GATE CSE NOTES

by

UseMyNotes

Basic Structure of Computers.

* Computer types :

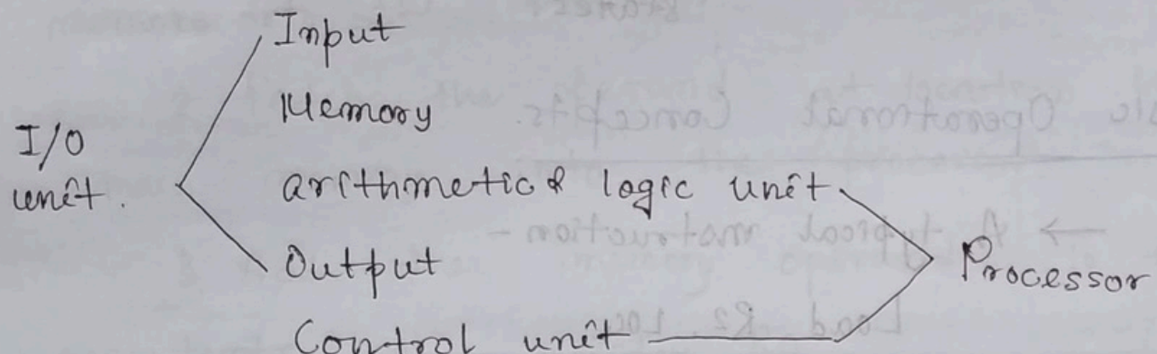
i) Embedded computers.

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 1/200th times of a vacuum tube.

iii) Power requirement of a transistor is 1/20th 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

Secondary - 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.

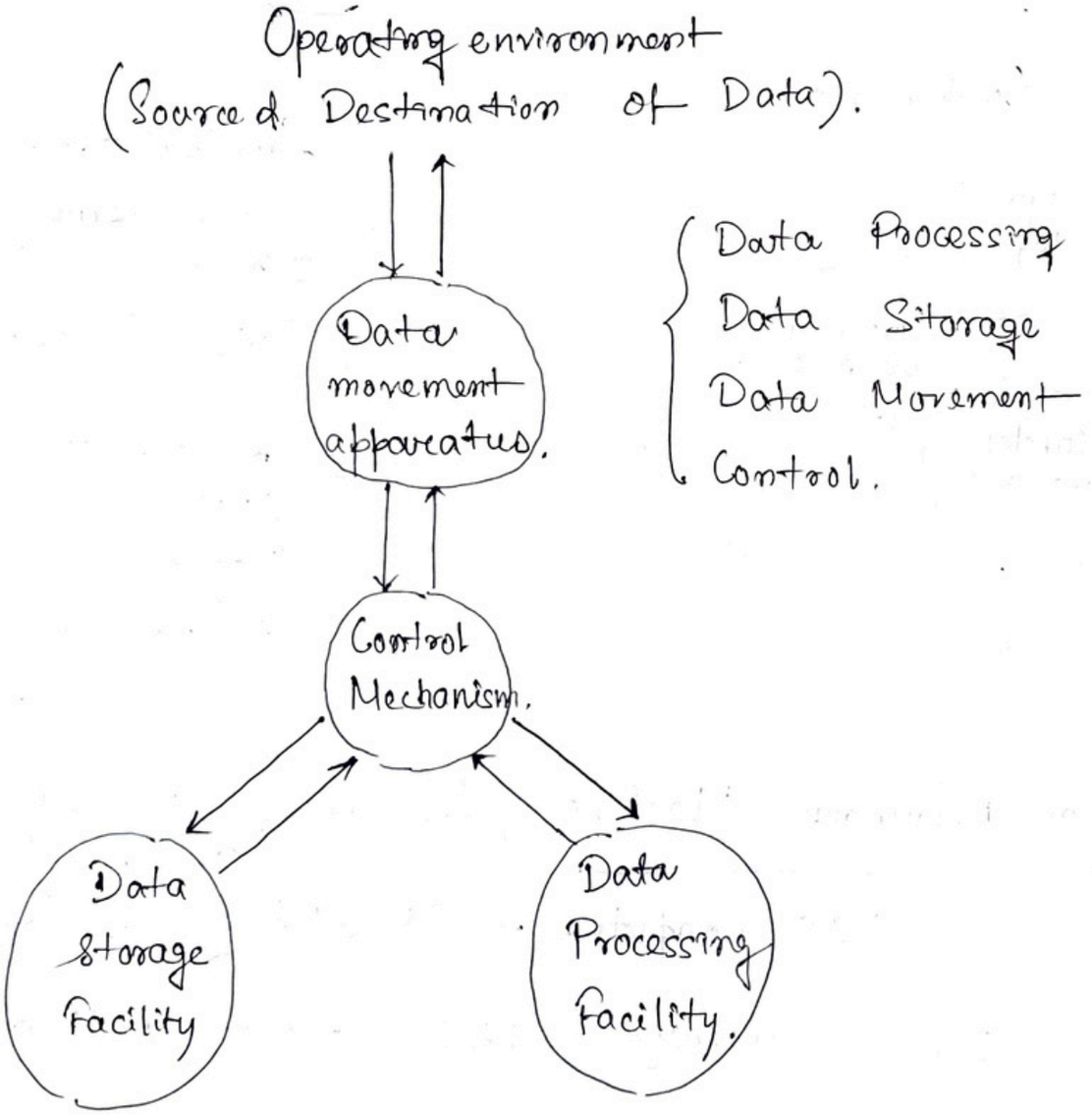
5. Program Counter (PC)

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

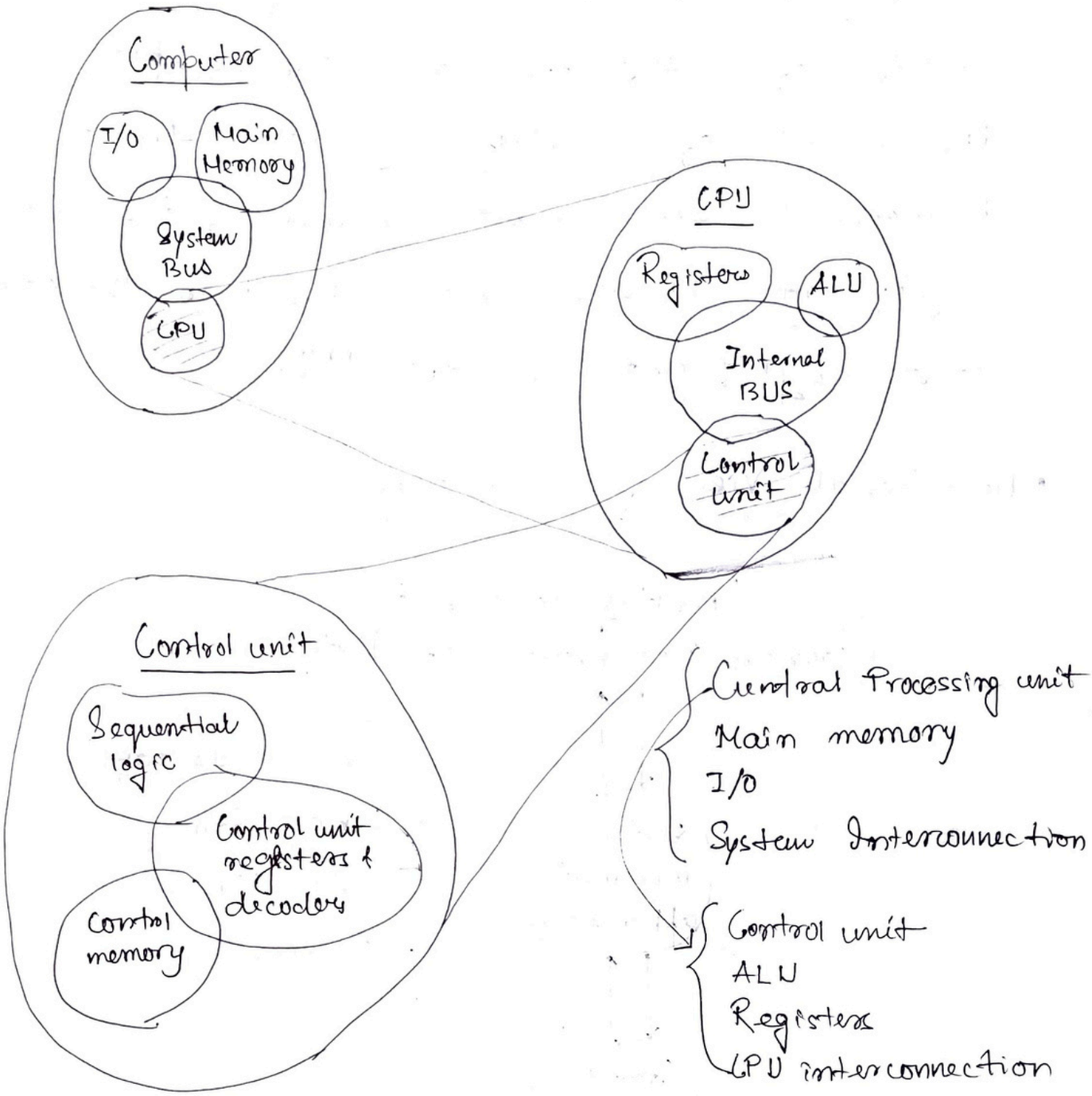
6) Accumulator (AC) & Multiplier Quotient (MQ)

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 ↄ

→ 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, $S = T/T'$

$T = t_n + t$.

