bytes with each of the following addresses be stored ?

i) 0001 0001 0001 1011

ii) 1100 0011 0011 0100

iii) 1101 0000 0001 1101

iv) 1010 1010 1010 1010.

c) Suppose the byte with addr. 0001 1010 - 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it ?

d) How many total bytes of memory can be stored in the cache ?

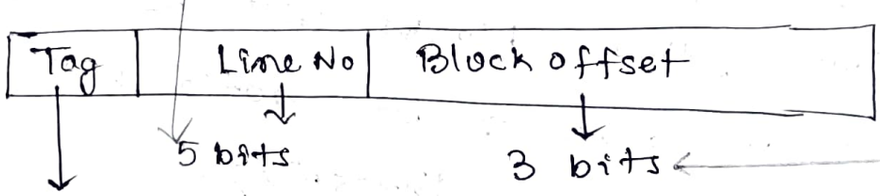
→ MM size = 2^{16} bytes

Block size = Cache line size = 8 bytes = 2^3 B

No. of blocks in MM = $\frac{2^{16}}{2^3} = 2^{13}$

Cache lines = 32 = 2^5.

(a)



16 - 5 - 3
= 8 bits

(b) (i) Line no.

00011 3

(ii) 00110 6

(iii) 00011 3

(iv) 10101 21.

(c) 0001 1010 0001 1010

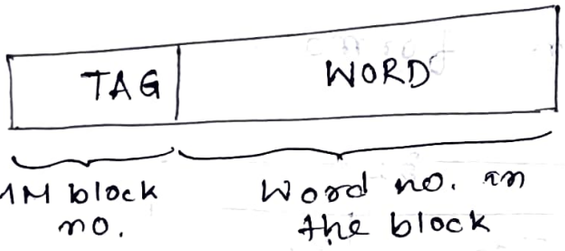
other bytes -

0001 1010 0001 1000 through

0001 1010 0001 1111.

(d) Total bytes that can be stored in the cache = 32 x 8 = 256 bytes.

2. Associative Mapping : Any memory block can be mapped to any cache line.



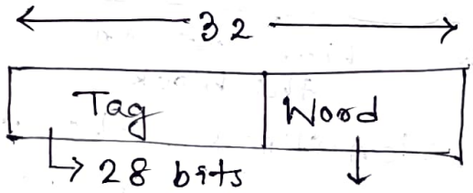
The tag field identifies the memory block number. Here, the tag field of the address needs to be matched with all tag contents of cache. The block read from MM is stored in any line if cache is not full. If cache is full, only replacement of a block in cache is required. Which block has to be replaced is decided by the replacement algorithms followed by the cache controller.

→ Merits - Offers total flexibility. No. of lines in cache is not fixed in associative mapping.

Demerits - Costly system for implementation. The cache controller becomes complex since the tags of all cache lines are matched simultaneously in order to minimise the cache controller delay. Parallel searching of all cache lines is done for a specific tag pattern (Associative search). A special memory called associative memory or constant addressable memory (CAM) is used for this purpose.

eg. A 32 bit computer has 32 bit mem. addr. Considering associative mapping, each cache line is 16 bytes. Show memory addr. format.

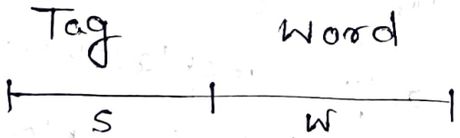
→



$16 = 2^4$

(specifying the word no. within the block)

→



Addr. length = S+W bits

No. of addressable units = 2^{S+W} words or bytes

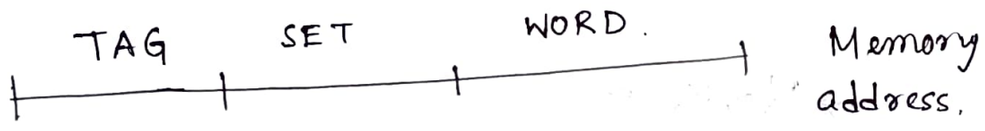
Block size = Line size = 2^W words or bytes

No. of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$

No. of lines in cache = undetermined

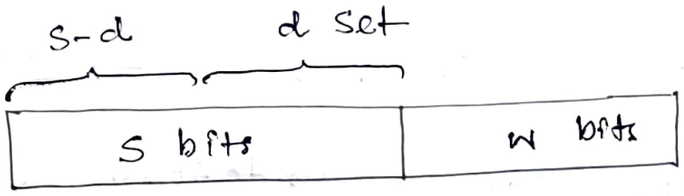
Size of tag = s bits.

3. Set-Associative Mapping : Combines concept of direct mapping with associative mapping to provide a reasonably flexible mapping scheme. Total number of cache lines are grouped into multiple sets. Each set has multiple lines of equal number. If there are k lines in each set, the mapping system is known as k-way set associative. A mm block can be mapped into a specific set only. Within a set, the memory block can be placed in any of the k lines.



Set field gives the set number.
Word field gives the word no. in a block.

When the processor performs a memory read operation, the cache controller uses the set field to access the cache. Tag field in the addr. is matched with the tag contents of k lines in the set. If there is only one set it is same as associative mapping. If there is one line, it is same as direct mapping.



S+w bits address.

Block size = line size = 2^w words or bytes

No. of blocks in MM = $\frac{2^{s+w}}{2^w} = 2^s$

No. of lines in set = k

No. of sets = $v = 2^d$

No. of lines in cache = $m = kv = k \times 2^d$

Size of cache = $k \times 2^{d+w}$ words or bytes

Size of tag = $(s-d)$

→ No. of lines in cache = no. of sets × no. of lines in each set

⇒ cache set no = (MM block no.) modulo (no. of sets)

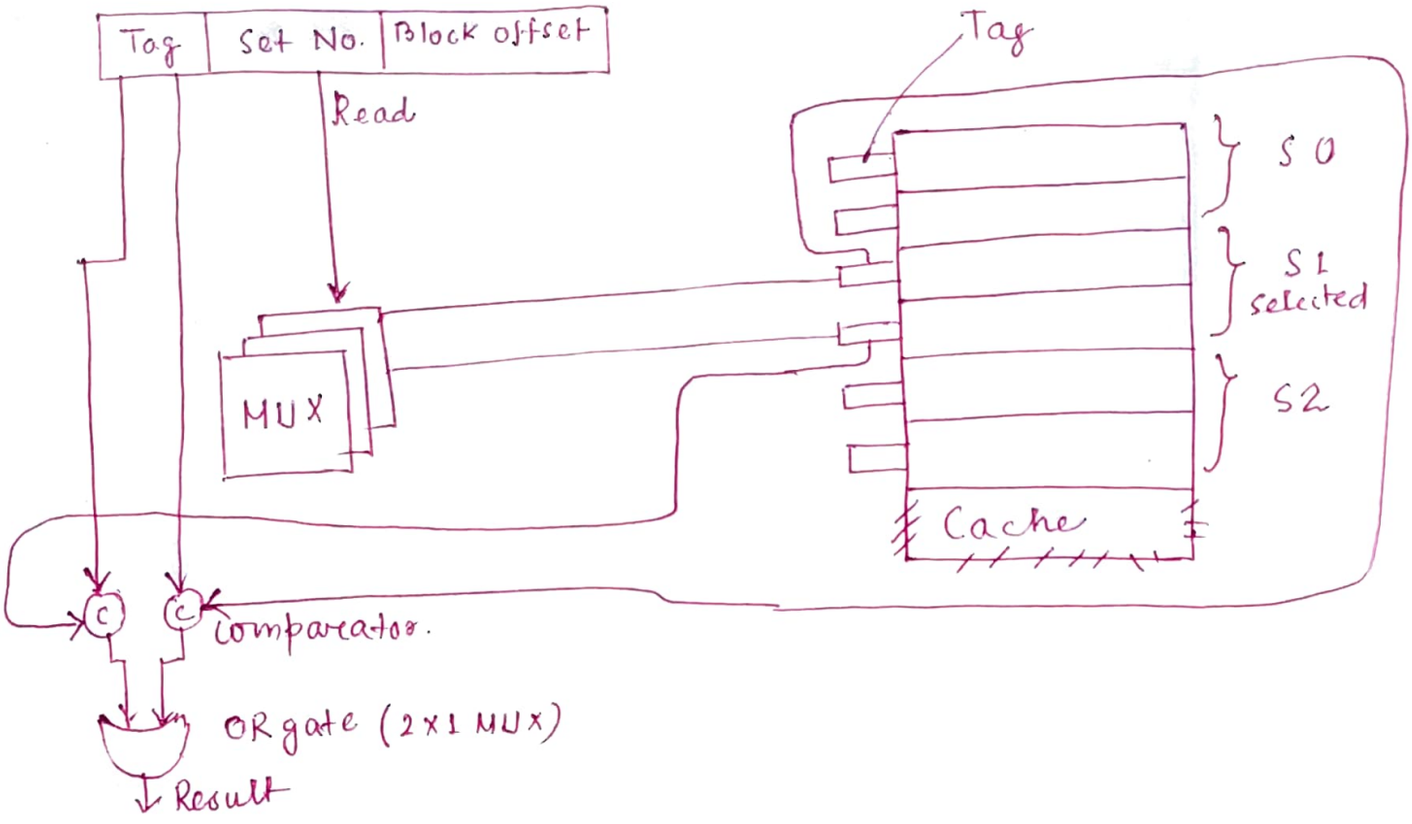
→ Merits ~

1. Compared to direct mapping, multiple choices are available for mapping a memory block.

2. During reading, the tag matching is limited to the number of lines in the set.

Demerits ~ Implementation cost is more than direct mapping.

- 2-way set associative mapping -



→ To output one complete tag to the comparator,
no. of mux's required = no. of bits in tag

→ If there are K lines in one set, no. of tags to output = K, thus
no. of mux required = K x no. of bits in one tag.

→ Each mux selects the tag bit for which it has been configured & outputs on the output line.

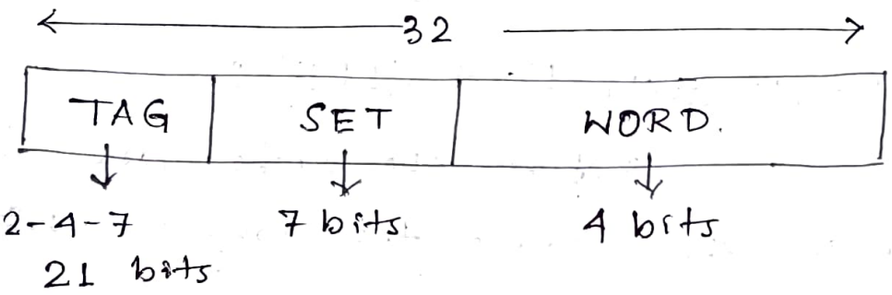
→ Complete tags as whole are sent to the comparators for comparison in parallel.

- Hit latency = mux latency + comparator latency + OR gate latency

- No. of comparators required = no. of lines in one set
Size of each comparator = no. of bits in the tag.

eg. A 32 bit computer has 32 bit memory address. It has 8 KB of cache memory. Follows 4-way set associative mapping. Each line size is 16 bytes. Show memory addr. format.

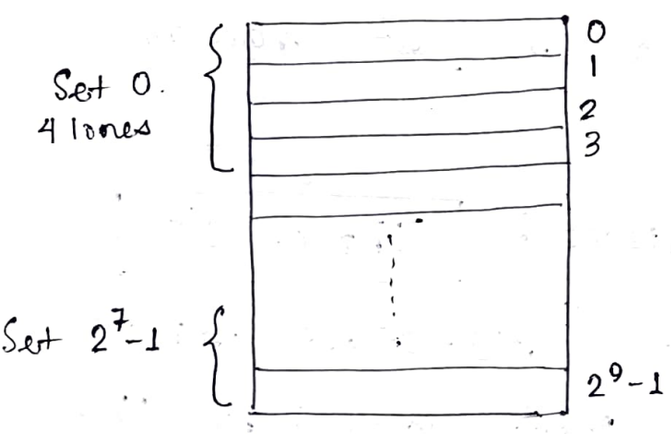
→

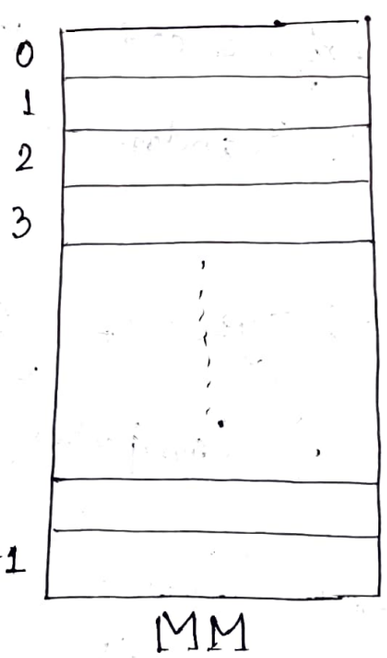


Line size = 16 bytes. = 2^4 bytes.

No. of cache lines = $\frac{8\ KB}{16\ B}$ = 512 = 2^9

No. of sets = $\frac{2^9}{4}$ = 2^7.





* Cache Replacement : Replacing existing contents in the cache for new entry. Portion of cache memory which is deleted is matter of interest.

→ Replacement Algorithms :

Direct mapping → No replacement algo. needed

Associative & Set associative mapping → Needed

1. LeastRecently Used (LRU):
Replace the cache line that has been in the cache the longest with no references to it.

2. FIFO : Replace the line that has been in the cache the longest.

3. Least Frequently used (LFU):
Replace the line that has experienced the fewest references.

4. Random : Pick a line at random

→ Types of Cache Misses : Three C's

i) Compulsory miss ~ (Cold-start miss)
Miss caused by the first access to a block that has never been in the cache.

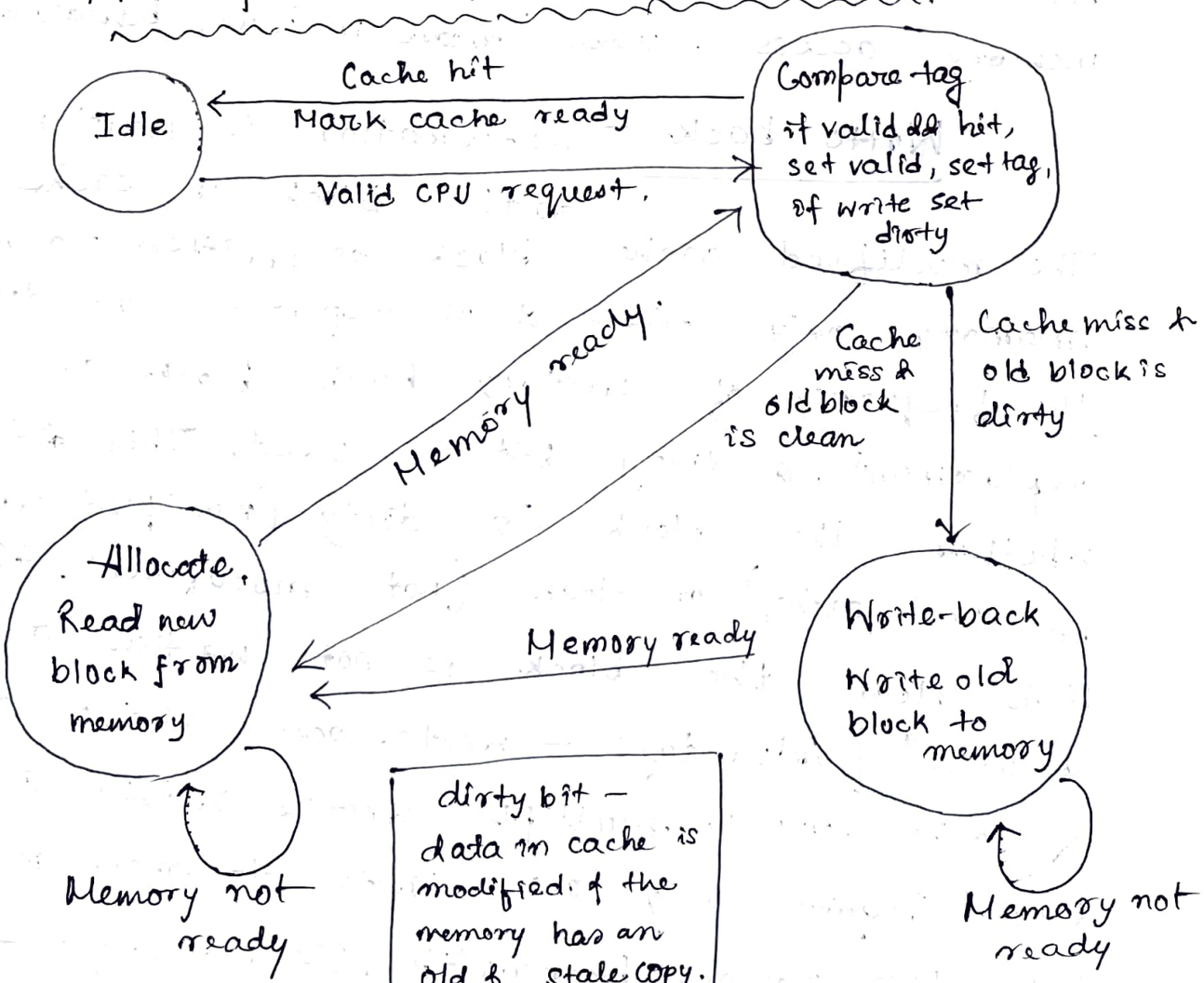
ii) Capacity miss ~ Occurred because the cache, even with full associativity, cannot contain all the blocks needed to satisfy the request.

iii) Conflict/Collision miss ~ Occurs in a set-associative or direct-mapped cache when multiple blocks compete for the same set & that are eliminated in a fully associative cache of same size.

→ Memory hierarchy design challenges ~

Cache size increase	—	Decreases capacity misses	— May increase access time
Associativity increases	—	Decreases miss rate due to conflict misses	— may increase access time
Block size increases	—	Decreases miss rate for a wide range of block sizes due to spatial locality	— Increases miss penalty. Very large block could increase miss rate.

→ FSM for a simple cache controller :



⇝ Interaction Policies with Main memory :

→ Read Policies –

Read Through – Reading a word from main memory to CPU.

No read through – Reading a block from main memory to cache and then from cache to CPU.

→ Write policies on write hit –

Write through – The information is written to both the block in the cache and to the block in the lower-level memory. [Advantage – read miss never results in writes to main memory, easy to implement, main memory always has the most current copy of the data (consistent). Disadvantage – write is slower, every write needs a main memory access, uses more memory bandwidth]

Write back – Information is written only to the block in cache. The modified cache block is written to main memory only when it is replaced. To reduce the frequency of writing back blocks on replacement, a dirty bit is used. This indicates whether the block is dirty (modified while in the cache) or clean (not modified). If it is clean the block is not written on a miss.* [Advantage – writes occur at the speed of cache memory, multiple writes within a block require only one write to main memory*, uses less memory bandwidth. Disadvantage –

harder to implement, main memory is not always consistent with cache, reads that result in replacement may cause writes of dirty blocks to main memory.*]

→ Write policies on write miss –

Write allocate – The block is loaded on a write miss, followed by the write-hit action.

No write allocate – The block is modified in the main memory and not loaded into the cache.

→ Although either write-miss policy could be used with write through or write back, write back caches generally use write allocate (hoping that subsequent writes to that block will be captured by the cache) and write through caches often use no-write allocate (since subsequent writes to that block will still have to go to memory.)

→ Possible combinations of interaction policies on write —

i) Write through with write allocate:

On hits it writes to cache and main memory. On misses it updates the block in main memory and brings the block to the cache.

*Bringing the block to cache on a miss does not make a lot of sense in this combination because the next hit to this block will generate a write to MM anyway.

ii) Write through with no write allocate:

On hits it writes to cache and MM. On misses it updates the block in main memory and ~~bring~~ does not bring the block to cache. Subsequent writes to the block will update main memory because write through is employed. So, some time is saved not bringing the block in the cache on a miss because it appears useless anyway.

iii) Write back with write allocate:

On hits it writes to cache setting dirty bit for the block, main memory is not updated. On misses it updates the block in MM ~~not~~ and brings the block onto the cache. Subsequent writes to the same block, if the block originally caused a miss, will hit in the cache next time, setting dirty bit for the block. That will eliminate extra memory accesses & result in very efficient execution compared with write through with write allocate combination.

iv) Write back with no write allocate:

On hits it writes to cache setting dirty bit for the block, main memory is not updated. On misses it updates the block in MM not bringing that block to the cache. Subsequent writes to the same block, if the block originally caused a miss, will generate misses all the way & result in very inefficient execution.

→ Cache Performance :

Assuming the costs of cache accesses that are hits are part of the normal CPU execution cycles.

$$\text{CPU time} = \left(\begin{array}{c}\text{CPU execution} \\ \text{clock cycles}\end{array} + \begin{array}{c}\text{Memory - stall} \\ \text{clock cycles}\end{array}\right) \times \begin{array}{c}\text{Clock cycle} \\ \text{time}\end{array}$$

Memory-stall clock cycles come primarily from cache misses.

$$\begin{array}{c}\text{Memory-stall} \\ \text{clock cycles}\end{array} = \begin{array}{c}\text{Read-stall} \\ \text{cycles}\end{array} + \begin{array}{c}\text{Write - stall} \\ \text{cycles.}\end{array}$$

$$\begin{array}{c}\text{Write - stall} \\ \text{cycles}\end{array} = \left(\frac{\text{Writes}}{\text{Program}} \times \begin{array}{c}\text{write miss} \\ \text{rate}\end{array} \times \begin{array}{c}\text{Write miss} \\ \text{penalty}\end{array}\right) + \begin{array}{c}\text{Write} \\ \text{buffer} \\ \text{stalls.}\end{array}$$

If we ~~use~~ assume that the write - buffer stalls are negligible, we can combine the reads & writes by using a single miss rate & miss penalty —

$$\begin{array}{c}\text{Memory-stall} \\ \text{clock cycles}\end{array} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{miss rate} \times \text{miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty.}$$

eg. Miss rate of an ins^h cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls & the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads & stores is 36%.

