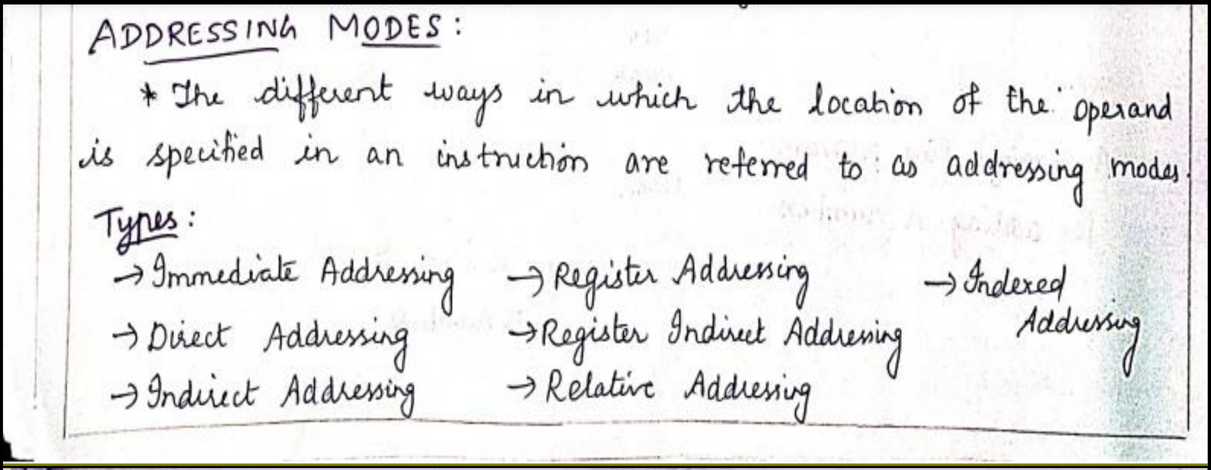
UNIT-1:

ADDRESSING MODES:

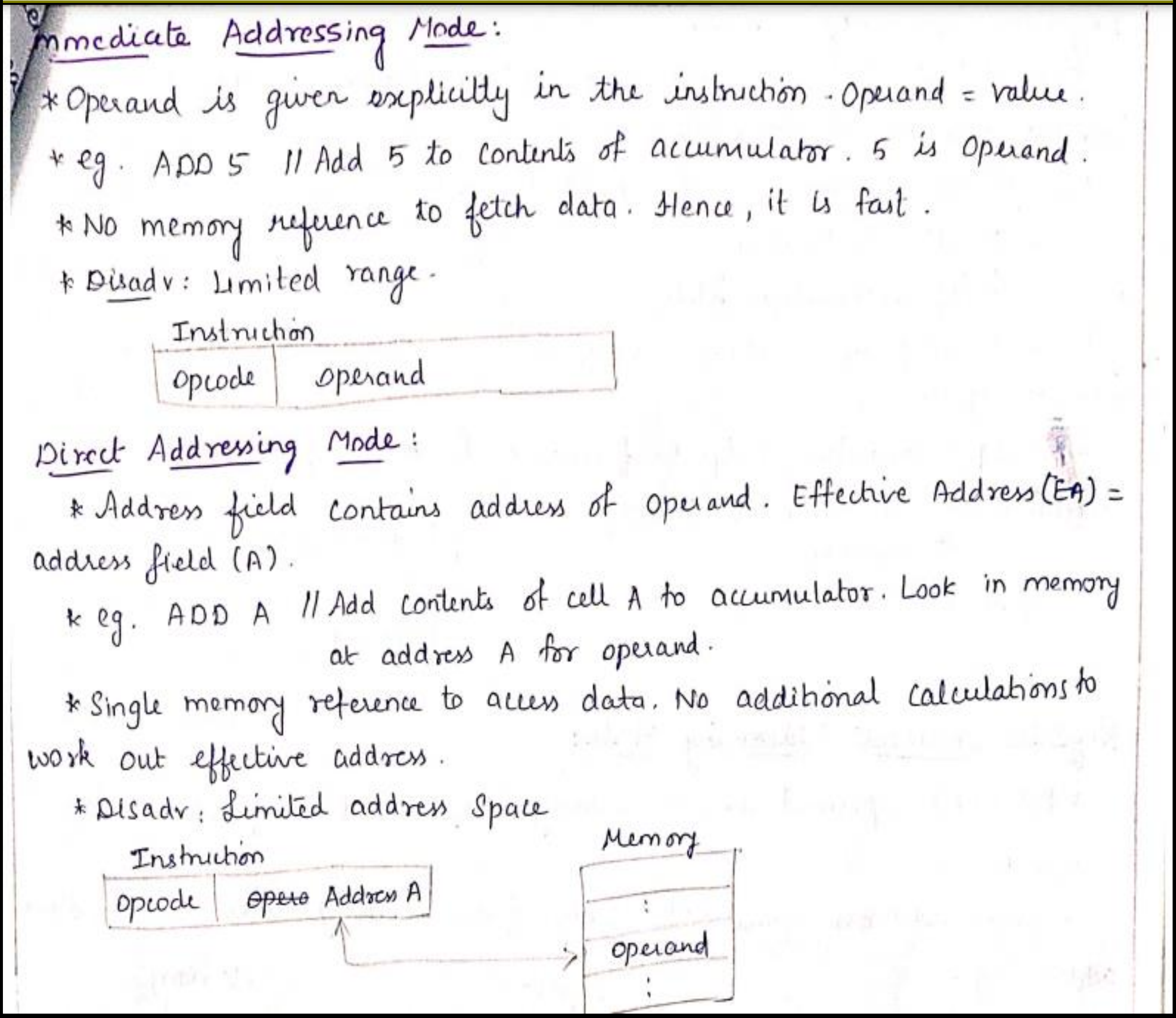


ADDRESSING MODES :

* The different ways in which the location of the operand is specified in an instruction are referred to as addressing modes.

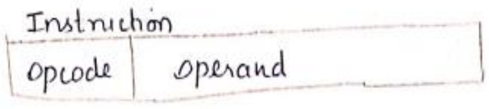
Types :

→ Immediate Addressing
→ Direct Addressing
→ Indirect Addressing
→ Register Addressing
→ Register Indirect Addressing
→ Relative Addressing
→ Indexed Addressing



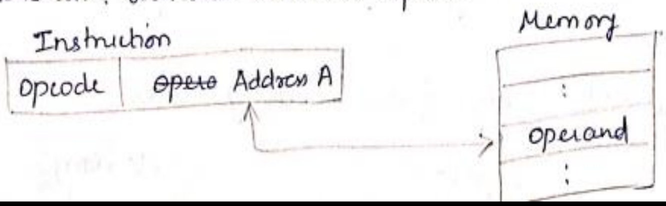
Immediate Addressing Mode :

* Operand is given explicitly in the instruction. Operand = value.
* eg. ADD 5 // Add 5 to contents of accumulator. 5 is operand.
* No memory reference to fetch data. Hence, it is fast.
* Disadv : Limited range.



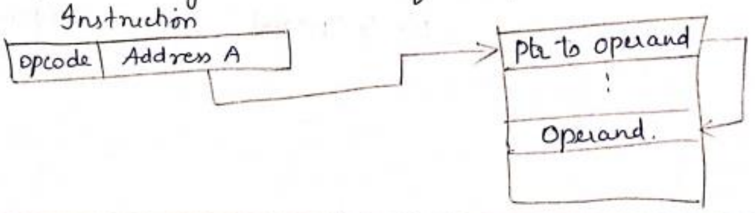
Direct Addressing Mode :

* Address field contains address of operand. Effective Address (EA) = address field (A).
* eg. ADD A // Add contents of cell A to accumulator. Look in memory at address A for operand.
* Single memory reference to access data. No additional calculations to work out effective address.
* Disadv : Limited address space



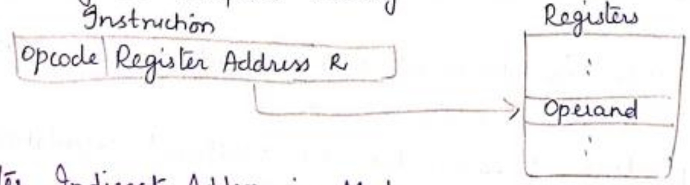
Indirect Addressing Mode:

* Memory cell pointed to by address field contains the address of (pointer to) the operand.
* EA = [A]. Look in A, find address of A and look there for operand.
* eg. ADD (A) // Add contents of cell pointed to by contents of A to accumulator.
* Large address space.
  2^n where n = word length
* It can be nested, multilevel, cascaded. eg. EA = (((A)))
* Multiple memory accesses to find operand. Hence it is slower.