$T' = t_n + t'$

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

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

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

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

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

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

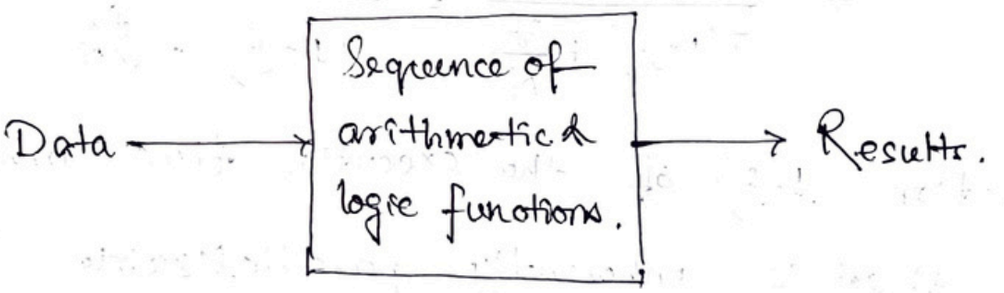
→ Von Neumann Architecture.

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

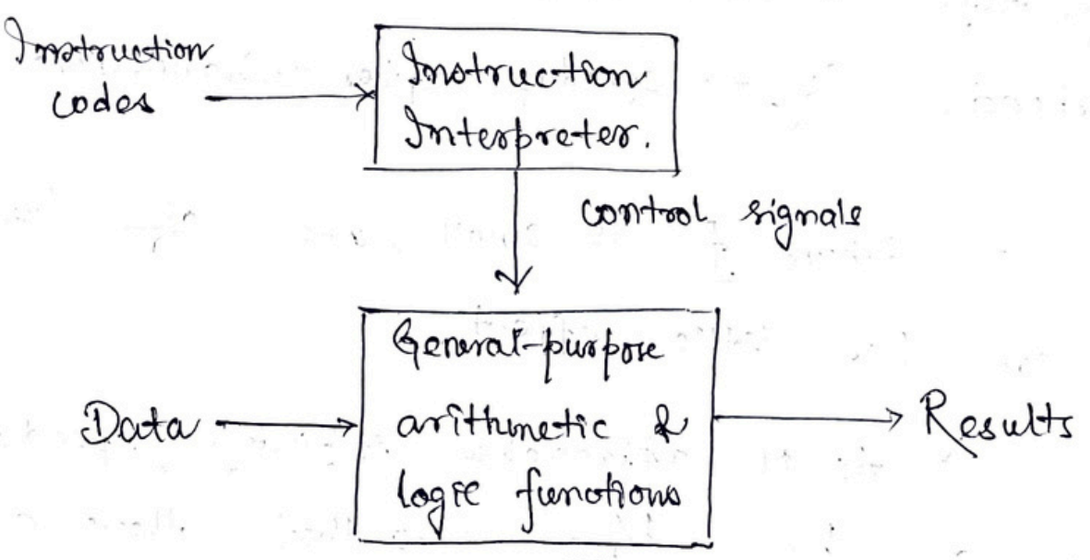
→ Contents of this memory are addressable by location, without regard to the type of data contained there.

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

→ Programming in hardware vs software.

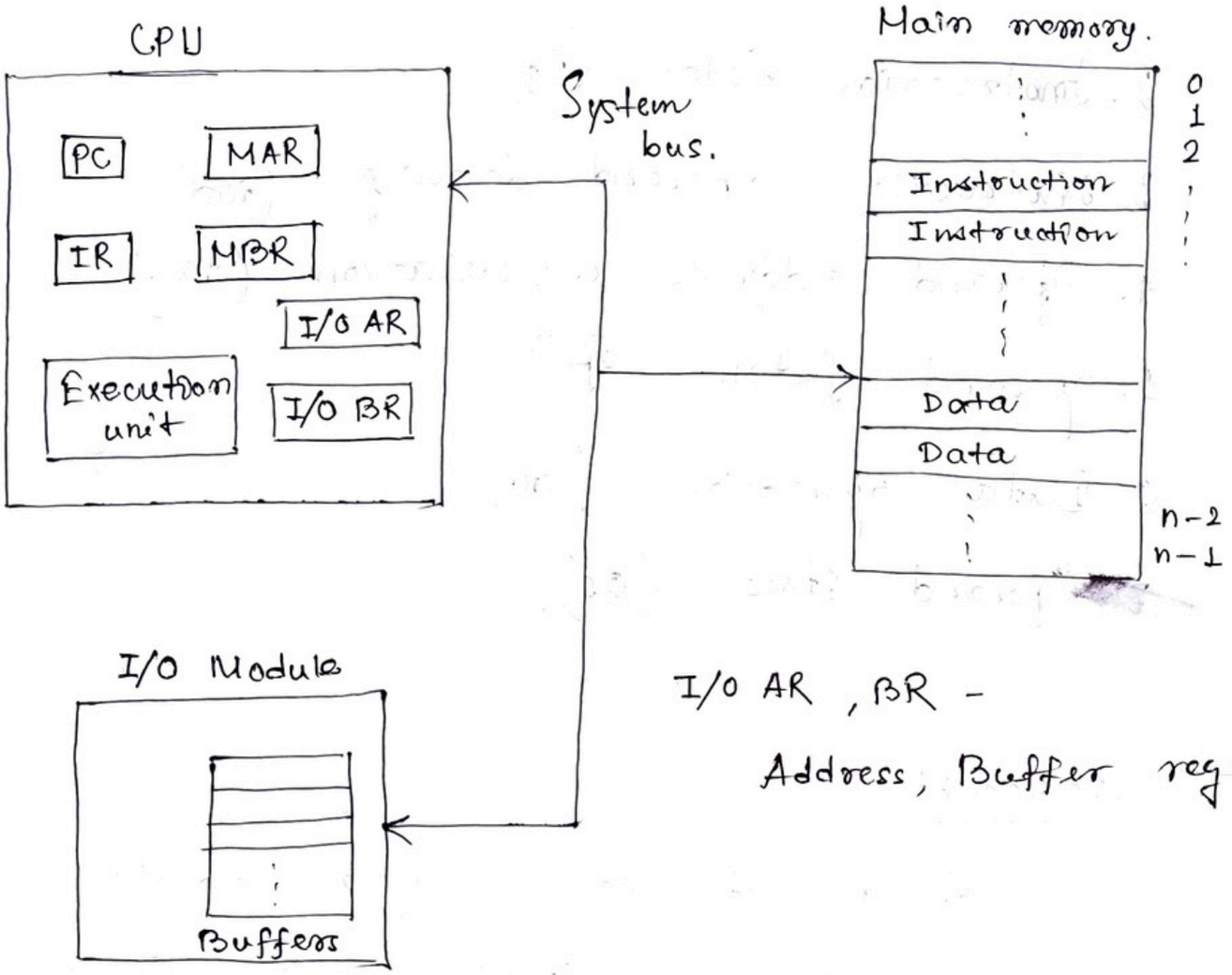


Programming in hardware.



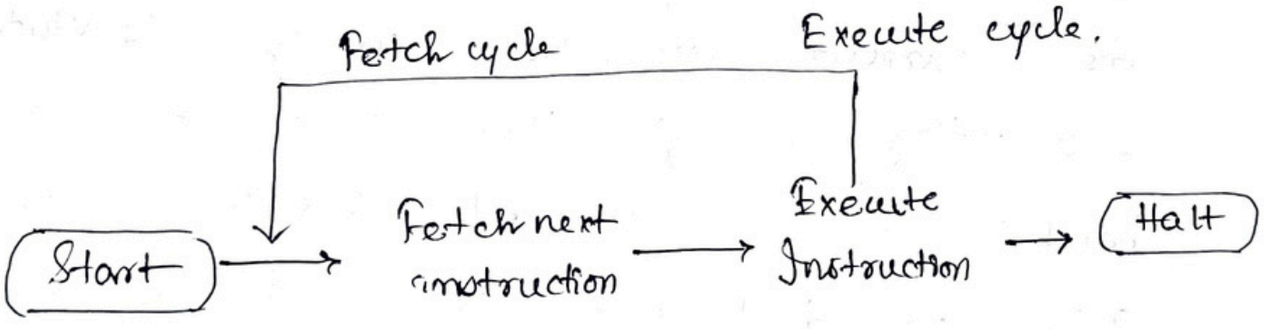
Programming in software.

→ 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.