→ Ins^h count = I

Ins^n miss cycles = I × 2% × 100 = 2I

Data miss cycles = I × 36% × 4% × 100

= 1.44 I

Total no. of memory-stall cycles = 3.44 I

Total CPI including memory-stalls =

2 + 3.44 = 5.44.

Ratio of CPU execution time =

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{stall} * \text{Clock cycle}}{I \times CPI_{perfect} \times \text{Clock cycle}}$$

$$= \frac{CPI_{stall}}{CPI_{perfect}} = \frac{5.44}{2} = 2.72$$

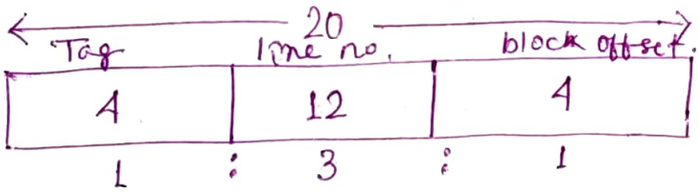
Performance with the perfect cache is better by 2.72

Q. G'16 A processor can support a maximum memory of 4 GB where the memory is word addressable (a word is 2 bytes). The size of the address bus of the processor is at least 31 bits.

→ $\frac{4GB}{2B} = 2^{31}$ addressable units.

Q. G'15 Consider a machine with a byte addressable main memory of 2^{20} B, block size of 16 B & direct mapped cache having 2^{12} cache lines. Let the address of two consecutive bytes in main memory be $(E201F)_{16}$ & $(E2020)_{16}$. What are the tag and cache line address for mm address $(E201F)_{16}$?

→



E 201 F.

Answer: E, 201.

Q. G'15 Assume that for a certain processor, a read request takes 50 ns on a cache miss & 5 ns on a cache hit. Suppose while running a program, it was observed that 80% of processor's read request result in a cache hit. The average read access time in ns is 14.

$(0.8 \times 5 + 0.2 \times 50) = 14$ ns.

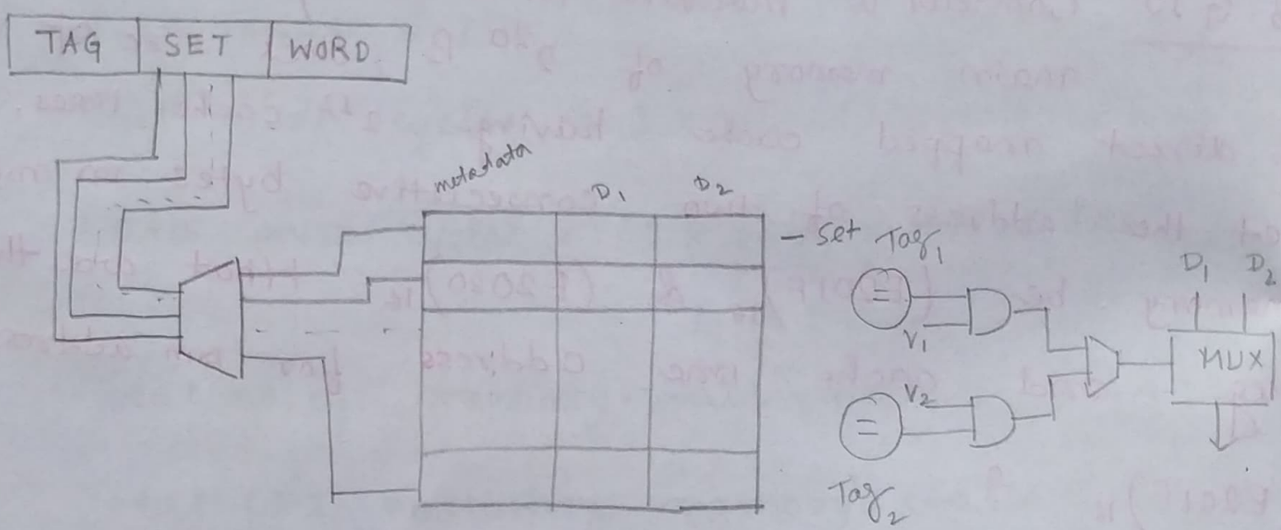
Q. G'14 If the associativity of a processor is doubled while keeping the capacity & block size unchanged which one of the following is guaranteed to be not affected?

a) Width of tag comparator. (t)

b) Width of set index decoder (s)

c) Width of way selection multiplexer. (N)

d) Width of processor to MM data bus.



Q. G'14 ** An access sequence of cache block address has length N & contains n unique block addresses. The number of unique block addresses between two consecutive accesses to the same block is bounded above by K. What is the miss ratio if the access sequence is passed through a cache of associativity A ≥ K exercising least recently used replacement policy?

Ans - $\left(\frac{n}{N}\right)$ [refer GO]

→ Access sequence

$N = 7$

$n = 5$

$a_1\ a_2\ a_3\ a_4\ a_5\ a_1\ a_2$

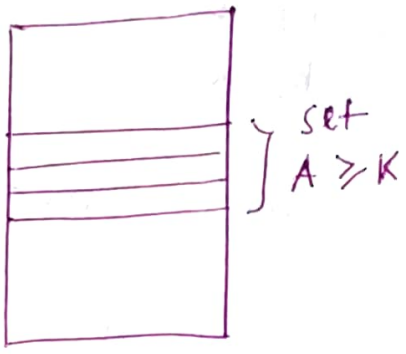
K

N - Total no. of addresses (7)

n - no. of unique references/addresses (5)

a b c d a

K



eg.

$a_1\ a_2\ a_3\ a_4\ a_1\ a_5\ a_6\ a_4$

$K = 4$

$N = 8$

$n = 6$.

First occurrence will always be cache ~~hit~~ miss.

3. In designing a computer's cache system, the cache block (or cache line) size is an important parameter. Which is correct?

a) A smaller block size implies better spatial locality. (X)

b) A smaller block size implies a smaller cache tag & hence lower cache tag overhead. (X)

c) A smaller block size implies a larger cache tag & hence lower cache hit time. (X)

✓ d) Smaller block size incurs a lowers cache miss penalty.

Q G'12 **

Consider 2 cache organizations : first one is 32KB 2 way set associative with 32 byte block size. Second is of same size but direct mapped. Size of an address is 32 bits in both cases. A 2 to 1 mux has latency of 0.6 ns while a K-bit comparator has a latency of K/10 ns. The hit latency of set associative organ is h_1, while that of the direct mapped one is h_2. Valu of h_1 is — 2.4 ns.
h_2 is. —

[refer GO]

→ h_1 = time to compare the data + time to selects the bloc in set.

= 0.6 + 18 (latency of 1-bit comparator)

= 0.6 + 18 × 1/10 = 2.4 ns.

set-associative.

18	9	5

Q G'12 A computer has a 256 KB, 4 way set associative write back data cache with block size of 32 B. Processor sends 32 bit addresses to the cache controller. Each cache tag directory entry contain in addition to address tag, 2 valid bits, 1 modified bit & 1 replacement bit. No. of bits in the tag field of an address is — 16.

→ No. of blocks = $\frac{256\ KB}{32\ B} = \frac{2^{18}\ B}{2^5\ B} = 2^{13}$

No. of sets = $\frac{2^{13}}{4} = 2^{11}$.

16	11	5
Tag	set	Block offset

Q. G'09 Consider a 4-way set-associative cache (initially empty) with total 16 cache blocks. The MM consists of 256 blocks & the request for memory blocks is in the order-

0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155.

Which one block will not be in cache if LRU replacement is used?

a) 3 b) 8 c) 129 ~~d)~~ d) 216.

→ No. of sets = $\frac{16}{4}$ = 4.

Set no. = (block address) mod 4.

0 - 0
255 - 3
1 - 1
4 - 0
3 - 3
8 - 0
133 - 1
159 - 3
216 - 0
129 - 1
63 - 3
8 - 0
48 - 0
32 - 0
73 - 1
92 - 0
155 - 3

Set 0: 0 ← 48, 4 ← 32, 8, 8, 216, 92

Set 1: 1, 133, 129, 73

Set 2:

Set 3: ~~255~~ ← ~~73~~ 155, 3, ~~155~~, 159, 63

For each set 0, 1, 2, 3, last 4 accesses will be in cache.

{8, 48, 32, 92} {1, 133, 129, 73} {} {155, 3, 159, 63}

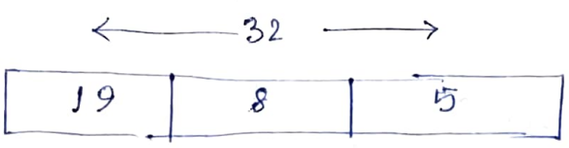
set 0 set 1 set 2 set 3

Ans 216.

Q G'11 An 8 KB direct mapped write-back cache is organised as multiple blocks, each of size 32 B. Processor generates 32 bit addresses. Cache controller maintains the tag info for each cache block comprising of the following. - 1 valid bit, 1 modified bit. As many bits as the minimum needed to identify the memory block mapped in the cache. What's the total size of memory needed at the cache controller to store metadata for the cache ?

→ No. of cache lines = $\frac{8\,KB}{32\,B} = 2^8$

$32 = 2^5$



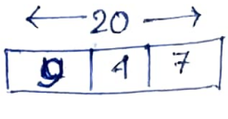
Meta data for 1 line = 1 + 1 + 19 = 21 bits required

Total memory = $21 \times 2^8 = 21 \times 256$ bits

= 5376 bits (Ans)

Q G'08 Consider a computer with a 4-way set associative mapped cache of characteristics: a total of 1 MB of MM, a word size of 1 B, a block size of 128 words and a cache size of 8 KB. While accessing the memory location 0C795H by the CPU, the contents of the TAG field of the corresponding cache line is ____. 0000 1100 0 (Ans)

$1\,MB = 2^{20}\,B$.



→ $\frac{2^3 \times 2^{10}}{2^7} = 2^6$ cache lines

$2^6/2^2 = 2^4$ sets.

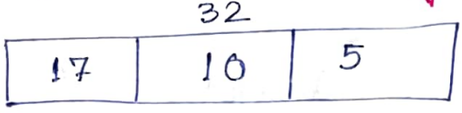
0C795H ≡ 0000 | 1100 | 0111 | 1001 | 0101

Hex

tag

Q. G'07 Consider a 4-way set-associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. The no. of bits in the TAG, LINE & WORD fields — 9, 5, 6.

Q. G'05 Consider a direct mapped cache of size 32 KB with block size 32 B. The CPU generates 32 bit addresses. The no. of bits needed for cache indexing & the no. of tag bits are — 10, 17



Q. G'06 A CPU has a 32 KB direct mapped cache with 128 byte block size. Suppose A is 2D array of size 512 x 512 with elements that occupy 8 bytes each. Consider following

```
P1: for(i=0; i<512; i++)
  { for(j=0; j<512; j++)
      x += A[i][j];
  }
```

```
P2: for (i=0; i<512; i++)
    { for (j=0; j<512; j++)
        x += A[j][i];
    }
```

P1 & P2 are executed independently with the same initial state, namely, the array A is not in the cache & i, j, x are in registers. Let the no. of cache misses experienced by P1 be M_1 & that for P_2 be M_2.

Values of M_1, M_1/M_2 are —

[refer GO]

→ No. of cache lines = $\frac{2^{15}}{2^5} = 256$

In 1 cache line = $\frac{128}{8} = 16$ elements

$M_1 = \frac{512 \times 512}{16} = 16384.$ (Ans)

[When A[0][0] is fetched, 128 consecutive bytes are moved to cache ; for nex

$\frac{128}{8} - 1 = 15$ references there won't be mi

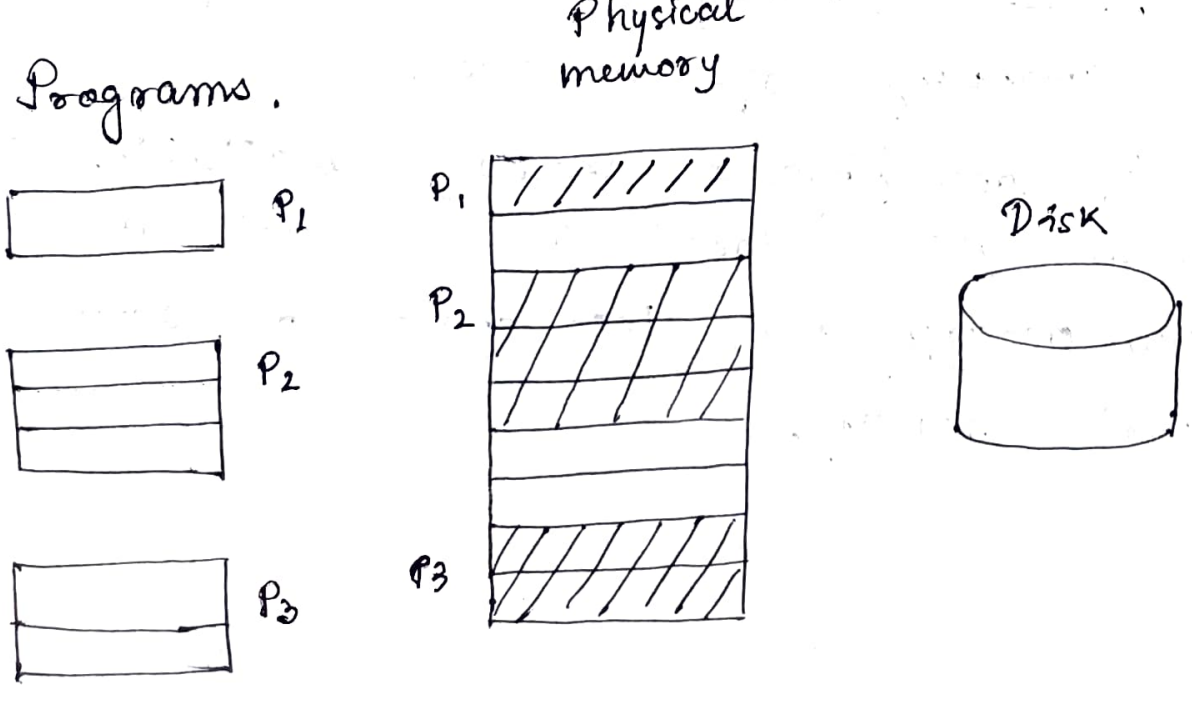
A[0][0]	miss	For P_2, for every element
A[0][1]	hit	there's a miss because
⋮		storage is row major (def
A[0][15]	hit	& we are accessing column w
A[0][16]	miss	
⋮		$M_2 = 512 \times 512.$
A[0][31]	hit	
⋮		

$M_1 / M_2 = 1/16$ (Ans)

* Virtual Memory

Memory that appears to exist as main storage although most of it is supported by data held in secondary storage, transfer between the two being made automatically as required.

→ Segmented memory ~ Segments are swapped between disk and main memory as needed. Program segments correspond to blocks of program code such as procedures or functions. Data segments correspond to data structures such as stacks, queues or graphs. Segments vary in size. The OS knows start and size of each segment in physical memory. Each segment is atomic, either the whole segment is in RAM or none of the segment is in RAM. A segment in memory can only be replaced by a segment of the same size or smaller. Segmentation can result in memory fragmentation; a lot of small segments with gaps in between. Large segments may not be allowed into memory very often. Segments can be pushed together to limit fragmentation & allow large segments to be loaded.



→ Paged Memory : Memory is split up into small equal sized sections called pages (or page frames). A single application may occupy multiple pages, which are not necessarily contiguous. Each application program has its own view of the memory, known as logical memory. A page table records where the different pages of a program are located. In physical memory. Unused pages may be paged out to a swap file on disk to make room for others. Pages are paged in when needed. When memory is low, excessive swapping can lead to disk thrashing and degrade performance.

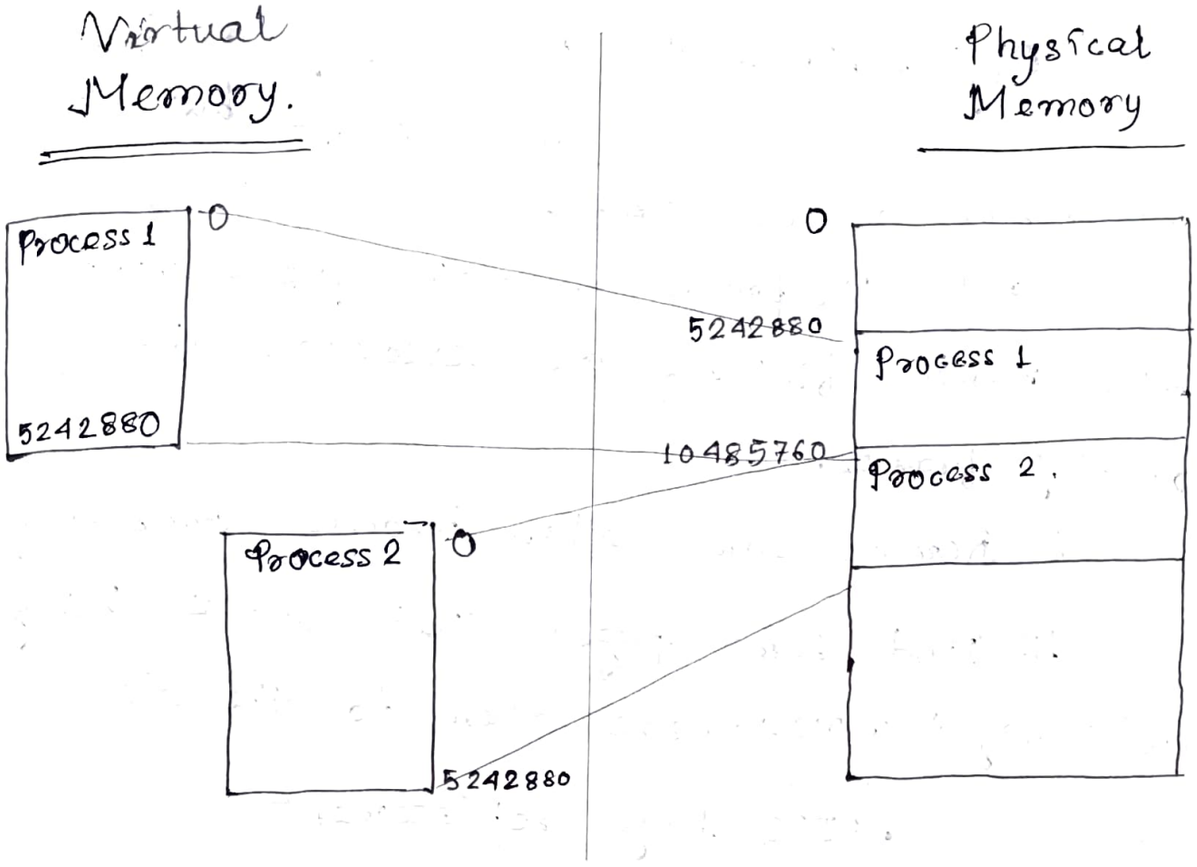
→ Virtual memory is a logical space where large programs can store themselves in form of pages while their execution & only the required pages or portions of processors are loaded into the MM.

Benefits ~

1. Large programs can be written, as virtual space available is huge compared to physical memory.

2. Less I/O required, leads to faster & easy swapping of processes.

3. More physical memory available., as programs are stored on virtual memory, so they occupy very less space on actual physical memory.



→ Page table ~ Data structure to store the mapping between virtual addresses and physical addresses.

→ Demand Paging ~ When a process is swapped in, all pages are not swapped in all at once. Rather they are swapped in only when the process needs them (on demand). This is also called lazy swapping.

→ When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered & following steps are followed ~

1. Memory address is first authenticated.
2. If it's valid, a free frame is located.
3. Process's page table is updated with the new frame number & invalid bit is changed to valid.

→ Page replacement : Process requests for more pages but no free memory is available.

i) Put the process in wait queue, until any other process finishes its execution thereby freeing frames.

ii) Remove some other process completely.