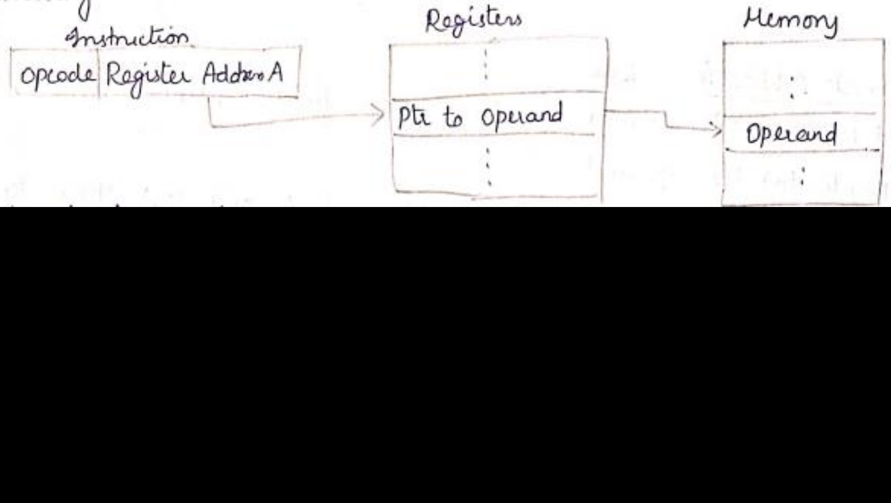
Register Addressing Mode:

* Operand is held in register named in address field. EA = R. Limited number of registers.
* Very small address field needed.
  - shorter instructions
  - faster instruction fetch
* No memory access. Hence, very fast execution. Very limited address space.
* Multiple registers help performance. Requires good assembly programming or compiler writing.



Register Indirect Addressing Mode:

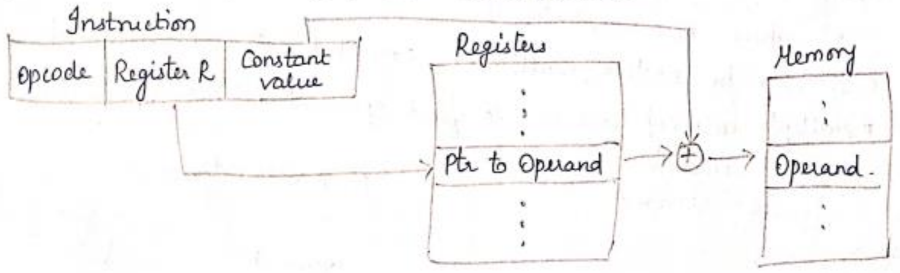
* EA = [R]. Operand is in memory cell pointed to by contents of register R.
* Large address space (2^n). Only fewer memory access than indirect addressing.



Indexed Addressing Mode:

* EA = X+[R]. Address field hold 2 values. X - constant value. It represents offset. R- register that holds address of memory location.

eg. Add 20(R1), R_2 (or) Add 1000(R1), R_2



lative Addressing Mode:

* A version of displacement addressing.

EA = X + (PC) (ie) get operand from X bytes away from current location pointed to by PC. Locality of reference and cache usage.

Auto increment Mode:

* The effective address of the operand is the contents of a register specified in the instruction.

* After accessing the operand, the contents of this register are automatically incremented to point to next item in the list.

* EA = [Ri]; Increment Ri — (Ri)+.

eg. Add (R2)+, R0.

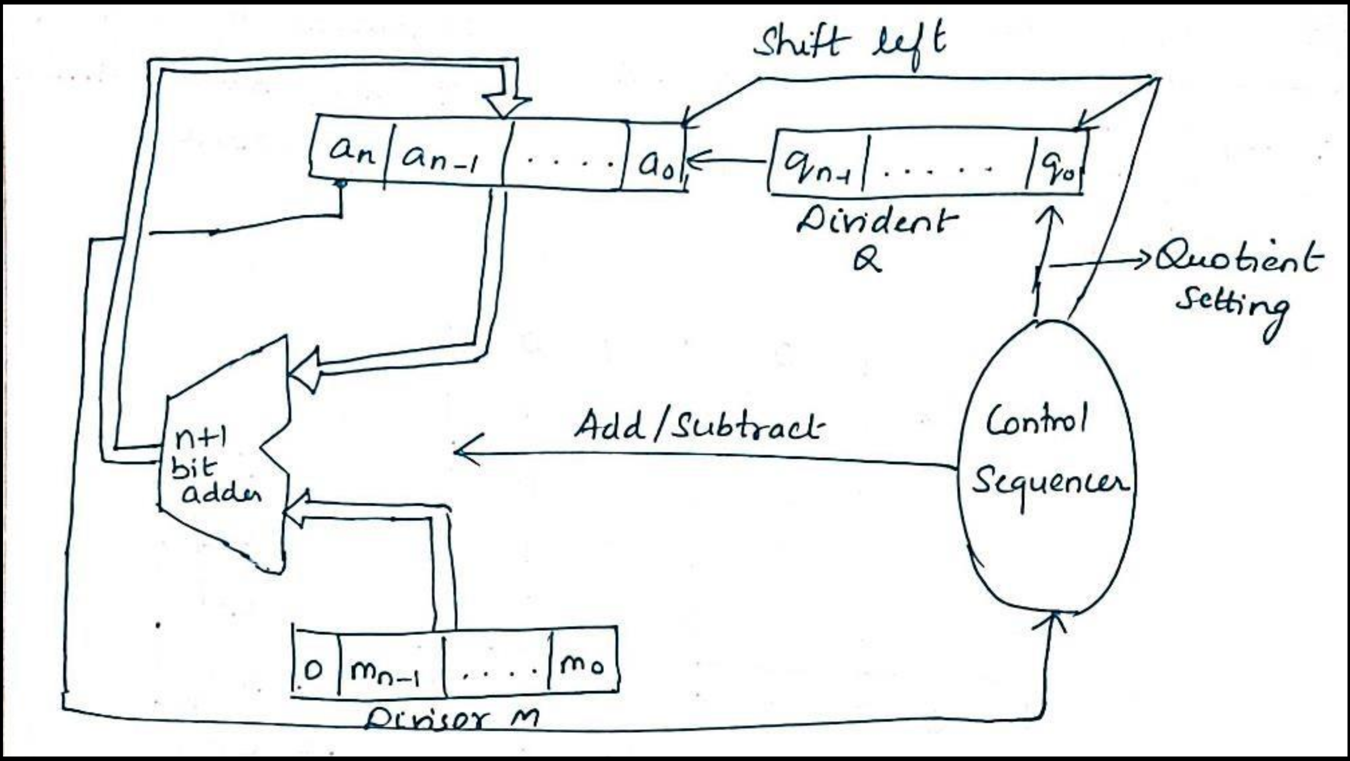
Auto Decrement mode:

* The contents of a register specified in the instruction are first automatically decremented and then used as the effective address of the operand.

eg. Add (R2) —, R0.