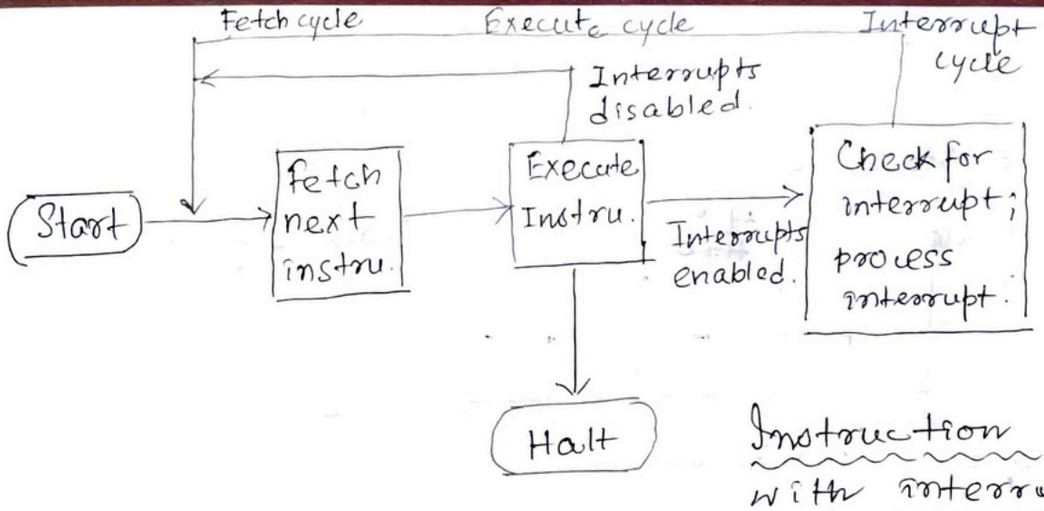
→ States for any given instruction cycle.

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 ~

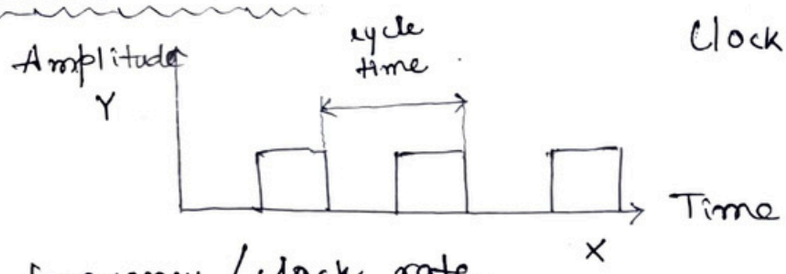
Program ~ Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.

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):

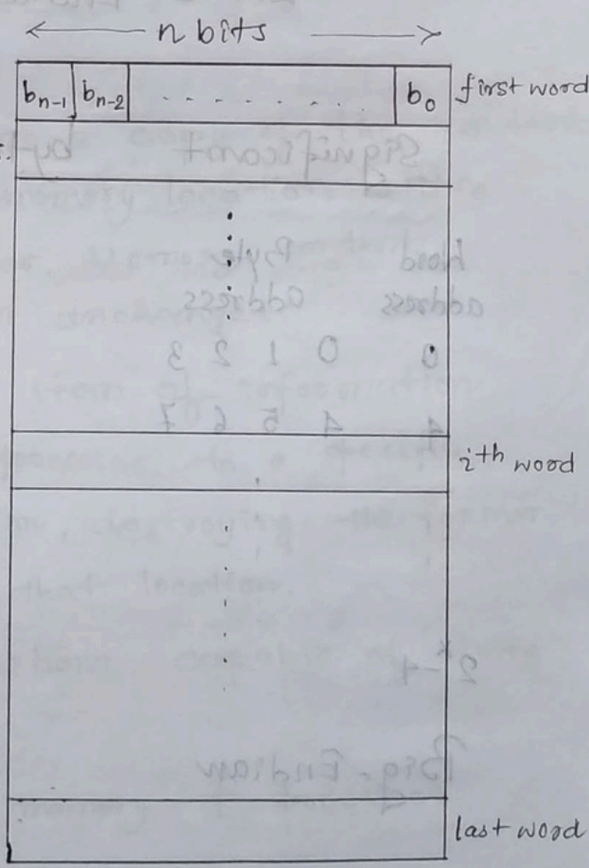
A complete instruction set of a processor specifying instructions, addressing methods used for the access of data operands & the processor registers available for use by the instructions.

* 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:

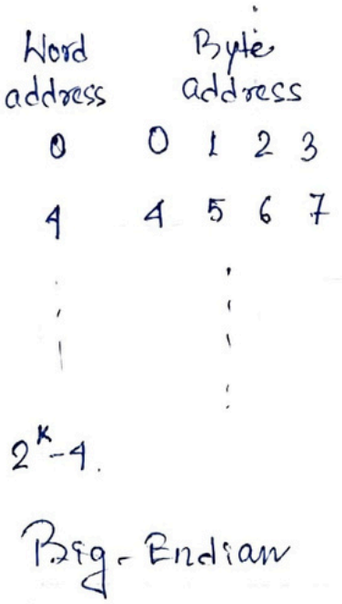
→ Every byte has its own unique address & can be accessed.

→ 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.

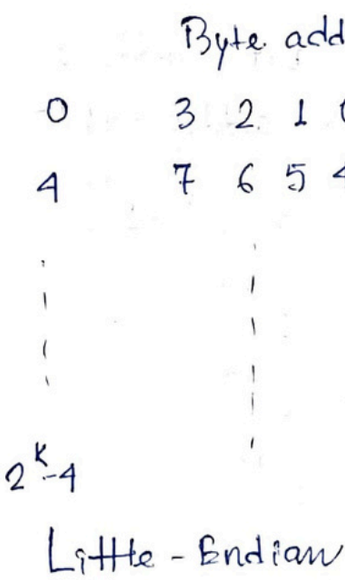
* Big-Endian & Little-Endian Assignments.

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 execut^n 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.

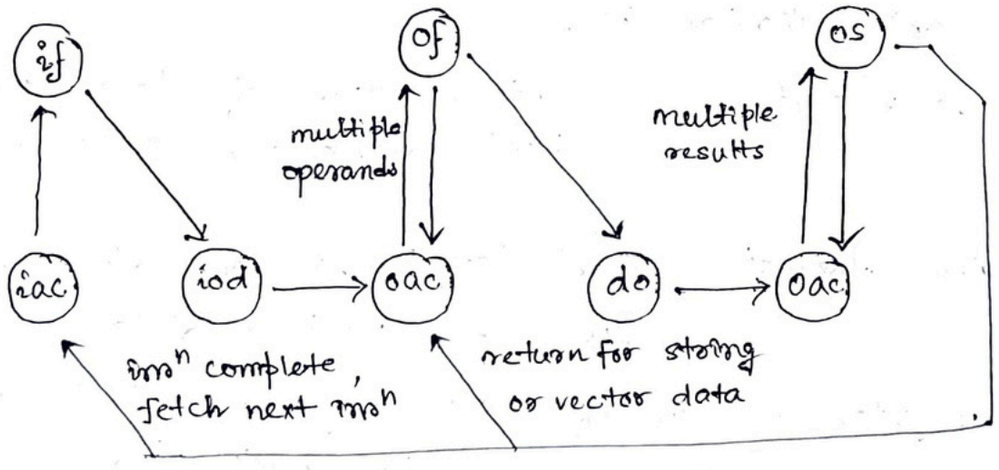
iii) Ins^{n} op.^{n} decoding (iod) – analyse ins^{n} to determine type of op^{n} to be performed & operands to be used

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}.

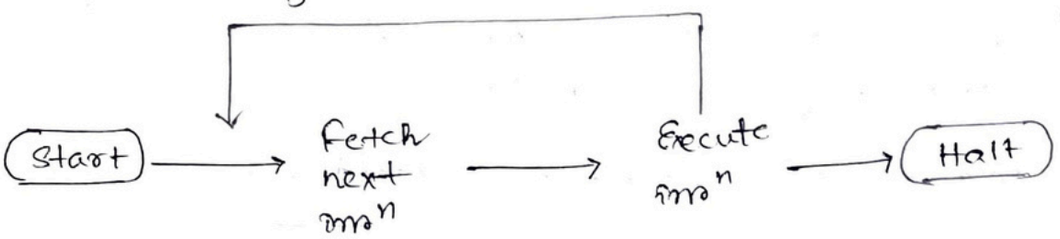
vii) Operand store (os) – Write the result into memory or out to I/O.



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

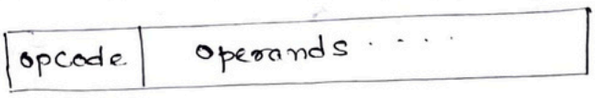
States on the lower part involve only internal processor operations.