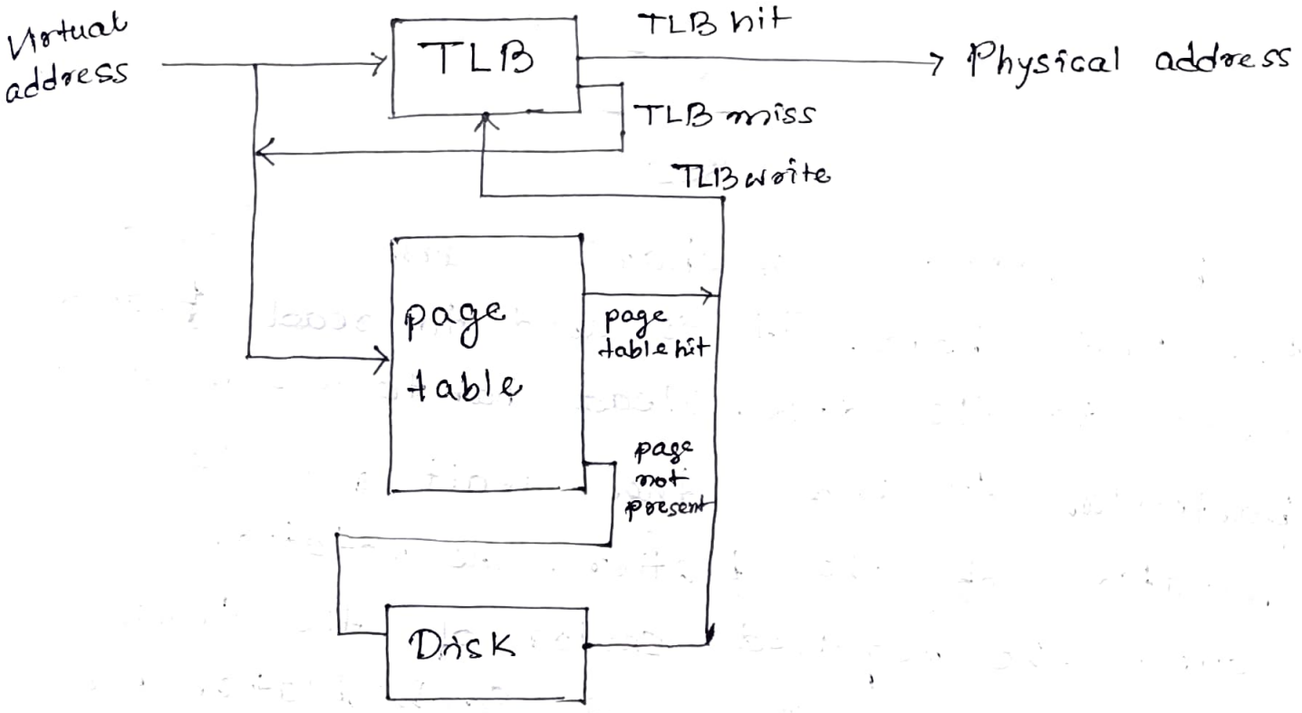
iii) Find some pages that aren't being used right now, move them to the disk.

- FIFO page replacement
- LRU page replacement

→ Thrashing ~ Spending more time paging than executing. Processes doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in & out very frequently to keep executing.

→ TLB (Translation Lookaside Buffer) ~

Memory cache that is used to reduce the time taken to access a user memory location. It's a part of the chip's MMU. It stores the recent translations of virtual memory to physical memory & can be called an address-translation cache.



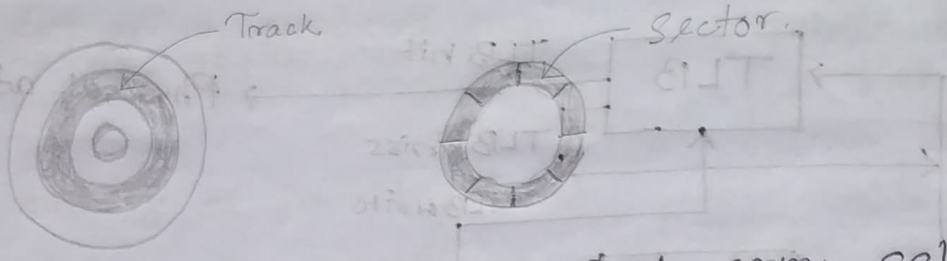
* Secondary Memory :

Used for storing data and instructions permanently. Also used for carrying data from one computer to another. Not directly accessible to CPU and slower for read/write, cheaper, non-volatile.

→ Magnetic disks (hard drives, floppy disks), optical disks (CD, CDROM), magnetic tape, pendrives, blu-ray disk

→ Magnetic Disks.

• Architecture ~ Entire disk is divided into platter (circular substrate constructed of nonmagnetic material coated with magnetic material). Each platter consists of concentric circles called as tracks. These tracks are further divided into sectors which are the smallest divisions in the disk. A cylinder is formed by combining the tracks at a given radius of a disk pack.



There exists a mechanical arm called as Read/Write head. It is used to read from & write to the disk. Head has to reach at a particular track & then wait for the rotation of the platter. The rotation causes the required sector of the track to come under the head. Each platter has 2 surfaces - top & bottom & both the surfaces are used to store the data. Each surface has its own read/write head.

⇝ CAV ~ The rate at which the disk is rotated at a fixed speed to scan the information.

For the magnetic disks, a bit near the center of a rotating disk travels ~~fast~~ fast a fixed point (such as a read/write head) slower than a bit on the outside. Therefore, the space between bits of information recorded on segments of the disk can be increased to compensate for the variation in speed so that the head can read all the bits at the same rate. By rotating the disk at CAV, we can scan information at same rate always. Disk is divided into a no. of pie-shaped sectors & into a series of concentric tracks. Hence, all tracks have same storage capacity.

→ Disk Performance Parameters :

The time taken by the disk to compensate the time of an I/O request is called disk service time or disk access time.

Factors affecting disk performance –

1. Seek time : The time taken by the read/write head to reach the desired track. (largest % of disk service time).

Specifications –

a) Full stroke – Time taken by the head to move across the entire width of the disk from the inner most track to the outermost.

b) Average – Average time taken to move from one random track to another.

Average seek time = $\frac{1}{3}$ x full stroke.

[pages.cs.wisc.edu/~remzi/OSFEP/file-disks.pdf | pg 9]

c) Track to track – Time to move between the adjacent tracks.

2. Rotational Latency : Time taken by the desired sector to come under the read/write head is called rotational delay.

Avg. rotational delay = $\frac{1}{2}$ x time taken for full rotation

3. Data Transfer Rate : Amount of data that passes under the head in a given amount of time. Time taken is transfer time.

Depends on – no. of bytes to be transferred, rotation speed of the disk, density of track, speed of electronics.

4. Controller overhead ~ The overhead imposed by the disk controller.

5. Queuing Delay ~ Time spent waiting for the disk to become free.

→ Formulae:

1. Disk access time =

seek time + rotational delay + transfer time + controller overhead + queuing delay

2. Average disk access time =

avg seek time + avg rotational delay + transfer time + controller overhead + queuing delay.

3. Average seek time =

$\frac{1}{3}$ x time taken on full stroke

or,

If time taken by the head to move from one track to adjacent track = t units & there are total k tracks, then average seek time =

$$\frac{0 + (k-1)t}{2} = \frac{k-1}{2} t.$$

4. Average rotational latency =

$\frac{1}{2}$ x time taken for one full rotation

5. Capacity of disk pack =

total no. of surfaces x no. of tracks per surface x no. of sectors per track x storage capacity of one sector

6. Formatting overhead =

no. of sectors × overhead per sector

7. Formatted disk space =

Total disk space − formatting overhead

8. Recording density / Storage density$_{track}$ =

$$\frac{\text{Capacity of track}}{\text{circumference of track.}}$$

9. Track capacity =

recording density of track × circumference

10. Data transfer rate =

no. of heads × bytes that can be read in one full rotation × no. of rotations in one second

= no. of heads × capacity of one track × no. of rotations in one second

11. Tracks per surface =

(Outer radius − inner radius) / Inner track gap

12. Transfer time = $\frac{b}{rN}$

b - no. of bytes to be transferred
r - rotation speed (rev/s)
N - no. of bytes on track

Total avg access time,

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

[T_s - avg seek time]

8. Consider a disk pack with 16 surfaces, 128 tracks per surface, 256 sectors per track & 512 B per sector.

- Capacity of disk pack = $16 \times 128 \times 256 \times 512$ B
  = 256 MB.
- No. of bits required to address sector. = $\log_2(16 \times 128 \times 256) = 19$.
- Format overhead is 32 B/sector.

  Total formatting overhead = 2^{19} sectors $\times$ 32B
  = 16 MB.

  Formatted disk space = (256 − 16) MB.
  = 240 MB.
- Format overhead is 64 B/sector.

  Memory lost due to formatting = $2^{19} \times 64$B
  = 32 MB.
- Diameter of innermost track 21 cm

  Storage capacity of a track = 256 × 512 B
  = 128 KB

  Circumference of innermost track = $2\pi r$
  = 65.94 cm

  Maximum recording density = (128 KB)/65.94 cm
  = 1.94 KB/cm
- Diameter of innermost track 21 cm with 2KB/cm recording density.

  Capacity of one track = 2KB/cm × $(\pi \times 21)$ cm
  = 132 KB.

- Disk rotating @ 3600 RPM.

No. of rotations/sec = $\frac{3600}{60}$ = 60 RPsec

Data transfer rate = 16 (heads) × (256 × 512) × 60 B/sec

= 120 MBps.

- Rotational speed 3000 RPM. | Avg. seek time = 11.5ms

Time taken for one full rotation = $\frac{60}{3000}$ s

= 20 ms

Average rotational delay = $\frac{1}{2}$ × 20 ms = 10 ms

Average access time = (11.5 + 10) = 21.5 ms.

Q What is the average access time for transferring 512 B. of data with following specifications -

avg. seek time = 5 ms

rotation speed = 6000 RPM

data rate = 40 KB/s

controller overhead = 0.1 ms.

→ Time taken for one rotation = $\frac{60}{6000}$ s = 10 ms

Average rotational delay = $\frac{1}{2}$ × 10 = 5 ms

Transfer time = $\frac{512\ B}{40\ KB/s}$ = 12.5 ms

Average access time = (5 + 5 + 12.5 + 0.1) ms

= 22.6 ms.

Q A certain moving arm disk storage with one head has the specs –

No. of tracks/surface = 200

Disk rotation speed = 2400 RPM.

Track storage capacity = 62500 bits

Average latency$_{rot}$ = P ms

Data transfer rate = Q bit/sec

Calculate P & Q.

→ Time taken for one rotation = $\frac{60}{2400}$ s = 25 ms

Average rotational latency = $\frac{1}{2} \times 25$ = 12.5 ms

Data transfer rate = $1 \times 62500 \times \frac{2400}{60}$ bit/sec

= 2.5×10^6 bits/sec.

Q A disk pack has 19 surfaces & storage area on each surface has an outer diameter of 33 cm & inner diameter of 22 cm. Max record density on any track is 200 bits/cm & min spacing between tracks is 0.25 mm. Calculate capacity of disk pack.

→ No. of tracks on each surface = $\frac{r_2 - r_1}{d}$ = $\frac{(16.5 - 11)\text{ cm}}{0.25\text{ mm}}$ = 220 tracks

Capacity of each track = (200 bits/cm) × ($\pi \times 22$ cm)

= 1727 B

Capacity of disk pack = 19 × 220 × 1727 B

= 6.88 MB

6. Consider a typical disk that rotates at 15000 RPM & has a transfer rate of 50×10^6 B/sec. If the average seek time of the disk is twice the avg rotational delay & controller's transfer time is 10 times the disk transfer time. What's avg time to read or write a 512 B sector of the disk?

→ Avg. rotational latency = $\frac{1}{2} \times \left(\frac{60}{15000}\right) s = 2$ ms

Avg. seek time = 2×2 ms = 4 ms

Disk transfer time = $\frac{512 \text{ B}}{50 \times 10^6 \text{ B/s}} = 0.01024$ ms

Controller's transfer time = 10×0.01024 ms

$= 0.1024$ ms

Ans = $(4 \text{ ms} + 2 \text{ ms} + 0.01024 + 0.1024)$ ms

$= 6.11$ ms.

8. A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces & 1000 cylinders. Address of a sector is given as a triple (c,h,s) where c is cylinder number, h is the surface number & s is sector number.

- <400, 16, 29> corresponds to which sector no?

→ We have to cross 400 cylinders.

Total no. of sectors that are crossed in 400 cylinders =

no. of cylinders × no. of surfaces/cylinder × no. of tracks/surface × no. of sectors/track.

$= 400 \times (10 \times 2) \times 1 \times 63 = 504000$

Now, after crossing 400 cylinders we are at cylinder-400.

To reach desired surface, we've to cross 16 surfaces.

Total no. of sectors that're crossed = 16 (surfaces) × 1 (tracks/surface) × 63 (sectors/track)

= 1008

Crossing 29 sectors, we ~~are~~ have crossed total (504000 + 1008 + 29) = 505037 sectors.

Required address of sector = 505037. (Ans)

- Address of 1039 sector is —

a) <0, 15, 31>

b) <0, 16, 31>

c) <0, 17, 31>

→

a) 0 + (15 × 1 × 63) + 31 = 976 ×

b) 0 + (16 × 1 × 63) + 31 = 1039. ✓

c) 0 + (17 × 1 × 63) + 31 = 1102 ×

ALU, DATA-PATH & CONTROL UNIT

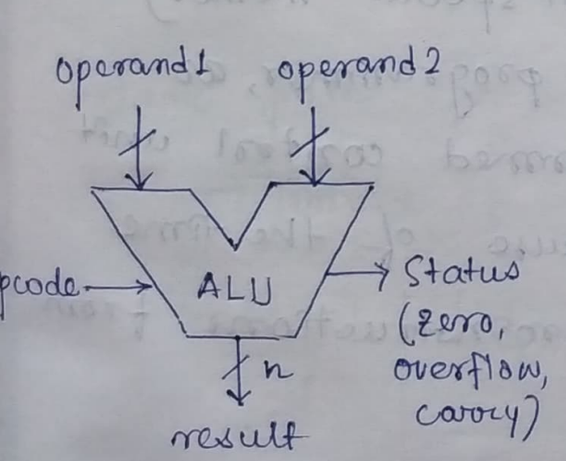
* ALU: Arithmetic and Logic unit is the subunit within a computer's CPU. ALU contains the logical circuit to perform mathematical operations like subtraction, addition, multiplication, division, logical operations & logical shifts on the values held in the processor registers or its accumulator.

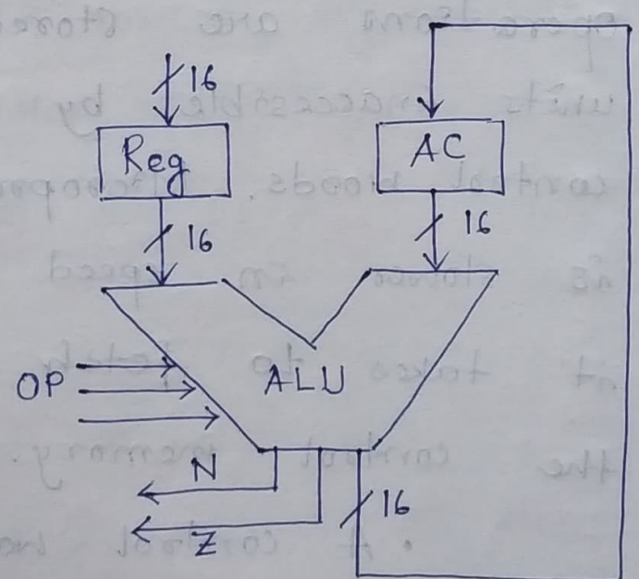
Control unit generates control signals to ALU to perform specific operations.

The operands & results of the ALU are machine words of two kinds: arithmetic words, that represent numerical values in digital form & logic words, which represent arbitary sets of digitally encoded symbols. Arithmetic words consist of digit vectors. (strings or digits).

ALU uses many types of flags.

Flag register (Status register/Program SR) holds the boolean value of status word used by the process.





* Control Unit: Responsible for decoding the opcode & operand bits in the instruction register, and then generating the control signals necessary to drive all other hardware in the CPU to perform the sequence of micro-operations that comprise the instruction.

→ Design of Control Unit ~

When the control signals are generated by hardwire using conventional logic design techniques, the CU is said to be hardwired.

When control signals are generated by a program similar to machine language programs, it is said to be micro-programmed.

→ Microprogrammed Control Unit:

The control signals associated with operations are stored in special memory units inaccessible by the programmer, as control words. Microprogrammed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

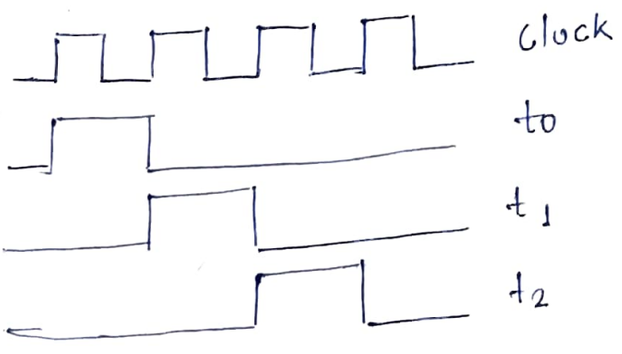
• A control word is a word whose individual bits represent various control signals.

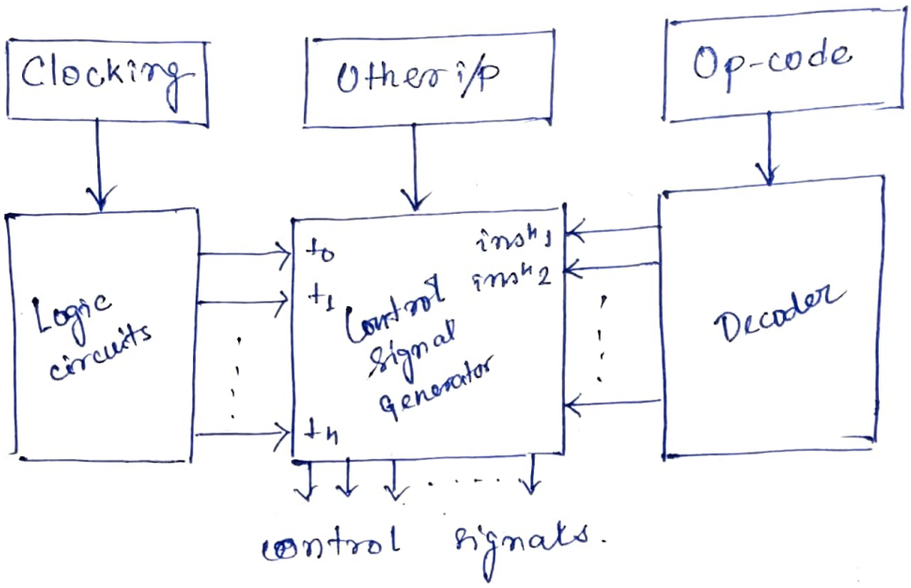
* Register Sets.

1. General Purpose Registers (GPR)
   - accessible to the programmers.
2. Memory access registers.
   - MAR, MDR.
3. Instruction Fetching Registers.
   - PC, IR.
4. Condition Registers.
   - Program Status Word
   - AC & flag registers.
5. Special purpose address registers.
   - Index Registers (Destination & Source Index DI, SI)
   - Segment Register (Code segment, Data segment, Stack segment, Extra segment)
   - DS : DI, DS : SI } EA
   - Stack Pointer (SP)

* Control Unit : Directs the main operations by sending control signals to the data path (data section of CPU).

→ Timing of Control signals.



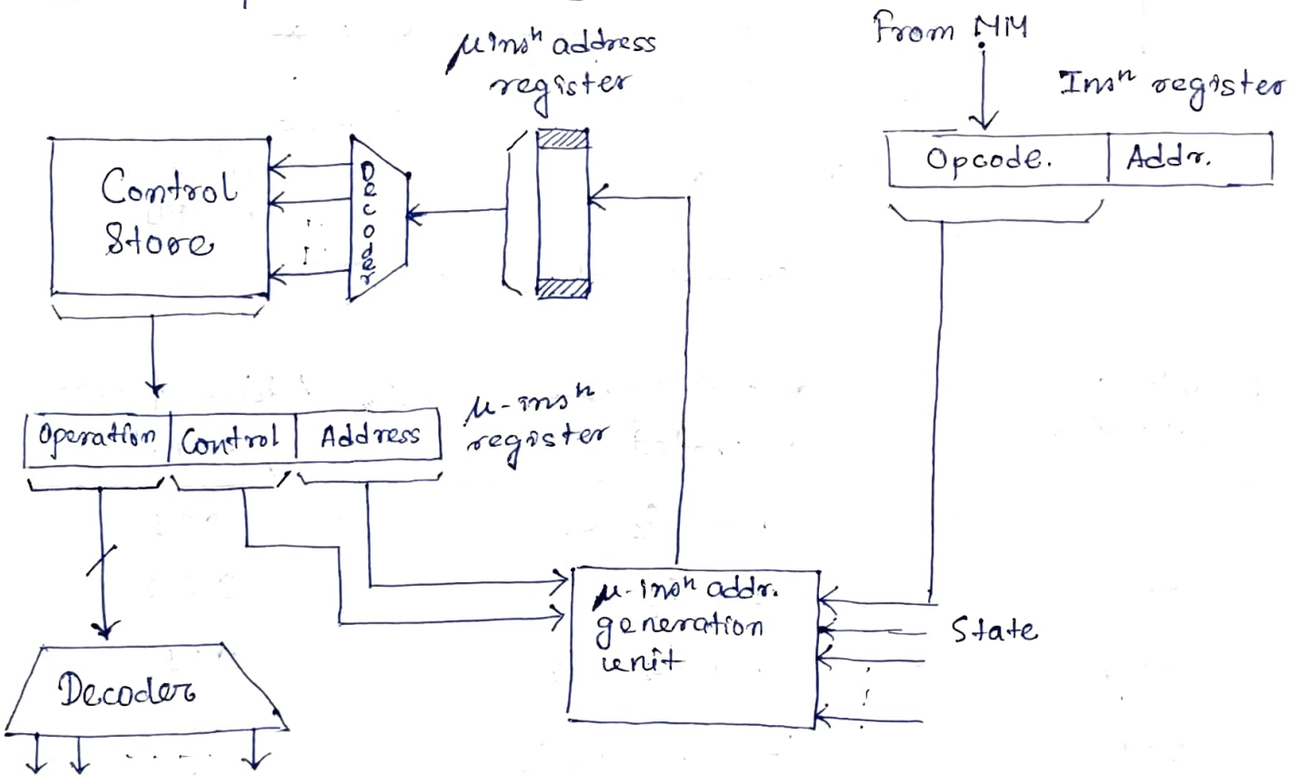

Clocking
Other i/p
Op-code
Logic circuits
Control signal generator
Decoder
t_0
t_1
t_n
$inst_1$
$inst_2$
control signals.