UNIT-2:-

RESTORING DIVISION:-



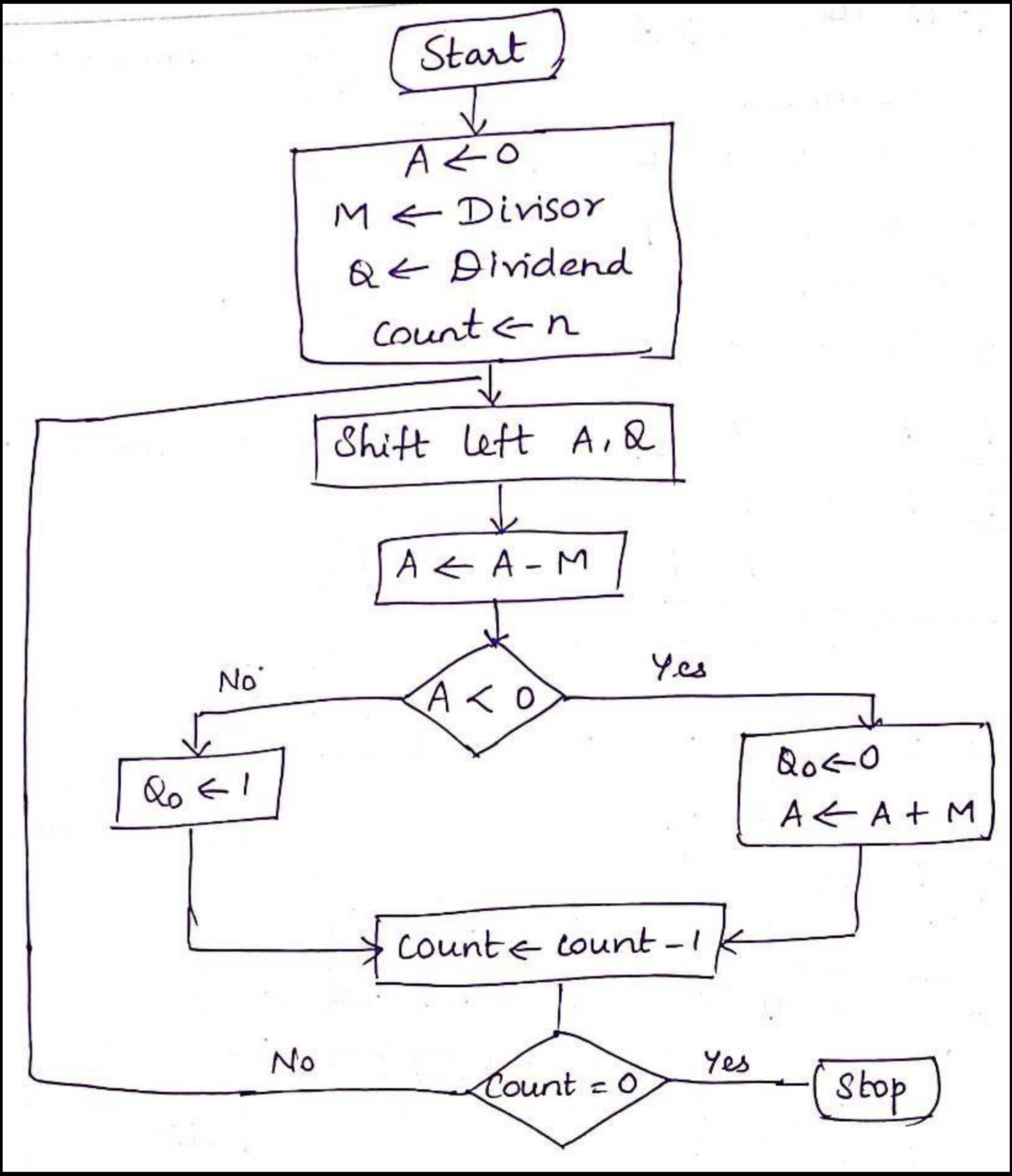
- The figure shows the logic circuit arrangement that implements restoring division.
- An n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at start. Register A is set to zero.
- The extra bit position at the left end of both A & M accommodates the sign bit during subtractions.
- At the end, the n-bit quotient is in register Q and the remainder is in register A.

ALGORITHM:-

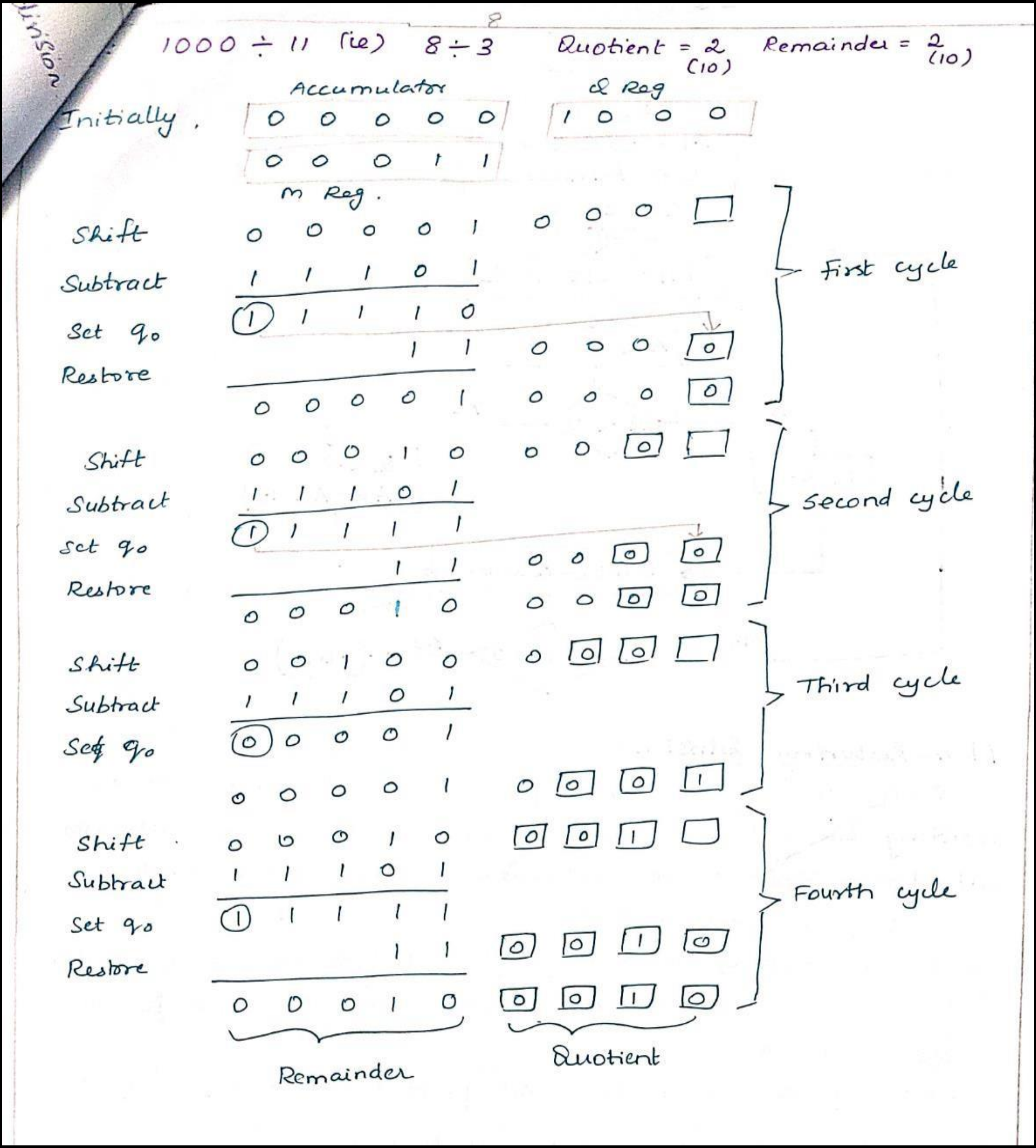
Do the following "n" times:

1. Shift A and Q left one binary position.
2. Subtract M from A, place the answer back in A.
3. If the sign of A is 1, set Q_0 to 0 and add M back to A (i.e., restore A); Otherwise, set Q_0 to 1.