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

* Ins^{n} types examples.

i) Data transfer : MOVE, LOAD, STORE, PUSH, POP, XCHG (swap contents of source & destination), CLEAR (Reset dest^{n} with all 0), SET (set dest^{n} 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 ins^{n} : 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
(S1, 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 on stack ; absolute address of operand is held on 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
MUL Z, C, D
DIV Y, Y, Z

3 ad. insn
```

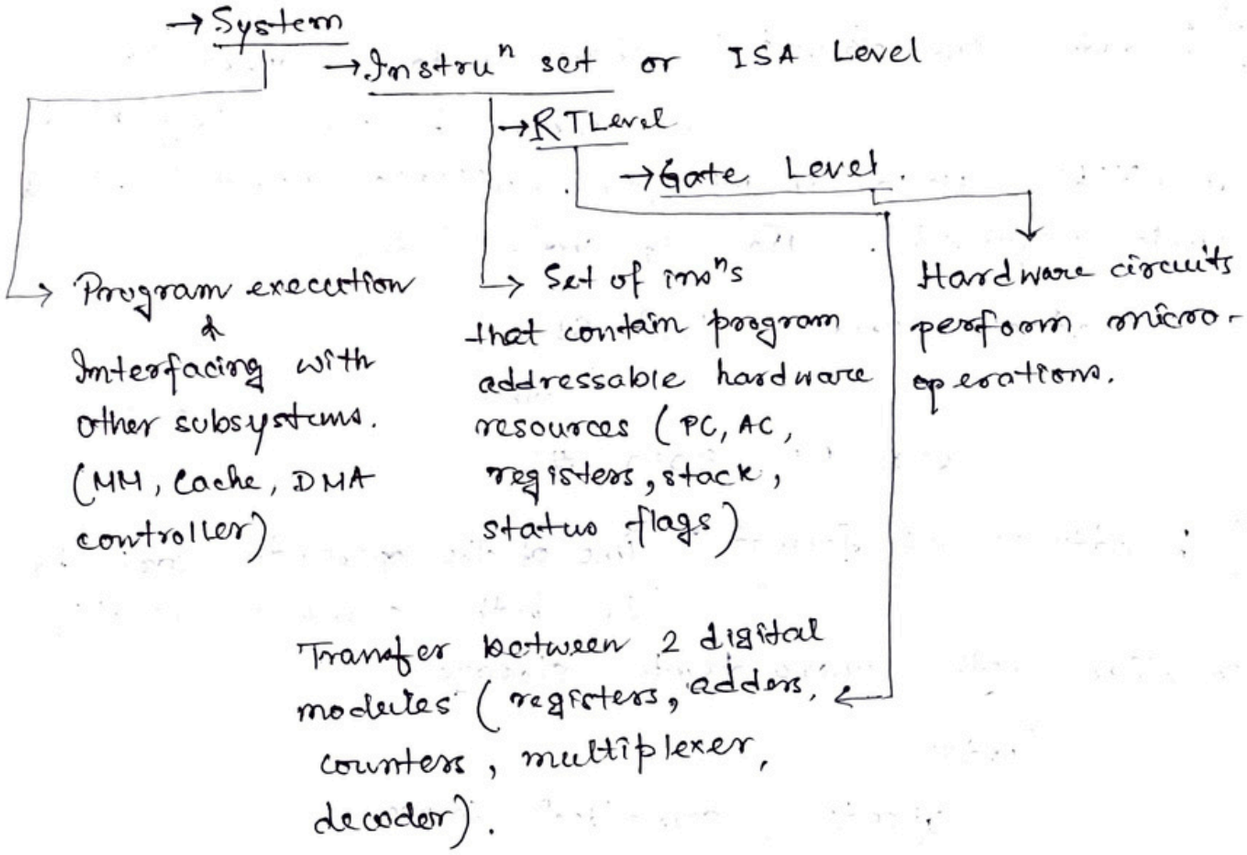
```
MOV Y, A
ADD Y, B
MOV Z, C
MUL Z, D
DIV, Y, Z

2 ad. insn
```

```
LOAD C
MUL D
STORE Y
LOAD A
ADD B
DIV Y
STORE Y.