• A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro routine for that instruction.

• Individual control word in the micro-routine is referred to as microinstruction.

• A sequence of micro-instructions is called a microprogram, which is stored in a ROM or RAM as control memory (CM).

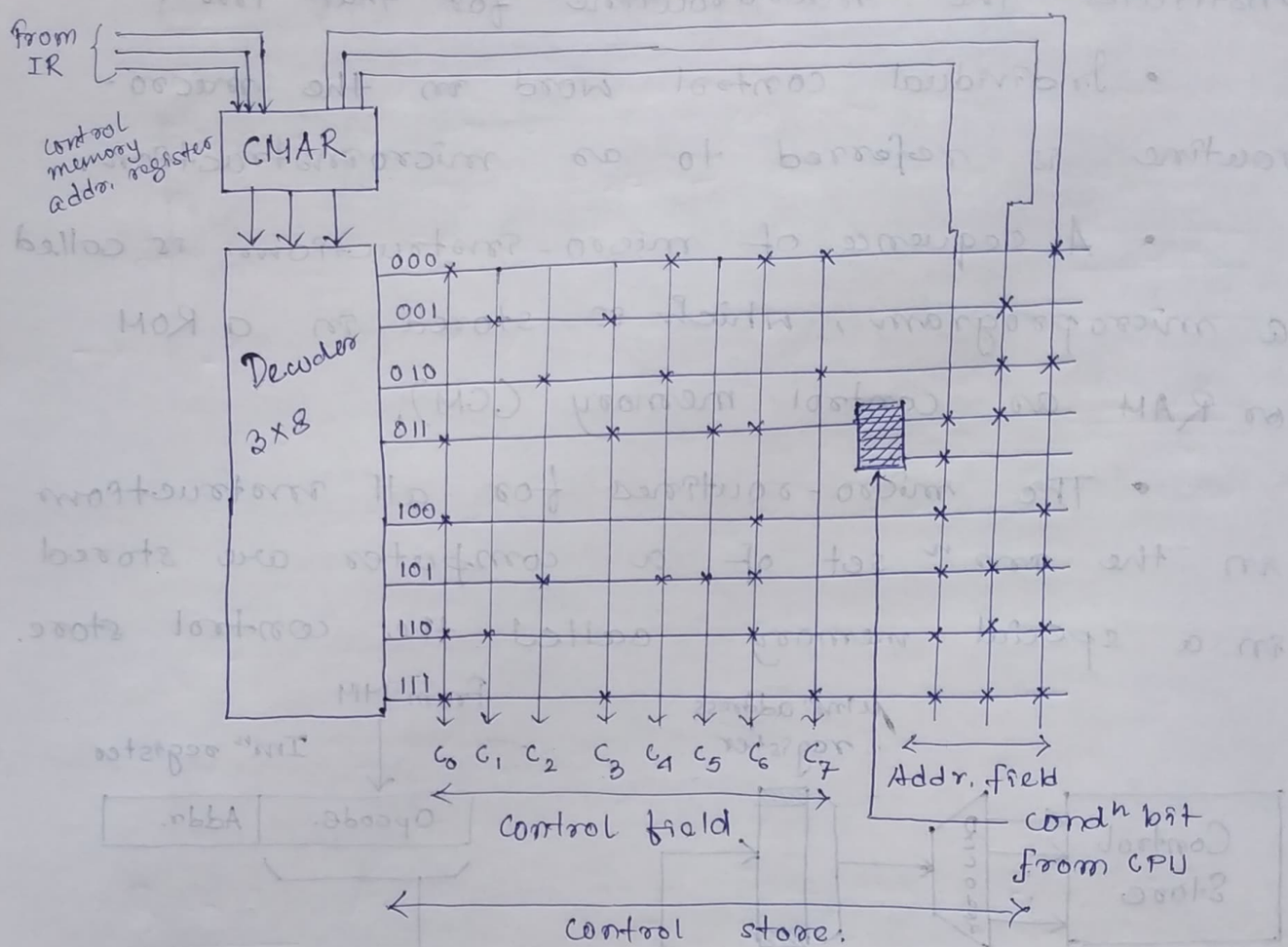
• The micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the control store.



• Format of Control Word.

Branch condition — Flag — Control field — Control memory address.

• Wilkes Control Circuit Design.



Decode line activated	Control signals Generated	Addr. of next μ-instn.
000	C_0, C_4, C_6, C_7	001
001	C_1, C_3	010
010	C_2, C_4, C_7	011
011	C_0, C_3, C_5, C_6	?
Either 011	C_6, C_3, C_5, C_6	110 (cond^n bit true)
110	C_6, C_1, C_6	111
111	C_6, C_3, C_7	load next instn to
or 011	C_0, C_3, C_5, C_6	100 (cond^n bit false)
100	C_0, C_6	101
101	C_2, C_4, C_5, C_6	111
111	C_0, C_3, C_7	Load next instn.

- Types of μ-programmed control unit :

Based on the type of control word stored in the control memory (CM) -

1. Horizontal μ-programmed control unit :

Each μ-instn is a series of bits each of which represents a single control line.

Control signal is expressed with decoded binary format per control signal. There is no need of external decoder to generate CS. It's bit flexible as compared to hardwired control unit since it allows modification easily as compared to hardwired control unit.

This concept is not in practice as managing the signals in the decoded format is not easy as it leads to longer instns.

- long formats
- ability to express a high degree of parallelism
- minimally encoded scheme
- many resources can be controlled
- operating speed is low.
- faster than vertical organisation.

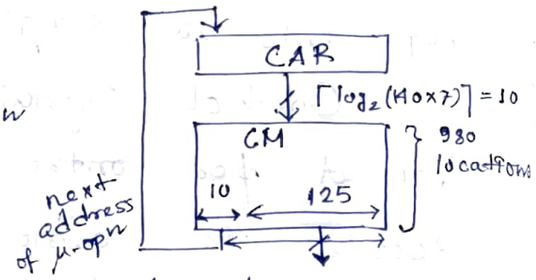
2. Vertical μ-programmed control unit :

Control signals are represented in the encoded binary format. For N control signals $\log_2 N$ bits are required.

Control signals expressed in encoded format. It requires external decoder. It provides ease for implementation of new instructions because of encoded binary format.

- Short formats.
- Limited ability to express microoperation parallelism.
- Operating speed is high.

Q. G'08 Consider a CPU where all instns require 7 clock cycles to complete execution. There are 140 instns in the instn set. It's found that 125 control signals are needed to be generated by the control unit. While designing the horizontal μ-p control unit, single address field format is used for branch control logic. What is the min. size of the control word & control address register?



→ 140 instns × 7 clock cycles/instn

= 980 cycles.

We need 10 bits $(2^{10} = 1024)$ to represent them.

(125 + 10) = 135 – control word

Answer – (135, 10)

Q. G'04 A CPU has only three instns I1, I2 & I3, which use the following signals in time steps T1 – T5 :

I1 : T1 : Ain, Bout, Cin	I2 : T1 : Cin, Bout, Din	I3 : T1 : Din, Aout
T2 : Cout, Bin	T2 : Aout, Bin	T2 : Ain, Bout
T3 : Zout, Ain	T3 : Zout, Ain	T3 : Zout, Ain
T4 : Bin, Cout	T4 : Bin, Cout	T4 : Dout, Ain
T5 : End	T5 : End	T5 : End

Logic fn for signal Ain (hardwired control) –

$$A_{in} = I_1(T1 + T3) + I2\,T3 + I3(T2 + T3 + T4).$$

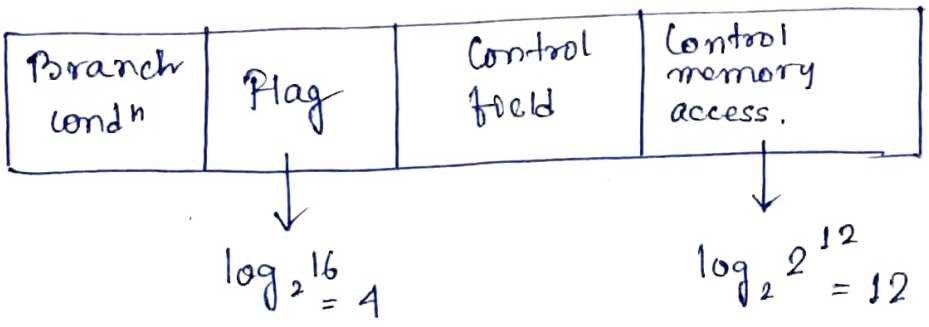
Q. G'02 Horizontal μ-programming

a) doesn't require use of signal decoders
b) results in larger sized μ-instns
c) uses one bit for each cs
✓ d) all

... processor have.

* Horizontal μ-pro. CU	Vertical μ-pro. CU.
1. Supports longer control word.	1. Shorter.
2. Allows higher degree of parallelism.	2. Lower degree of parallelism.
3. No additional hardwire.	3. Decoder reqd.
4. Faster.	4. Slower.
5. Less flexible.	5. More flexible.
6. Every bit in control field attaches to a control line.	6. Code is used for each action to be performed.
7. Less use of ROM encoding.	7. More use of ROM encoding.

eg. CU supports 4K words. The hardwire contains 64 control signals & 16 flags.



~ horizontal –

64 bits / control signal

control word size = 4 + 64 + 12 = 80 bits

Control memory = $\frac{4 \times 80}{8}$ KB = 40 KB

~ Vertical –

$\log_2 64$ = 6 bits for 64 control signals

control word size = 4 + 6 + 12 = 22 bits

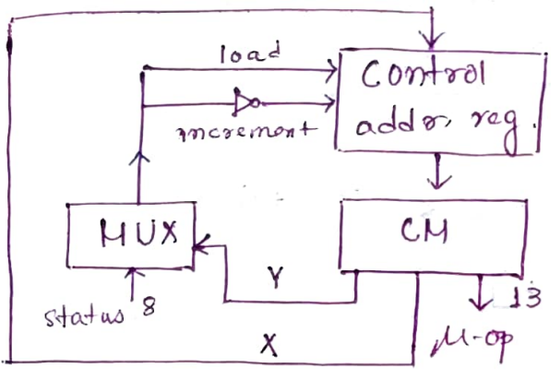
Control memory = $\frac{4 \times 22}{8}$ KB = 11 KB.

* Hardwired CU	μ-programmed CU.
1. Generates control signals using logic circuits.	1. Uses μ-instructions.
2. Faster.	2. Slower.
3. Difficult to modify.	3. Easy to modify.
4. More costlier.	4. Less costlier.
5. Can't handle complex instrs as circuitry becomes complex.	5. Can handle.
6. Limited no. of instrs are used.	6. For many instrs cs can be generated.
7. Used in RISC.	7. Used CISC.

d) all

Q, 6'04 The μ-instrs stored in the CM of a processor have a width of 26 bits. Each μ-instr is divided into 3 fields - a μ-operⁿ field (13 bits), a next address field (X), & a MUX select field (Y). There are 8 status bits in the inputs of the MUX.

Calculate X, Y & size of CM.



→ MUX has 3 select lines. ($2^3 = 8$).

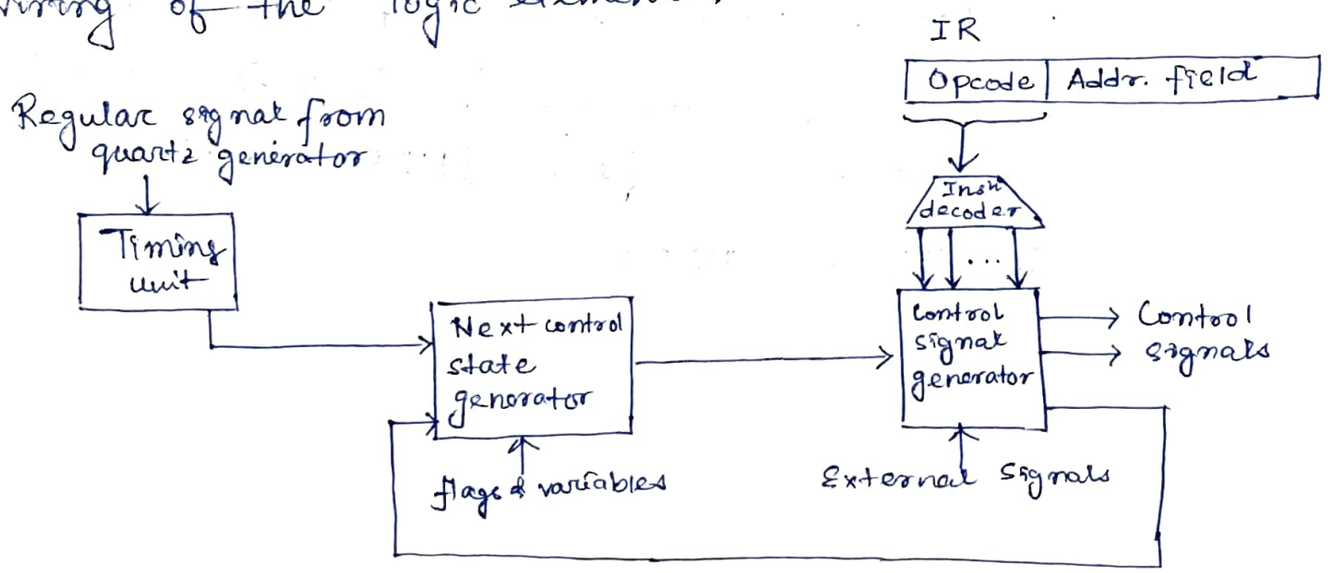
Y = 3

No. of bits in next addr. = 26 - 13 - 3
= 10.

X = 10.

CM size = 2^{10} words.

→ Hardwired Control Unit: Fixed logic circuits that correspond directly to the boolean expressions used to generate control signals. Hardwired control is faster than μ-programmed control. A controller that uses this approach can operate at high speed. The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of instr register, the condition codes & the external inputs. The outputs of the state machine are the control signals. The sequence of operations carried out by this machine is determined by the wiring of the logic elements.



→ Micro-operations for different ins^{n}s.

– Fetch ins^{n}

Single bus data-path.

t_0 MAR ← (PC) ; A ← (PC)

t_1 MDR ← M[MAR] ; PC ← (A)+4

t_2 IR ← (MDR)

Three bus data-path

t_0 MAR ← (PC) ; PC ← (PC)+4

t_1 MDR ← M[MAR]

t_2 IR ← (MDR)

– Execution ins^{n} ADD R1, R2, R0 (R0: des^{n})

Single bus datapath

t_0 A ← (R1)

t_1 B ← (R2)

t_2 R_0 ← (A) + (B)

Two bus datapath

t_0 A ← (R1) + (R2)

t_1 R0 ← (A)

Three bus datapath

t_0 R0 ← (R1) + (R2)

– Fetching a word from memory.

1. MAR ← [R1]
2. Request memory read & put data to addr. reg
3. Wait for memory fetch cycle signal & put result from [MDR] to R2.
4. R2 ← [MDR]

- Storing a word in memory

1. MAR ← [R1]
2. MDR ← [R2]
3. Request memory write
4. Wait for MFC signal.

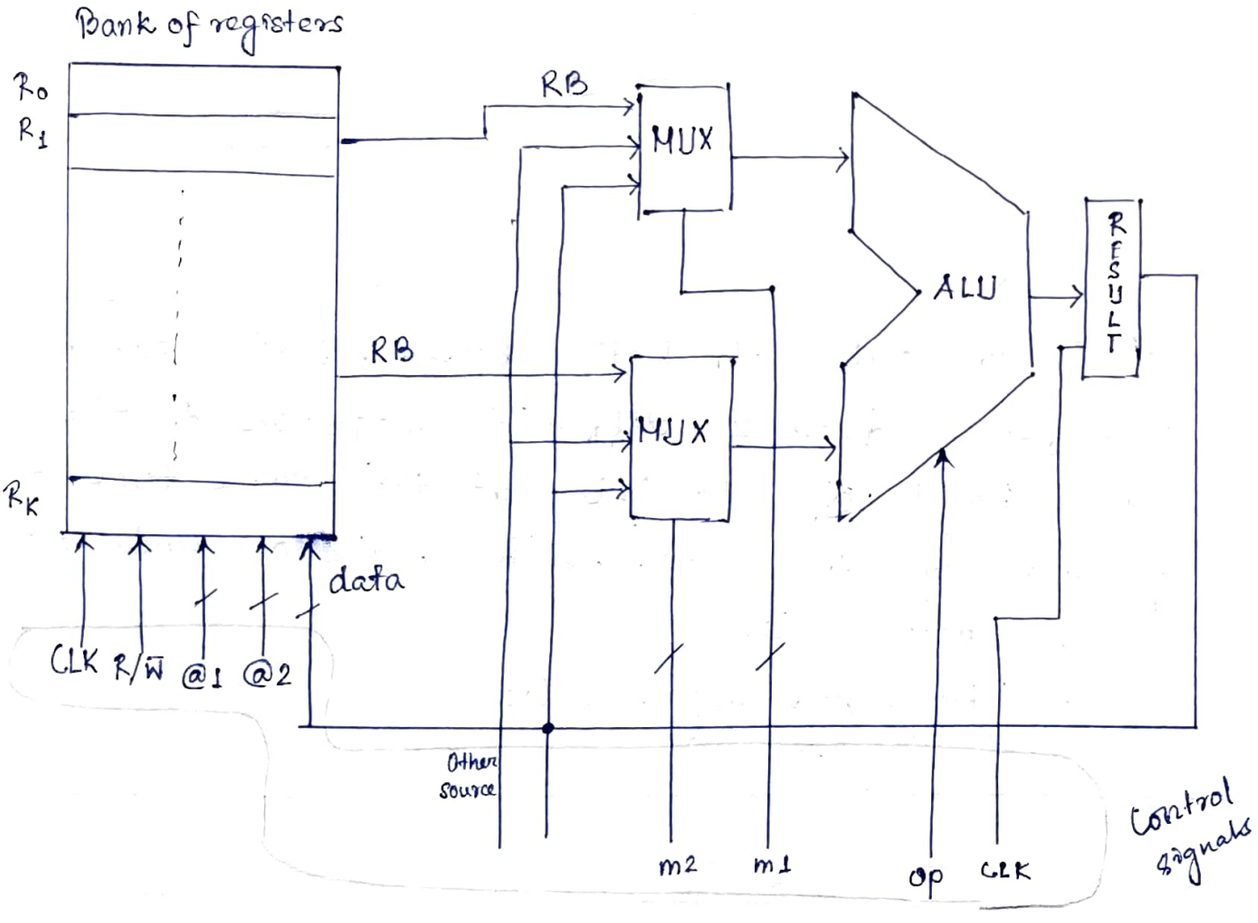
- Interrupt handling

t_1 MDR ← (PC)

t_2 MAR ← addr. 1 (where to save old PC);
PC ← addr. 2 (interrupt handling routine)

t_3 M[MAR] ← (MDR)

* Datapath : The registers, ALU & the interconnecting bus are collectively referred to as datapath.



eg. R3 = R1 + R2

① @1 = R1 , @2 = R2 , Read
m1 = m2 = RB , op = Add

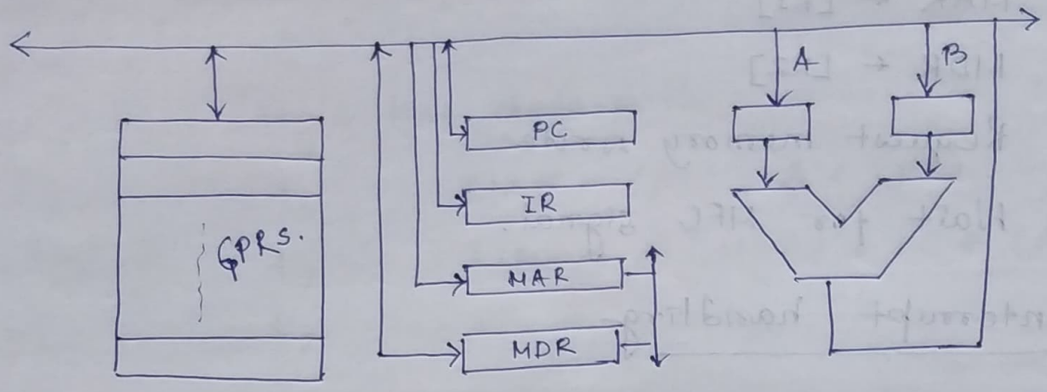
② (Result = R1 + R2)
R/W̄ = Write , @1 = R3.