FLOWCHART:-


Start
A ← 0
M ← Divisor
Q ← Dividend
count ← n
Shift left A, Q
A ← A - M
A < 0
No
Yes
Q0 ← 1
Q0 ← 0
A ← A + M
count ← count - 1
Count = 0
No
Yes
Stop

EXAMPLE PROGRAM:-



1000 ÷ 11 (ie) 8 ÷ 3   Quotient = $2_{(10)}$ Remainder = $2_{(10)}$

	Accumulator	Q Reg	
Initially	0 0 0 0 0	1 0 0 0	
	0 0 0 1 1		
	M Reg		
Shift	0 0 0 0 1	0 0 0 $\boxed{}$	First cycle
Subtract	1 1 1 0 1		
Set q_0	$\textcircled{1}$ 1 1 1 0		
	1 1	0 0 0 $\boxed{0}$	
Restore	0 0 0 0 1	0 0 0 $\boxed{0}$	
Shift	0 0 0 1 0	0 0 $\boxed{0}$ $\boxed{}$	Second cycle
Subtract	1 1 1 0 1		
Set q_0	$\textcircled{1}$ 1 1 1 1		
	1 1	0 0 $\boxed{0}$ $\boxed{0}$	
Restore	0 0 0 1 0	0 0 $\boxed{0}$ $\boxed{0}$	
Shift	0 0 1 0 0	0 $\boxed{0}$ $\boxed{0}$ $\boxed{}$	Third cycle
Subtract	1 1 1 0 1		
Set q_0	$\textcircled{0}$ 0 0 0 1		
	0 0 0 0 1	0 $\boxed{0}$ $\boxed{0}$ $\boxed{1}$	
Shift	0 0 0 1 0	$\boxed{0}$ $\boxed{0}$ $\boxed{1}$ $\boxed{}$	Fourth cycle
Subtract	1 1 1 0 1		
Set q_0	$\textcircled{1}$ 1 1 1 1		
	1 1	$\boxed{0}$ $\boxed{0}$ $\boxed{1}$ $\boxed{0}$	
Restore	0 0 0 1 0	$\boxed{0}$ $\boxed{0}$ $\boxed{1}$ $\boxed{0}$	
	Remainder	Quotient	

NON-RESTORING DIVISON:-

The restoring division algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction. Subtraction issaid to be unsuccessful if the result is negative.

ALGORITHM:-

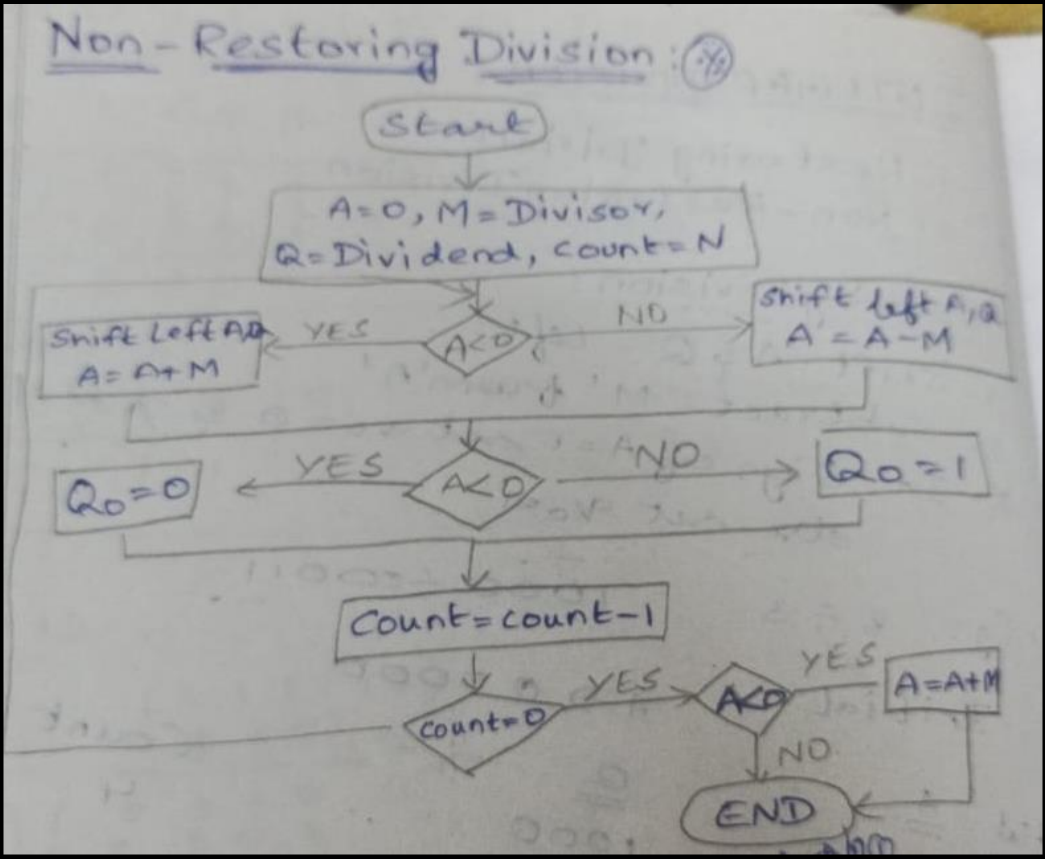
Step 1 - Do the following "n" times:

i. If the sign of A is 0, shift A and Q left one bit position and subtract Mfrom A; Otherwise, shift A and Q left and add M to A.
ii. Now, if sign of A is 0, set Q_0 to 1; Otherwise,

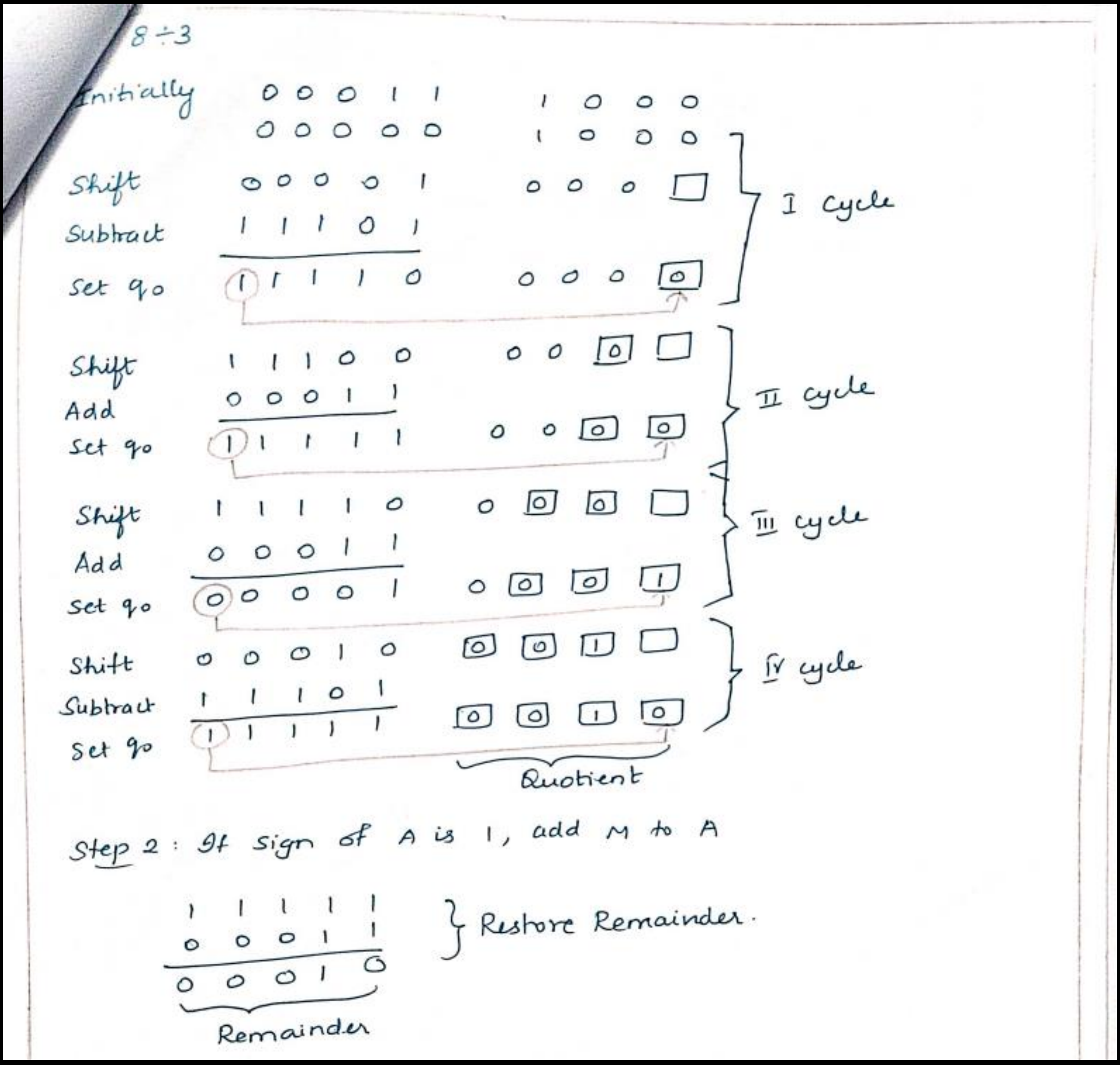
set Q_0 to 0.Step 2 - If the sign of A is 1, add M

to A.

FLOWCHART:-



EXAMPLE PROGRAM:-



8 ÷ 3

Initially	0 0 0 1 1	1 0 0 0	
	0 0 0 0 0	1 0 0 0	I cycle
Shift	0 0 0 0 1	0 0 0 $\boxed{}$	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0	0 0 0 $\boxed{0}$	
Shift	1 1 1 0 0	0 0 $\boxed{0}$ $\boxed{}$	II cycle
Add	0 0 0 1 1		
Set q_0	1 1 1 1 1	0 0 $\boxed{0}$ $\boxed{0}$	
Shift	1 1 1 1 0	0 $\boxed{0}$ $\boxed{0}$ $\boxed{}$	III cycle
Add	0 0 0 1 1		
Set q_0	0 0 0 0 1	0 $\boxed{0}$ $\boxed{0}$ $\boxed{1}$	
Shift	0 0 0 1 0	$\boxed{0}$ $\boxed{0}$ $\boxed{1}$ $\boxed{}$	IV cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1	$\underbrace{\boxed{0}\ \boxed{0}\ \boxed{1}\ \boxed{0}}_{\text{Quotient}}$	

Step 2 : If sign of A is 1, add M to A

$$\begin{array}{r} 1\ 1\ 1\ 1\ 1 \\ 0\ 0\ 0\ 1\ 1 \\ \hline \underbrace{0\ 0\ 0\ 1\ 0}_{\text{Remainder}} \end{array} \Bigg\}\ \text{Restore Remainder.}$$

UNIT-3:-

PIPELINING CONCEPT:-

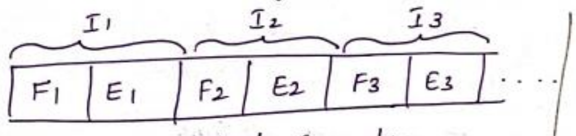
LINING - BASIC CONCEPT:

* The speed of execution of programs is influenced by many factors. One way is to arrange the hardware so that more than one operation can be performed at the same time.

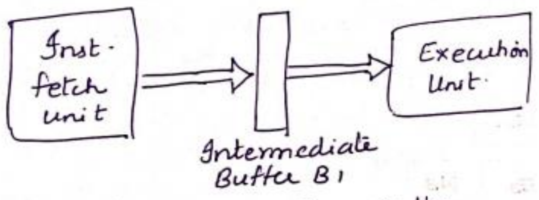
* Pipelining is an effective way of organizing concurrent activity in a computer system.

* The processor executes a program by fetching and executing the instructions, one after the other. Let F_i and E_i refer to the fetch and execute steps for instruction I_i.

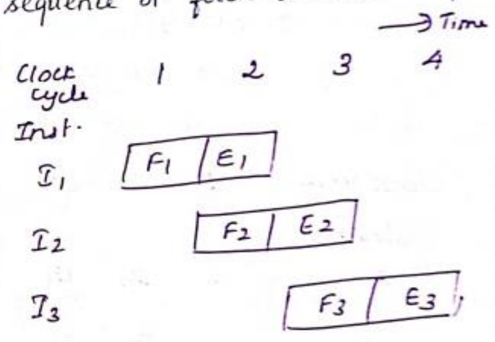
* Execution of program consists of a sequence of fetch & execute step.



Sequential Execution.



Hardware Organization



Pipelined Execution.

* Assume, computer has 2 seperate hardware units, one for fetching instructions and another for executing them.

* The instruction fetched by fetch unit is deposited in an intermediate storage buffer B1. This buffer enables execution unit to execute the instruction while the fetch unit is fetching the next instruction.

* The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle.

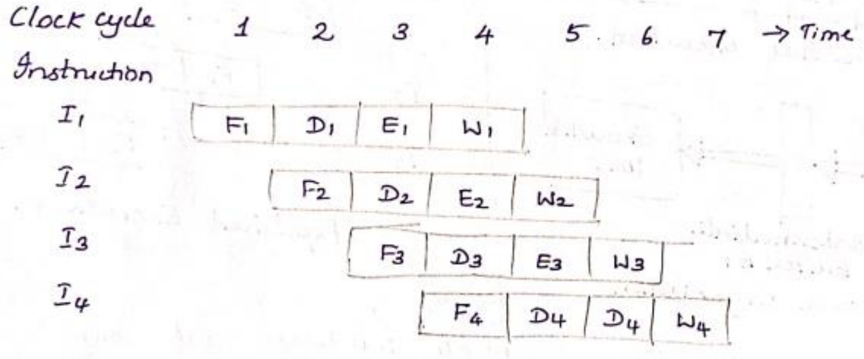
* In 1st clock cycle, fetch unit fetches I1 and stores it in buffer B1 at the end of the clock cycle.

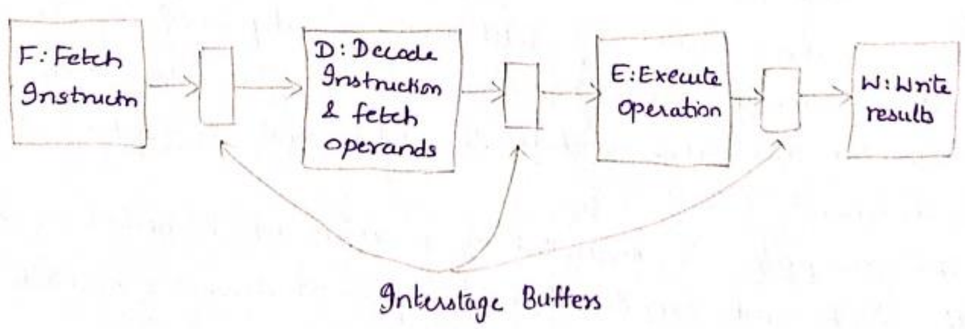
* In 2nd clock cycle, the fetch unit proceeds with the fetch
Meanwhile, execution unit executes I_1 and is completed. I_2
stored in B_1 replacing I_1.

* The fetch and execute unit constitutes a 2-stage pipeline in the
which each stage performs one step in processing an instruction.

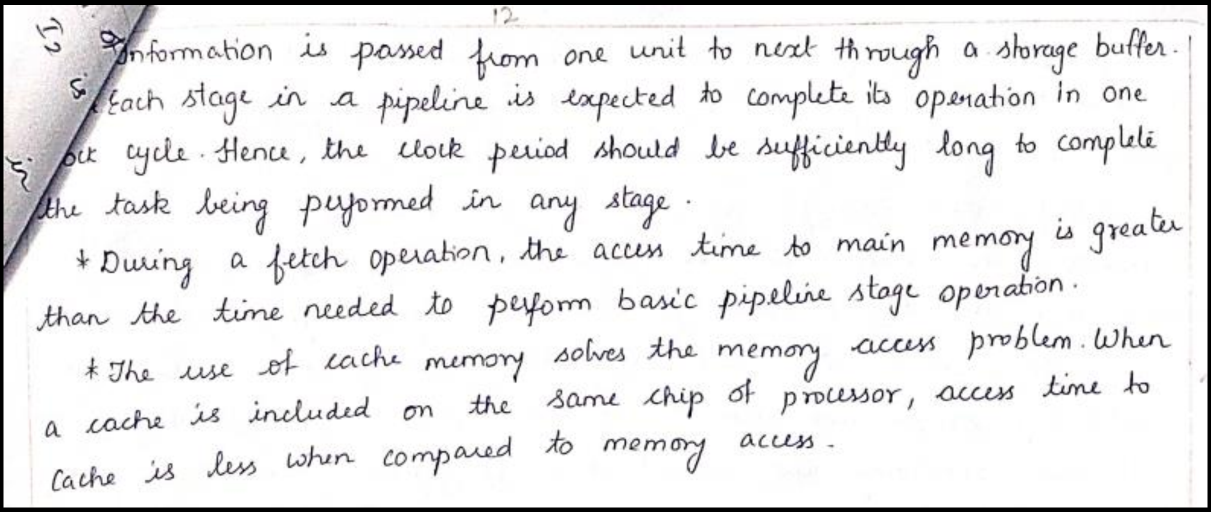
* The processing of an instruction need not be divided into only 2 steps. For eg. a pipelined processor may process each instruction in 4 steps as follows:

F [Fetch] : Read the instruction from the memory.
D [Decode] : Decode the instruction and fetch the source operand(s).
E [Execute] : Perform the operation specified by the instruction.
W [Write] : Store the result in the destination location.





* Four instructions are in progress at any given time. Hence, four distinct hardware units are needed, capable of performing their tasks simultaneously and without interfering with one another.



12

Information is passed from one unit to next through a storage buffer.
Each stage in a pipeline is expected to complete its operation in one clk cycle. Hence, the clock period should be sufficiently long to complete the task being performed in any stage.

* During a fetch operation, the access time to main memory is greater than the time needed to perform basic pipeline stage operation.

* The use of cache memory solves the memory access problem. When a cache is included on the same chip of processor, access time to cache is less when compared to memory access.

PIPELINE CONFLICTS:-

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. Timing Variations : All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards : When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching : In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts : Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency : It arises when an instruction depends upon the result of a previous instruction, but this result is not yet available.

UNIT-4:-

FLYNN'S CLASSIFICATION:-

FLYNN'S CLASSIFICATION:

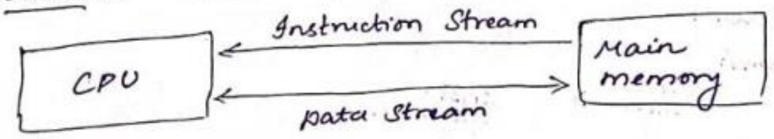
* Proposed by Michael Flynn in 1972. Introduced the concept of instruction and data streams for categorizing of computers. This classification is based on instruction and data streams.

* Instruction Cycle:

* The instruction cycle consists of a sequence of steps needed for the execution of an instruction in a program.

* The control unit fetches instructions one at a time. The fetched instruction is then decoded by the decoder. The processor executes the decoded instructions. The result of execution is temporarily stored in Memory Buffer Register (MBR).

* Instruction Stream and Data Stream.



* The flow of instructions is called instruction stream. The flow of operands between processor and memory is bi-directional. This flow of operands is called data stream.

*Flynn's Classification is done based on multiplicity of instruction streams and data streams observed by the CPU du program execution.

1) Single Instruction and Single Data Stream (SISD)
2) Single Instruction and multiple Data Stream (SIMD)
3) Multiple Instruction and Single Data Stream (MISD)
4) Multiple Instruction and Multiple Data Stream (MIMD).

← Prev Next →

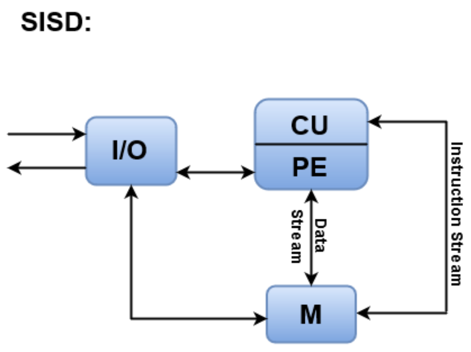
SISD

SISD stands for ***'Single Instruction and Single Data Stream'***. It represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

Instructions are executed sequentially, and the system may or may not have internal parallel processing capabilities.

Most conventional computers have SISD architecture like the traditional Von-Neumann computers.

Parallel processing, in this case, may be achieved by means of multiple functional units or by pipeline processing.



Where, CU = Control Unit, PE = Processing Element, M = Memory

Instructions are decoded by the Control Unit and then the Control Unit sends the instructions to the processing units for execution.

Data Stream flows between the processors and memory bi-directionally.

Examples:

Older generation computers, minicomputers, and workstations

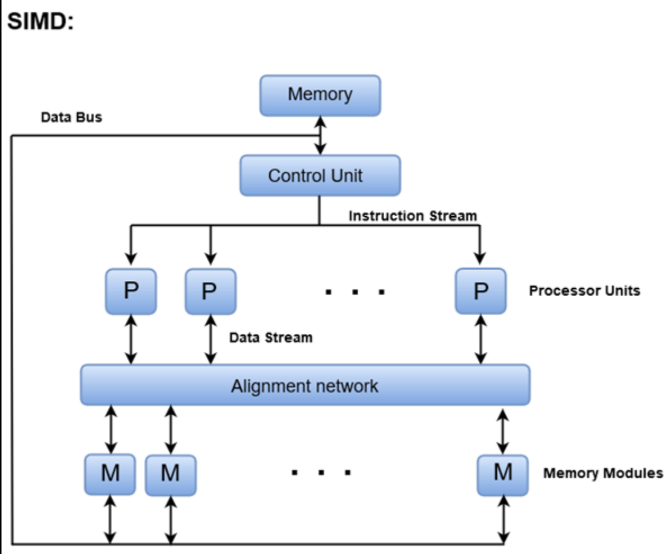
← Prev Next →

SIMD

SIMD stands for ***'Single Instruction and Multiple Data Stream'***. It represents an organization that includes many processing units under the supervision of a common control unit.

All processors receive the same instruction from the control unit but operate on different items of data.

The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.



SIMD is mainly dedicated to array processing machines. However, vector processors can also be seen as a part of this group.

MISD

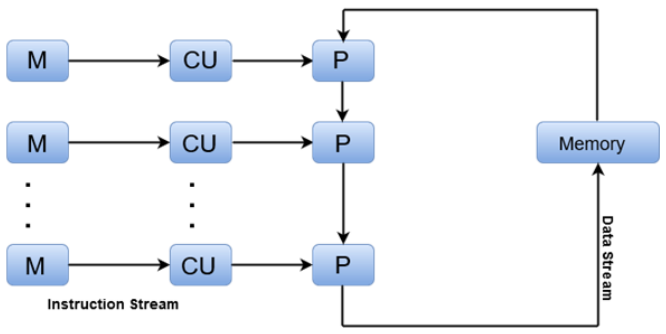
← Prev Next →

MISD stands for ***'Multiple Instruction and Single Data stream'***.

MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

In MISD, multiple processing units operate on one single-data stream. Each processing unit operates on the data independently via separate instruction stream.

MISD:



Where, M = Memory Modules, CU = Control Unit, P = Processor Units

Example:

The experimental Carnegie-Mellon C.mmp computer (1971)

MIMD

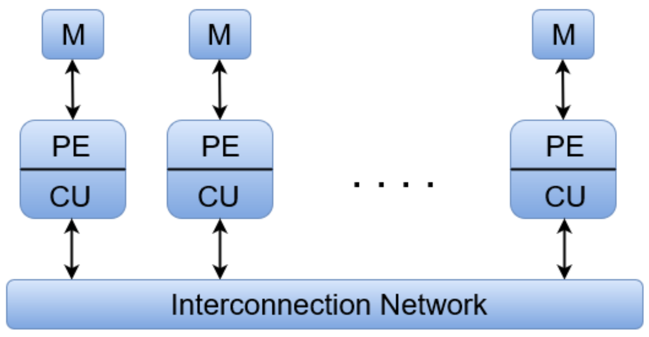
← Prev

MIMD stands for ***'Multiple Instruction and Multiple Data Stream'***.

In this organization, all processors in a parallel computer can execute different instructions and operate on various data at the same time.

In MIMD, each processor has a separate program and an instruction stream is generated from each program.

MIMD:

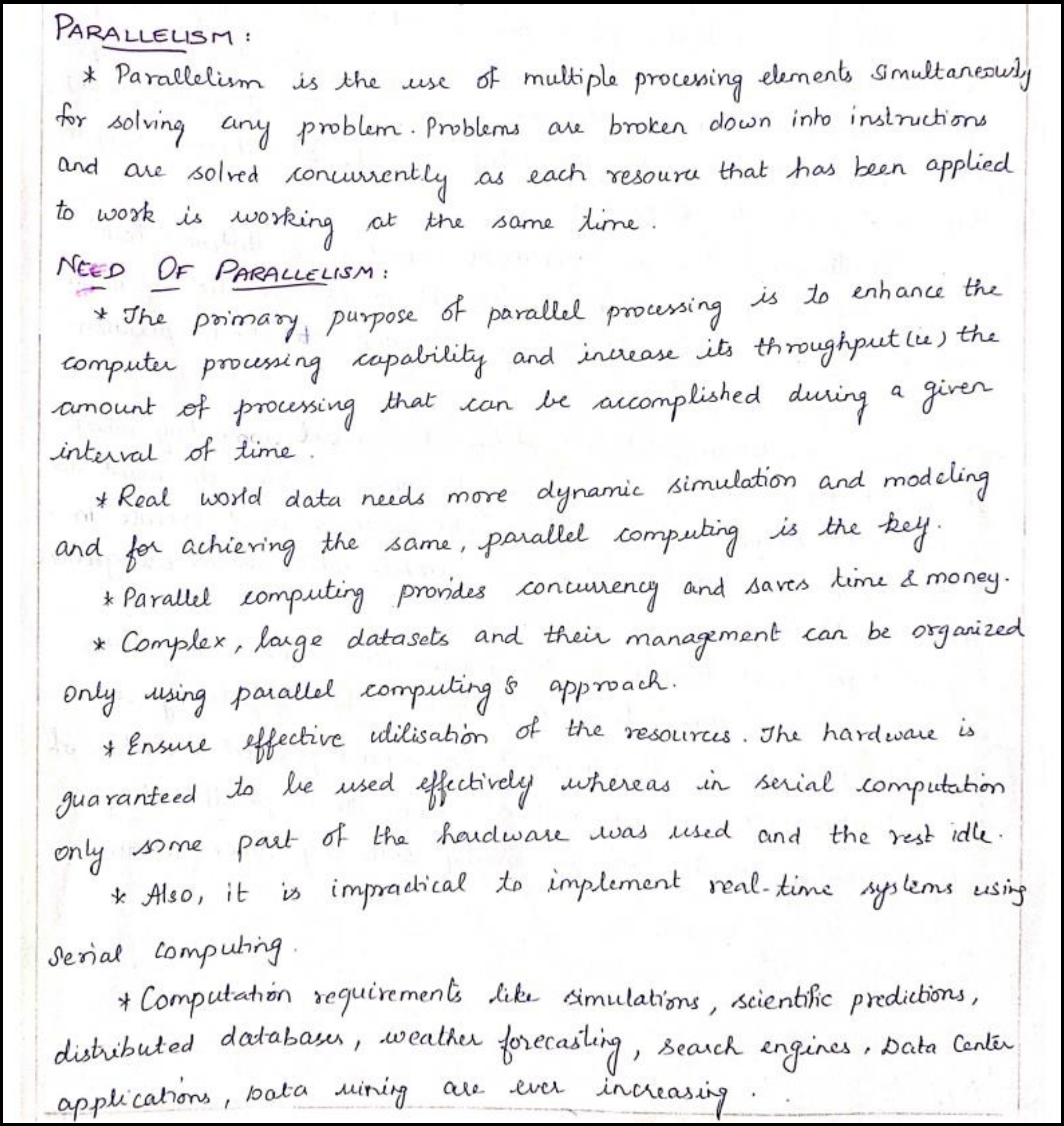


Where, M = Memory Module, PE = Processing Element, and CU = Control Unit

Examples:

Cray T90, Cray T3E, IBM-SP2

PARALLELISM:-



PARALLELISM :

* Parallelism is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource that has been applied to work is working at the same time.

NEED OF PARALLELISM :

* The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput (ie) the amount of processing that can be accomplished during a given interval of time.

* Real world data needs more dynamic simulation and modeling and for achieving the same, parallel computing is the key.

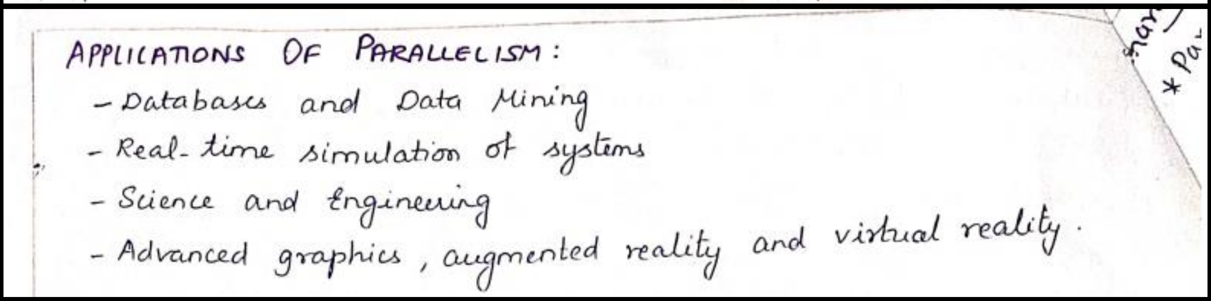
* Parallel computing provides concurrency and saves time & money.

* Complex, large datasets and their management can be organized only using parallel computing & approach.

* Ensure effective utilisation of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of the hardware was used and the rest idle.

* Also, it is impractical to implement real-time systems using serial computing.

* Computation requirements like simulations, scientific predictions, distributed databases, weather forecasting, search engines, Data Center applications, Data mining are ever increasing.



APPLICATIONS OF PARALLELISM :

- Databases and Data Mining
- Real-time simulation of systems
- Science and Engineering
- Advanced graphics, augmented reality and virtual reality.

TYPES OF PARALLELISM:

1) Data Parallelism - It means concurrent execution of the same task on each multiple computing core. Eg. Summing the contents of an array of size N. For a single-core system, one thread would simply sum the elements [0].... [N-1]. In a dual-core system, thread A, running on one core 0, could sum the elements [0]...[N/2-1] and while thread B, running one core 1, could sum [N/2]...[N-1]. So 2 threads run in parallel on seperate computing cores.

2) Task Parallelism - It means concurrent execution of different task on multiple computing cores. An example of task parallelism might involve 2 threads, each performing a unique statistical operation on the array elements.

3) Bit-Level parallelism - It is a form of parallel computing which is based on increasing processor word size. Increasing the word size reduces the number of instructions the processor must execute in order to perform an operation on variables whose sizes are greater than word length.

4) Instruction Level Parallelism - It means simultaneous execution of multiple instructions from a program. While pipelining is a form of ILP, we must exploit it to achieve parallel execution of the instructions in the instruction stream. In a parallel loop, every iteration of the loop can overlap with any other iteration.

UNIT-5:-

VIRTUAL MEMORY:-

3 8 * Virtual memory is an architectural solution to increase the effective size of the memory system.

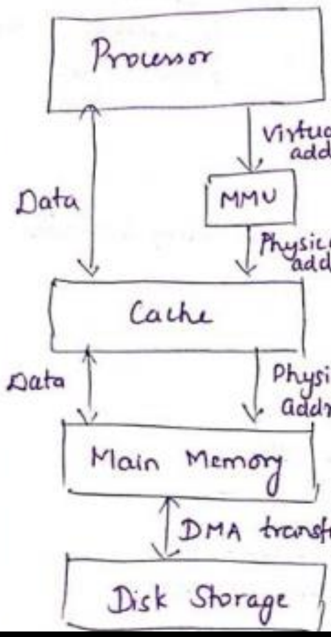
* Physical main memory typically ranges from few hundred megabytes to 1 GB bytes. Large programs that cannot fit completely into main memory have their parts stored on secondary storage devices such as magnetic disks.

* Operating system automatically transfers data between main memory and secondary storage. Application programmer need not be concerned with this transfer.

* Techniques that automatically move program and data between main memory and secondary storage when they are required for execution are called virtual memory techniques.

* Programs and processors reference an instruction or data independen of the size of the main memory. Processor issues binary addresses for instructions and data. These binary addresses are called logical or virtual addresses.

* Virtual addresses are translated into physical addresses by hardwa and software subsystem.



* Memory Management Unit (MMU) translates virtual addresses into physical addresses.

* If the desired data or instructions are in the main memory they are fetched.

* If the desired data or instructions are not in main memory, they must be transferred from secondary storage to main memory.

* MMU causes OS to bring the data from secondary storage into main memory.

Address Translation :

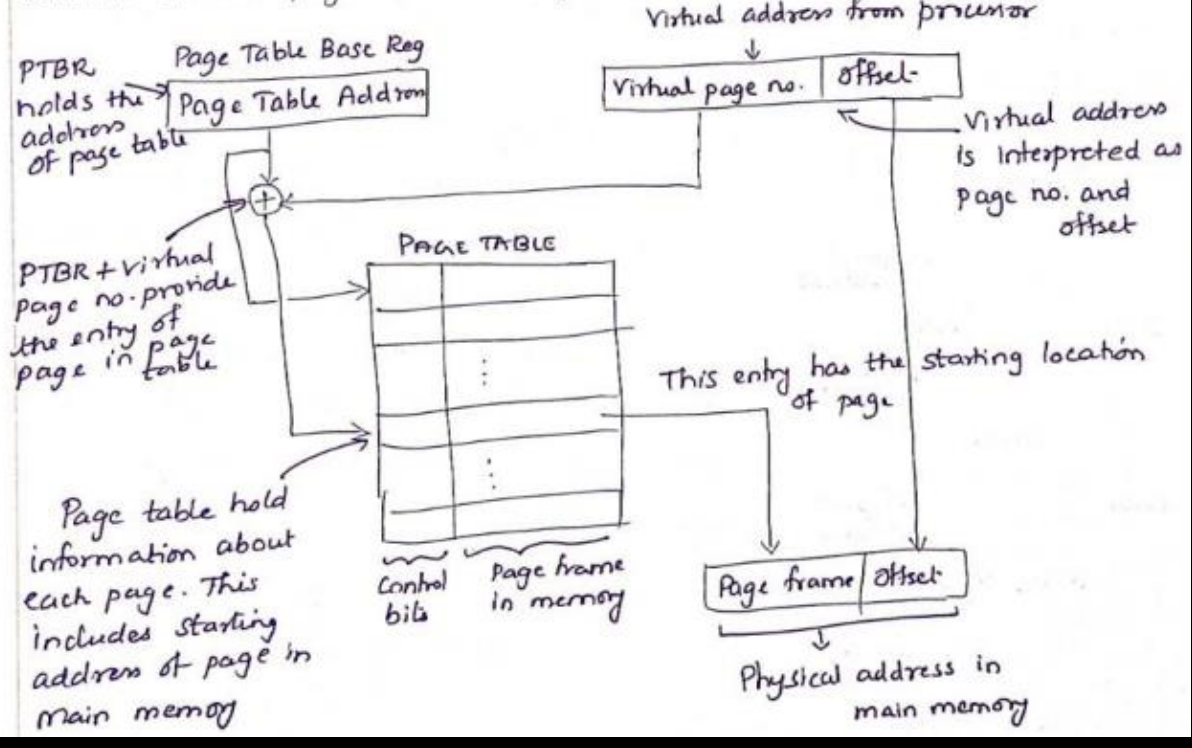
* Page is a basic unit of information that is transferred between secondary storage and main memory. A page consists of block of words that occupy contiguous locations in the main memory. Size of a page ranges from 2k to 16k bytes.

* Each virtual or logical address generated by processor is interprete as a virtual page no (high-order bits) plus an offset (low-order bits) tha specifies the location of a particular byte within that page.

* Information about the main memory location of each page is kep in the page table. Area of main memory that can hold a page is called as page frame. Starting address of page table is kept in page table base register.

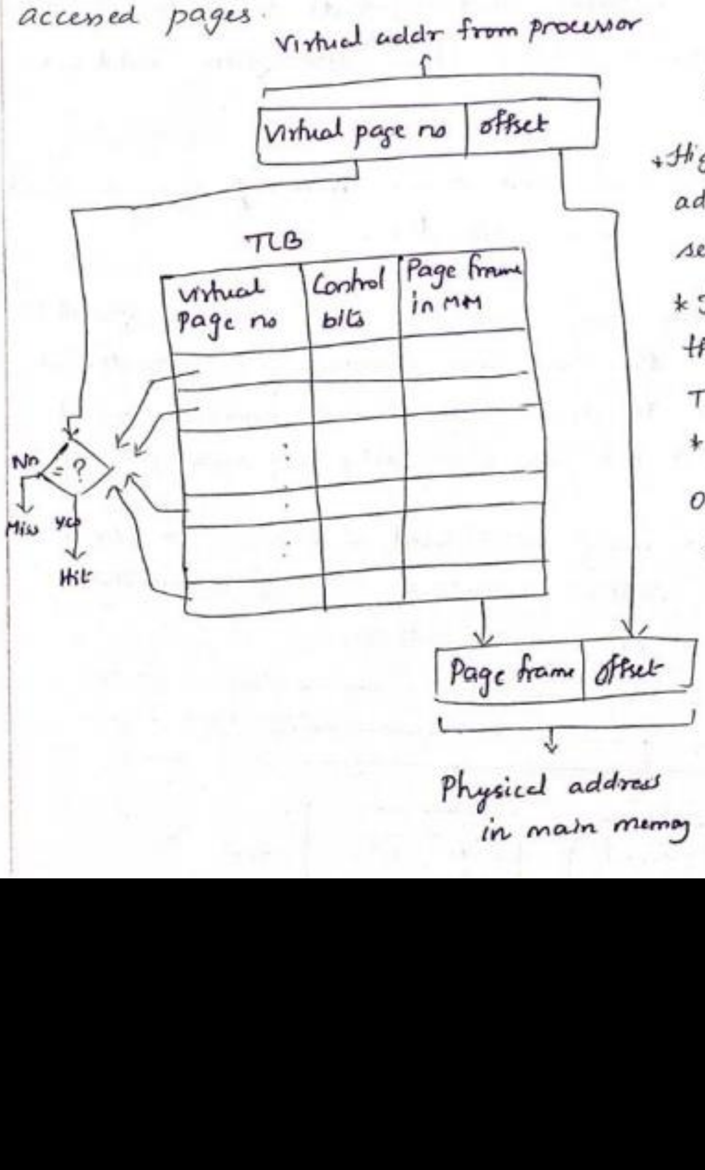
*Virtual page no. generated by the processor is added to contents of the PTBR. This provides the address of the corresponding entry in the page table.

* The contents of this location in the page table gives the starting address of the page if the page is currently in main memory.


Virtual address from processor
Page Table Base Reg
PTBR holds the address of page table
Page Table Address
Virtual page no.
offset
Virtual address is interpreted as page no. and offset
PTBR + virtual page no. provide the entry of the page in page table
PAGE TABLE
This entry has the starting location of page
Page table hold information about each page. This includes starting address of page in main memory
Control bits
Page frame in memory
Page frame
offset
Physical address in main memory

age table entry for a page also includes some control bits which cribe the status of page while it is in main memory.
- One bit indicates the validity of the page.
- One bit indicates whether the page has been modified during its residency in the main memory.

* The page table is used by MMU for every read/write access. Page table is quite large.
* MMU is implemented as part of the processor chip. Impossible to include a complete page table on the chip. Page table is kept in main memory. A copy of the small portion of page table can be accomodated within MMU.
* A small cache called as Translation Lookaside Buffer (TLB) is included in MMU. TLB holds page table entries of the most recently accessed pages.



Associative mapped TLB.

* High order bits of the virtual address generated by processor select the virtual page.
* These bits are compared to the virtual page numbers in TLB.
* If there is a match, a hit occurs and the corresponding address of page frame is read.
* If there is no match, a miss occurs and the page table within the main memory must be consulted.