1 ad. insn.
```

* 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^n'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.

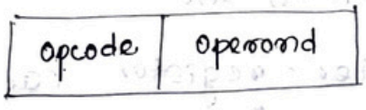
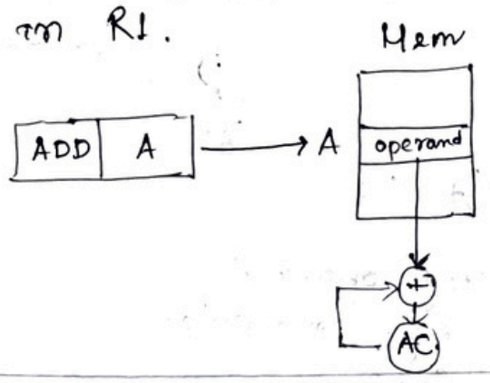
- Refers to the way in which the operand of an ins^n is specified.
- Multiple addressing modes give flexibility to the programmer in writing efficient programs.
- Addressing mode is indicated to the CU in any of the following ways -

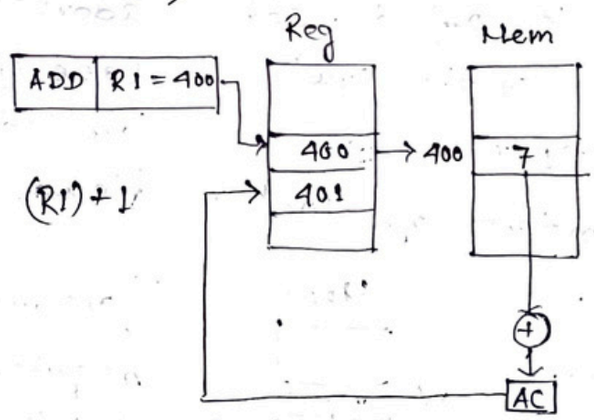
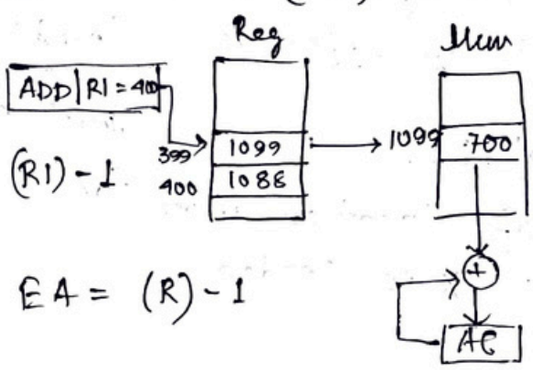
1. A separate mode field in the ins^n indicates the addressing mode used.
2. The opcode itself explicitly specifies the mode.

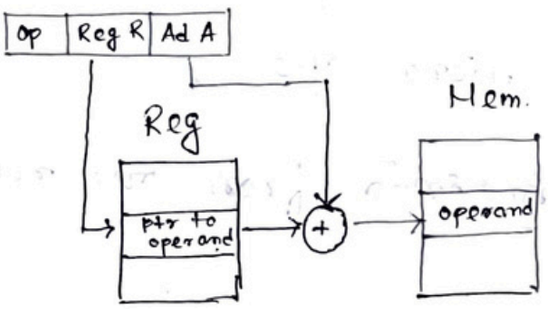
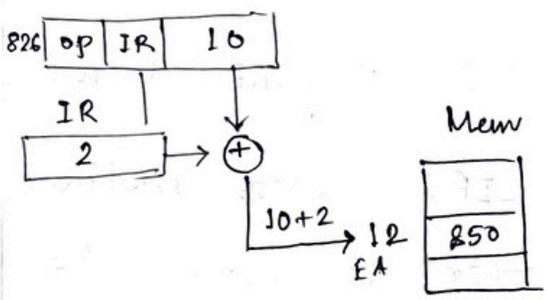
- Effective address / offset - offset determined by adding any comb^n of 3 address elements - displacement, base, index.

contents of base register (BX or BP)

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 s.t.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 ins^ns of a program & store the results in the accumulator.

→ Ins^n 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 in registers or in 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 ins^n size. ii) Ins^n cycle takes less time. (as fetching takes " ")

→ Disadvantages ~

i) Memory size increases for complex expressions because of many short ins^ns, ii) exe^n 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

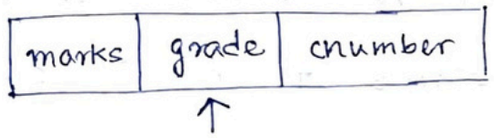
a) Post-increment addressing mode (R1)+