R3 = R1 + R2 + R1

① → ② (Result) = R1 + R2
@1 = R1 , R/W̄ = Read
m1 = RB , m2 = Result
op = Add

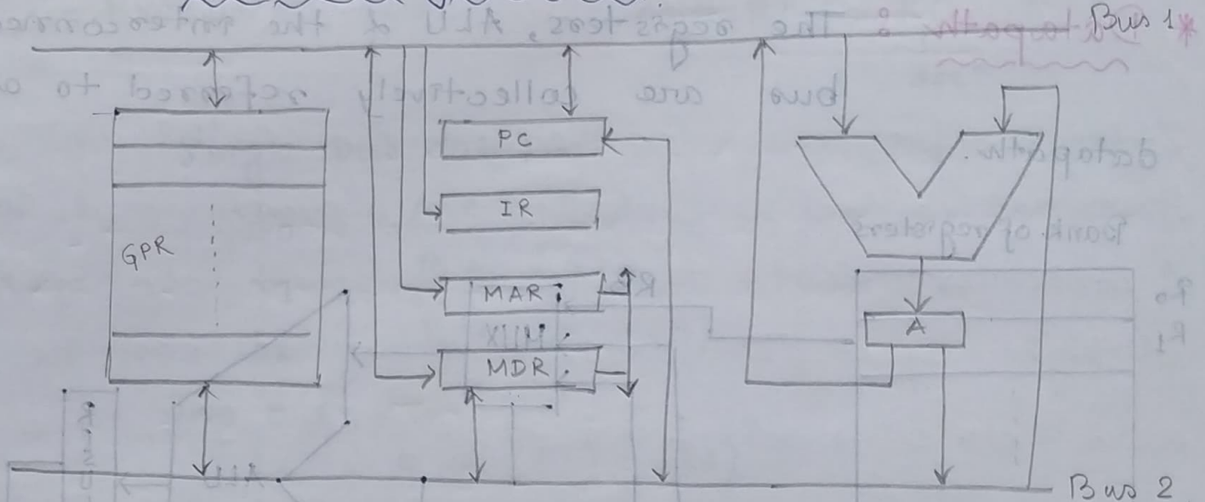
③ (Result = R1 + R2 + R1)
R/W̄ = Write , @1 = R3.

→ One-bus organisation.



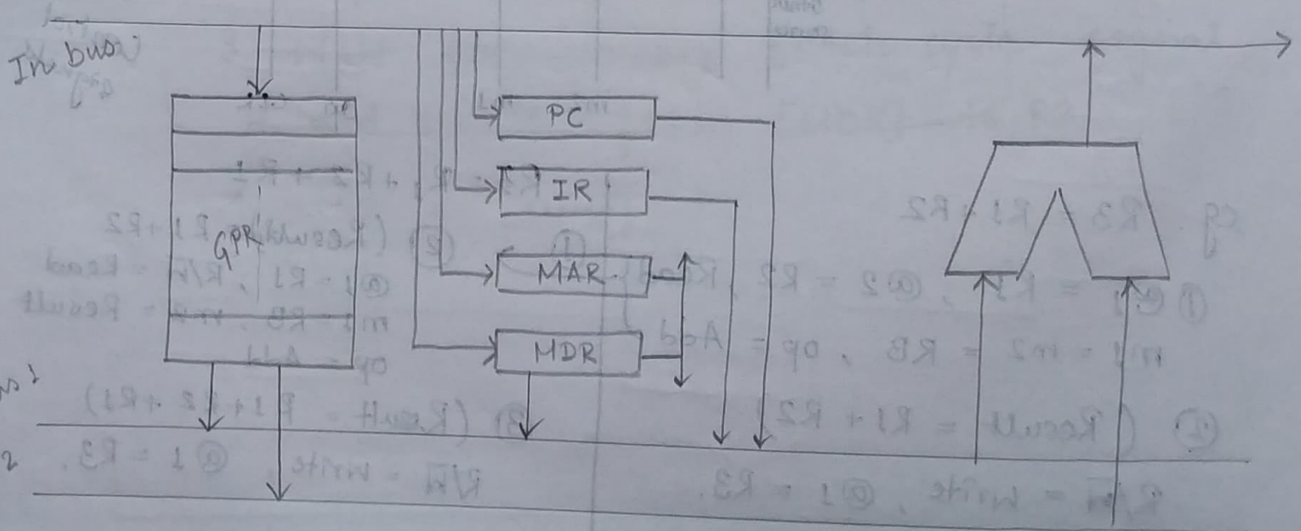
CPU registers & ALU use a single bus. Bus can handle only a single data movement within one clock cycle.

→ Two-bus organisation.



GPRS. connected to both buses. Data can be transferred from 2 different registers to the input point of the ALU at the same time. 2 operand operation can fetch both operands in the same clock cycle.

→ Three-bus organisation.



Source buses move data out of registers (out-bus) & the destination bus may move data into a register (in-bus). Each of the 2 out-buses is connected to an ALU input point. The o/p of the ALU is connected directly to in-bus.

* RISC (Reduced instn set computer).

Instn set composed of a few basic steps for loading, evaluating & storing operations.

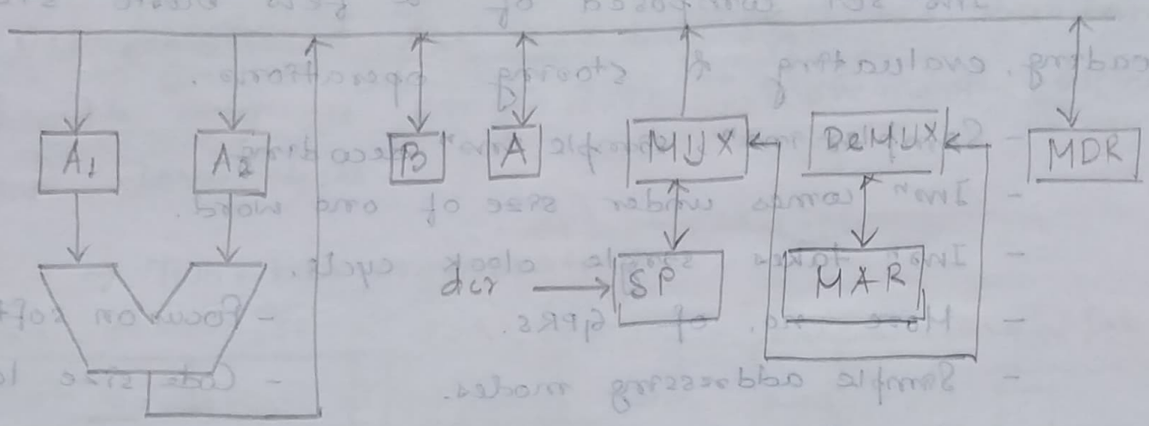
- Simpler instn, simple instn decoding.
- Instn comes under size of one word.
- Instn takes single clock cycle.
- More no. of GPRs.
- Simple addressing modes.
- Less data types.
- Pipelining can be achieved.
- Reduces the cycles per instn at the cost of the no. of instns per program.
- Focus on software
- Code size large

* CISC (Complex instn set computer)

Single instn for tasks like ADD, MUL.

- Complex instn, complex instn decoding.
- Instns are larger than one word.
- Instn may take more than one clock cycles.
- Less no. of GPRs as opn gets performed in memory itself.
- Complex addr. modes
- More data types.
- Focus on hardware.
- Code size small
- Minimises the no. of instns per program but at the cost of increase in no. of cycles / instn.

Q. G'01 Consider following datapath of a simple non-pipelined CPU. The registers A, B, A_1, A_2, MDR, the bus & the ALU are 8-bit wide. SP & MAR are 16 bit registers. The MUX is of size $8 \times (2:1)$ & the demux is of size $8 \times (1:2)$. Each memory op^n takes 2 CPU clock cycles & uses MAR & MDR. SP can be decremented locally.



The CPU has an instruction "push r", where r = A or B, has the specification

assembly language:
M[SP] ← r
SP ← SP − 1

How many clock cycles are needed to execute the push r instruction?

→ t_1, t_2 : MAR ← SP.

t_3 : MDR ← r ; SP ← SP − 1

t_4, t_5 : M[MAR] ← MDR.

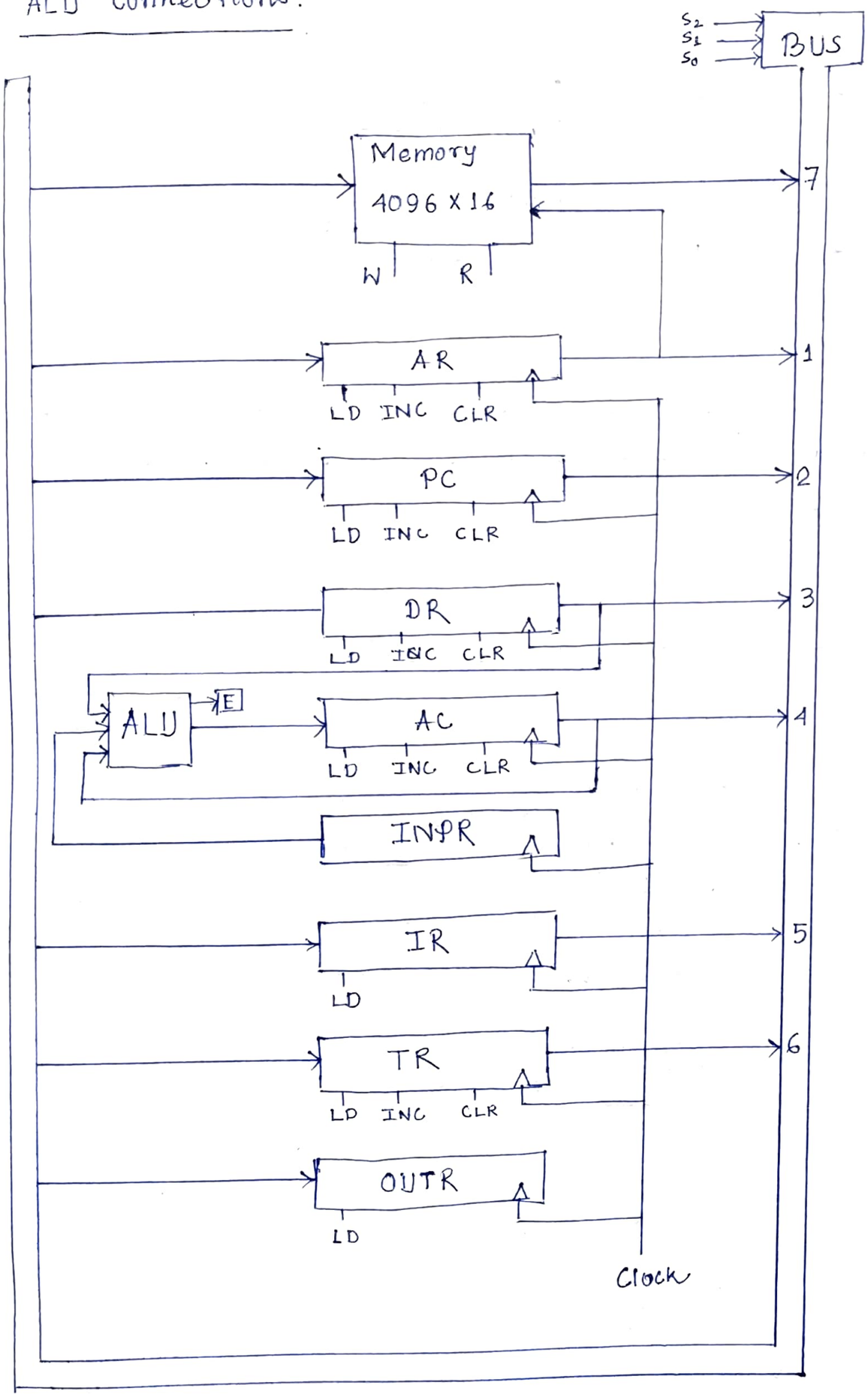
We need to move 16 bit SP to 16 bit MAR via a 8-bit bus. So, 2 cycles.

MDR & r both 8 bit. So, 1 cycle. (SP local decrement no more cycle)

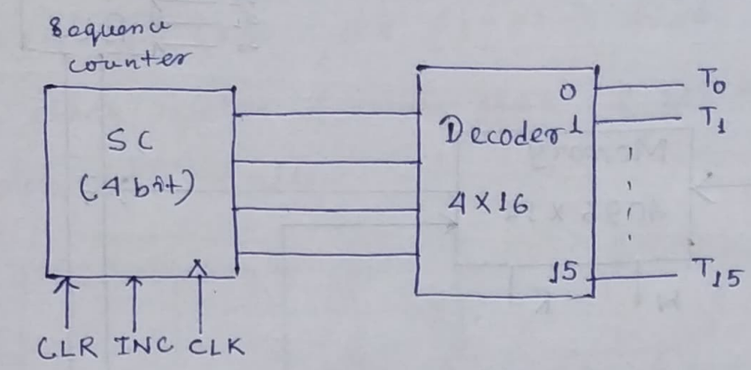
MAR is 16 bit, while MDR is 8 bit. So 2 cycles.

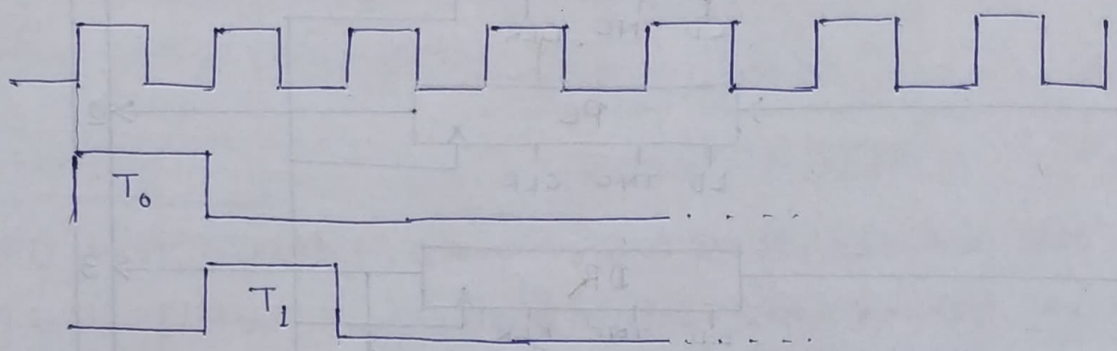
Answer → 5 clock cycles.

ALU connections.

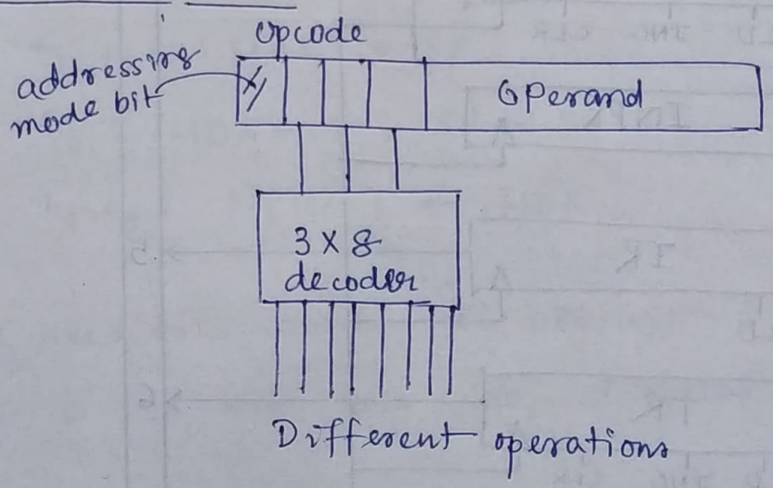


Timing Circuitry.





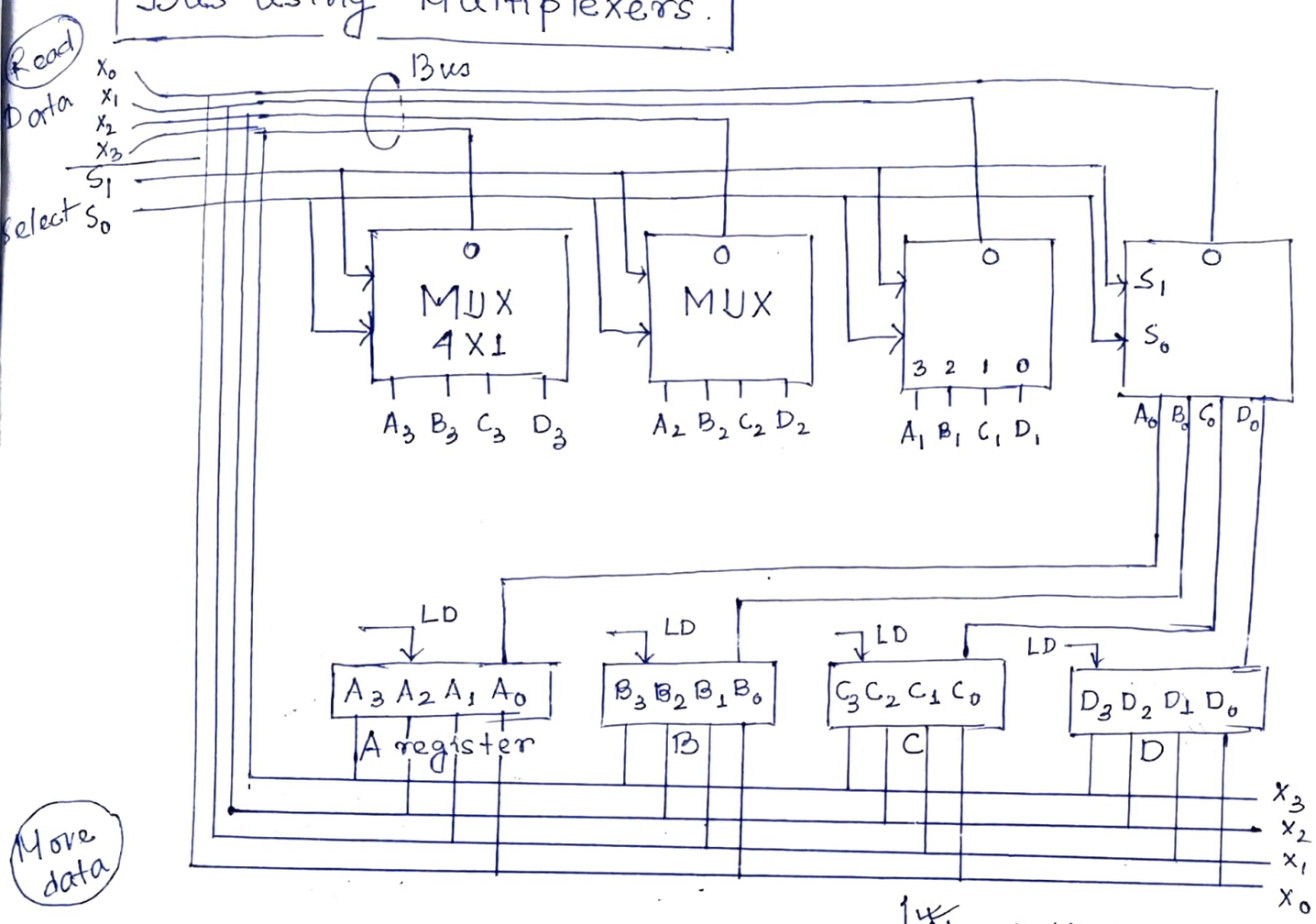
Insⁿ Decoder.



* Different Insⁿ types -

$D_7 I T_3$	I/O
$D_7 \bar{I} T_3$	Register
$\bar{D}_7 I T_3$	Memory (EA)
$\bar{D}_7 \bar{I} T_3$	Stall

Bus using Multiplexers.



→ Read from C.

1) Select lines –

S_1	S_0
0	1

2) Read from $X_3 X_2 X_1 X_0$.

→ Move A ← D

1) Select lines

S_1	S_0
1	1

2) Data of D at $X_3 X_2 X_1 X_0$

3) LD (load) control signal 1 at register A.

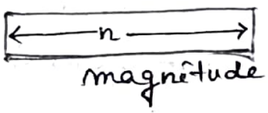
4) Data loaded to A. from bus.

No. of MUX = no. of bits of one register

Size of MUX = no. of registers x 1

* Number Representation.

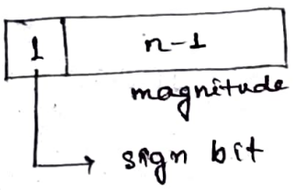
→ Unsigned



0 to $2^n - 1$

$> 2^n - 1$ +ve overflow.

→ Signed.



If $n = 4$ then

$0111 \Rightarrow +7$

$1111 \Rightarrow -7$

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

$< -(2^{n-1} - 1)$ -ve overflow

$> (2^{n-1} - 1)$ +ve overflow

0 - 2 representations

0000 +ve 0

1000 -ve 0

(ambiguous)

Using 2's complement.

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

2's complement$_2$ = 1's complement$_2$ + 1_2

If $n = 4$ then

$0111 \Rightarrow +7$

$1111 \Rightarrow -1$

0 - only 1 representation

0000.

$< -2^{n-1}$ -ve overflow

$> 2^{n-1} - 1$ +ve overflow

→ 2's complement on n bit architecture

2^n - number.

eg. $0101_2 \xrightarrow{1's} 1010 \xrightarrow{2's} 1011$.

$2^4 = 16 = 10000_2$

$$\begin{array}{r} 10000 \\ -\ 0101 \\ \hline 1011 \checkmark \end{array}$$

$$\left\{\begin{array}{l} (1111) - (0101) + 1 \\ (1111 + 1) - (0101) \\ 10000 - 0101 \end{array}\right\}$$

→ $\overset{7\ 6\ 5\ 4\ \ 3\ 2\ 1\ 0}{0111\ \ 0110}$

$= (2^7 - 2^4) + (2^3 - 2^1) = 118_{10}$

→ $0.\underbrace{111\cdots1}_{n} = 1 - 2^{-n}$

- Floating Point Number Representation.

$\pm$ Significand / mantissa $\times$ Base $^{\pm \text{Exponent}}$

Normalisation rules of floating point number.