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 instruction 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. loc^n 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 instr^n 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.

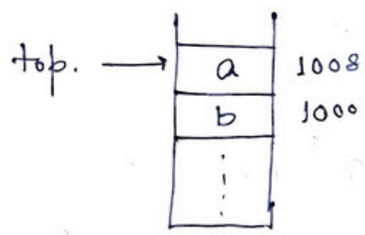
a) ADD (X)-, (X)

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.

8 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 instn size (True)

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

** 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 instn, 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.

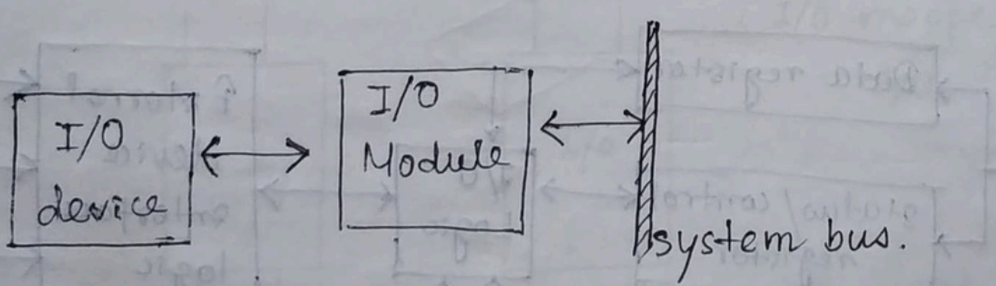
* Reasons for not connecting peripherals directly to the system bus :

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 :

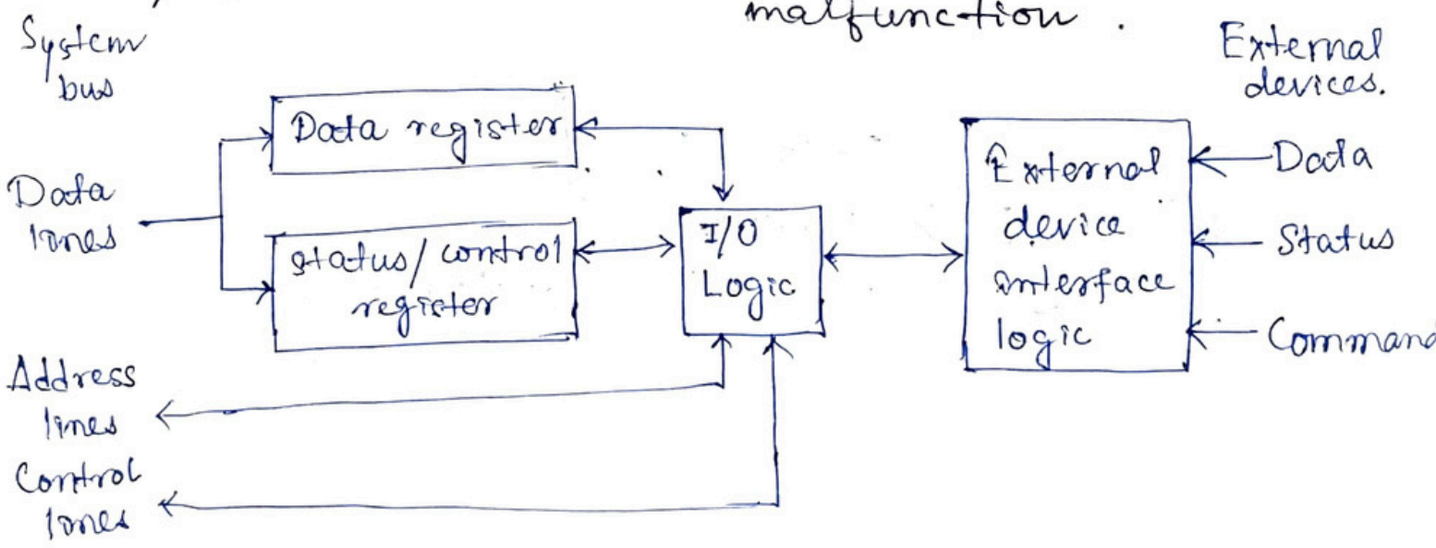
i) Control & timings : Coordinates the flow of traffic between internal & external resources.

ii) Processor communication : Communicate with processor during I/O operation. It involves command decoding, data exchange, status 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.

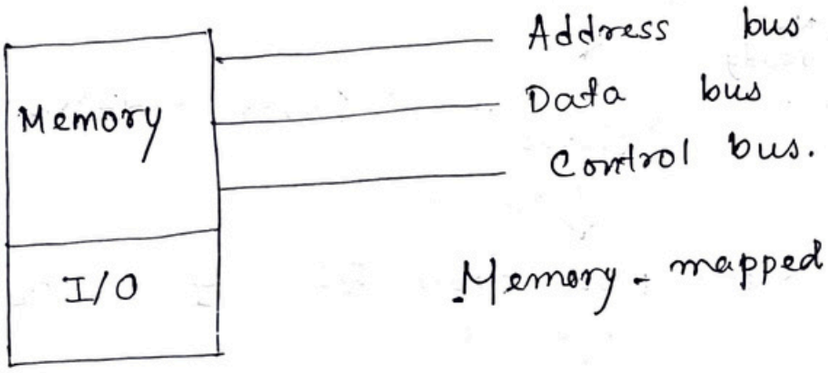


* Memory-mapped I/O : There is a single address space for memory locations and I/O devices. The processor treats the status & address register of I/O module as memory location. Processor & I/O are mapped using the memory address.

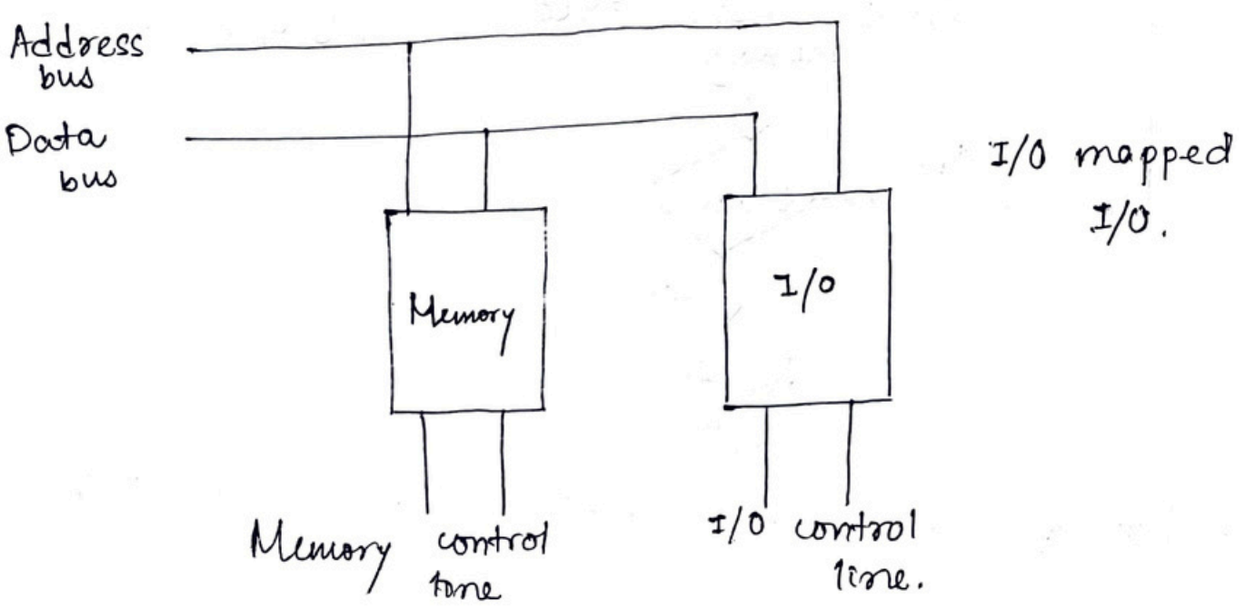
Available addresses for memory are less.

* 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

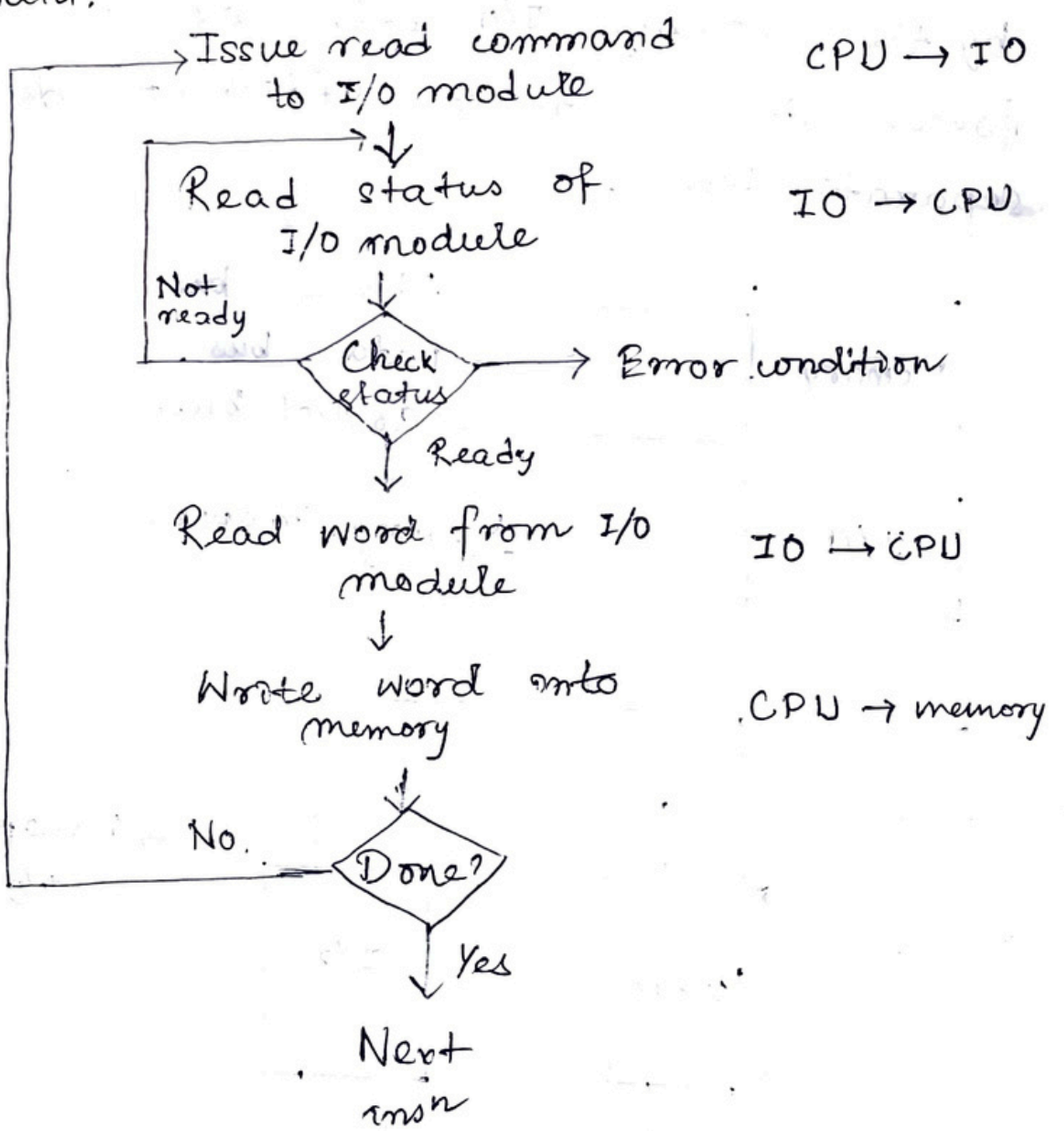


I/O mapped I/O.

* 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.

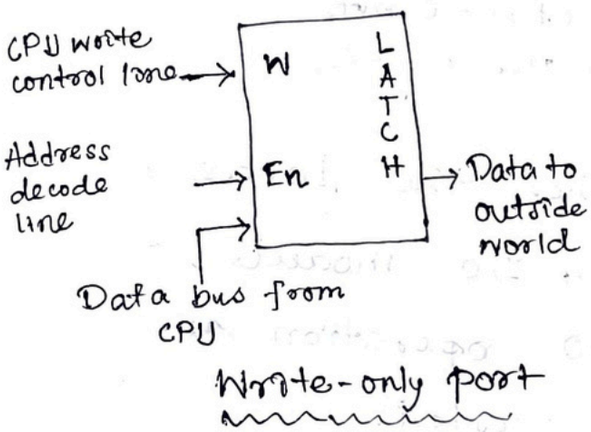
* Programmed I/O : The processor executes a program that gives its direct control of the I/O operation, including sensing device status, sending a read or write command & transferring the data.



Input of block of data in Programmed I/O.

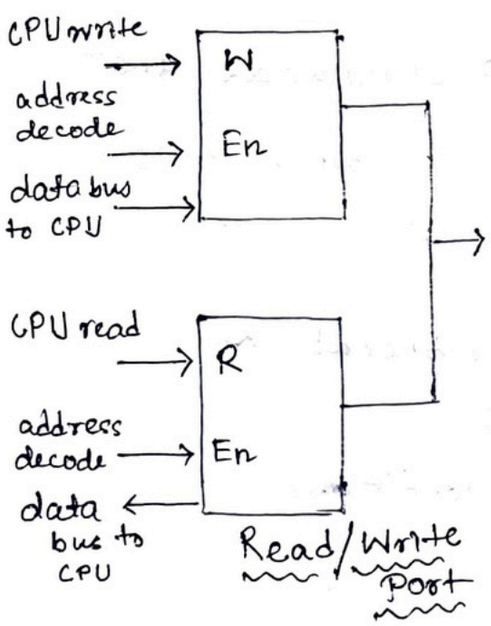
→ I/O Port: Device that looks like a memory cell but contains connections to the outside world. I/O port uses a latch. When the CPU writes to the address associated with the latch, the latch device captures the data & makes it available on a set of wires external to the CPU & memory system.

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



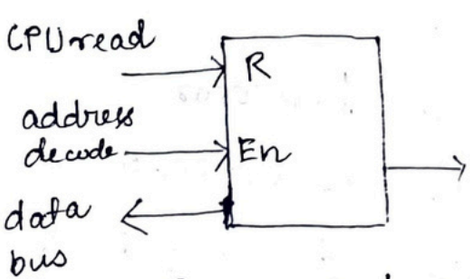
Write-only port

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.



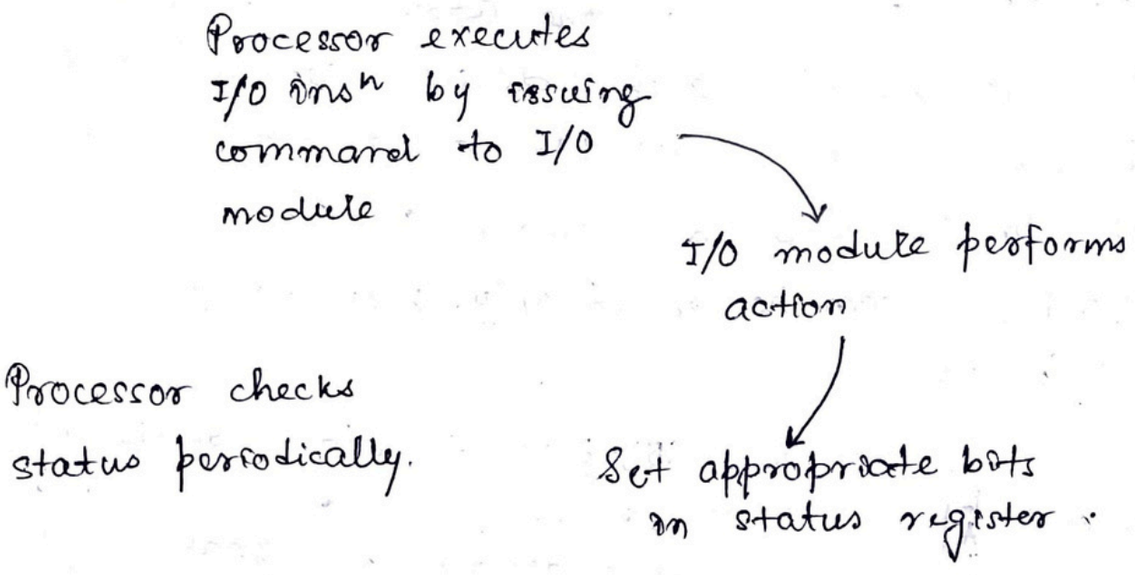
Read/Write Port

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.

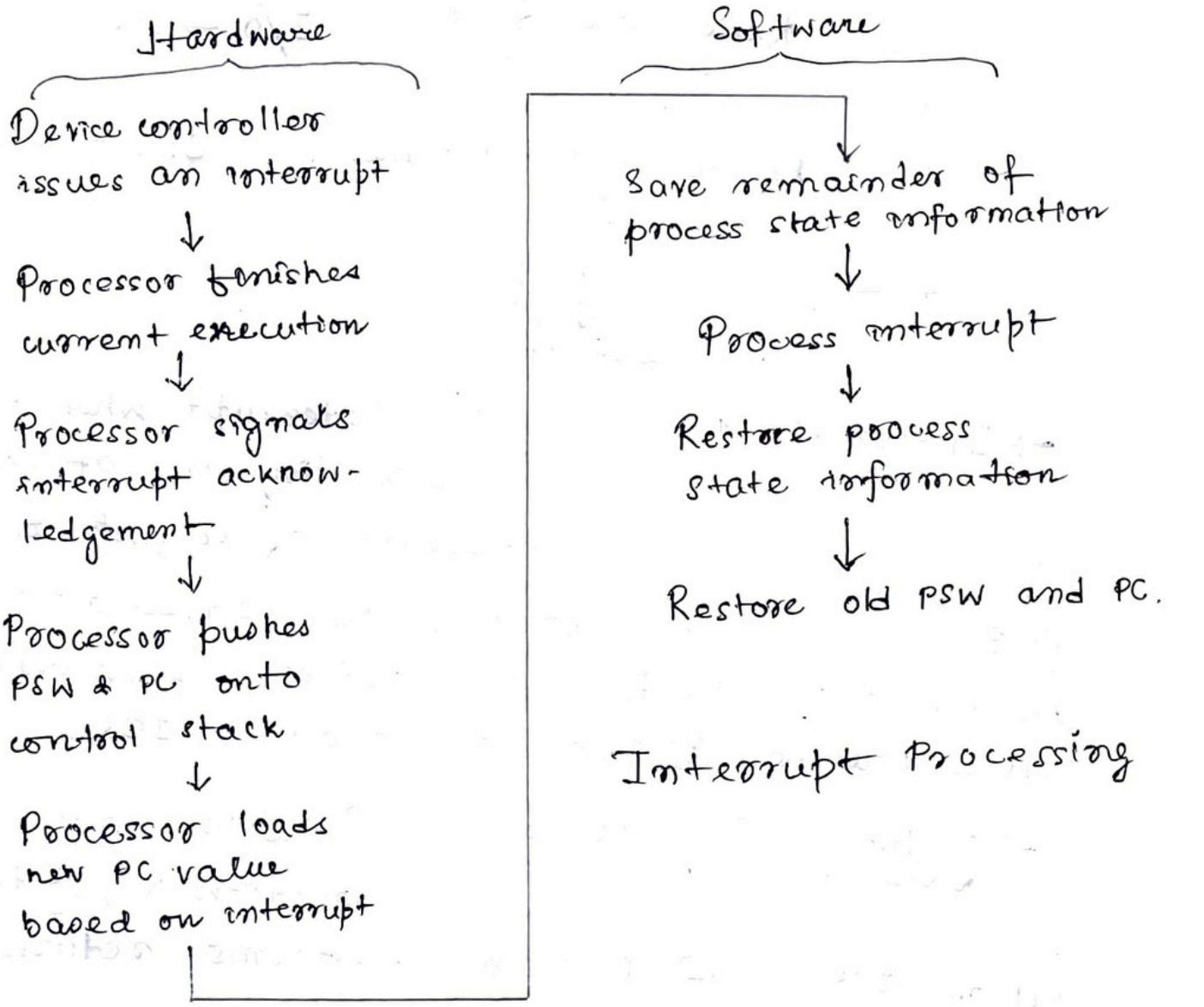
c) Read

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.

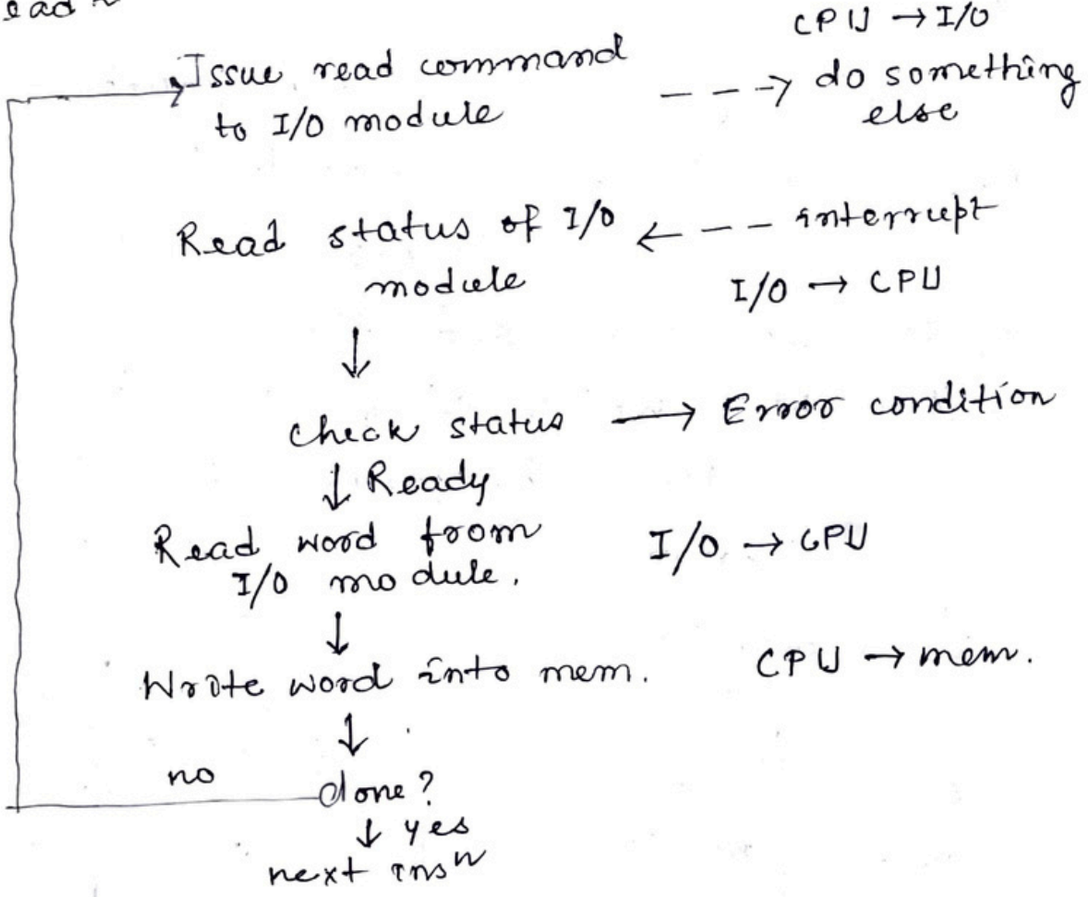


Interrupt Processing

→ 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.

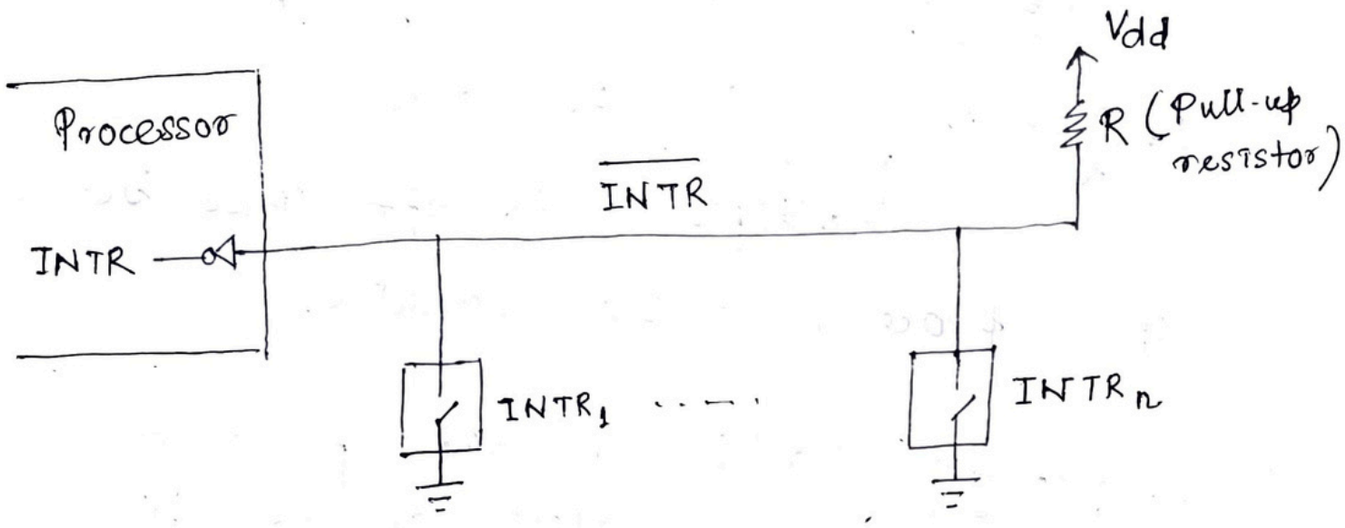
→ Return from interrupt : ① Return instn in location x, at the end of ISR. ② Processor performs return from ISR. ③ While returning, processor restores value of PC, PSW from control stack. ④ Processor starts executing user's interrupted program.

→ 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).

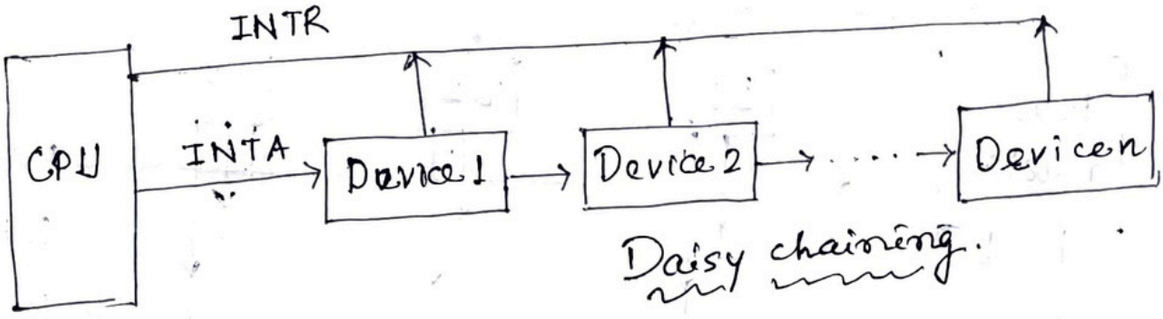
→ Device identification techniques.