1. The integer part should be zero.
2. $0 \cdot d_1 d_2 \cdots d_n \times B^{\pm E}$ then $d_1 > 0$ & except all $d_i \geq 0$.

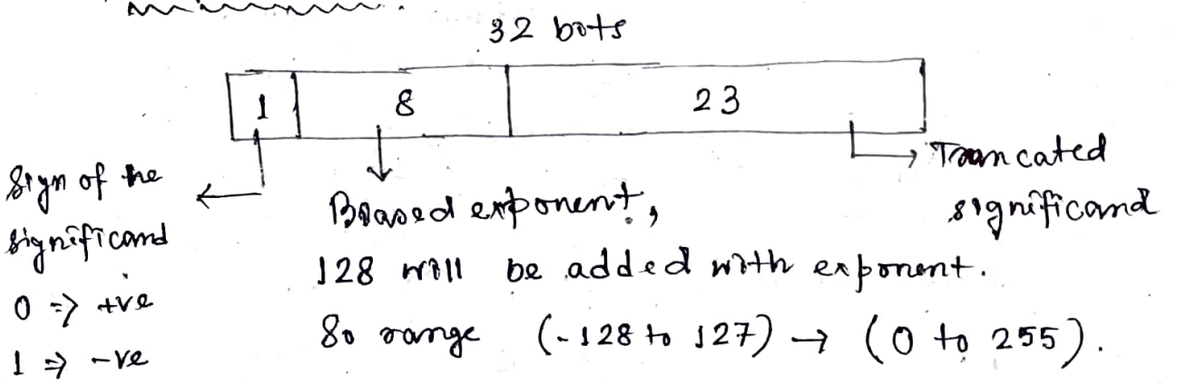
$0.123 \times 10^4 = 0.0123 \times 10^5 = 1.23 \times 10^3$

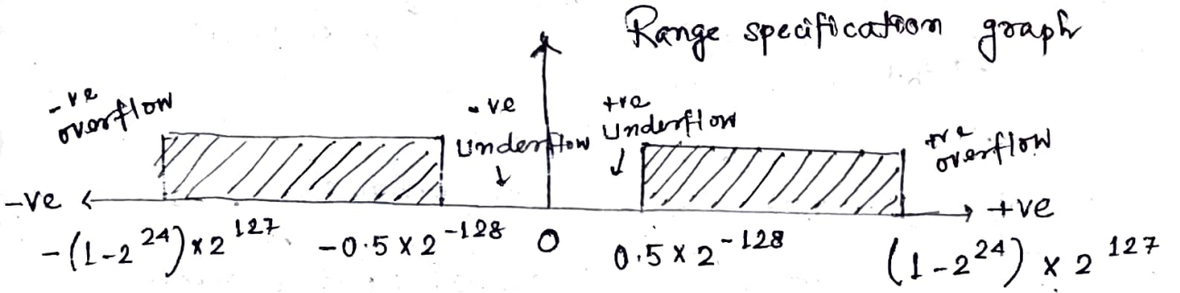
✓ ✗ ✗

2 representation techniques.

1. Single Precision (32 bit)
2. Double Precision (64 bit)

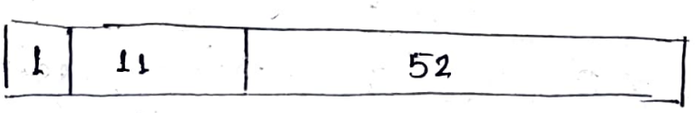
Single Precision.





Double Precision.

64 bits.



eg.

$0.1\underline{10111} \times 2^{100101}$

single precision –

~~0 10100101~~

0 | 10100101 | 10111000...0

(23)

$-0.1\underline{10111} \times 2^{-10010}$

1 | 01011011 | 10111000...0

(23)

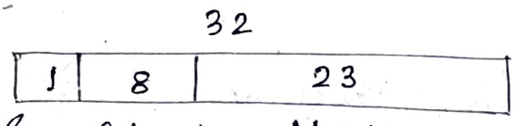
| 00100101
| 10000000
|---
| 10100101

| 00100101
| ↓ 2's
| 11011

| 11011011
| 10000000
|---
| 1 | 01011011

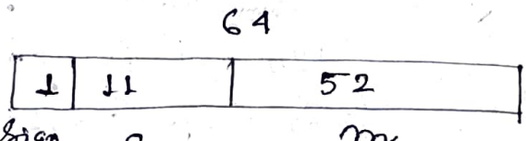
- Floating Point Number Representation
  IEEE 754 Floating Point Rep^{n}.

Single Precision.



$$X_{FP32} = (-1)^{S} \times 2^{e-127} \times 1.m$$

Double Precision



$$X_{FP64} = (-1)^{S} \times 2^{e-1023} \times 1.m$$

e	m	Inference.
255	0	NaN (not a number)
255	$\neq 0$	Infinite number
0	0	$X = (-1)^{S} \times 2^{-126} \times (0.m)$
0	$\neq 0$	0, $(-1)^{S} \times 0$, again +0 & -0 are possible

eg. Decimal equivalent of 40400000_{He}

$$40400000_{H} = \underbrace{0}_{s}\ \underbrace{1000000}_{e}\ \underbrace{0100}_{m}\ \underbrace{0 \ldots 0}_{20\ 0's}$$

(1 | 8 | 23)

$S = 0$

$e = 128$

$m = 0.5$

$$X_{FP32} = (-1)^{0} \times 2^{128-127} \times 1.5$$

$$\therefore 2 \times 1.5 = 3.$$

eg. Express 10_{10} on IEEE 754 format (single precision).

→ $10 = (-1)^0 \times 2^{130-127} \times 1.25$.

$10 = 8 \times 1.25$

$= 2^3 \times 1.25$

0 | 10000010 | 010 ···· 0

23

$4120\ 0000_{Hex}$ (Ans)

eg. $40A00000_H$ to decimal.

= 0100 | 0000 | 1010 | 0 ···· 0

s e m 20 0's

$X = (-1)^0 \times 2^{129-127} \times 1.25$

$= 4 \times 1.25 = 5_{10}$

- Floating Point arithmetic

Add/Subtract.

1. Compare the magnitudes of two exponent & make suitable alignment to the number with the smaller magnitude of exponent.
2. Perform the addition/subtraction.
3. Perform normalization by shifting the resulting mantissa & adjusting the resulting exponent.

eg. Add 1.1100×2^4 & 1.1000×2^2.

alignment : 1.1000×2^2

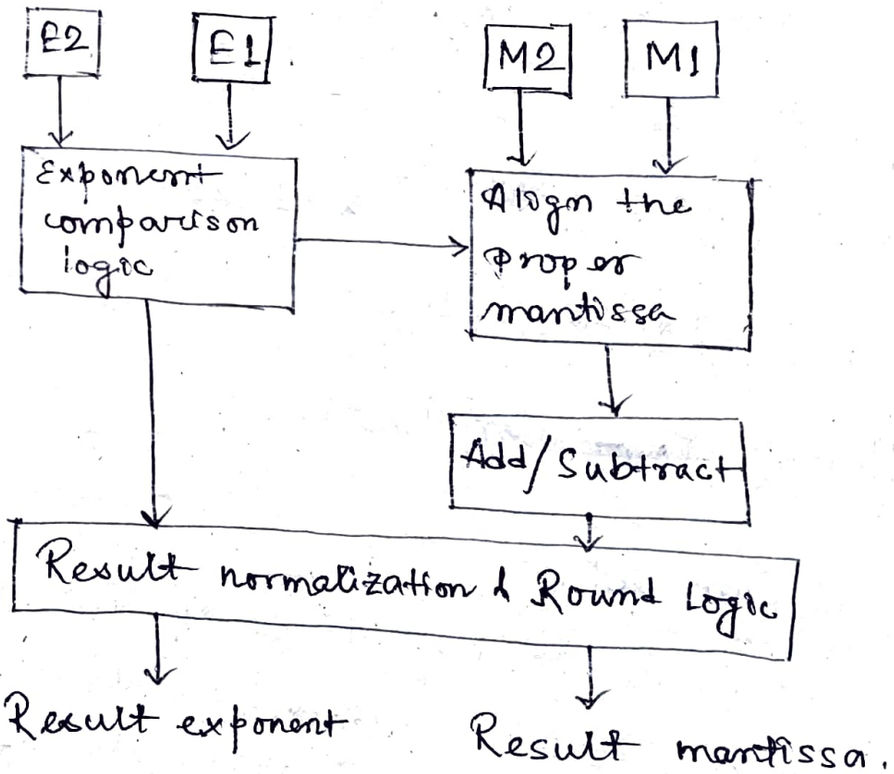
$\downarrow$

0.0110×2^4

addition : 10.0010×2^4

normalization : 0.1000×2^6.

Hardware circuitry –



- Multiplication.

FP numbers $X = m_x \times 2^a$ & $Y = m_y \times 2^b$ is represented as

$$X * Y = (m_x \times m_y) * 2^{a+b}$$

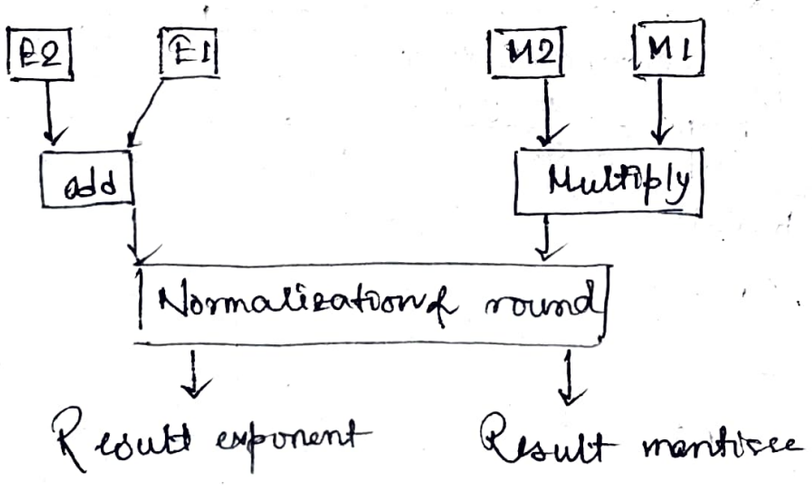
1. Compute the exponent of the product by adding the exponents together.
2. Multiply 2 mantissas.
3. Normalise & round the final product.

eg. $X = 1.000 \times 2^{-2}$

$Y = -1.010 \times 2^{-1}$

1. $(-1-2) = -3$
2. $1.000 \times (-1.010) = -1.010000$
3. -1.0100×2^{-3}.

Hardware circuitry



- **Division.** $X = m_x \times 2^a$ & $Y = m_y \times 2^b$

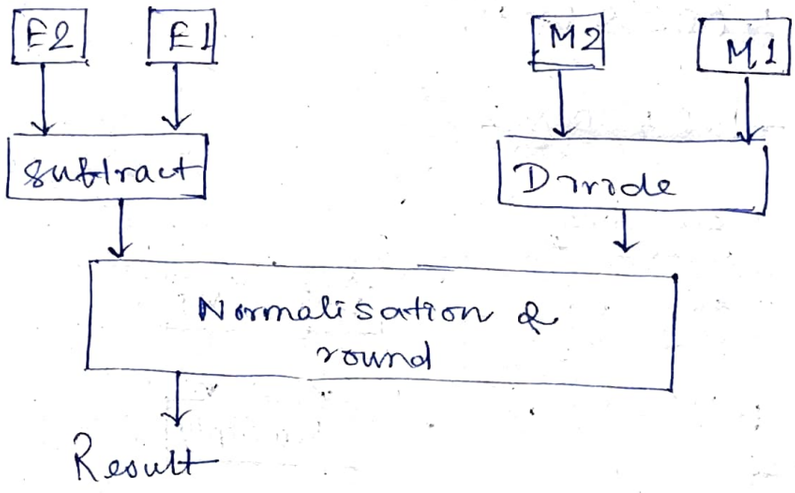
$$X/Y = (m_x/m_y) \times 2^{a-b}.$$

1. Compute exponent of the result by subtracting.
2. Divide the mantissa & decide the sign of the result.
3. Normalise & round.

eg. $X = 1.0000 \times 2^{-2}$

$Y = -1.0100 \times 2^{-1}$

1. $-2 - (-1) = -1$
2. $1.0000 \div (-1.0100) = -0.1101$
3. -0.1101×2^{-1} (Ans).

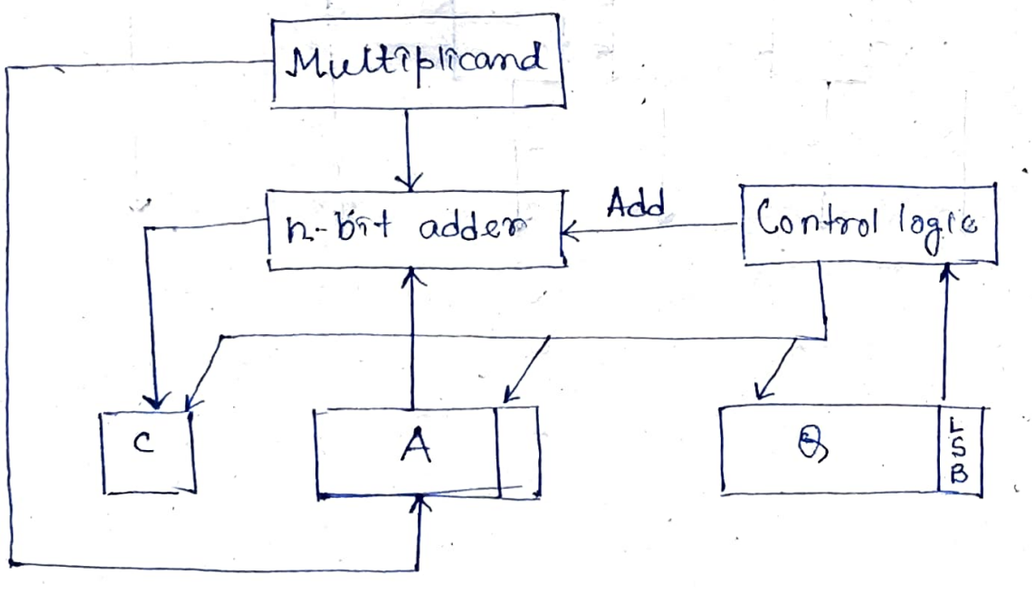


* Multiplication using Add-Shift Method:

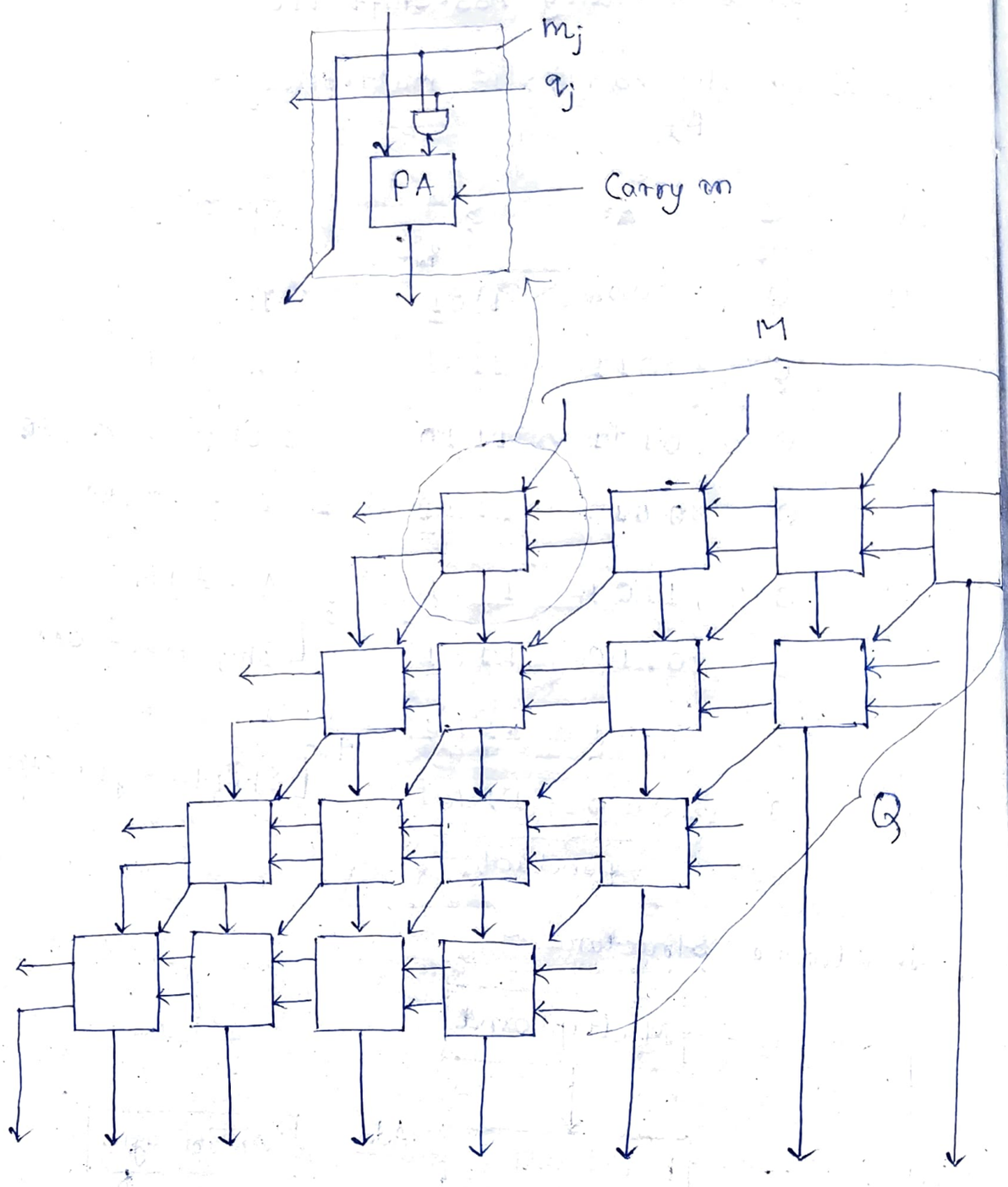
→ 11 (multiplicand) × 13 (multiplier)
M Q

M	C (carry)	A	Q (q0)	Operation
1011	0	0000	1101	Init
	0	1011	1101	1 { A = A + M
	0	0101	1110	1 { Shift-right CAQ
	0	0010	1111	2 Shift-right CAQ
	0	1101	1111	3 { A = A + M
	0	0110	1111	3 { Shift-right CAQ
	1	0001	1111	4 { A = A + M
	0	1000	1111	4 { Shift-right CAQ
		product		

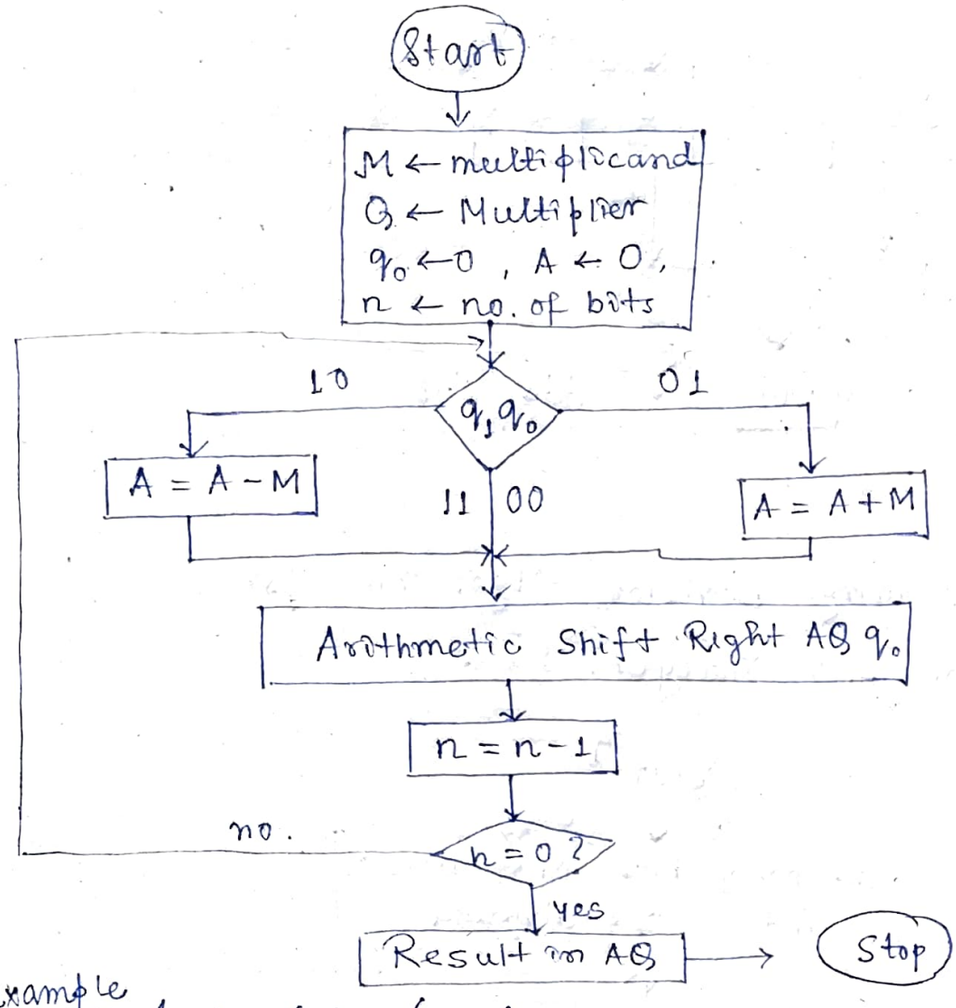
Hardware Structure ~



* Multiplication using Array Multiplier:



* Booth's Algorithm for Signed Multiplication:



example: $(-7) \times (3) = (-21)$

$M = -7_{10} = 1001_2$

$-M = 0111_2$

n	A	Q $q_4 q_3 q_2 q_1$	q_0	action
4	0000	0011	0	init
	0111	0011	0	A = A − M
3	0011	1001	1	ASR AQ q_0 | n--
2	0001	1100	1	ASR AQ q_0 | n--
	1010	1100	1	A = A + M
1	1101	0110	0	ASR AQ q_0 | n--
0	1110	1011	0	ASR AQ q_0 | n--

1's comp

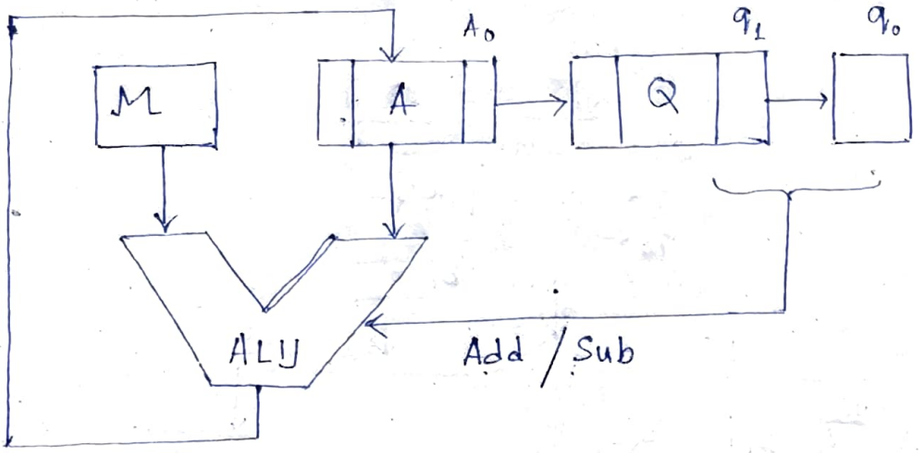
0001 0100

+ 1

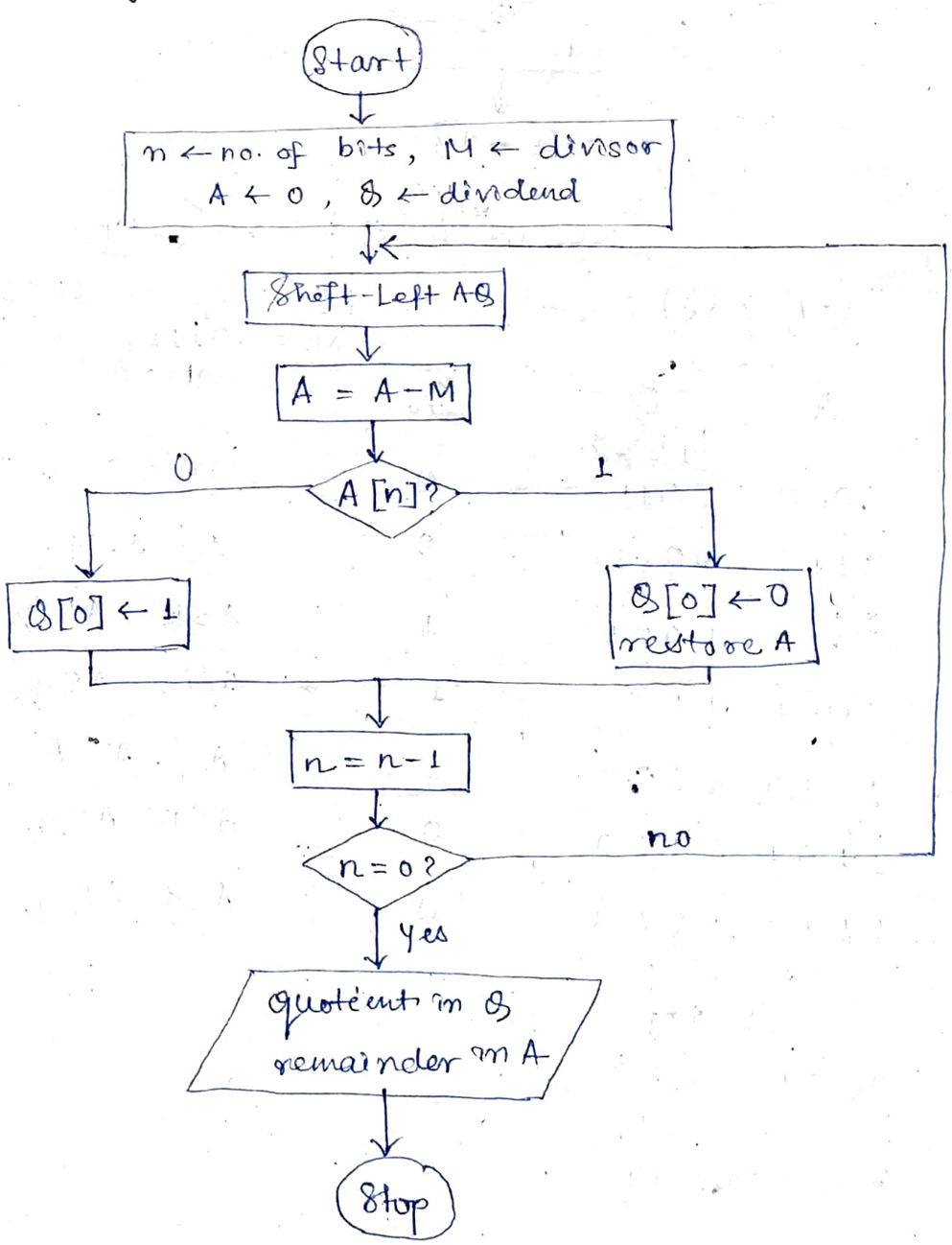
0001 0101

-21_{10}

Hardware Structure.



* Restoring Division Algorithm for Unsigned Integer. :

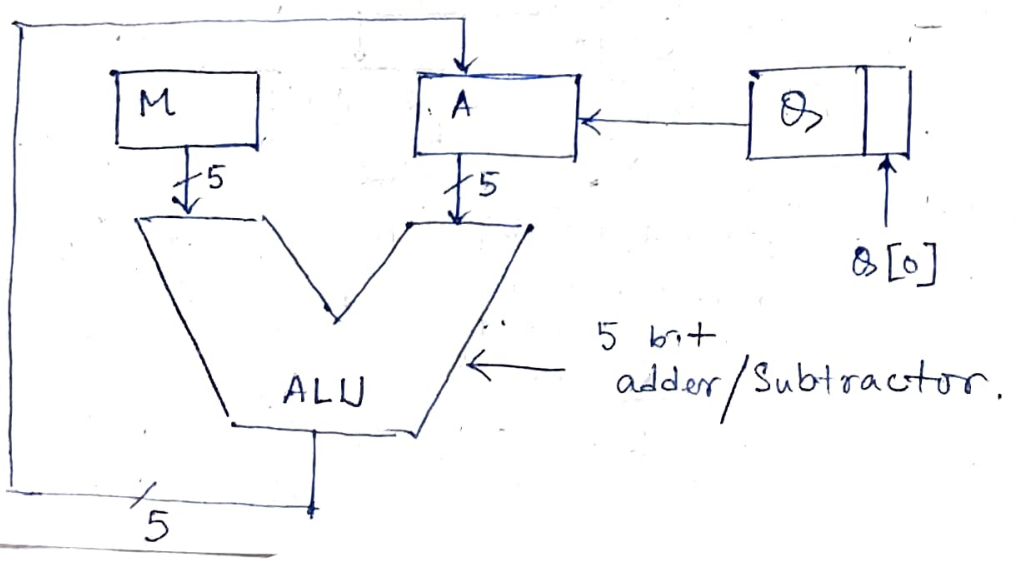


example. 11 (dividend) / 3 (divisor)
= 3 (quotient) & 2 (remainder)

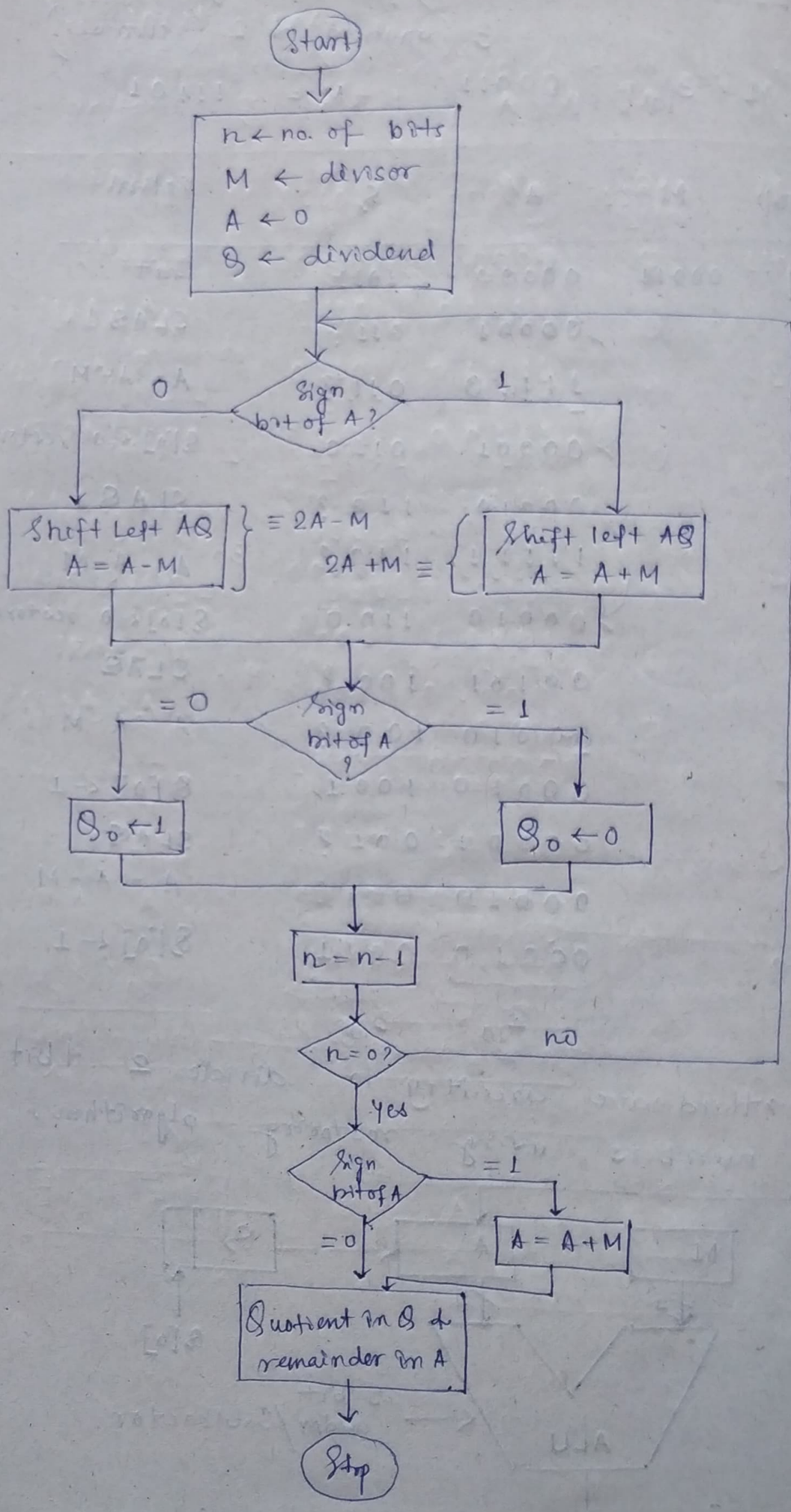
$M = 3_{10} =$ 00011 | $-M =$ 11101

n	M	A	Q	action
4	00011	00000	1011	init :
		00001	011 ?	SLAQ
		11110	011 ?	A = A - M
3		00001	0110	Q[0] ← 0 restore A
		00010	110 ?	SLAQ
		11111	110 ?	A = A - M
2		00010	1100	Q[0] ← 0 restore A
		00101	100 ?	SLAQ
		00010	100 ?	A = A - M
1		00010	1001	Q[0] ← 1
		00101	001 ?	SLAQ
		00010	001 ?	A = A - M
0		00010	0011	Q[0] ← 1
		2_{10}	3_{10}	

→ Hardware circuitry to divide 2 4 bit numbers using restoring algorithm.



Non-restoring division algorithm :



example. $11 \div 3$ $\quad M = 00011 \mid Q = 1011$, $-M = 11101$

n	M	A	Q	action
4	00011	00000	1011	init
		00001	011 ?	SLAQ
		11110	011 ?	A = A-M
3		11110	0110	$Q_0 \leftarrow 0 \mid n--$
		11100	110 ?	SLAQ
		11111	110 ?	A = A+M
2		11111	1100	$Q_0 \leftarrow 0 \mid n--$
		11111	100 ?	SLAQ
		00010	100 ?	A = A+M
1		00010	1001	$Q_0 \leftarrow 1 \mid n--$
		00101	001 ?	SLAQ
		00010	001 ?	A = A-M
= 0		00010	0011	$Q_0 \leftarrow 1 \mid n--$

2_{10} $\quad 3_{10}$

Pipelining

go.doc.

→ A pipelined processor allows multiple instructions to execute at once and each instn uses a different functional unit in the datapath.

→ This increases throughput, so programs run faster.

→ We may need to perform several operations in the same cycle.

Increment PC & add registers at same time.

Fetch one instn while another one reads/writes.

→ Design of a basic pipeline ~ In a pipelined processor, a pipeline has 2 ends, the ip end & the o/p end. Between these ends, there are multiple stages/segments s.t. o/p of one stage is connected to i/p of next stage & each stage performs a specific operation. Interface registers are used to hold the intermediate o/p between 2 stages. These interface registers are also called latch or buffer. All the stages in the pipeline along with the interface registers are controlled by a common clock.

→ Pipeline stages ~ RISC processor has 5 stages instn pipeline to execute all the instns in the RISC instn set.

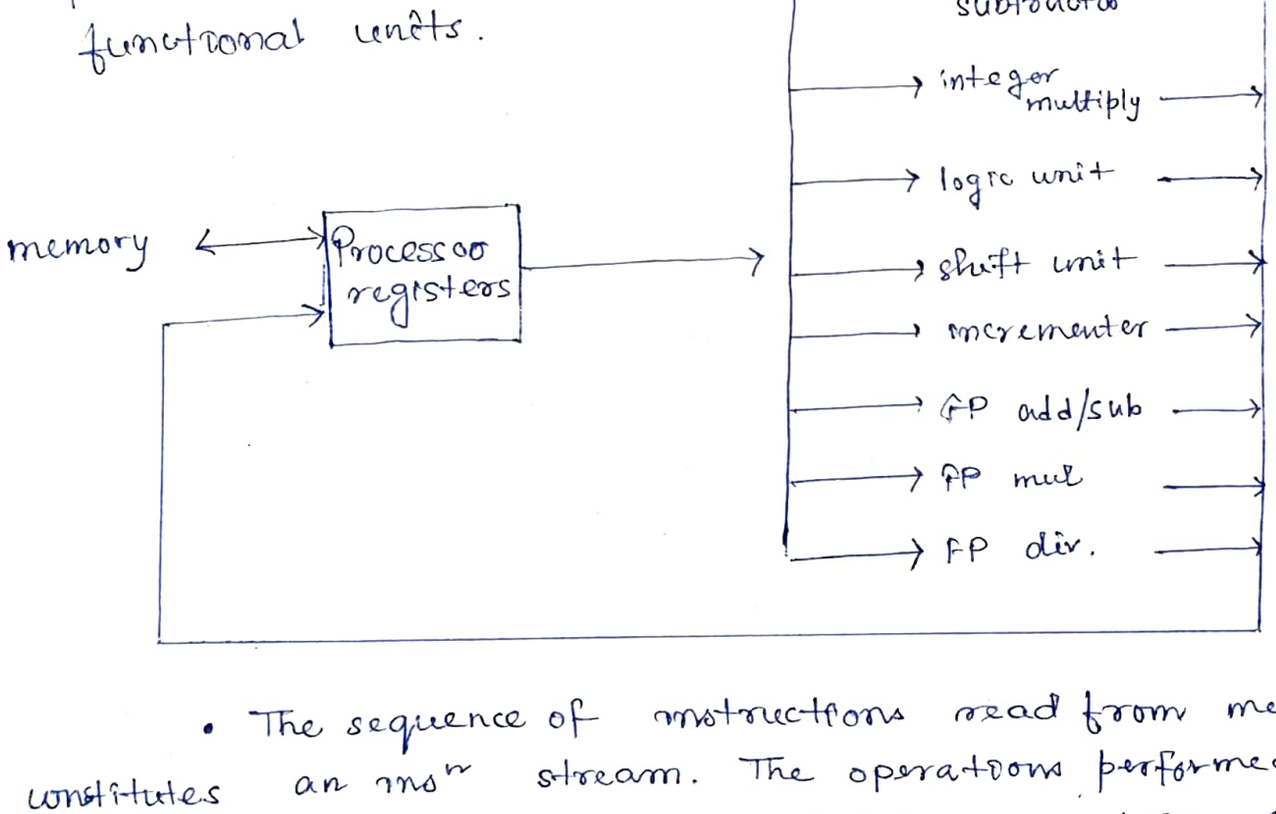
1. Stage 1 (Instn fetch) ~ CPU reads instns from the addr. in the memory whose value is present in the PC.
2. Stage 2 (Instn decode) ~ Instn is decoded & the register file is accessed to get the values from the registers used in the instn.
3. Stage 3 (Instn execute) ~ ALU operations are performed.
4. Stage 4 (Memory access) ~ Memory operands are read & written from/to the memory that is present in the instn.
5. Stage 5 (Write Back) ~ Computed/fetched value is written back to the register present in the instns.

→ Flynn's Taxonomy.

• Parallel Processing ~ Jobs are broken into discrete parts that can be executed concurrently. Each part broken down to further instructions. Ins^n's from each part execute simultaneously on different CPUs.

Parallel systems deal with the simultaneous use of multiple computer resources. Processor with multiple functional units deals with parallel processing.

Processor with multiple functional units.



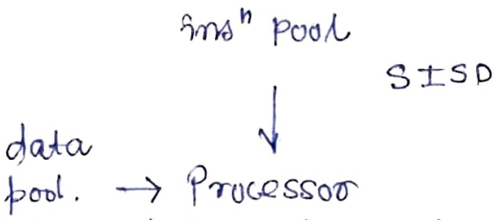
• The sequence of instructions read from memory constitutes an ins^n stream. The operations performed on the data in the processor constitutes a data stream.

• Based on the number of instructions & data stream that can be processed simultaneously, computing system are classified into four major categories ~

Data Stream \ Ins^n Stream	one	many
one	SISD Von Neumann single CPU computer	MISD may be pipelined computers
many	SIMD Vector processors parallel computers	MIMD multi computer multi processor

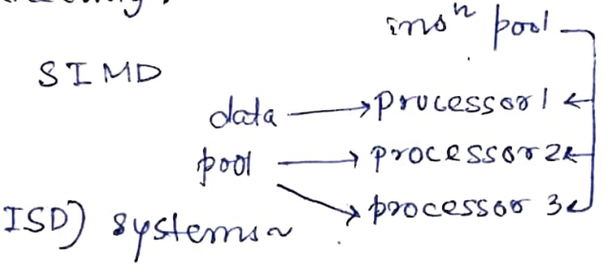
1. Single Ins^n, Single Data (SISD) systems ~

Executing a single ins^n operating on a single data stream. Machine ins^{n}s are processed in a sequential manner. All ins^{n}s & data have to be stored in main memory.



2. Single ins^n, multiple data (SIMD) systems ~

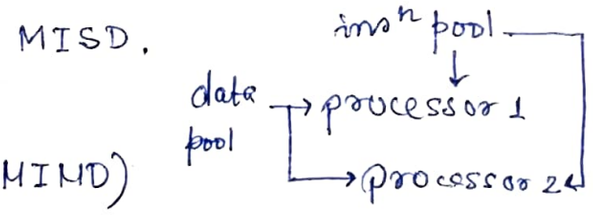
Multiprocessor machine capable of executing the same ins^n on all the CPUs but operating on different data streams. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.



3. Multiple ins^n single data (MISD) systems ~

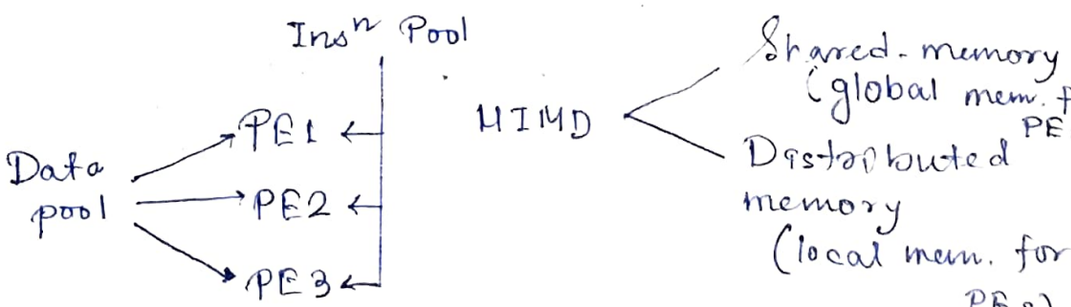
Multiprocessor machine capable of executing different ins^{n}s on different processing elements but all operating on same dataset. $Z = \sin x + \tan x + \cos$

Machines with MISD are not useful on most case

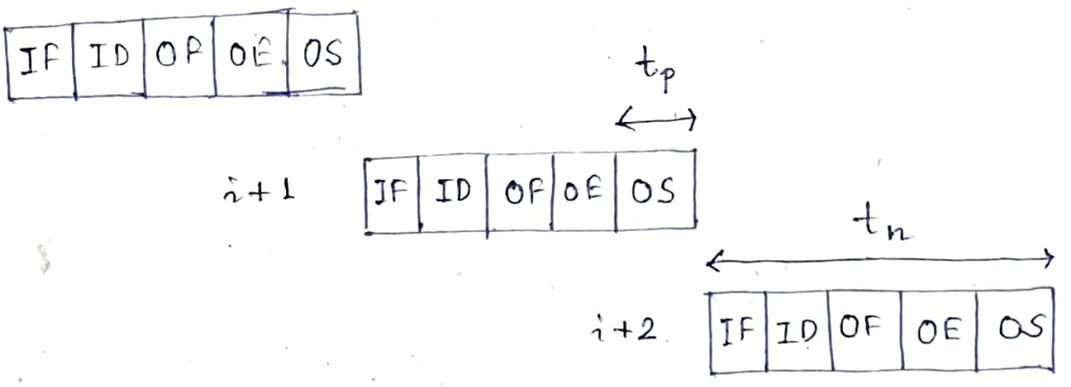


4. Multiple ins^n multiple data (MIMD) systems ~

Multiprocessor machine capable of executing multiple ins^{n}s on multiple data sets. Each processing element in the MIMD has separate data & ins^n stream. PEs work asynchronously unlike SIMD, MISD.



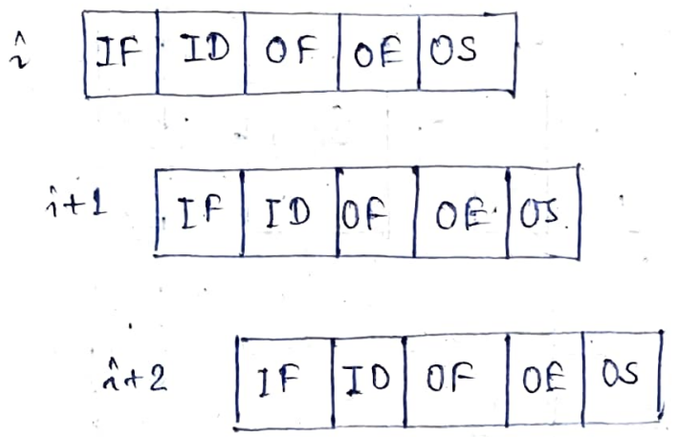
Pipelining.



Non-pipelined architure.

→ instruction-wise interleaved.

t_1 t_2 t_3 t_4 t_5 t_6 t_7



Pipelined architecture ins^n execution.

→ Phase-wise interleaved.

Performance in Pipelining.

→ No. of clock cycles = $K + n - 1$

$K \rightarrow$ No. of segments

$n \rightarrow$ No. of tasks.

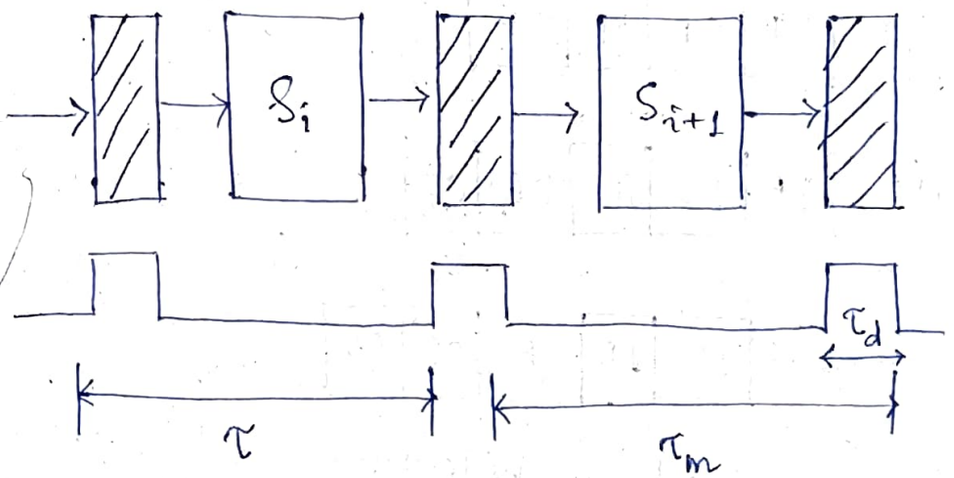
→ Speed-up ratio (S_K) =

$$\frac{\text{Non-pipelined execution time}}{\text{Pipelined execution time}}$$

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

($n \times t_n$ ← n instⁿ execⁿ time)

[assuming all tasks take same time t_p]



$$\tau = \max(\tau_i) + \tau_d$$

($\max(\tau_i)$ ← max phase duration; τ_d ← latch delay)

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

(τ ← clock cycle duration)

if $n \gg K$, $\therefore S_K = \frac{n \times t_n}{n \times \tau} = \frac{t_n}{\tau}$.

→ Frequency $= \frac{1}{\tau}$

→ Throughput, $H_K = \frac{n}{(K+n-1)\tau}$

no. of ins executed / unit time

$\therefore H_K = \frac{nf}{K+n-1}$

→ Pipeline efficiency, $E_K = \frac{S_K}{K}$

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

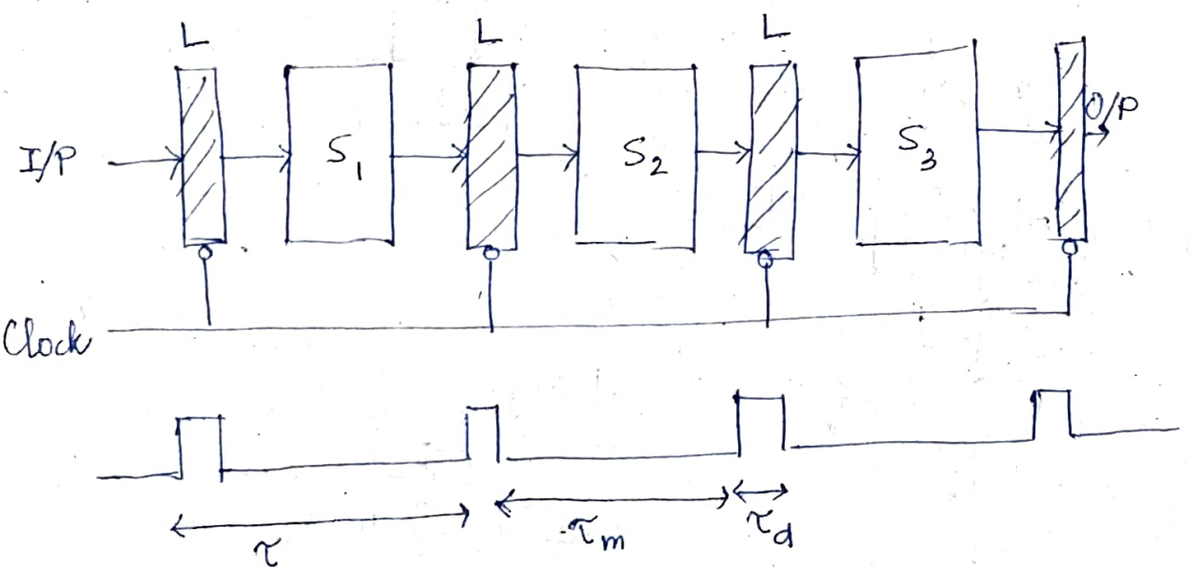
$$= \frac{nK}{K+n-1}$$

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

(t_p: one task duration) (τ: clock cycle dur^n)

→ t_p needs not be equal to τ.

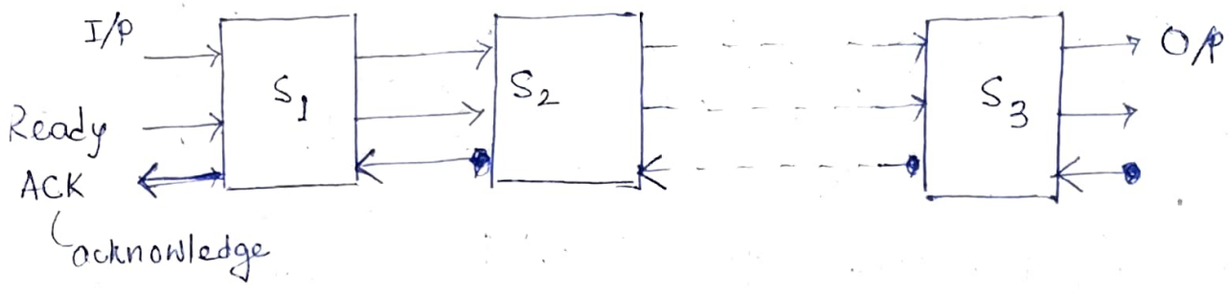
⑧ Synchronous Pipelining



$$\tau = \max\left(\tau_i\right)_{i=1}^{K} + \tau_d$$

$$= \tau_m + \tau_d$$

⊛ Asynchronous Pipelining



⊛ Arithmetic Pipeline Architecture

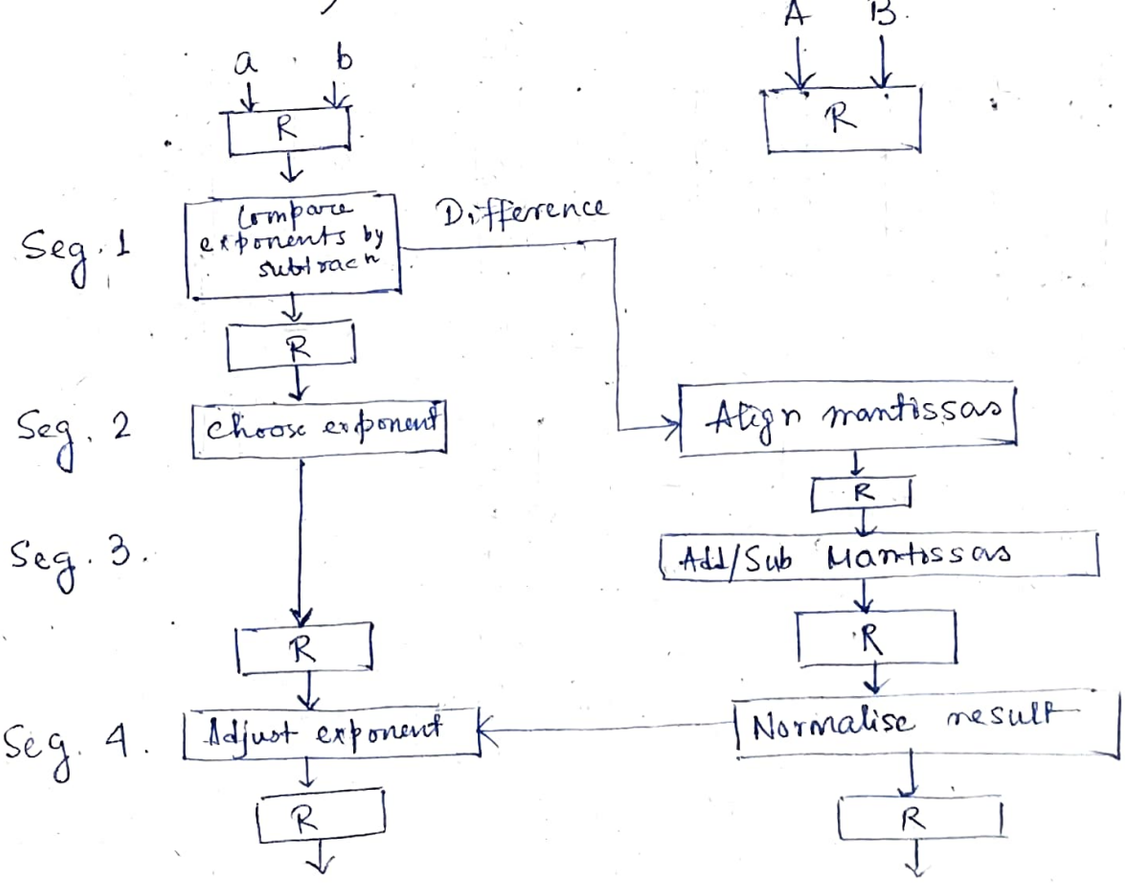
eg. Floating point adder pipeline

$$X = A \times 2^{a}$$

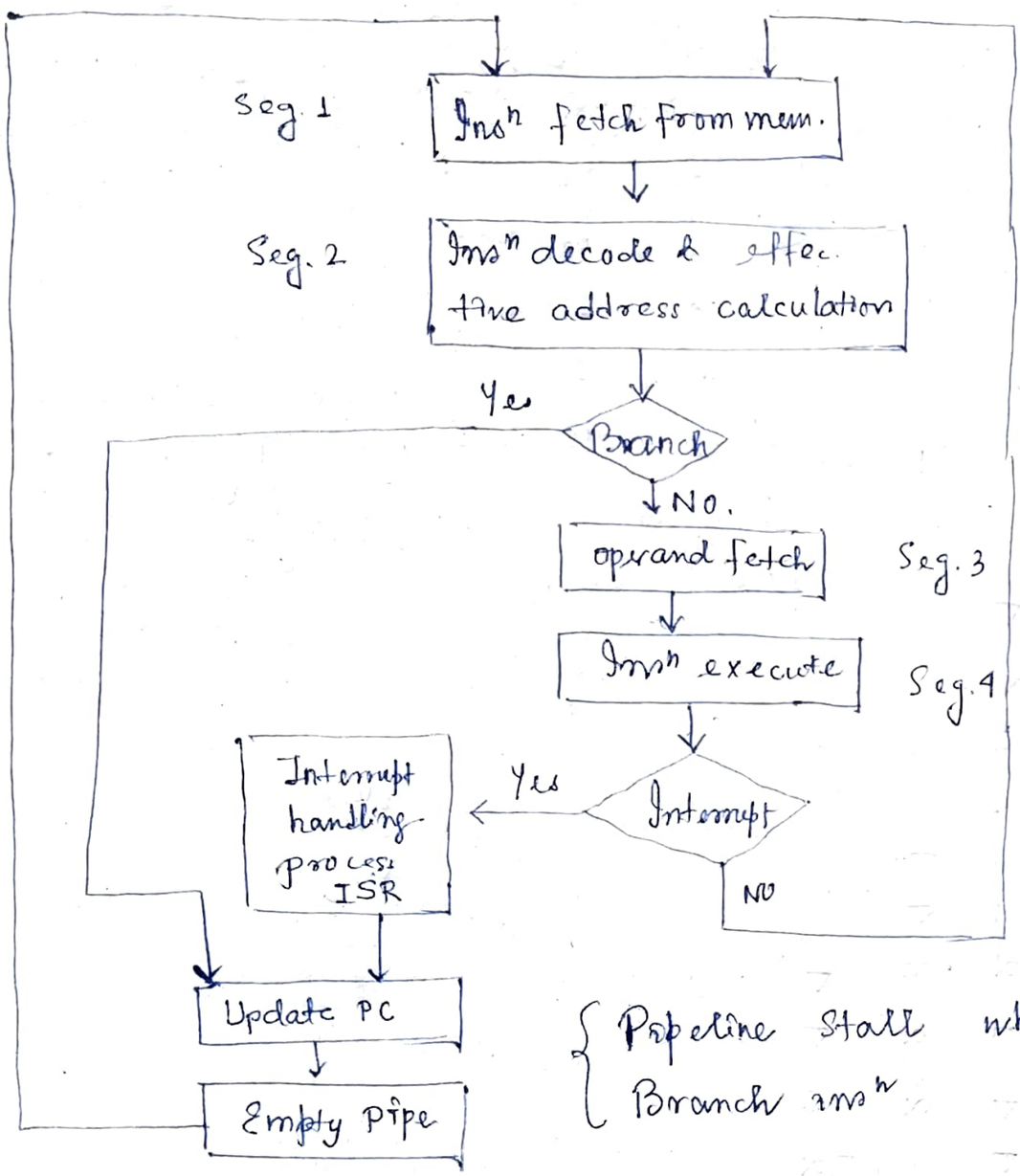
$$Y = B \times 2^{b}$$

Can be done in 4 segments -

1) Compare the exponents
2) Align the mantissas
3) Add or subtract the mantissa
4) Normalize the result.



⊛ Instruction Pipelone. Architecture



Step	1	2	3	4	5	6	7	8	9	10	11	12	13
Insⁿ: 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5					-	-	-	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

Timing of insⁿ pipeline

Q. Consider a 4-stage pipeline processor. No of cycles needed by the 4 m/c's I1, I2, I3, I4 in stages S1, S2, S3, S4 is shown

	S_1	S_2	S_3	S_4
I_1	2	1	1	1
I_2	1	3	2	2
I_3	2	1	1	3
I_4	1	2	2	2

What's the no. of cycles needed to execute the following loop?

```
for i = 1 to 2 {
    I1; I2; I3; I4;
}
```

→ 16 / 23 / 25 / 29

G'09

Ans.

23. clock cycles needed.

Q4 05 We have 2 designs D1 & D2 for a synchronous pipeline processor. D1 has 5 pipeline stages with execution time of 3ns, 2ns, 4 ns, 2ns, 3 ns. While the design D2 has 8 pipeline stages each with 2ns exe^{n} time. How much time can be saved using design D2 over design D1 for executing 100 insts ? → 214 / 202 / 86 / 200 ns

→ D1 → 5 stages.

cycle time = Max (3, 2, 4, 2, 3) = 4 ns.

$(\text{Exe}^{n} \text{ time})_{D1}$ = (5 + 100 − 1) × 4 ns

= 104 × 4 = 416 ns

D2 → 8 stages.

cycle time = 2 ns.

$(\text{Exe}^{n} \text{ time})_{D2}$ = (8 + 100 − 1) × 2 ns

= 107 × 2 ns

= 214 ns.

(416 − 214) = 202 ns. (Ans)

Q. eg. Pipeline has 4 phases with duration 60, 50, 90, 80 ns. Latch delay is 10 ns.

Non

→ Pipeline cycle time =

$$\max\left(\tau_i\right)_{i=1}^{k} + \tau_d = (90 + 10)\ ns.$$

$$= 100\ ns$$

→ Non-pipeline exec^{n} time =

$$(60 + 50 + 90 + 80) = 280\ ns.$$

→ Speedup ratio (S_k) = $\frac{280}{100} = 2.8$

→ Pipeline time for 1000 tasks =

$$(n+k-1) \times \tau = (1000 + 4 - 1) \times 100.$$

$$= 100300\ ns.$$

→ Sequential time for 1000 tasks =

$$(280 \times 1000)\ ns.$$

→ Throughput = H_k = $\frac{1000}{1003 \times 100}$

* Calculation of important parameters:

Considering a pipelined architecture consisting of k-stages. Total no. of instns to be executed is n.

1. Calculating cycle time ~

There is a global clock that synchronizes the working of all the stages. Frequency of clock is set s.t. all the stages are synchronised. At the beginning of each clock cycle, each stage reads the data from its register & process it. Cycle time is the value of one clock cycle.

Case 1 - All the stages offer same delay.

cycle time = delay offered by one stage including the delay due to its register.

Case 2 - All the stages don't offer same delay.

cycle time = max. delay offered by any stage including the delay due to its register.

→ frequency = 1/cycle time.

2. Calculating non-pipelined execution time ~

No. of clock cycles taken by each $instn$ = k clock cycles.

Non-pipelined execution time =

Total no. of $instns$ x Time taken to execute one $instn$

= n x k clock cycles.

3. Calculating pipelined execution time ~

No. of clock cycles reqd. by first $instn$ = k clock cycles. After first $instn$ is completely executed, one $instn$ comes out of the $instn$ set per clock cycle. So, no. of clock cycles taken by each remaining $instn$ = 1 clock cycle.

Pipelined execution time =

Time taken to execute first $instn$ + Time taken to execute remaining $instns$

= 1 x k clock cycles + (n-1) x 1 clock cycles

= (k + n - 1) clock cycles.

* Aim of pipelined architecture is to execute one complete $instn$ in one clock cycle (CPI = 1).

Ideally, it's assumed CPI = 1.

* Maximum speedup = no. of stages [when efficiency is 100%].

* Ideally speedup = k.

* Experiments show that 5 stage pipelined processor gives the best performance.

* In case only one instn has to be executed, then non-pipelined execution gives better performance because delays are introduced due to registers in pipelining.

* High efficiency of pipelined processor is achieved when- all the stages are of equal duration, there are no conditional branch instns, there are no interrupts, there are no register & memory conflicts.

* Difficulties in instn pipelining:

1. Resource conflicts ~ Caused by access to memory by 2 segments at the same time. Most of these conflicts can be solved by using separate instn & data memories. This conflict arises when instn fetch phase tries to access the memory for reading the code & register write back phase tries to access the memory to store the results.

2. Data dependency conflicts ~ Arises when an instn depends on the result of a previous instn but this result is not yet available. When an instn is updating some register & next instn is trying to read the update on decoding phase, but till that time no update is made on the actual register, it can lead to Read before write hazard in the system.

3. Branch difficulties ~ Arises from branch & other instns that change the value of PC. In some instns, the result of branch operation is present after the completion of the execution or in some other phase of execution; so, some stalls are created for successful execution.

* Hazard is any condition that causes the pipeline to stall.