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 ~~refer~~ referred to as a vector & is either the address of the I/O module or some other unique identification. (Vectored interrupt).



Daisy chaining.

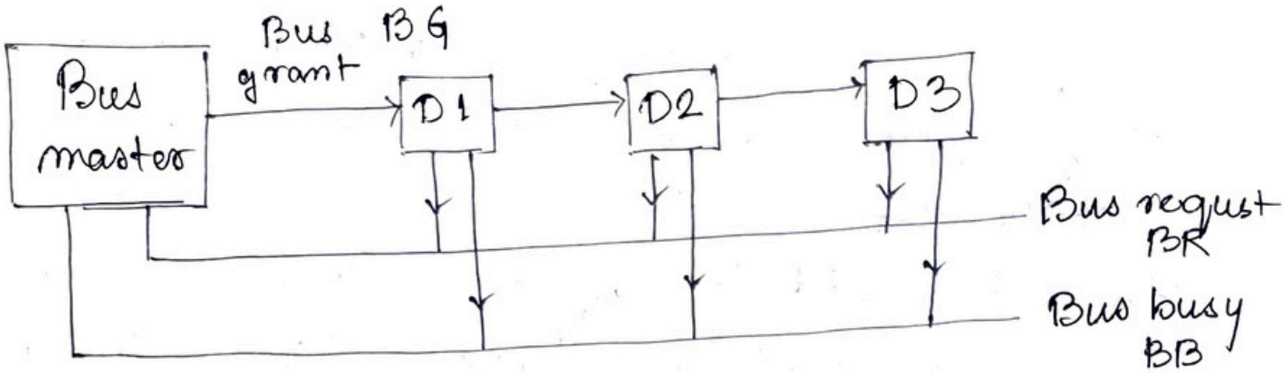
4) Bus Arbitration: I/O module must first gain control of the bus before it can raise the interrupt request lines. Arbitration refers to the process by which the current bus master (controller that has access to a bus at an instance) accesses & then leaves the control of the bus & passes it to the another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

There are 2 approaches to bus arbitration-

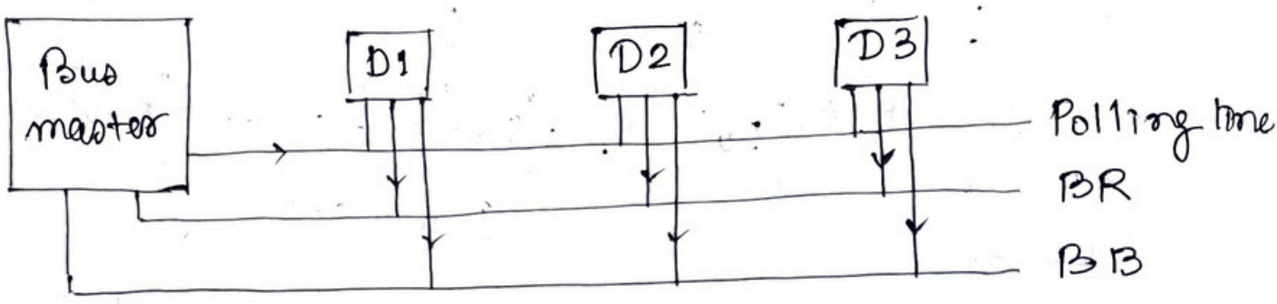
a) Centralised: A single bus arbiter performs required arbitration. Three different schemes that use this approach are -

i) Daisy Chaining:

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

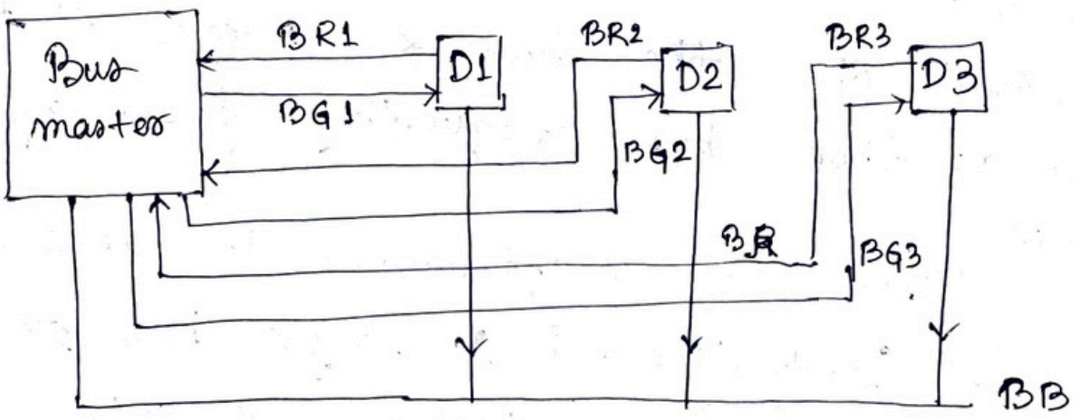


ii) Polling:



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:

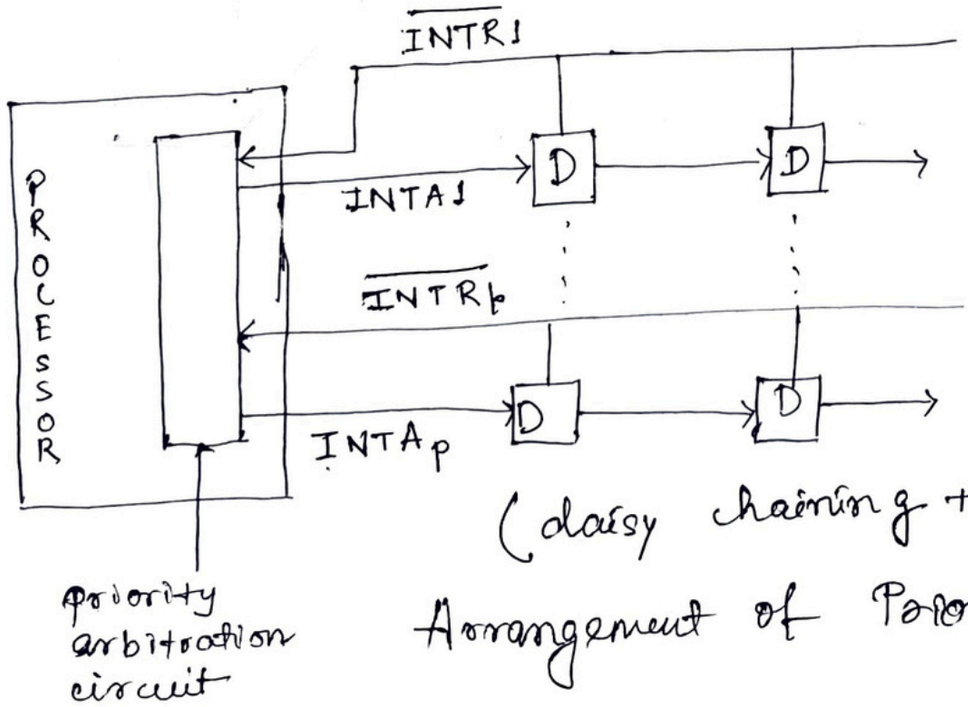
i) With multiple lines, the processor picks the interrupt line with highest priority.

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 in the daisy chain.

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

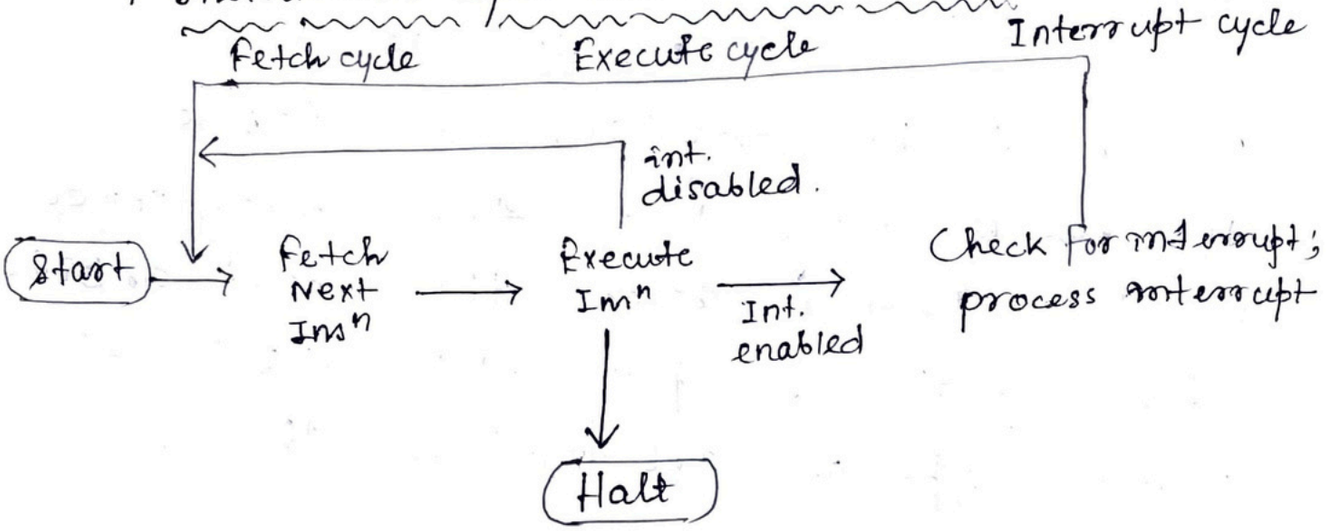
→ Combining identification techniques:



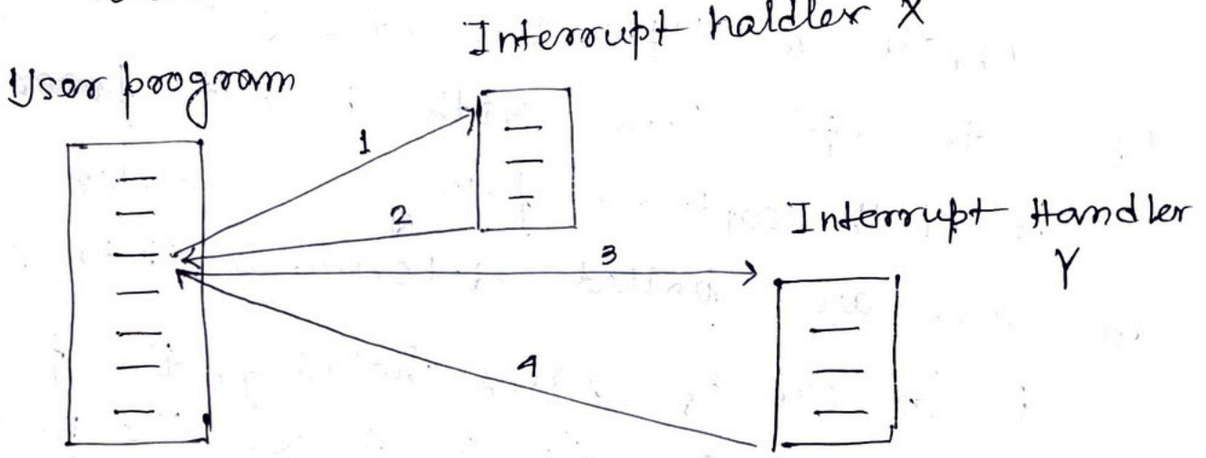
(daisy chaining + polling)

Arrangement of Priority groups.

→ Instruction cycle with interrupts:



→ Sequential Interrupt Processing:



→ Nested Interrupt Processing: