CS8491 COMPUTER ARCHITECTURE

L	T	P	C
3	0	0	3

UNIT I BASIC STRUCTURE OF A COMPUTER SYSTEM 9

Functional Units – Basic Operational Concepts – Performance – Instructions: Language of the Computer – Operations, Operands – Instruction representation – Logical operations – decision making – MIPS Addressing.

UNIT II ARITHMETIC FOR COMPUTERS 9

Addition and Subtraction – Multiplication – Division – Floating Point Representation – Floating Point Operations – Subword Parallelism

UNIT III PROCESSOR AND CONTROL UNIT 9

A Basic MIPS implementation – Building a Datapath – Control Implementation Scheme – Pipelining – Pipelined datapath and control – Handling Data Hazards & Control Hazards – Exceptions.

UNIT IV PARALLELISIM 9

Parallel processing challenges – Flynn's classification – SISD, MIMD, SIMD, SPMD, and Vector Architectures - Hardware multithreading – Multi-core processors and other Shared Memory Multiprocessors - Introduction to Graphics Processing Units, Clusters, Warehouse Scale Computers and other Message-Passing Multiprocessors.

UNIT V MEMORY & I/O SYSTEMS 9

Memory Hierarchy - memory technologies – cache memory – measuring and improving cache performance – virtual memory, TLB's – Accessing I/O Devices – Interrupts – Direct Memory Access – Bus structure – Bus operation – Arbitration – Interface circuits - USB.

TOTAL : 45 PERIODS

UNIT 1
BASIC STRUCTURE OF A COMPUTER SYSTEM

Functional Units

- A computer consists of 5 functionally independent main parts.

  * Input
  * Memory
  * Arithmetic & logic
  * Output
  * Control.

I/P - The Input unit accepts coded information from human Operators, from electromechanical devices such as Keyboards.

ALU - The information received is either stored in the Computer's memory for later reference or

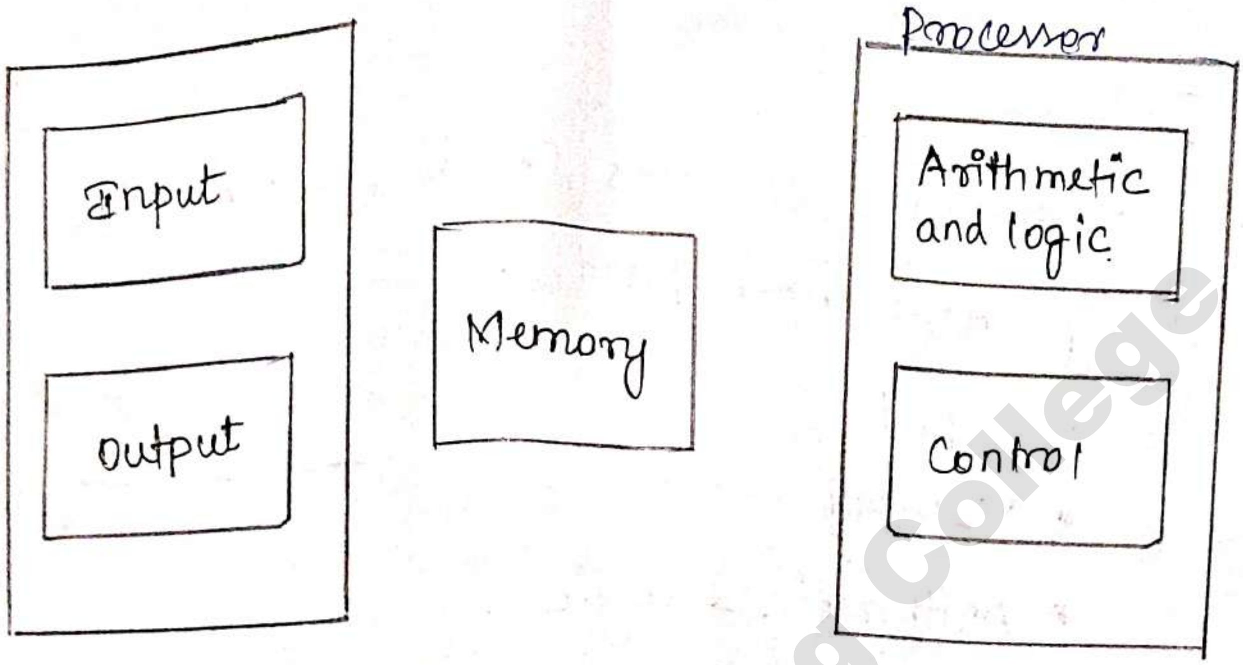
- Immediately stored in computer memory / by the ALU to perform desired Operation.

processor - The processing steps are determined by a program Stored in the ~~prog~~ Memory.

O/P - finally the result are sent through output Unit.

Ctrl unit - All of these actions are coordinated by the Control unit.

- ALU in conjunction with main control circuits & as the processor



- A list of Instructions that performs a task is called a program. Usually pgm stored in memory.

- The processor then fetches the instructions that make up the program from the memory. One after another & perform desired operation.

- The computer is completely controlled by stored Program.

- The Data are numbers & Encoded characters that are used as operands by the instructions.

- Data → Mean any digital infn.

- The Task of compiling a high-level language source program into a list of machine instructions Constitute a Machine language program called Object Program.

The source pgm is the Input data to compiler program which translates into Machine lang pgm.

- Each numbers, characters / Instructions is encoded as a string of binary digits called bits
- Bits - 0 or 1.
- Alphanumeric characters are expressed in terms of binary codes.
- Two schemes are ASCII - 7 bit code,
  EBCDIC - 8-bit to denote a character.
  (Extended Binary-coded Decimal Interchange code).

Input Unit :

- Computer acept coded information through input units, which reads the data.
- The Well-Known Input device is keyboard, when a key is pressed, the corresponding letter is automatically translated into corresponding binary code & transmitted over a cable to the Memory or processor.
- Other devices are Joysticks, trackballs & mouses
- Microphones used to capture audio input is then sampled & converted into digital codes for storage & processing.

Memory Unit :

- functions of the memory unit is to store progra. and data
- Two classes of storage
  - * primary
  - * Secondary.
- Primary storage is a fast memory that operates at Electronic speeds.
- programs must be stored in memory while they are being executed.
- Instructions & data can be written into the memory or read under the control of processor.
- It is possible to access any word location in the memory as Quickly.
- Memory in which any location can be reached in a short & fixed amount of time after specifying its address called RAM.
- The time required to access one word is called Memory access time.
- Small, fast RAM units called caches which are tightly coupled with processor.

* Secondary storage is used when large amount of data and many programs can be stored.

Some of the Secondary Storage devices are Magnetic disks, tapes & Optical disks (CD-ROMS).

Arithmetic & Logic Unit.

- ALU Operations such as Multiplication, division or comparisons of numbers
- ALU is Inititated by bringing the required operands into the Processor, where the operation is performed by ALU.
- Eg: Two number located in Memory are to be added.
  - numbers are brought into the Processor
  - Actual addition is carried by ALU
  - Sum stored in Memory or Retained in Processor for immediate Use.
- When Operands are brought into the processor, they are stored in high-Speed storage elements called registers.
- Each registers can store one word of data.
- The control & ALU are faster than other devices Connected to a computer which enables a Single processor to control a no. of External devices such as Keyboard, display, Magnetic & Optical disks.

Output Unit:

- It function is to send proccesed result to the outside world.
- Eg: Printer which employ Mechanical device impact heads, ink jet streams, to perform printing.
- Some units such as Graphic displays provide both I/O functions.

Control unit:

- The control unit is the nerve center that sends control signals to other Units & senses their states.
- The actual timing signals that governs the transfers are generated by the control circuits.
- Timing signals are signals that determine when a given action is to take place.
- Data transfers b/w the Processor & memory are controlled by control unit through timing signals
- A large set of control lines carries the signals used for timing & synchronisation of events in all Units.

* The operation of a computer are,

- Computer accepts infn in form of pgms & data through an Input unit & stores it in Memory.

Infn stored in the memory is fetched, under pgm ctrl into an ALU where it is processed.

- processed infn leaves the computer through an o/p unit.

- All activities inside the machine are directed by the control unit.

Basic Operational Concept

- To perform a given task, its appropriate program must be stored in a memory.
- Individual Instructions are brought from the memory into the processors which executes the specified Operations.
- Data to be used as operands are also stored in the memory.
- Eg: ADD LOCA, R_0
  - This instruction add the Operand at ML LOCA to the Operand in a register R_0 & place the sum or result into the register R_0.
  - Here the data of LOCA will be unchanged, whereas the value on Register R_0 will be overwritten.
- Transfers between the memory & processor are started by sending the address of the to be accessed to memory by issuing appropriate signals

Another Eg: Load LOCA, R1
Add R1, R0

- here content of ML LOCA is transferred into Processor Register R1.
- 2nd instructions adds the contents of registers R1 & R0 & places sum into R0.

* The processor contains a number of registers used for several different purposes.
  - Some of the register are :

- Instruction Register:
  - It holds the instruction that is currently being executed. its o/p is available to control circuits.

- program counter:
  - This is specialised register.
  - It keeps track of the execution of a program.
  - It contains the memory address of the next instructions to be fetched and executed.

- Memory Address Register (MAR):
  - It holds the address of the location to be accessed.

- Memory Data Register (MDR):
  - It contains the data to be written into or read out of addressed Location.

General Purpose Register :

- The processor also contains GPR R_0 to R_{n-1} .

Operating Steps :

- Program reside in the memory .
- Execution of the program starts when the
  * PC is set to point to the first instruction of the pgm.
  * The contents of PC are transferred to the MAR
  * Read control signal is sent to the memory .
  * The first instrution is read out of the memory
  * this instrution will be loaded in MDR .
  * FROM MDR, the content are transferred to IR .
  * At this point , the instrution is ready to be decoded & Executed .
- If the Instruction involves an operation to be performed by ALU , it necessary to obtain the required operands, it have some steps
  * If operand is in memory, it has to fetched by sending its address to the MAR & initiating a Read cycle .
  * After reading the operand, it is transferred from MDR to ALU .
  * After one or more operands are fetched , the ALU can perform the desired operation .

* If the result is to transferred into memory.
then it send to MDR.

* The address of the location where the result is to stored is send to MAR & a write cycle is initiated.

* The contents of PC are incremented, so that it points to next instruction to be executed.

* As soon as the execution of current instruction is completed, a new instruction fetch may be started

- In addition, Some Machine Instruction with the ability to handle I/o transfers are provided.

* Normal execution of programs may be preempted if some device requires urgent servicing.

<u>Eg:-</u> A Mointoring device in a computer-controlled industrial process may detect a dangerous condition.

* An interrupt signal is a request from an Input/output device for service by the processor.

* The processor provides the requested Service by executing an appropriate interrupt Service routine.

* Before servicing the interrupt, internal state of the processor must be saved in ML.

* Normally, the content of PC, GPR & some

Performance

Performance:

* Performance of computer depends on many factors such as modern software systems and wide range of performance improvement in hardware side.

* Here it can also be evaluated by its speed

Response time or Execution time

* Total time required for the computer to complete the task including disk access, memory access, I/O activities, CPU Execution.

Throughput (or) Bandwidth:

* Total amount of work done in a given time

↳ To understand the relationship between throughput and Response time.

↳ To maximize performance:

* Minimise Response time or Execution time for some task

Relate Performance and Execution time for a computer

$$\text{Performance}_x = \frac{1}{\text{Execution time}_x}$$

Then for two computers x &y

↳ performance x is greater then the performance of y

$$\boxed{\text{Performance}_x > \text{performance}_y}$$

(ie) $$\frac{1}{\text{Executiontime}_x} > \frac{1}{\text{Execution time}_y}$$

(ie) $\text{Executiontime}_y > \text{Execution time}_x$

(ie) Execution time on y is larger than that on x

* If x is n time than y

$$\frac{\text{Performance}_x}{\text{Performance}_y} = n$$

Then Execution time will be

$$\frac{\text{Execution time}_y}{\text{Execution time}_x} = n$$

(ie) y is n times longer than on x

Eg: Computer A runs a program in 10s, Computer B runs same program in 15s, how much faster A is than B

We know that A is n times faster than B

$$\frac{\text{Performance}_A}{\text{performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is $\frac{15}{10} = 1.5$

Then A is 1.5 times faster than B

Measuring performance:

* Time is the measure of computer performance

It can defined as

→ wall clock time

→ Response time

→ Elapsed time

CPU Exection time (or) CPU time:

*Actual time that the CPU spends computing for a specific task.

* It has two types

i) User CPU time:

* CPU time spent in the programs

System Cpu time:

↳ The CPU time spent in the operating System Performing task on behalf of the programs.

↳ computer designers construct the computer with a clock which determines when the Events to be take place

Clock cycles:

* It is also called tick, clock cycles

* It is the time for one clock period, usually of the processor clock, which runs at constant rate

Clock period:

* Length of each clock cycle called clock period

CPU performance and its factors:

* A Simple formula relates the basic Metrics to CPU time,

CPU Execution time for a program = CPU Clock cycles for a program × Clock cycle time

Alternatively, clock rate and clock cycle time are inverse then,

$$\text{CPU Execution time for a program} = \frac{\text{CPU Clock cycles for a program}}{\text{Clock rate}}$$

↳ performance can be improved by

* Reducing the length of clock cycle or Number of Clock cycles required for a program

Instruction Performance:

* Execution time is equals to the number of instructions executed multiplied by average time per instructions

* The Number of clock cycles required for a program can be written as

$$\text{CPU Clock cycles} = \text{Instructions for a program} \times \text{Average Clock cycles Per Instruction}$$

Clock cycle Per Instruction (CPI)

* CPI is the average Number of clock cycle per Instruction for a program or program fragment

* CPI is the average of all Instructions Executed in the program.

Classic CPU Performance Equation:

* Instruction Count:

The number of Instructions executed by the program

$$\text{CPU time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock cycle time}$$

* Clock rate is inverse of clock cycle time

$$\text{CPU time} = \frac{\text{Instruction Count}}{\text{Clock rate}} \times \text{CPI}$$

Instructions

Representing Instructions in a computer System:

* An instruction is an order given to a computer processor by a computer program.

* At the lowes level, each instruction is a sequence of 0's and 1's that describe a physical operation the computer is to perform and depending on the particular type the operation is varied.

* The specification of special storage areas called registers that may contain data to be used in carrying out the instruction or the location in computer memory of data.

* Instructions are kept in the computer as a series of high and low electronic signals and it may be represented as numbers.

* To represent the instructions we need to use the Instruction format specified by particular language.

Instruction Format

It is a form of representation of an instruction composed of fields of binary numbers

Registers

* In computer hardware registers are referred to in instructions.

* But instructions are represented as numbers so we must convert register names into numbers

* In MIPS assembly language, registers $so to $s7 map onto registers 16 to 23 and registers $to to $t7 map onto registers 8 to 15

* $so means 16, $s1 means 17, $s2 means 18 and so on as like this $to means registers, $t1 means register 9 and so on.

Machine Language

* Assembly Language instructions use exactly 32 bits and the same size as a data word.

* All MIPS instructions are 32 bits long so we need to focus some numeric versions of instruction called machine language

* Machine Language is a binary representation used for communication within a computer system

* Instruction used in machine language are called machine code.

Eg:

Translating a MIPS assembly instruction into a machine instruction, let us consider MIPS Instruction

add $to, $s1, $s2

Translate first as a combination of decimal numbers and then a binary numbers.

Answer:

The decimal representation is

0	17	18	8	0	32

* Each of these segments of an instruction is called a field

* The first and last fields (0 & 32) in combination tell the MIPS computer that this instruction perform addition

* Second field gives the number of the register that is the first source operand of the addition operation (17 $s1)

* Third field gives the other source operand for the addition (18 $s2)

* Fourth Field Contains the number of register that is to receive the Sum (8 $t0)

* Fifth field is unused in this instruction so it is set to 0

* This instruction adds register $s1 to register $s2 and places the sum in register $t0

000000	10001	10010	01000	00000	100000
6bits	5bits	5bits	5bits	5bits	6bits

Hexadecimal Number:

* Computer can use binary numbers to read and write data. In binary number format for small value also it requires large amount of bits

* So we can use higher base that can be easily converted into binary.

* All computer data sizes are multiples of 4 in that hexadecimal numbers are popular

* Base value of hexadecimal number is 16 and it is power of 2

Hexadecimal to binary Conversion

Hexadecimal	Binary	Hexadecimal	Binary
0hex	0000two	8hex	1000two
1hex	0001two	9hex	1001two
2hex	0010two	Ahex	1010two
3hex	0011two	Bhex	1011two
4hex	0100two	Chex	1100two
5hex	0101two	Dhex	1101two
6hex	0110two	Ehex	1110two
7hex	0111two	Fhex	1111two

MIPS Field:

MIPS fields has two kind of format such as

1. R-type or R-format (for register)

2. I-type or I-format (for immediate)

1. R-format

op	rs	rt	rd	Shamt	funct
6bits	5bits	5bits	5bits	5bits	6bits

op- basic operation of the instruction called as opcode

↳ opcode denotes the operation and format of an instruction rs- the first register source operand

rt- the second register, source operand

rd- the register destination operand, it gets result

Shamt- Shift amount

funct- This field called function or function code, selects the specific variant of the operation in the op field

2. I- Format

op	rs	rt	Constant or address
6bits	5bits	5bits	16 bits

* Used by the immediate and data transfer Instructions

Drawback of different Format

*Multiple formats Complicate the hardware

* It increase the complexity

Operations and Operands

Operations of the Computer Hardware:

* Every computer will perform different kinds of operations based on the application areas.

* Every computer must be able to perform arithmetic operantions.

* The MIPS assembly language notation for Performing addition operation is

$$\boxed{\text{add} \quad a, b, c}$$

↳ add = addition operation
a,b,c are variables

* It instructs a computer to add the two Variables band c and Put their sum in a.

Example: Suppose we need to Place the sum of four variables b, c, d and e into variable a means the following sequence of instructions can be executed.

add a, b, c : the sum of b and c is placed in a
add a, a, d : the sum of b, c, and d is now in a
add a, a, e : the sum of b,c,d and e is now in a

Three instructions to Sum the four variables

Category	Instruction	Example	Meaning	Comment
Arithmetic	add	add $S1, $S2, $S3	$S1=$S2+$S3	Three register operands
	Subtract	sub $S1, $S2, $S3	$S1= $S2-$S3	Three register operands
	add immediate	addi $S1, $S2, 20	$S1= $S2+20	Used to add constants
Data transfer	load word	lw $S1, 20($S2)	$S1= Memory [$S2+ 20]	word from memory to register
	Store word	sw $S1, 20($S2)	Memory[$S2+20] = $S1	word from register to memory
Logical	add	and $S1,$S2, $S3	$S1= $S2&$S3	Three register operands; bit by bit AND
	or	or $S1, $S2 $S3	$S1= $S2| $S3	Three register operands; bit by bit OR
conditional branch	branch on equal	beq, $S1, $S2, 25	if($S1=$S2) go to PC+4+100	Equal test; PC-relative branch
	branch on not equal	bne $S1, $S2, 25	if($S1!=$S2) goto PC+4+100	Not equal test PC relative
Unconditional jump	jump	j 2500	goto 10000	Jump to target address
	jump register	jr $ra	goto $ra	For switch, procedure return

* To keep the hardware simple, every instruction has to have exactly three operands no more and noless.

* Hardware for a variable number of operands is more complicated than hardware for a fixed number.

* Three design principles of hardware technology has

Design Principles 1: Simplicity favors regularity

Design principles 2: Smaller is faster

Design principles 3: Good design demands good compromises

Design principle 1: Simplicity favors regularity:

Compiling two C assignment statements into MIPS

* C program contains the five variables a, b, c, d and e

$a = b + c$

$d = a - e$

* The translation from C to MIPS assembly language instructions is performed by the compiler. Show the MIPS code produced by a compiler.

Solution:

* A MIPS instruction operates on two source operands and places the result in one destination operands.

* So the two simple statements above compile directly into these two MIPS assembly language instructions.

```
        add   a, b, c
        sub   d, a, e
```

Operands of the Computer Hardware:

* operand is a variable used to perform any kind of operations.

* unlike high level languages, the operands of arithmetic instructions are restricted

* operand must be from a limited number of special locations built directly in hardware called registers.

* Registers are primitives used in hardware design that also visible to the programmer when the computer is completed.

* MIPS architecture has 32 bits registers and group of 32 bits are called word.

Design principle 2: Smaller is faster

* Registers is a very small amount of very fast memory that is built into the cpu.

* A very large number of registers may increase the clock cycle time because it takes electronic signals longer to travel it.

* Another reason for why more than 32 bit registers are not used means it takes in the instruction format.

*MIPS convention use two character names following a dollar sign to represent a register.

Example:

* Compiling a C assignment using registers. It is the compilers job to associate program variables with registers. Take for instance, the assignment statement from our Example is

$$f = (g+h) - (i+j);$$

The variables f, g, h, i and j are assigned to the registers $ s0, $s1, $s2, $s3 and $s4 respectively. what is the compiled MIPS code?

Answer:

add $t0, $s1, $s2: register $t0 contains g+h

add $t1, $s3, $s4: register $t1 contains i+j

Sub $s0, $t0, $t1: f gets $t0-$t1, which is (g+h) - (i+j)

Memory operands:

* programming language have simple variables that contain single data element, but they also have more complex data structures like arrays and structures

* Complex data structures contains large amount of elements than the registers in a computer.

Data transfer instructions:

* Data transfer instruction is a command that moves data between memory and registers.

Address: →A value used to define the location of a specific data element within a memory array called Address.

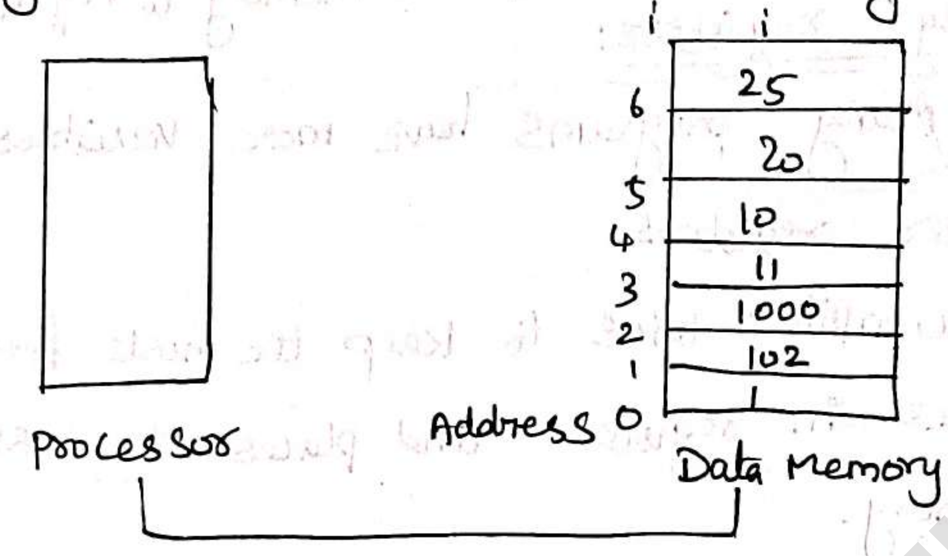
* To access a word in memory, the instruction must supply the memory address.

Memory:

* Memory is a large, single dimensional array, with the address acting as the index to that array starting at 0

Example :

Memory addresses and contents of memory at those location



* The address of the third data element is 2 and the value of memory [2] is 1000.

* The data transfer instruction copies data from memory to a register that process is called load.

Alignment Restriction:

*In MIPS, words must start at addresses that are multiples of 4. This requirement is called an alignment restriction.

Big Endian and Little Endian:

* Address of the left most byte is called "big end" and right most byte is called "little end".

Store:

* The instruction complementary to load is called store.

* Store copies data from a registes to memory.

* Load copies data from memory to register

Spilling registers:

* Many programs have more variables than the computer registers.

* compiler tries to keep the most frequently used variables in registers and places the rest of variable in memory.

* Using loads and stores we can move variables between registers and memory.

* "The process of putting less commonly used variables into memory is called spilling registers".

Constant or Immediate operands:

* constant variables are used as one of the operand for many arithmetic operations in MIPS architecture.

* Constant have been placed in memory when the program was loaded.

* To avoid load instruction used in arithmetic instruction we can use one operand is a constant.

* This quick add instruction with one constant operand is called add arithmetic or addi

Eg: addi \$S3,\$S3,4 ; \$S3=\$S3+4

* Sometimes a program may need to use a constant in an operation. Those operands are called immediate operands.

Instruction Representation

* An instruction is an order given to a computer processor by a computer program.

* At the lowes level, each instruction is asequence of 0's and 1's that describe a physical operation the computer is to perform and depending on the particular type the operation is varied.

* The specification of special storage areas called registers that may contain data to be used in carrying out the instruction or the location in computer memory of data.

* Instructions are kept in the computer as a series of high and low electronic signals and it may be represented as numbers.

* To represent the instructions we need to use the Instruction format specified by particular language.

Instruction Format

It is a form of representation of an instruction composed of fields of binary numbers

Registers

* In computer hardware registers are referred to in instructions.

* But instructions are represented as numbers so we must convert register names into numbers

* In MIPS assembly language, registers $s0 to $s7 map onto registers 16 to 23 and registers $t0 to $t7 map onto registers 8 to 15

* $s0 means 16, $s1 means 17, $s2 means 18 and so on as like this $t0 means registers, $t1 means register 9 and so on.

Machine Language

* Assembly Language instructions use exactly 32 bits and the same size as a data word.

* All MIPS instructions are 32 bits long so we need to focus some numeric versions of instruction called machine language

* Machine Language is a binary representation used for Communication within a computer System

* Instruction used in machine language are Called machine code.

Eg:

Translating a MIPS assembly instruction into a machine instruction, let us Consider MIPS Instruction

add $to, $s1, $s2

Translate first as a combination of decimal numbers and then a binary numbers.

Answer:

The decimal representation is

0	17	18	8	0	32

* Each of these segments of an instruction is called a field

* The first and last fields (0 & 32) in Combination tell the MIPS computer that this instruction perform addition

* Second field gives the number of the register that is the first source operand of the addition operation (17 $s1)

* Third field gives the other source operand for the addition (18 $s2)

* Fourth Field contains the number of register that is to receive the sum (8 $t0)

* Fifth field is unused in this instruction so it is set to 0

* This instruction adds register $s1 to register $s2 and places the sum in register $t0

000000	10001	10010	01000	00000	100000
6bits	5bits	5bits	5bits	5bits	6bits

Hexadecimal Numbers:

* Computer can use binary numbers to read and write data. In binary number format. For small Value also it requires large amount of bits

* So we can use higher base that can be easily converted into binary.

* All computer data sizes are multiples of 4 in that hexadecimal numbers are popular

* Base value of hexadecimal number is 16 and it is power of 2

Hexadecimal to binary Conversion

Hexadecimal	Binary	Hexadecimal	Binary
0hex	0000two	8hex	1000two
1hex	0001two	9hex	1001two
2hex	0010two	Ahex	1010two
3hex	0011two	Bhex	1011two
4hex	0100two	Chex	1100two
5hex	0101two	Dhex	1101two
6hex	0110two	Ehex	1110two
7hex	0111two	Fhex	1111two

MIPS Field:

MIPS fields has two kind of format such as

1. R-type or R-format (for register)

2. I-type or I-format (for immediate)

1. R-format

op	rs	rt	rd	Shamt	funct
6bits	5bits	5bits	5bits	5bits	6bits

op- basic operation of the instruction called as opcode

↳ opcode denotes the operation and format of an instruction rs- the first register source operand

rt- the second register, source operand

rd- the register destination operand, it gets result

Shamt- Shift amount

funct- This field called function or function code, Selects the specific variant of the operation in the op field

2. I- Format

op	rs	rt	Constant or address
6bits	5bits	5bits	16 bits

* Used by the immediate and data transfer Instructions

Drawback of different Format

* Multiple formats Complicate the hardware
* It increase the Complexity

Decision Making

Decision making is commonly represented in programming Languages using the if statement, sometimes combined with go to statements and labels.

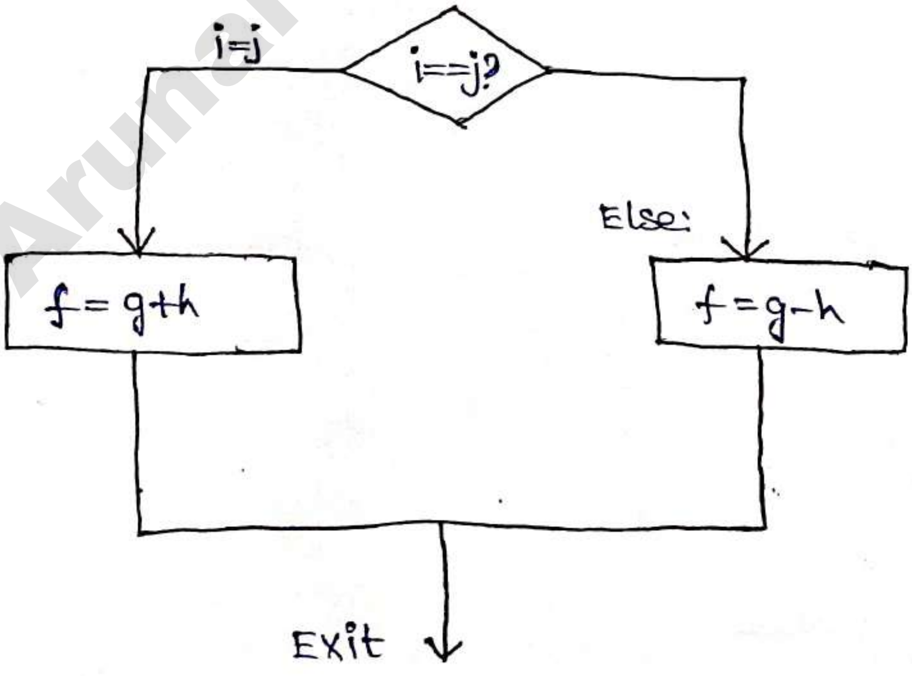
MIPS assembly language includes two decision-making instructions, similar to an if statement with a go to. The first instruction is

beq register1. register2. L1

The mnemonic beq stands for branch if equal.
The second instruction is

bne register1. register2. L1

The mnemonic bne stands for branch if not equal.
These two instructions are traditionally called conditional branches.



MIPS Addressing

Addressing Modes:

* Addressing mode is one of several addressing regimes delimited by their varied use of operands and/or address.

* The different ways that a processor can access data are referred to as addressing schemes or addressing modes.

* An address computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction is known as Effective Address (EA).

* An Effective Address can be made up of three elements, The base, index & displacement.

MIPS has the following addressing modes

1) Immediate addressing
2) Register addressing
3) Base or displacement addressing
4) PC - relative addressing
5) Pseudo - direct addressing

1. Immediate Addressing

op	rs	rt	immediate

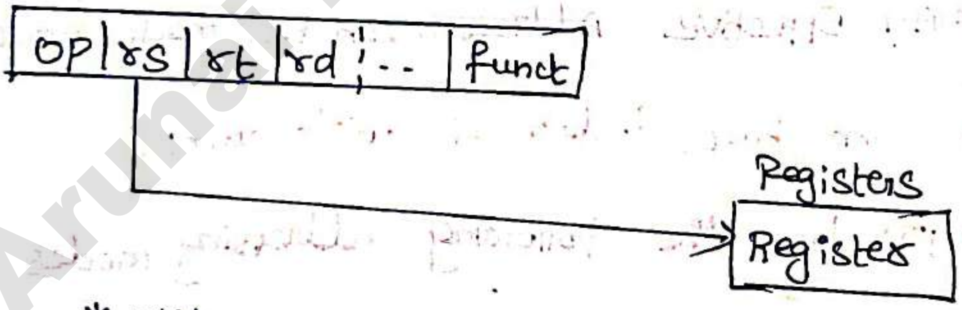
* The operand is given explicitly in the instruction.

Eg MOV #20, A

This instruction copies operand 20 in the register A. The sign # in front of the value of an operand is used to indicate that this value is an immediate operand.

2. Register Addressing:

* The operand is a register. (ie) The operand is the contents of processor register. The name of register is specified in the instruction.



* Either compiler or assembler must break large constants into pieces and then reassemble them into a register.

* In this type of addressing mode the name of the register is given in the instruction where the data to be read or result is to be stored.

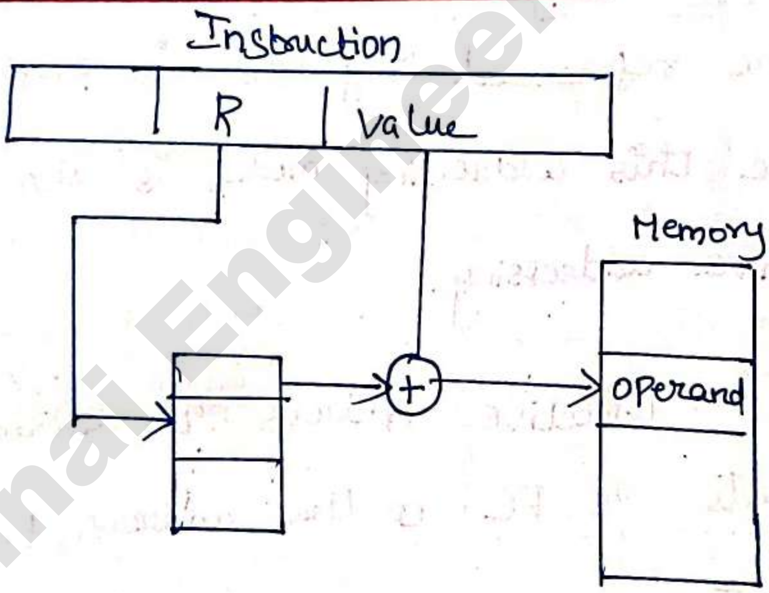
* In immediate instructions the size will be the problem for performing a load and store operations.

* To solve this problem registers are used for temporary

Eg: MOV R2, R1:

This instruction copies the contents of register R2 to register R1.

3. Base or Displacement Addressing:



* This addressing mode combines the capabilities of direct addressing and register indirect addressing.

* In this addressing mode, instruction has two address fields:

> Value

> Referenced register

* The effective address is computed by adding contents of referenced register to value.

$$EA = value + (R)$$

* Three common variation of displacement addressing are

i) Relative Addressing

ii) Base register addressing

iii) Index addressing

4. i) Relative Addressing mode.

* The referenced register is program counter (PC) and hence this addressing mode is also known as PC-relative addressing.

* The Effective Address is determined by adding the contents of PC to the address field

EA = PC + Address part of Instruction

* The address part is a signed number so that it is possible to have branch target location either before or after the branch instruction. This addressing node is commonly used to specify the target address in branch instruction.

Eg JNZ BACK

This instruction causes program execution to go to the branch target location identified by the name BACK, if the branch condition is satisfied.

ii) Base register addressing:

* In this addressing mode, the referenced register contains the main memory address and address field contains the displacement.

* Displacement is usually unsigned integer number

$$EA = (R) + Displacement$$

Eg: MOV [R+8], A

* This instruction copies the contents of memory whose address is determined by adding the contents of register R and displacement 8 to the register A.

iii) Index Addressing mode:

* In this addressing mode, the address field references the main memory and the referenced register contains a positive displacement from the address

$$EA = Memory\ address + (R)$$

$$EA = X + (R)$$

* Used for Array type data structures

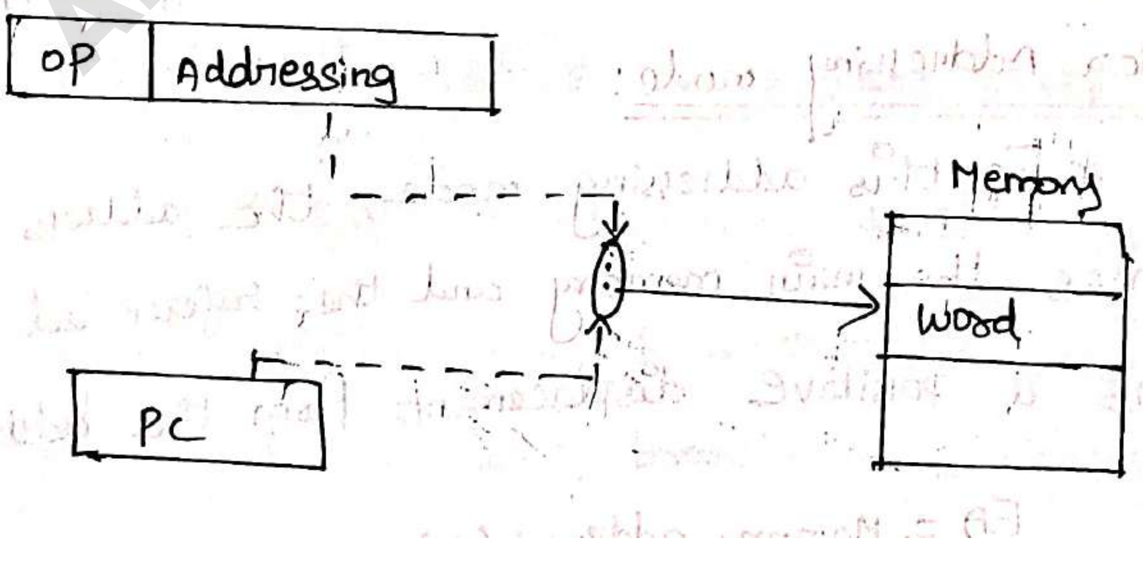
* The indexing is a technique that allows programmer to point or refer the data (operand) stored in sequential memory locations one by one. It is an efficient mechanism for performing iterative operations.

Eg MOV[R1 +RI], R

* In this instruction main memory address is given by register R_1 and the referenced register RI gives the positive displacement. The contents of the memory address generated by the addition of main memory address and displacement is copied to register R

5. PseudoDirect Addressing:

* The jump address is the 26 bits of the instruction concatenated with the upper bits of the PC



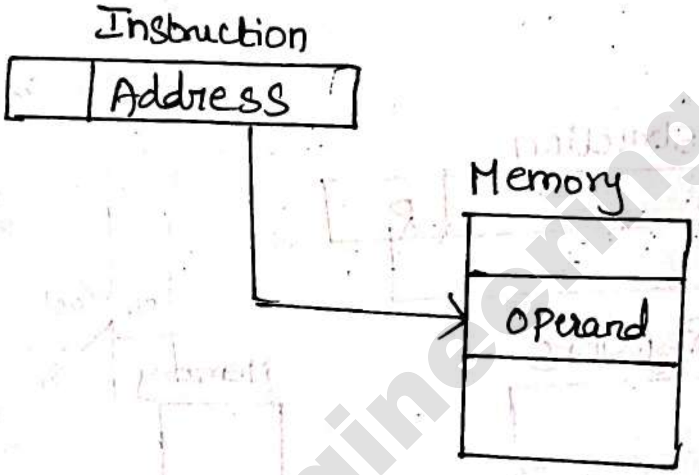
Other Addressing Modes:

6. Absolute or Direct addressing mode:

The address of the location of the operand is given explicitly as a part of the instruction.

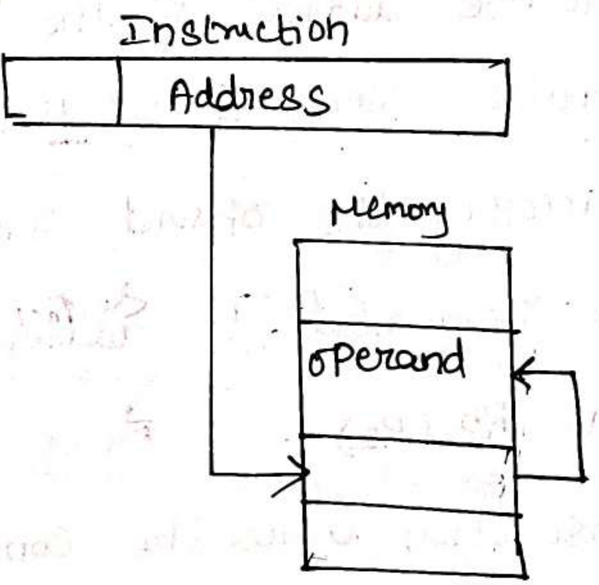
Eg MOV 2000, A

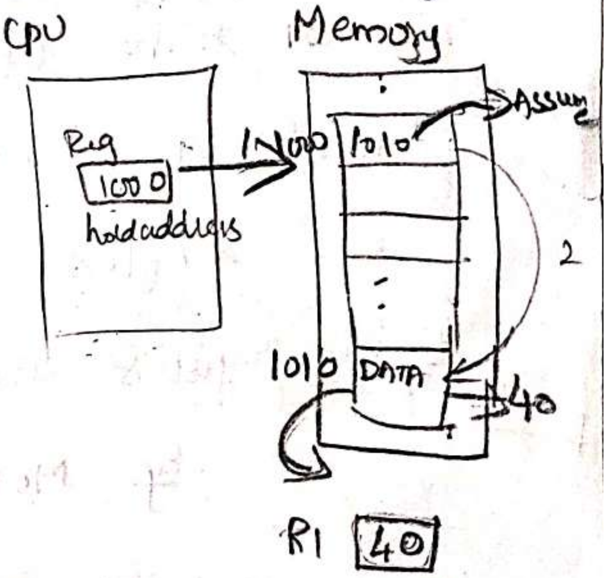
This instruction copies the contents of memory location 2000 into the A register.



7. Indirect Addressing mode:

* In this addressing mode, the instruction contains the address of memory which refers the address of the operand



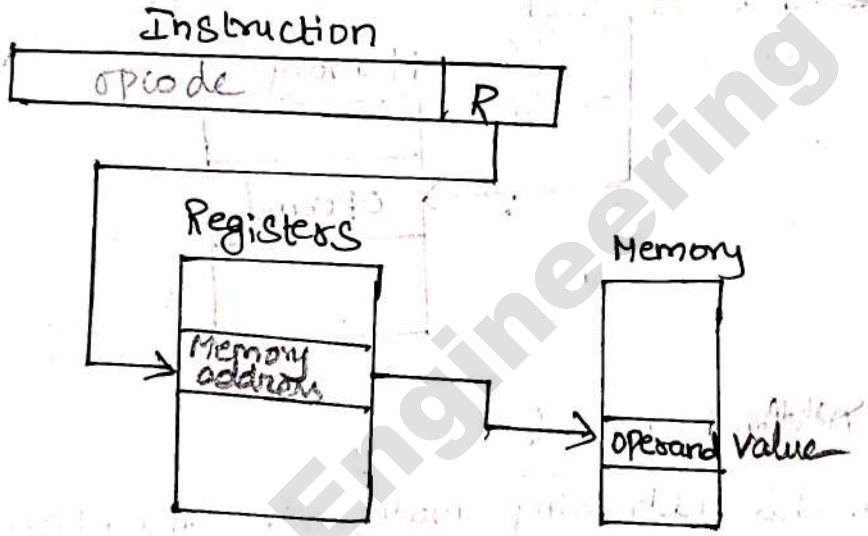


8. Register Indirect addressing mode:

* The effective address of the operand is the contents of a register or the main memory location whose address is given explicitly in the instruction.

Eg MOV (R0), A

* This instruction copies the contents of memory addressed by the contents of register R0 into the register A.



9. Autoincrement addressing 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 incremented to address the next locatio

Eg MOV R0, (R2)+

* The instruction copies the contents of registers R0 into the memory location whose address is specified

by the contents of register R2. After copy operation, the contents of register R2 are automatically incremented by 1.

10. Autodecrement addressing mode:

* The contents of a register specified in the instruction are decremented and then they are used as an effective address to access a memory location.

eg MOV −(R0), R1

* This instruction, initially decrements the contents of register R0 and then the decremented contents of register R0 are used to address the memory location

* Finally the contents from the addressed memory locations are copied into the register R1

11. Stack addressing mode:

* A stack is linear array of reserved memory locations. It is associated with a pointer called stack pointer (SP).

* In stack addressing mode, stack pointer always contains the address of TOP of stack (TOS) where the operand is to be stored or located.

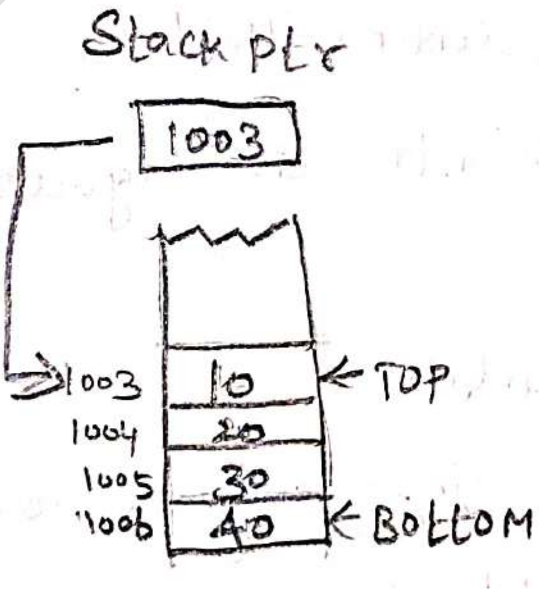
* Thus, the address of the operand (source or destination) is the contents of stack pointer.

* This addressing mode is the special case of register indirect addressing where referenced register is a stack pointer.

* Usually, stack grows in the direction of descending addresses (descending stack), starting from a high address and progressing to lower one.

* SP (Stack pointer) is decremented before any items are appended (pushed) on stack

* SP is incremented after any items are popped from the stack.



UNIT 2
ARITHMETIC FOR COMPUTERS

Addition

* Digits are added bit by bit from right to left, which carries passed to the next digit to the left.

Eg:- Adding 6_{ten} to 7_{ten} in Binary.

```
  0000 0000 0000 0000 0000 0000 0000 0111two
+ 0000 0000 0000 0000 0000 0000 0000 0110two
  0000 0000 0000 0000 0000 0000 0000 1101two
```

```
(0)  (0)  (1)  (1)  (0)
 0    0    0    1    1    1
 0    0    0    1    1    0
(0) 0 (0) 0 (0) 1 (1) 1 (1) 0 (0) 1
```

* Overflow occurs when the result of addition cannot be represented with the available hw.

* Adding Operands with different sign cannot cause the Overflow.

Eg:- Add 10 with -4, the result is 6.

here No overflow occurs.

* If overflow does not occur, then it is good it will speed the ALU operation.

* Generally Adding two 32 bits numbers require 33 bits for result to Expressfully.

* 33rd bit Used to when Overflow Occur, the sign bit is set with the value of the result

* Overflow occurs, when adding 2 positive no's & Sum is negative or vice versa.

– Carry out Occured in a sign bit.

Addition can be performed in 2 ways.

- * Signed Binary no.
- * Unsigned "

Signed numbers :

* sign bit 0 → (+ve) no.
1 → (-ve) no.

* If the binary no is signed,
leftmost bit ⇒ sign
Rest of the bit ⇒ number

Sign	Number

* If the binary no is unsigned,
Leftmost bit ⇒ Most significant bit of the number

Eg:- 01001

9 unsigned

+9 signed ⇒ Leftmost bit ⇒ 0.

11001

25 unsigned

-9 signed ⇒ leftmost bit ⇒ 1.

Eg:- 2 operands is positive & result is negative, overflow occur.

$$\begin{array}{rcl} +98 & = & 01001\ 1000 \\ +87 & = & 0\ 1000\ 0111 \\ \hline & & 1]0001\ 1111 \end{array}$$

1 ⇒ -ve., overflow occured.

eg:- -83 = 11000 0011

-24 = 10010 0100

Discard bit ← $\underline{1}\,\underline{0}$1010 0111

0 → +ve Value, Overflow occured.

Unsigned Numbers:

- Unsigned integers used for Memory addresses because Overflow are ignored.
- Overflow is not detected, for some cases the computer designer must provide a way to ignore Overflow.

* The MIPS computer has method to ignore overflow in signed integers.

1. Add (add), add immediate (addi), sub) cause Exception on Overflow.
2. Add unsigned (addu), add immediate unsigned (addiu), Subtract unsigned does not cause Exception on Overflow.

* MIPS detects Overflow with an Exception called Interrupt ⇒ An exception that comes from outside the processor

* Exception is an unscheduled Event that disrupts program.

Execution Used to detect Overflow.

Overflow condi: for addit & Sub

Operation	operd A	operd B	Result Indicating Overflow
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0

Subtraction

* Inverse Operation of addition

```
Eg:  9 => 1001     1000
    -6 =>  110       11
          ----     ----
           011       01
```

*

Eg:. sub. 6_2 from 7_2

```
0000 0000 0000 0000 0000 0000 0000 0111
0000 0000 0000 0000 0000 0000 0000 0110
---------------------------------------
                                   0001 = 1.
```

or via addition, using 2's compl. representation of -6.

```
0000 0000 0000 0000 0000 0000 0000 0111
1111 1111 1111 1111 1111 1111 1111 1010
---------------------------------------
0000                          0000 0001 => 1
```

Overflow in subtraction

- Opposite to the overflow in addition.
- When the sign of the operands are same, overflow cannot occur.
- It occur in 2 cases.

Case 1: Consider 2 4bit no's.

X = +15 => 01111

Y = -12 => 11100

```
Direct sub:  01111
             11100
             -----
             10011
```

the result is (-ve), overflow occured.

Case 2:

X = -12 = 11100

Y = 15 = 01111

```
01101
```

Table

Operation	Operand A	B	Result indicating overflow
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A−B	≥ 0	< 0	< 0
A−B	< 0	≥ 0	≥ 0

* In MIPS, to ignore the overflow is signed numbers,

1. Sub cause Exception on overflow
2. Sub unsigned does not cause Exception on overflow

* Mips includes a register called Exception PC

EPC ⇒ contains the addr of the instruction that caused the Exception.

Multiplication

Multiplication:

- * The first operand is called Multiplicand
- * The Second operand is called Multiplier
- * Final Result is called Product

Basic rules for Binary digits multiplication

```
0x0  = 0
0x1  = 0
1x0  = 0
1x1  = 1
```

Eg Multiplying 1010 by 1000

```
Multiplicand          1010
Multiplier          x 1000
                 ----------
                      0000
                     0000
                    0000
                   1010
                 ----------
Product            1010000
                 ----------
```

* Take the digits of the multiplier one at the time from right to left, Multiplying the

multiplicand by the Single digit of the multiplier.

*Shifting the intermediate product one digit to the left of the earlier intermediate products.

* The numbers of digits in the Product is larger than the number ofs digits in either the multiplicand or the multiplier

* Multiplication is usually implemented by Some form of repeated addition

* To compute x×y is to add the multiplicand y to itself x times, where x is the multiplier

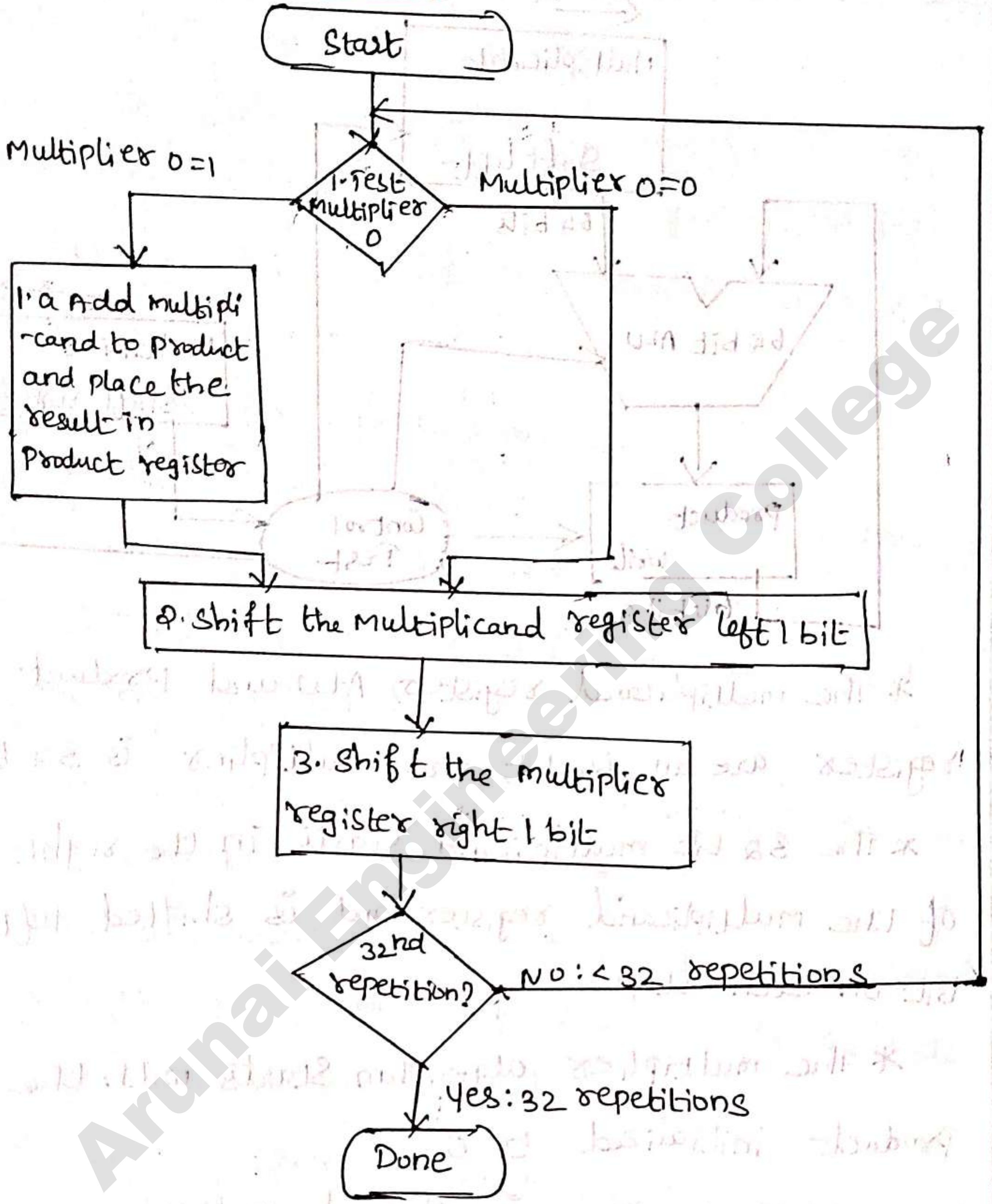
Example :

$$4 \times 8 = 32$$

Here x=4 and y=8 now we add y itself x number of times, that will produce the product

$$\begin{array}{r} 8 \\ 8 \\ 8 \\ 8 \\ \hline 32 \\ \hline \end{array}$$

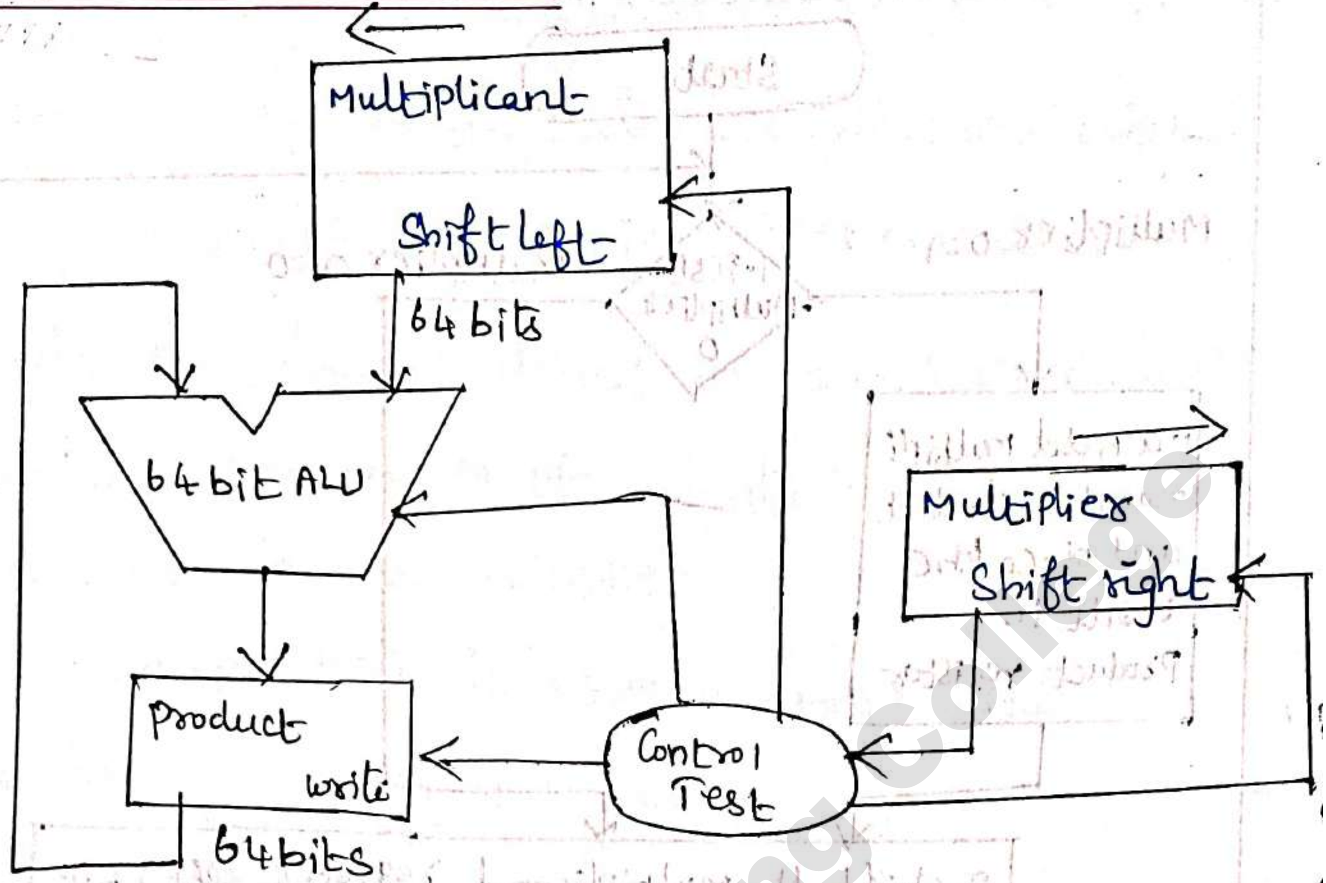
Multiplication Algorithm:


Start
Multiplier 0=1
1. Test Multiplier 0
Multiplier 0=0
1.a Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

The first Multiplication Algorithm:

Step1: The least significant bit of the multiplier (multiplier 0) determines whether the multiplicand is added to the product register

Multiplication Hardware:



* The multiplicand register, ALU and product register are all 64 bit and multiplier is 32 bits

* The 32 bit multiplicand starts in the right half of the multiplicand register and is shifted left 1 bit on each step.

* The multiplier algorithm starts with the product initialized to 0.

* The multiplier is shifted in the opposite direction at each step.

* control decides when to shift the multiplicand and multiplier registers and when to write new values into the product register.

Step 2: The left shift has the effect of moving the intermediate operands to the left.

Step 3: The shift right gives the next bit of the multiplier to examine in the following iteration.

Step 4: These three steps are repeated 32 times to obtain the product.

Clock cycles for multiplication algorithm

* This algorithm requires almost 100 clock cycles to multiply two 32 bit numbers.

* Multiply take multiple clock cycle without affecting performance

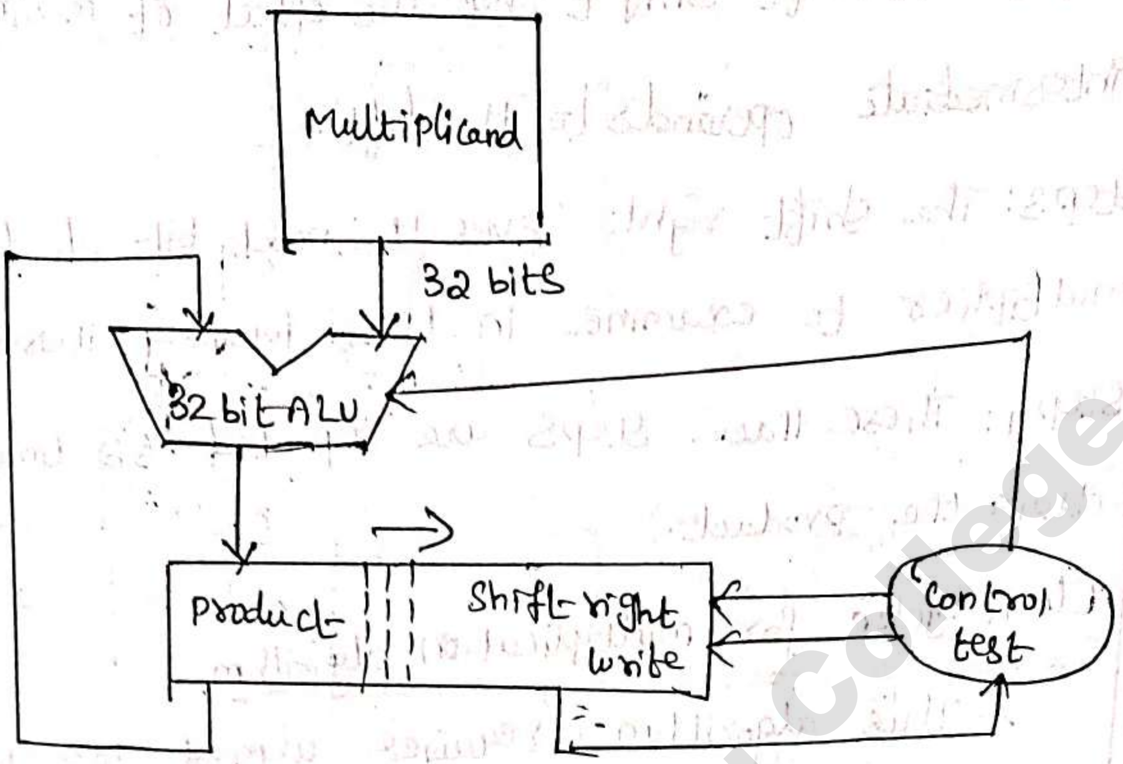
* To increase the speed of process by performing the operations in parallel manner.

* The multiplier and multiplicand are shifted while the multiplicand is added to the product if the multiplier bit is 1.

Refined version of multiplication Hardware:

* The hardware is used to ensure that is tests and right bit of the multiplier and gets the pre-shifted version of the multiplicand

* Hardware will take 1 clock cycle per step



* The multiplicand register, ALU and multiplier registers are all 32-bits, the product register only has 64 bits. The product is shifted right.

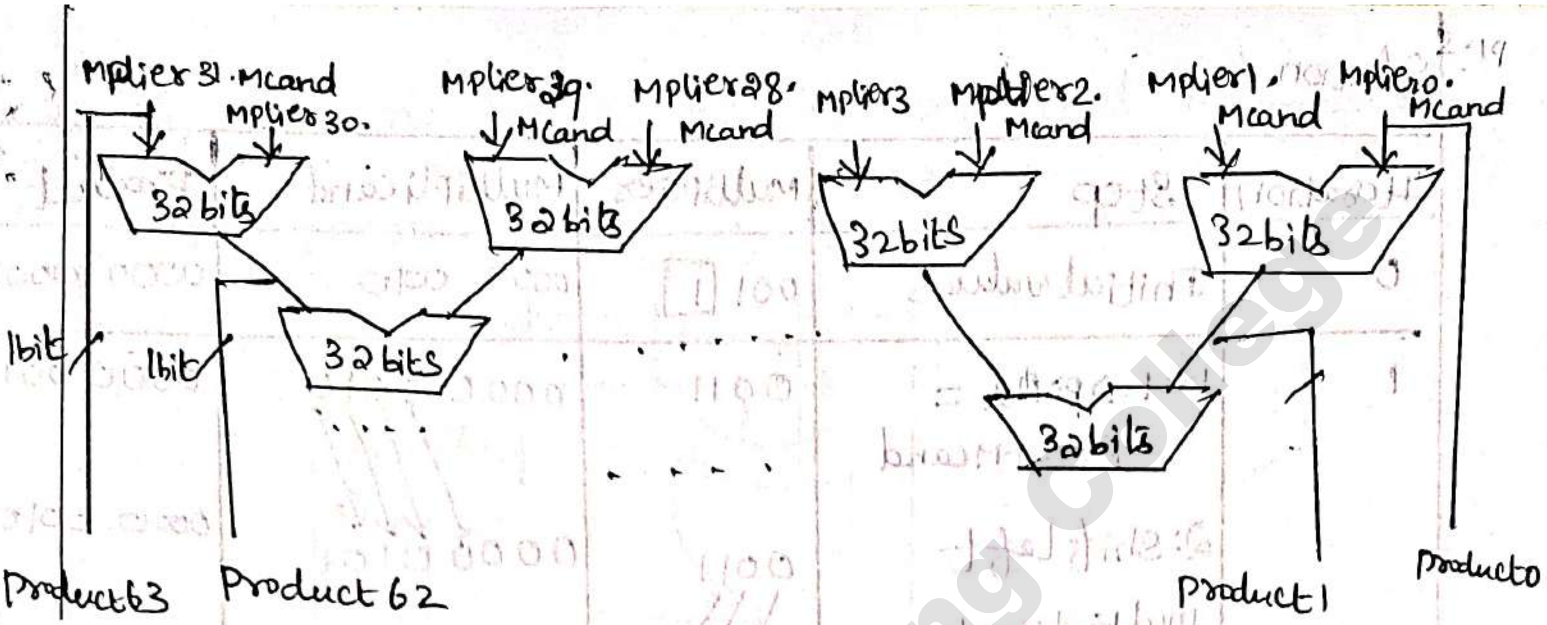
* The separate multiplier register also disappeared and the multiplier is placed instead in the right half of the Product register

Signed Multiplication:

* For Signed multiplication first convert the multiplier and multiplicand into positive numbers and then remember the original signs

Faster Multiplication:

* Moore's law has provides much more resources is there the hardware designers can build much faster multiplication hardware.



* Faster Multiplications are possible by providing one 32 bit adder for each bit of the multiplier

* One input is the multiplicand ANDed with a multiplier bit and the other is the output of a prior adder.

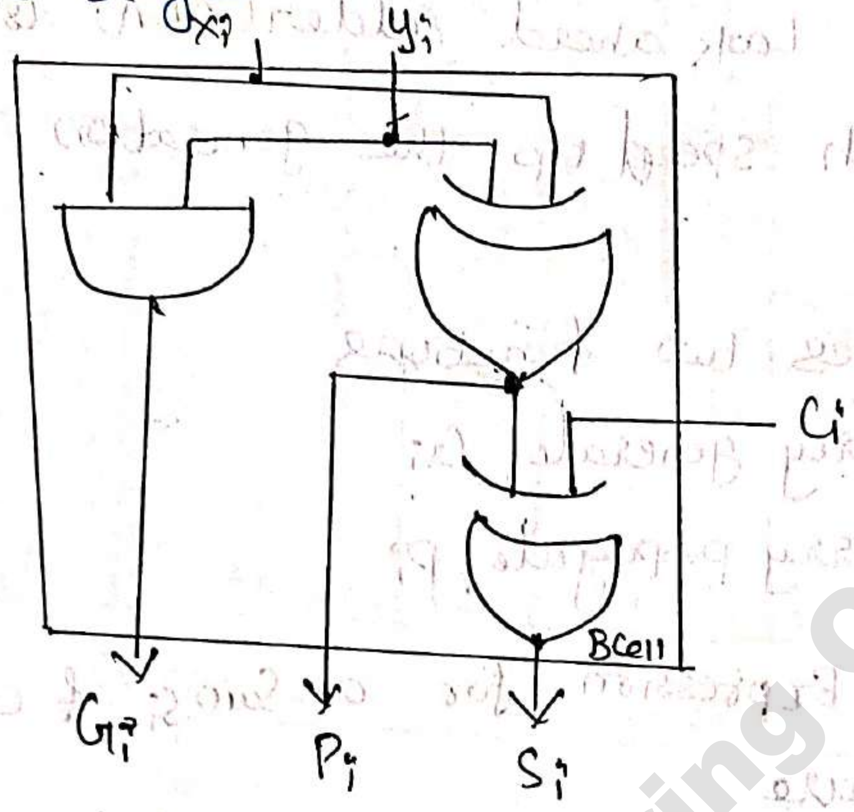
Example:

Using 4 bit numbers to save space, multiply 2_{ten} 3_{ten} or 0010_{two} 0011_{two}.

Solution :

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	001$\boxed{1}$	0000 0010	0000 0000
1	1a: 1 => prod = prod + Mcand	0011	0000 0010	0000 0010
	2: Shift Left multiplicant	0011	0000 0100	0000 0010
	3: Shift right multiplier	000$\boxed{1}$	0000 0100	0000 0010
2	1a: 1 => prod = prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left multiplicand	0001	0000 1000.	0000 0110
	3: shift Right Multiplier	000$\boxed{0}$	0000 1000	0000 0110
3.	1: 0 => No operation	0000	0000 1000	0000 0110
	2: Shift Left multiplicand	0000	0001 0000	0000 0110
	3: Shift Right Multiplier	000$\boxed{0}$	0001 0000	0000 0110
4	1: 0 => Nooperation	0000	0001 0000	0000 0110
	2: shift Left Multiplicand	0000	0010 0000	0000 0110
	3: Shift Right Multiplier	0000	0010 0000	0000 0110

The following basic cell can be used in each bit stage



* Expanding C_i in terms of $i-1$ and substitute into C_{i+1}, then following expression will be

$$\boxed{C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} C_{i-1}}$$

* The final expression for carry variable is

$$\boxed{C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} \cdots P_1 G_0 + P_i P_{i-1} \cdots P_0 C_0}$$

* Thus all carries can be obtained three gate delays after the input signals x, y, C_0 are applied

Division

Division:

*Division is a reciprocal operation of multiplication.

*Division operation is less frequent and is faster. It will Perform invalid operation when we Perform dividing by 0.

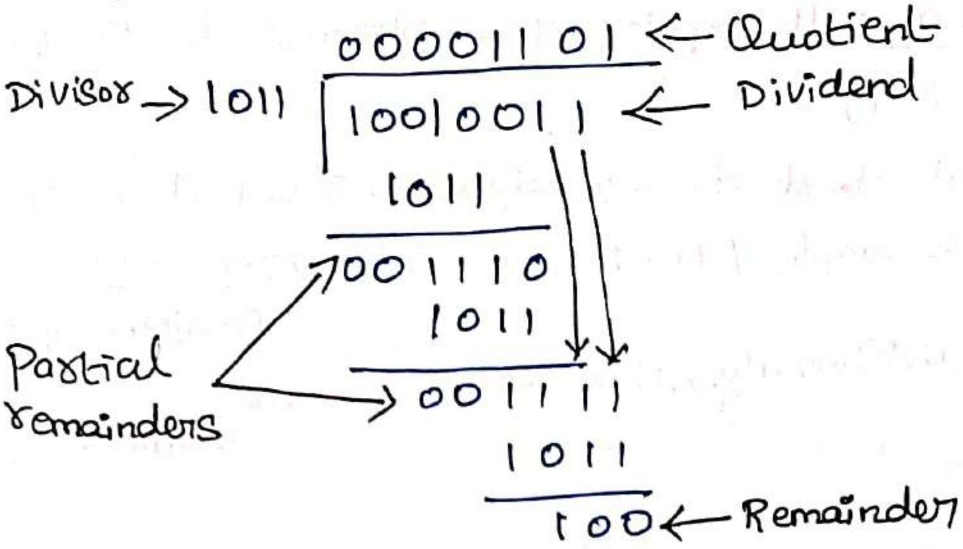
* Divide's use two operand called the dividend and divisor and produce the results called quotient and remainder.

* Hence the another way to express the relationship between the Components.

Dividend = Quotient * Divisor + Remainder.

Let us assume that both the dividend and divisor are positive and hence the quotient and the remainder are non-negative.

Example



Algorithm for Restoring Division:

Step1: Shift Remainder register left logically (Sll) one binary position

Step2: subtract divisor from left (upper half) of the remainder register and the result in the left half of the Remainder register

Step3: If the sign bit of Remainder register is 1, then shift Quotient register (Q) left by on binary position and set Q_0 to 0 and add divisor back to remainder register (ie restore remainder register); otherwise set Q_0 to 1.

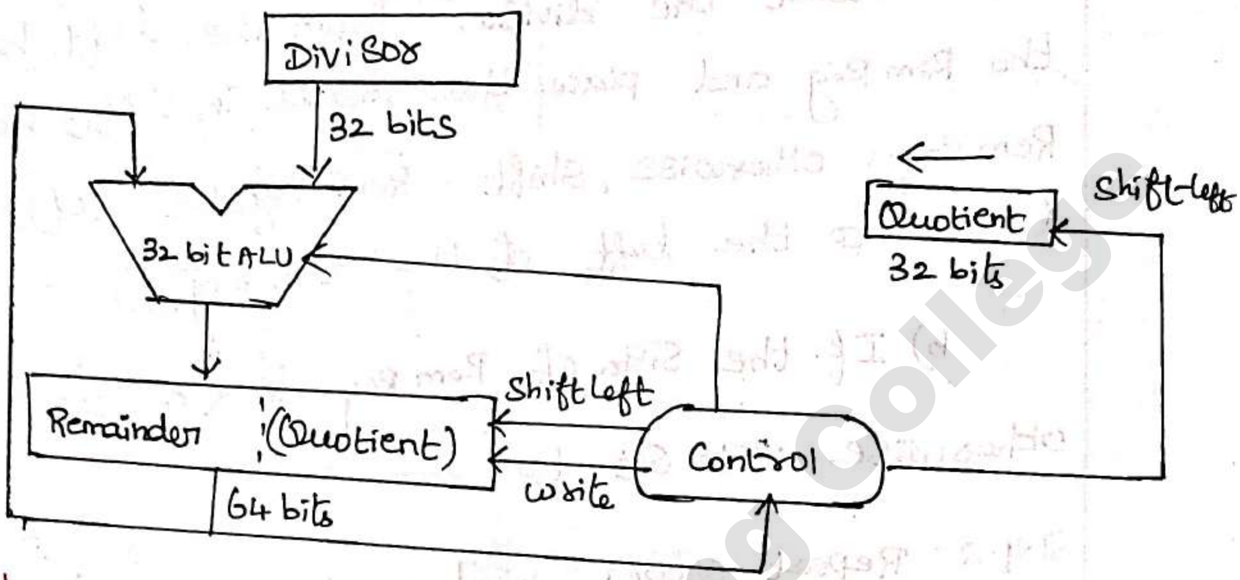
Step4: Repeat step 1, 2 and 3 for <33 times to obtain the reminder and quotient result.

For simplicity, use 4 bit version to perform division, same concept can expanded to 32 bit number.

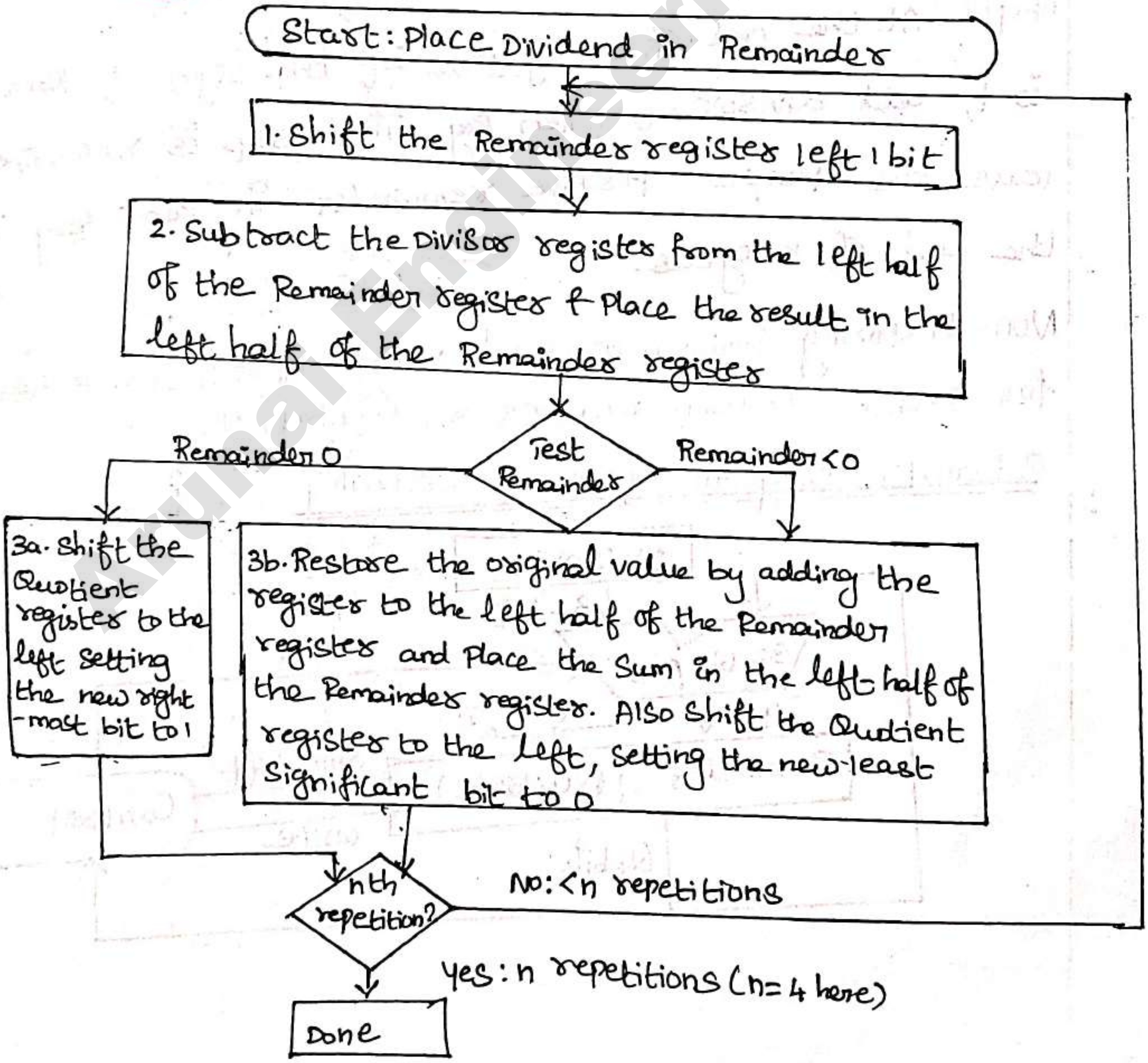
Restoring Division (or) Second version of Division :

Hardware of 32-bit Boolean number representation

Schematic Diagram of ALU circuitry



Flowchart


Start: Place Dividend in Remainder
1. Shift the Remainder register left 1 bit
2. Subtract the Divisor register from the left half of the Remainder register & Place the result in the left half of the Remainder register
Test Remainder
Remainder 0
Remainder < 0
3a. Shift the Quotient register to the left setting the new right-most bit to 1
3b. Restore the original value by adding the register to the left half of the Remainder register and Place the Sum in the left half of the Remainder register. Also Shift the Quotient register to the left, setting the new least significant bit to 0
nth repetition?
No: < n repetitions
Yes: n repetitions (n = 4 here)
Done

Algorithm for Non-Restoring Division:

Step1: a) If the sign of Remainder Register (Rem Reg) is 0, Shift Rem Reg left logically by one binary position and subtract the divisor from the left half of the Rem Reg and place the result in left half of Rem Reg. Otherwise, shift Rem Register left and add divisor to the left of the rem reg.

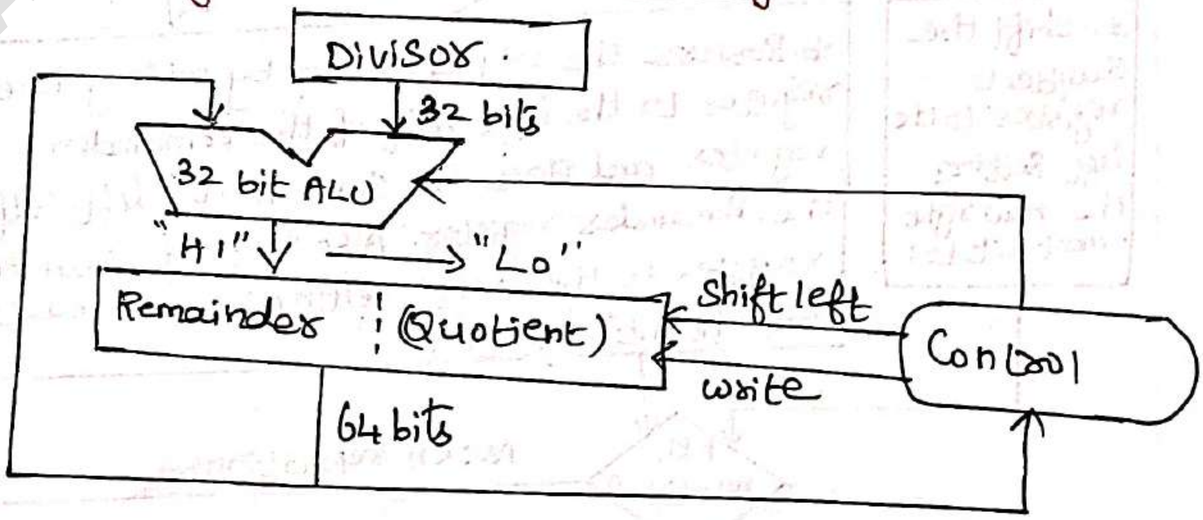
b) If the sign of Rem Reg is <0, set Q_0 to 0 otherwise set Q_0 to 1.

Step 2: Repeat steps 1 and 2 for < 33 times

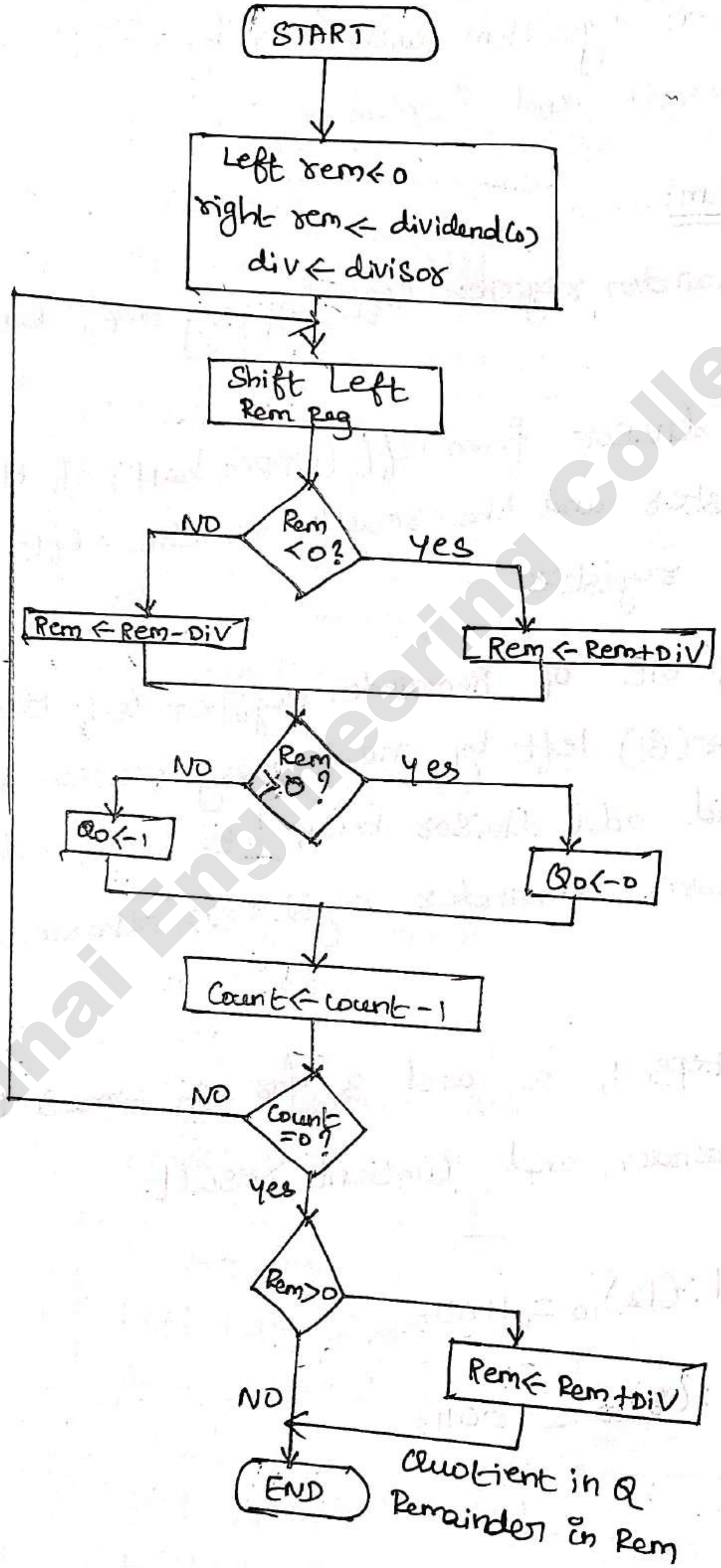
Step 3: At the end of n cycle, If the sign of Rem Reg is 1, add divisor to Rem Reg. This step is required to leave the proper positive remainder in Rem Reg at the end of n cycles.

Non-Restoring Division or Third version of Division Hardware for 32-bit Boolean number representations

Schematic Diagram of ALU Circuitry



Flowchart:


START
Left rem ← 0
right rem ← dividend(Q)
div ← divisor
Shift Left Rem Reg
Rem <0?
NO
yes
Rem ← Rem-DiV
Rem ← Rem+DiV
Rem ≥0?
NO
yes
Q0 ← 1
Q0 ← 0
Count ← count - 1
Count =0?
NO
yes
Rem>0
Rem ← Rem+DiV
NO
END
Quotient in Q
Remainder in Rem

Example:

Divide $(12)_{10}$ by $(3)_{10}$ Using the Restoring and Non-restoring division algorithm with step by step intermediate result and Explain

Restoring Algorithm:

Step1: Shift Remainder register left logically one binary position.

Step2: Subtract divisor from left (upper half) of the remainder register and the result in the left half of the Remainder register

Step3: If the sign bit of Remainder Register is 1, then shift Quotient Register(Q) left by one binary position and Set Q_0 to 0 and add divisor back to remainder register (ie restore remainder register); otherwise, Set Q_0 to 1

Step4: Repeat steps 1, 2 and 3 for <n times to obtain the remainder and quotient result

Dividend : $(12)_{10} = 1100_2$

Divisor : $(3)_{10} = 0011_2$

Iteration	Steps	Q (Quotient)	Remainder (Rem)	
			Left of the rem reg Set to 0	Dividend
0	Initial value	0000	0000	1100
1	Sll Rem Reg	0000	0001	1000
	Left half Rem reg ← left Rem - Divisor		0011	
			1110	
	Rem<0, restore Rem reg (+divisor), Sll Q, $Q_0=0$	0000	0001	1000
2	Sll Rem Reg	0000	0011	0000
	Left half Rem reg ← left Rem - Divisor		0011	
			0000	
	Rem>0, Sll Q, $Q_0=1$	0001	0000	0000
3.	Sll Rem Reg		0000	0000
	Left half Rem reg ← left Rem - Divisor		0011	
			1101	
	Rem<0, restore Rem reg (+divisor), Sll Q, $Q_0=0$	0010	0000	0000
4	Sll Rem Reg Left half Rem reg ← left Rem - Divisor	0010	0000 0011	0000
			1101	
	Rem<0, restore Rem reg (+divisor), Sll Q, $Q_0=0$	0100 Quotient $(4)_{10}$	0000 Remainder $(0)_{10}$	0000

Non-Restoring Algorithm:

Step 1: a) If the sign of Remainder Register (Rem Reg) is 0, shift Rem Reg left logically by one binary position and subtract the divisor from the left half of the Rem Reg and place the result in left half of Rem Reg. otherwise, shift Rem Register left and add divisor to the left of the rem reg.

b) If the sign of Rem Reg is 0, set Q_0 to 0, otherwise set Q_0 to 1

Step 2: Repeat steps 1 and 2 for <n times

Step 3: At the end of n cycle, If the sign of Rem Reg is 1, add divisor to Rem Reg. This step is required to leave the proper positive remainder in Rem Reg at the end of n cycles

Dividend : $(12)_{10} = 1100_2$

Divisor : $(3)_{10} = 0011_2$

Iteration	Steps	Remainder Register (Rem Reg)	
		Left Of Rem Reg Set to 0	Righ Half Of Rem Reg Dividend/Q (Quotient)
0	Initial values	0000	1100
1	Sll Rem Reg	0001	$100Q_0$
	Left half Rem reg ← left Rem − Divisor	0011	
		1110	
	Rem < 0, $Q_0 = 0$	1110	1000
2	Sll Rem Reg	1101	$000Q_0$
	Left half Rem reg ← left Rem + Divisor	0011	
		0000	
	Rem > 0, $Q_0 = 1$	0000	0001
3	Sll Rem Reg	0000	$001Q_0$
	Left half Rem reg ← left Rem − Divisor	0011	
		1101	
	Rem < 0, $Q_0 = 0$	1101	0010
4	Sll Rem Reg	1010	$010Q_0$
	Left half Rem reg ← left Rem + Divisor	0011	
		1101	
	Rem < 0, $Q_0 = 0$	1101	0100
		Sign of Rem Reg = result in negative. Restore: Remainder add with divisor 1101 + 0011 = 0000 Remainder $(0)_{10}$	Quotient $(4)_{10}$

A standard scientific notation for reals in normalised form offers 3 advantages.

* It simplifies Exchange of data that includes floating point numbers.
* Simplifies the FP arithmetic algorithms to know that no's will always be in this form.
* It increases the accuracy of the numbers that can be stored in a word.

Floating point Representation:

Fraction:

The value generated b/w 0 and 1, placed in the fraction field.

Exponent:

- In numerical representation system of floating point arithmetic, the value is placed in the Exponent field.

* Floating point numbers are usually a Multiple of size of a word.
* Increasing the size of a fraction, Increases the precision of the fraction
* Increasing the size of a Exponent, increases the Range of numbers that can be represented.

Two types of Representation:

* Single Precision format
* Double Precision format.

Floating Point Representation

fraction:

The value generated b/w 0 and 1, placed in the fraction field.

Exponent:

- In numerical representation system of floating point arithmetic, the value is placed in the Exponent field.

* floating point numbers are usually a Multiple of size of a word.

* Increasing the size of a fraction, Increases the precision of the fraction.

* Increasing the size of a Exponent, increases the Range of numbers that can be represented.

Two types of Representation:

* Single precision format
* Double precision format.

Representation of MIPS floating point numbers | Single Precision format:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	...	2	1	0

S	Exponent	Fraction.
1 bit	8 bits	23 bits.

S ⇒ Sign of the floating point number
(1 ⇒ Negative, 0 ⇒ Positive).

E ⇒ Exponent, 8-bit field.

F ⇒ Fraction, 23-bit number.

In general, floating point number are generally of the form

$$(-1)^S \times F \times 2^E$$

F ⇒ Value in fraction field

E ⇒ Value in Exponent field.

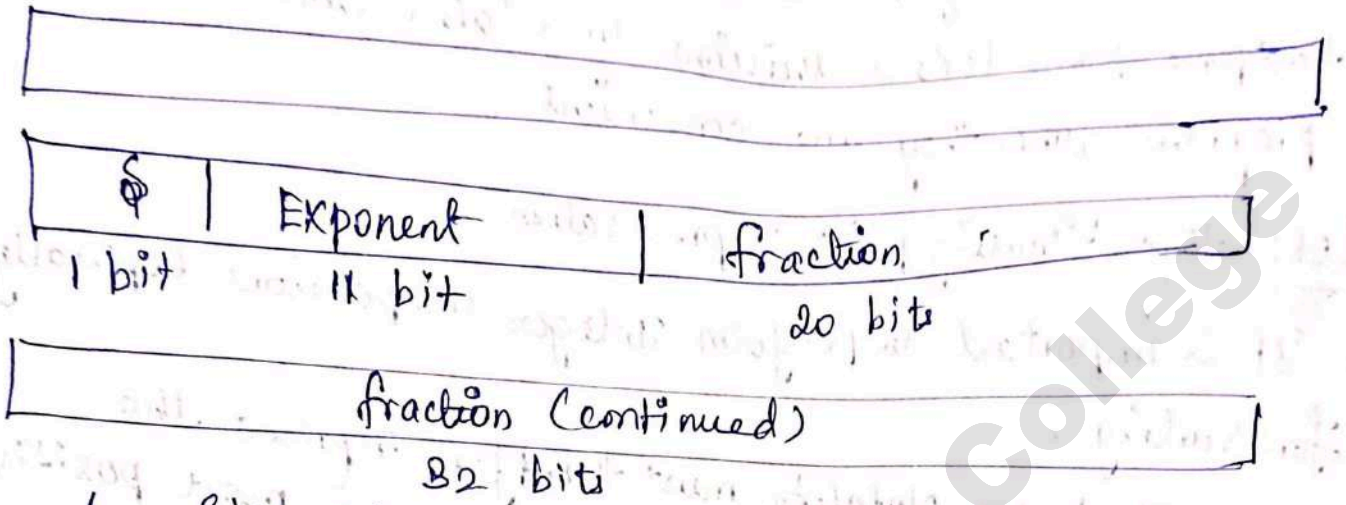
* size of Exponent & fraction gives MIPS computer arithmetic an Extra Ordinary range, which cause the overflow interrupts.

Overflow:
A situation in which a positive Exponent becomes too large to fit in the Exponent field.

Underflow:
A situation in which a Negative Exponent becomes too large to fit in the Exponent field.

Double precision:

- floating point value represented in ~~2~~ 2 MIPS words.



S - Sign of the number.

E ⇒ Value of 11-bit Exponent field.

F ⇒ Fraction 52-bit no's is the fraction

Adv:

- Double precision is a greater precision because of much larger fraction.

IEEE 754 floating point standard

- This standard was found in virtually every computer since 1980.
- This standard improved the Quality of computer arithmetic and Ease of porting fp. programs.
- This standard used to pack even more bits into the significant
- It makes the leading 1 bit of normalized binary no.
  - * the number is actually 24-bits long in single precision.
  - * 53-bits long in Double precision (1+52)

- It has Special Symbols to represent unusual events (∞)
- It has Symbol for the result of invalid Operation (NaN)

* The purpose of NAN's is to allow programmers to Postpone Some tests & decisions to a later time in the program when they are convenient.

IEEE 754 floating point Representation :

- It is important to perform Integer comparisons especially for sorting.

* The Desirable Notation must therefore represent the Most negative Exponent as $00\dots00_2$ and most positive Exponent as $11\dots11_2$ this convention is called biased notation

* with the bias being the Number subtracted from the normal unsigned representation to determine the real value.

* IEEE 754 Uses a bias of 127 for single precision

So, -1 is represented by the bit pattern of the value :

$-1 + 127_{10}$ (or) $126 = 01111110_2$

$+1 \Rightarrow +1+127$. (or) $128 = 1000\,0000_2$.

* Biased Exponent Means that the Value represented by a floating point number is really,

$$\boxed{(-1)^S \times (1+\text{fraction}) \times 2^{(\text{Exponent} - \text{Bias})}}$$

- Exponent Bias for Double precision is 1023

$$(-1)^s \times (1 + (S_1 \times 2^{-1}) + (S_2 \times 2^{-2}) + (S_3 \times 2^{-3}) + \dots) \times 2^E$$

Floating Point Operations

- Numbers with fractions .

$3.14159265 \ldots (\pi)$

$2.71828 \ldots (e)$

0.0000001 (or) 0.1×10^{-7} .

0.000000001 (or) 0.1×10^{-9} (seconds in ns)

3.15576000 (or) 3.15576×10^{9}.

Scientific Notation :

- A Notation that renders numbers with a single digit to the left of the decimal point

Eg: $11_{two} / 2^{2} \Rightarrow 0.11_{two} \times 2^{0}$

- A Number is scientific Notation that has No. leading 0's

Normalised

- To keep a binary number in normalized form, a base value is needed so that we can increase or decrease by exactly the No. of bits the number must be shifted to have one non-zero digit to the left of the decimal point .

1.1×2^{1} .

Floating Point :

- Computer arithmetic that represents numbers in which the binary point is not fixed.

Representation of MIPS floating point numbers | Single Precision format:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	...	2	1	0

S	Exponent	Fraction .
1 bit	8 bits	23 bits .

S ⇒ Sign of the floating point number
(1 ⇒ Negative , 0 ⇒ Positive).

E ⇒ Exponent , 8-bit field.

F ⇒ Fraction , 23-bit number.

In general, floating point number are generally of the form

$$(-1)^S \times F \times 2^E$$

F ⇒ Value in fraction field

E ⇒ Value in Exponent field.

* size of Exponent & fraction gives MIPS computer arithmetic as Extra Ordinary range , which cause the overflow interrupts .

Overflow:

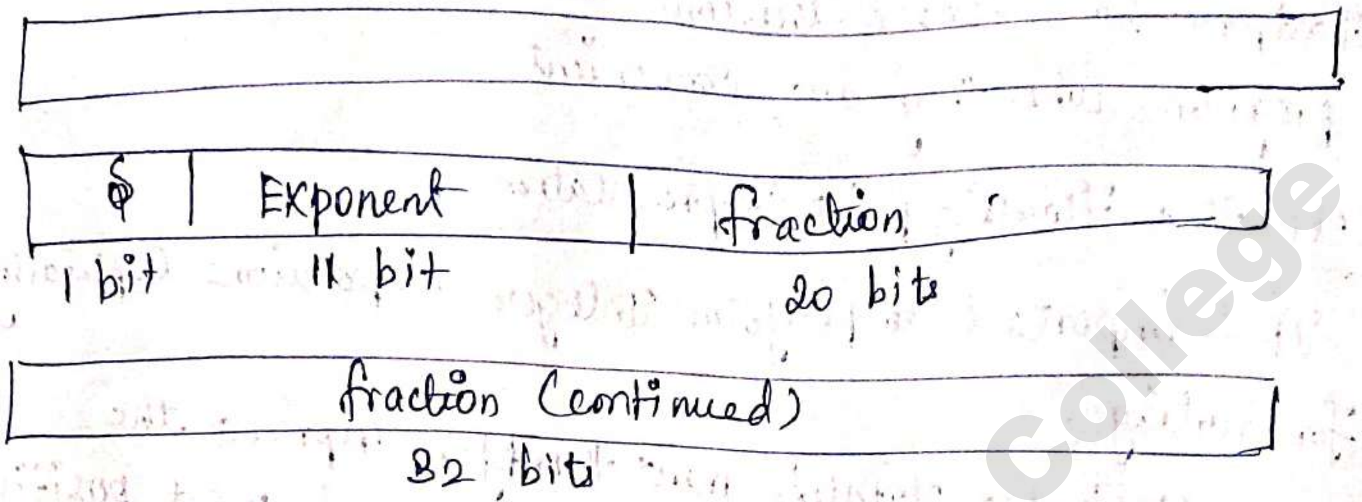
A situation in which a positive Exponent becomes too large to fit in the Exponent field.

Underflow:

A situation in which a Negative Exponent becomes too large to fit in the Exponent field.

Double precision:

- floating point value represented in 2 Mips words.



S - Sign of the number.

E ⇒ Value of 11-bit Exponent field.

F ⇒ Fraction 52-bit no's in the fraction

Adv:

- Double precision is a greater precision because of much larger fraction.

IEEE 754 floating point standard

- This standard was found in virtually every computer since 1980.
- This standard improved the Quality of computer arithmetic and Ease of porting fp. programs.
- This standard used to pack even more bits into the significant
- It makes the leading 1 bit of normalized binary no.
  - * the number is actually 24-bits long in Single precision.
  - * 53-bits long In Double precision (1+52)

It has special symbols to represent unusual events (∞)

- It has symbol for the result of invalid operation (NaN)

* The purpose of NAN's is to allow programmers to postpone some tests & decisions to a later time in the program when they are convenient.

IEEE 754 Floating point Representation

- It is important to perform Integer comparisons especially for sorting.

* The Desirable Notation must therefore represent the most negative Exponent as $00\ldots00_2$ and most positive Exponent as $11\ldots11_2$ this convention is called biased notation

* with the bias being the Number subtracted from the normal unsigned representation to determine the real value.

* IEEE 754 Uses a bias of 127 for single precision

So, -1 is represented by the bit pattern of the value:

$-1+127_{10}$ (or) $126 = 01111110_2$

$+1 \Rightarrow +1+127$ (or) $128 = 10000000_2$.

* Biased Exponent means that the value represented by a floating point number is really.

$$\boxed{(-1)^S \times (1+\text{fraction}) \times 2^{(\text{Exponent} - \text{Bias})}}$$

• Exponent Bias for Double precision is 1023

$$(-1)^S \times (1+(S_1 \times 2^{-1}) + (S_2 \times 2^{-2}) + (S_3 \times 2^{-3}) + \ldots) \times 2^E.$$

Subword Parallelism

* Parallelism will improve the performance of computer.

* A subword is a lowerprecision unit of data contained within a word.

* In subword parallelism, pack multiple subwords into a word and then process whold word with the appropriate subword boundaries.

* This technique results in parallel processing of subwords.

* Subword parallelism also known as data level parallelism or vector or SIMD (single Instruction Multiple Data) processing.

* Since the same instruction applies to all the subwords within the word, this is a form of small scale SIMD processing.

* It is possible to apply subword parallelism to non contiguous subwords of different sizes within a word.

* In practice, however, implementations are much simpler if we allow only a few subword size and if a single instruction operates on contiguous

Subwords that are all of the same size.

*Data parallel programs that benefit from subword parallelism tend to process data that are of the same size.

Eg If the word size is 64 bits, some useful subword sizes are 8, 16 and 32bits

* An instruction operates on 8bit subwords, four 16-bit subwords, two 32-bit subwords or one 64-bit subword in parallel.

* Data parallelism refers to an algorithm execution of the same program module on different sets of data.

* Subword parallelism is an efficient and flexible solutions for media processing, because the algorithms exhibit a great deal of data parallelism on lower precision data.

Advantage:

1. It allows general-purpose processors to exploit wider word sizes even when not processing high-precision data.

2. The processor can achieve more sub-word parallelism on lower precision data rather than wasting much of the word-oriented data paths and registers.

UNIT 3
PROCESSOR AND CONTROL UNIT

Basic MIPS Implementation

Basic MIPS Implementation:

* In examining the implementation we can get more information to increase the processor performance. Through the implementation the following things can be identified by the user.

1. How the instruction set architecture determines?
2. How various implementation strategies affect the clock rate?
3. How implementation affects the CPI cycles per instruction of the computer?

* MIPS has three kinds of core instructions such as

1. The memory-reference instructions→load word (lw) and store word (sw)
2. The arithmetic-logic instructions→add, sub, AND, OR and slt
3. Branch Instructions→Jump (j) and branch equal (beq)

TO implement the above three types we have Same method but independent of the exact Class of instruction.

* For every instruction, the first two steps are identical.

1. Send the program Counter(PC) to memory that Contains the code and fetch the instruction from that memory.

2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require that we Read two registers

* These two steps are common for all the instruction set.

* After these two steps, the action required to complete the instruction depend on the instruction class.

↳ MIPS instruction set has Simplicity and regularity and it will Simplify the implementation by making the execution of many of the instruction Classes Similar.

Example:

* All instruction classes, except jump, use the Arithmetic logical Unit (ALU) after reading the registers.

* The memory reference instruction use the

↳ ALU for an address calcultion

↳ Arithmetic logical Instructions for the operation execution

↳ Branches for comparison.

* After using the ALU, the actions required to complete the task differs from various instruction classes.

* A memory reference instruction will need to access the memory either to read data for a load or write data for a store.

* An arithmetic logical or load instruction must write the data from the ALU or memory back into a register.

* Branch instruction need to change the next instruction address based on the comparison, otherwise the PC should be incremented by 4 to get the address of the next instruction.

* All instructions start by using the program counter to supply the instruction address to the instruction memory.

* After the instruction is fetched, the register operands used by an instruction are specified by fields of the instruction.

* once the register operands have been fetched, they can be operated to do the following tasks.

1. To compute a memory address
2. To compute an arithmetic result
3. To compare for a branch.

* If the instruction is arithmetic logical instruction then the result from the ALU must be written to a register.

* If the operation is a load or store, the ALU result is used as an address to either store a value from the register or load a value from memory into the registers.

* The result from the ALU or memory is written back into the register file.

*Branch instruction require the use of the ALU output to determine the next instruction address, which comes either from the ALU or from an adder that increments the current PC by 4.

*It Shows most of the flow of data through the processor but it omits two important aspects of instruction execution.

1. It Shows data going to a particular unit as coming from two different sources.

2. several of the units must be controlled depending on the type of instruction

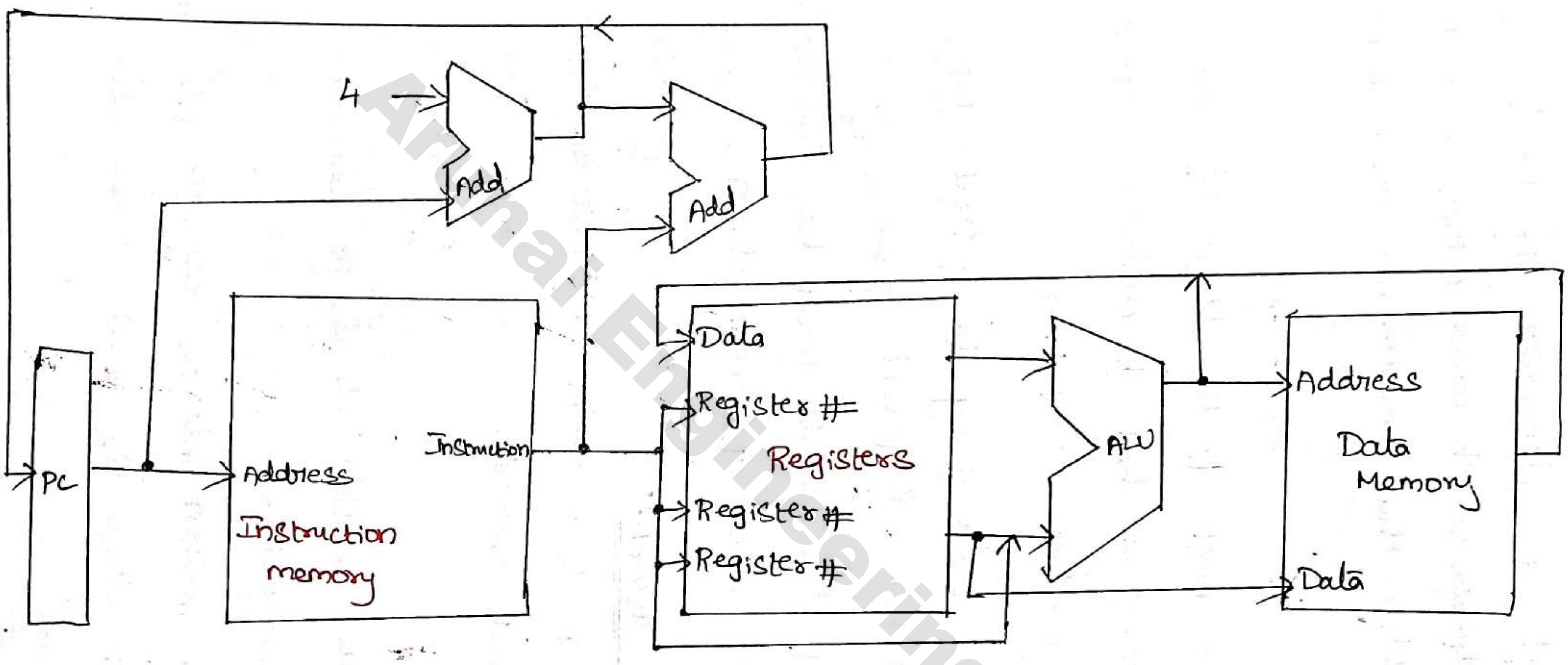
Problems in first aspect:

* Data going to a particular unit as coming from two different sources.

Example:

The value written into the PC can come from one of two address.

The data written into the register file can come come from either the ALU or the data memory.



An abstract view of the Implementation of the MIPS subset showing the major functional units and the major connection between them

Thicklines - Interconnecting the functional units represent buses, which consist of multiple Signals

Arrows -> To guide the reader in knowing how information flows

* The second input to the ALU can come from a register or the immediate field of the instruction.

* These data lines cannot simply be wired together for practical purpose. So we must add a logic element that chooses from among the multiple sources and move one of those sources to its destination.

* This selection is commonly done with a device called a Multiplexer and it is also called data Selector.

* Multiplexer is a combinational circuit which selects from among several inputs based on the setting of its control lines.

* The control lines are set based on information taken from the instruction being executed.

Problem in second aspect :

* In that figure several of the units must be controlled depending on the type of instruction.

* The data memory must read on a load and write on a store.

* The register file must be written only on a load or an arithmetic logical instruction.

* ALU must perform several operations. TO overcome these problems we can use another circuit with Some modification made in previous circuit

* The basic Implementation of MIPS required three multiplexers and one major unit as Control Lines

Control Unit:

* control unit has the instruction as an input and it is used to determine how to set the control lines for the functional Units and two of the multiplexers

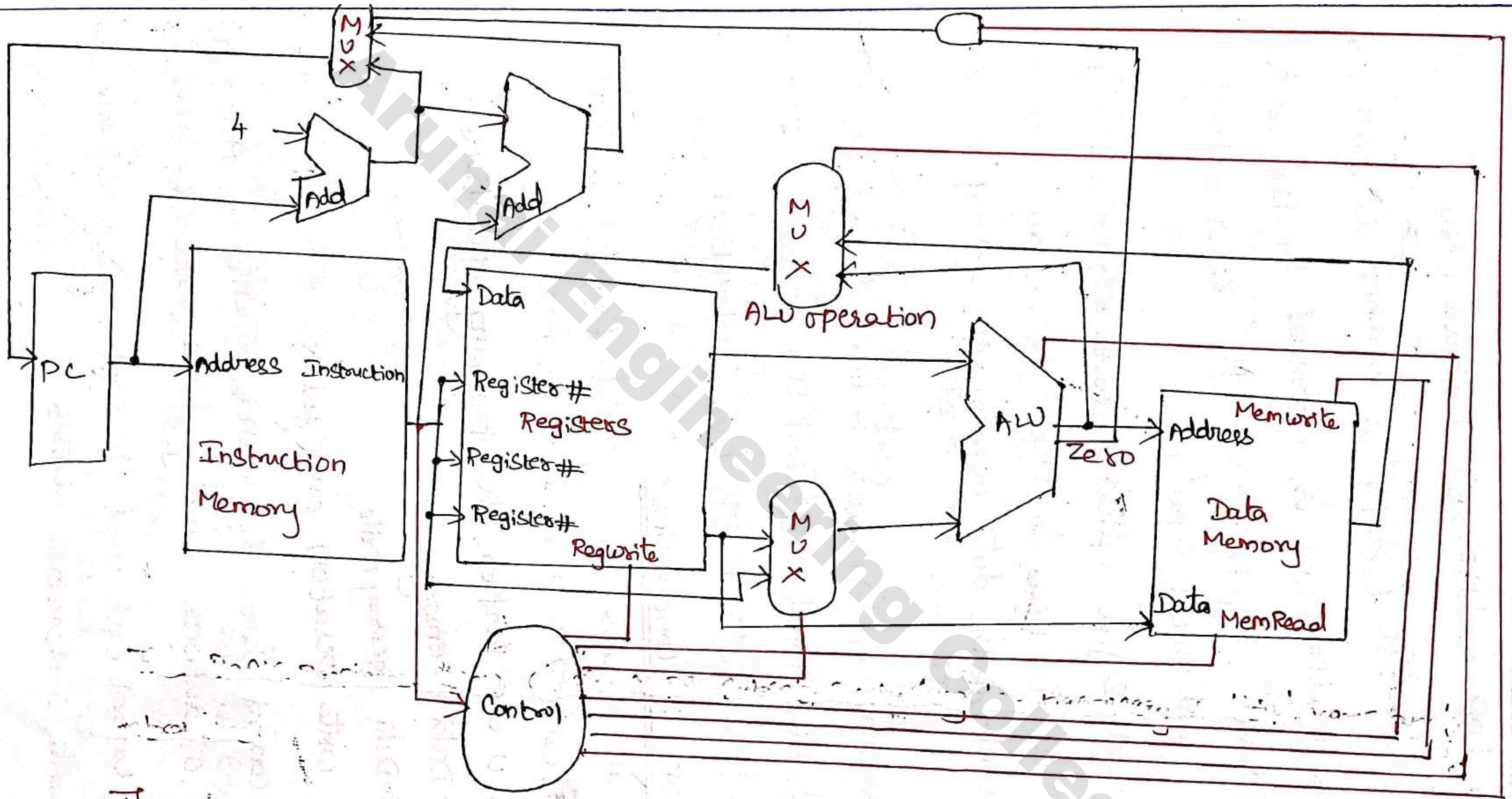
Function of third Multiplexer:

* Third multiplexer determines whether PC+4 or branch destination address is written into the PC.

* It is set based on the zero output of the ALU and it is used to perform the comparison of a beq, instruction

* The regularity and simplicity of the MIPS instruction set a simple decoding process.

* Simple decoding process used to determine how to set the control line.



The Basic Implementation of the MIPS subset, Including the necessary Multiplexers and control lines

Functions of the Multiplexers:

* The top multiplexer controls what the value replaces the PC [PC+4 or branch destination address]

* The multiplexer is controlled by the gate that "AND" together with the zero output of the ALU and control signal.

* control signal indicates the branch instruction. Middle multiplexer is used to steer the output of the ALU or the output of the data memory for writing into a register file.

* Bottom multiplexer is used to determine the second ALU input is from the registers or from the offset field of the instruction

Function of control line:

* Control lines are straight forward and determine the operation performed at the ALU

ALU can perform the following operations

1. Data memory read
2. Data memory write
3. write operation on register

* control line determines whether the ALU perform which operations among three mentioned operations

* Control unit used to control actions taken for different Instruction classes.

Building Datapath

* To consider three kinds of instruction classes, so far such as

1. R-type instructions
2. Load and store instructions
3. Branch instruction.

*It is important to know the flow of control through the data path for these three different instruction classes

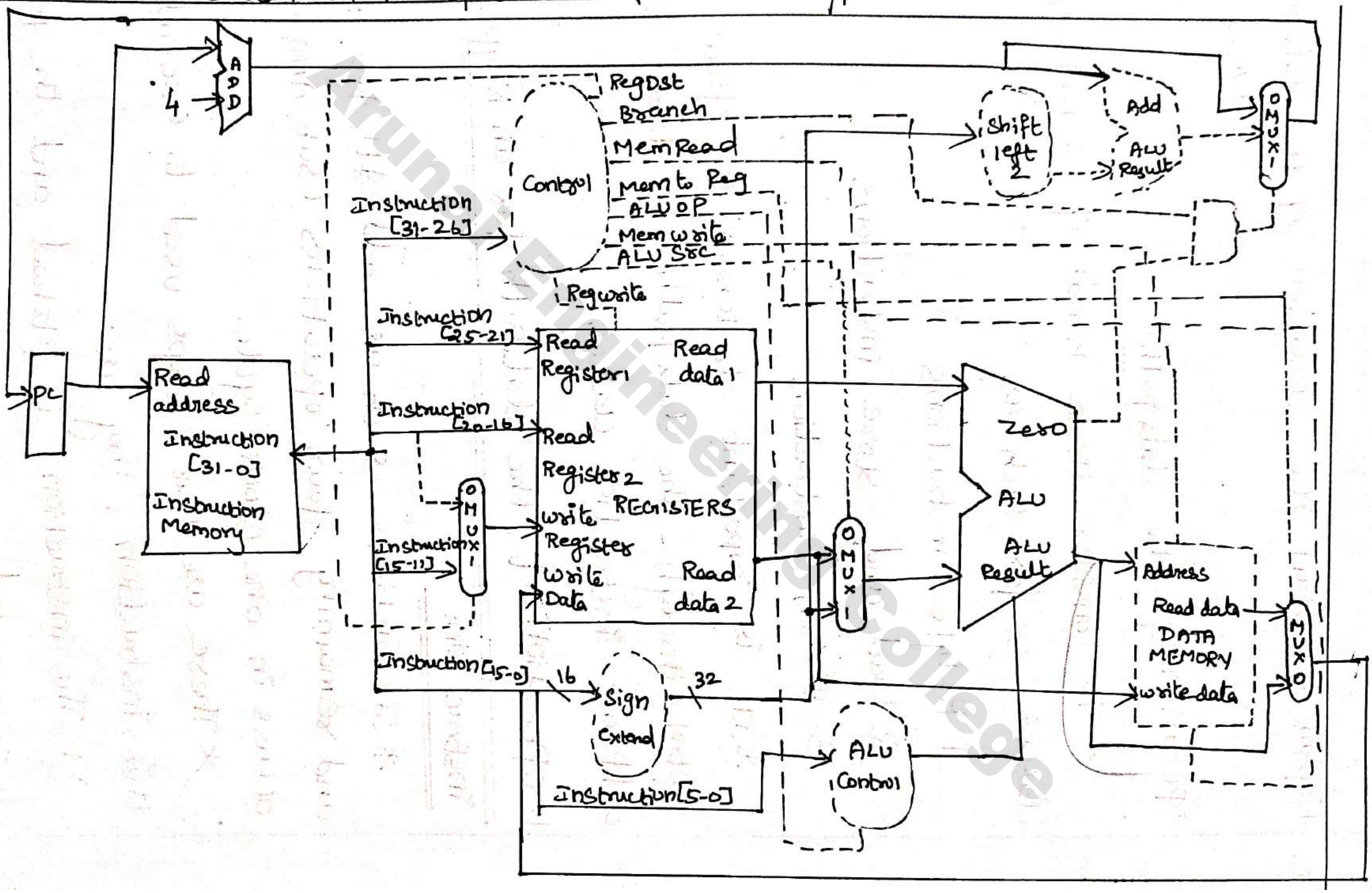
1. Operation of the datapath for an R-type instruction:

* Instructions such as add $t1, $t2, $t3 and remaining four operations (sub, AND, OR, $lt) occurs in one clock cycle.

* There are four steps used to execute this instruction. They are:

1. The instruction is fetched and the pc is incremented.

Data Path in operation for an R-type Instruction



2. Two registers, $t2 and $t3, are read from the register file.

3. The ALU operates on the data read from the register file, using the function code to generate the ALU function

4. The result from the ALU is written into the register file in the destination register ($t1)

2. operation of the datapath for a load word Instruction.

* Instructions such as lw $t1, offset ($t2) comes under this category.

* Five steps involved in execution of load instruction.

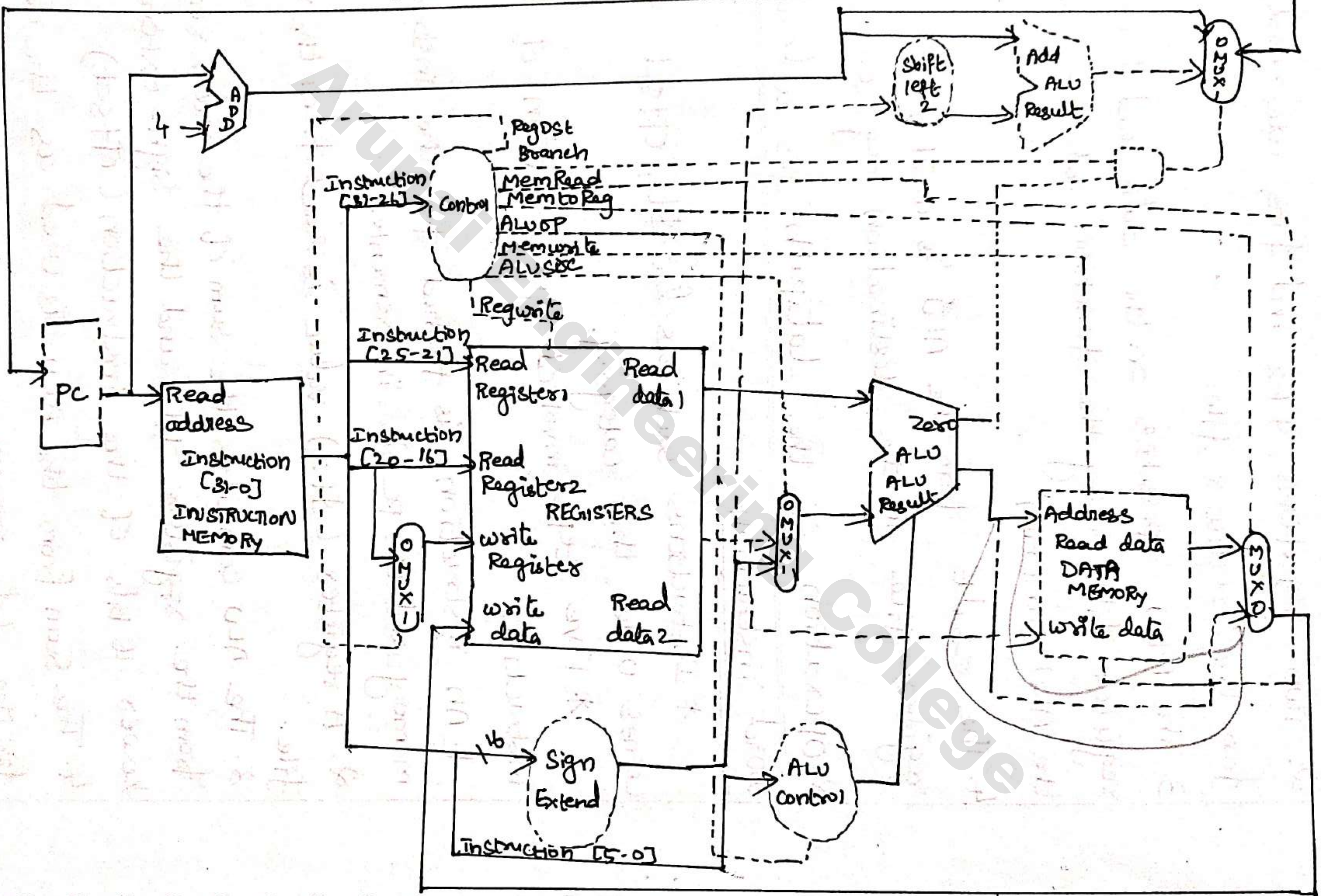
1. An instruction is fetched from the instruction memory and the PC is incremented.

2. A register ($t2) value is read from the register file.

3. The ALU computes the sum of the value read from the register file and the sign-extended, lower 16 bits of the instruction (offset)

4. The sum from the ALU is used as the address for the data memory.

Data path operation for load instruction



5. The data from the memory unit is written into the register file in the destination register ($t1)

3. operation of the datapath for branch-on-equal instruction.

* Instructions such as beq $t1, $t2, offset comes under this category. Four steps are needed to execute the instruction. They are

1. An instruction is fetched from the instruction memory and the pc is incremented.

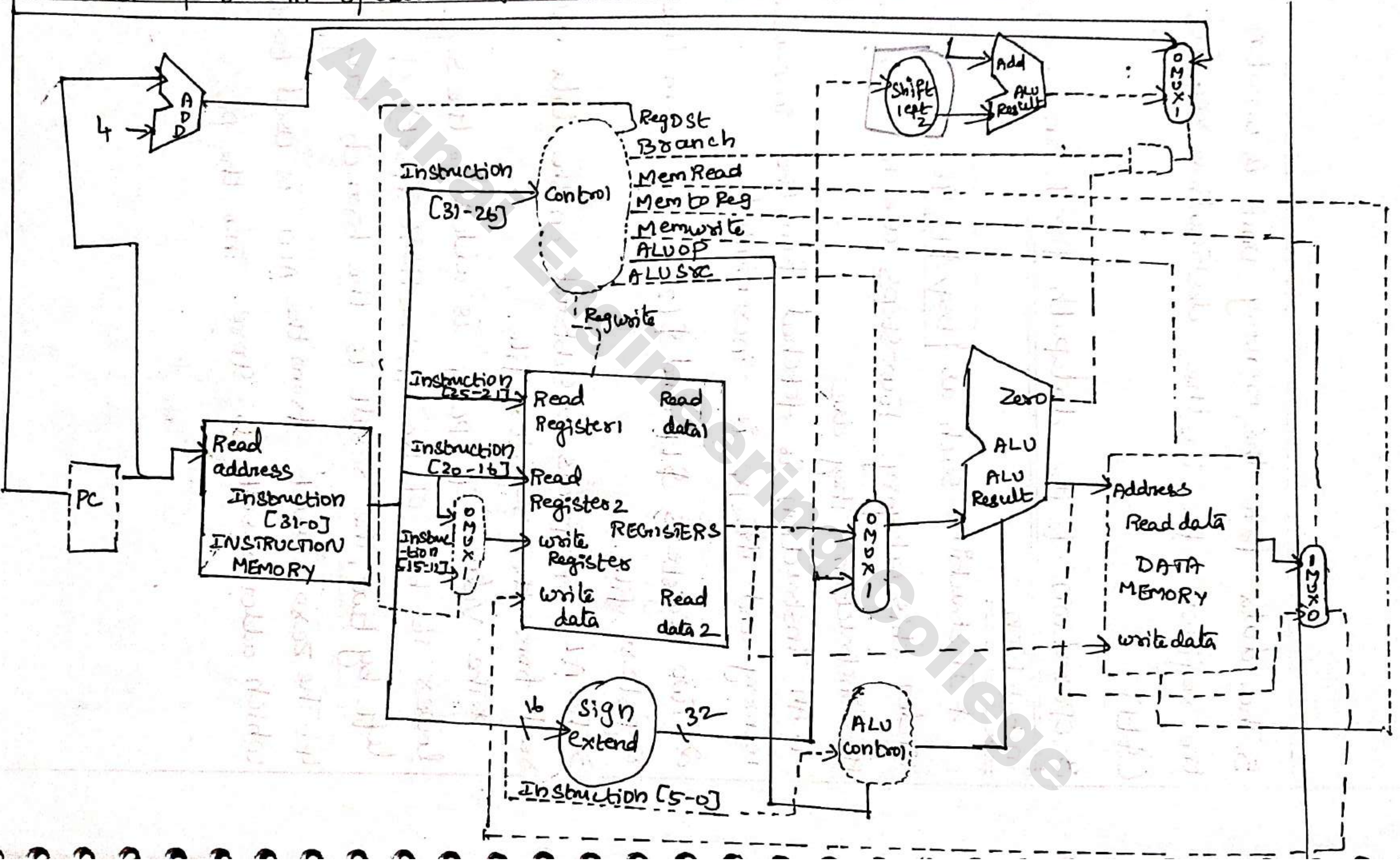
2. Two registers, $t1 and $t2 are read from the register file.

3. The ALU performs a subtract on the data values read from the register file.

*The value of PC+4 is added to the sign-extended, lower 16 bits of the instruction(offset) shifted left by two, the result is the branch target address.

4. The zero result from the ALU is used to decide which adder result to store into the PC.

Data path in operation for branch on equal instruction



Control Implementation Schema

operation of the data path:

* To consider three kinds of instruction classes, so far such as

1. R-type instructions
2. Load and store instructions
3. Branch instruction.

* It is important to know the flow of control through the data path for these three different instruction classes

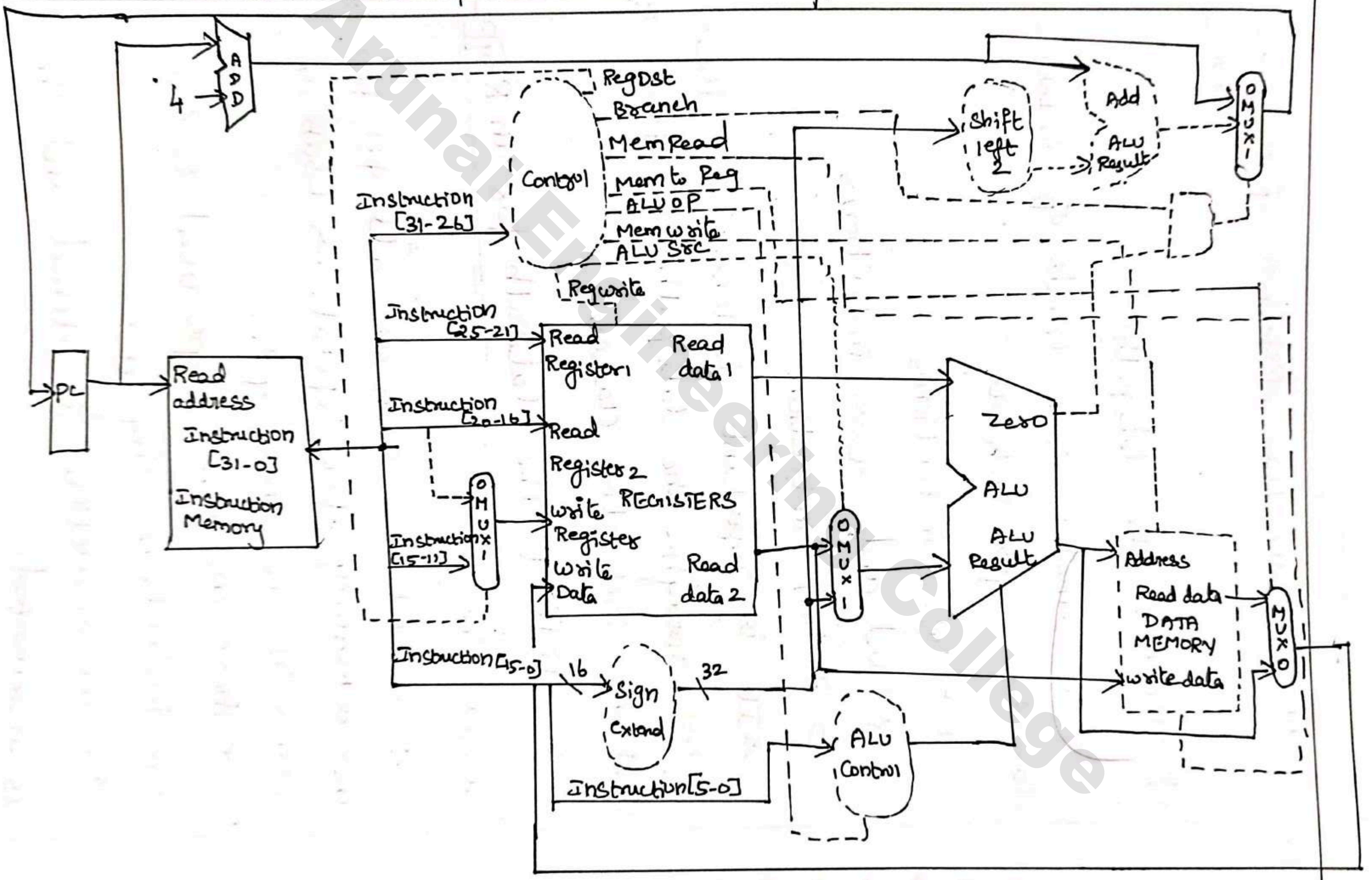
i. operation of the datapath for an R-type instruction:

* Instructions such as add $t1, $t2, $t3 and remaining four operations (sub, AND, OR, slt) occurs in one clock cycle.

* There are four steps used to execute this instruction, They are:

1. The instruction is fetched and the PC is incremented.

Data Path in operation for an R-type Instruction



2. Two registers, \$t2 and \$t3, are read from the register file.

3. The ALU operates on the data read from the register file, using the function code to generate the ALU function

4. The result from the ALU is written into the register file in the destination register (\$t1)

2. operation of the datapath for a load word Instruction.

* Instructions such as lw \$t1, offset (\$t2) comes under this category.

* Five steps involved in execution of load instruction.

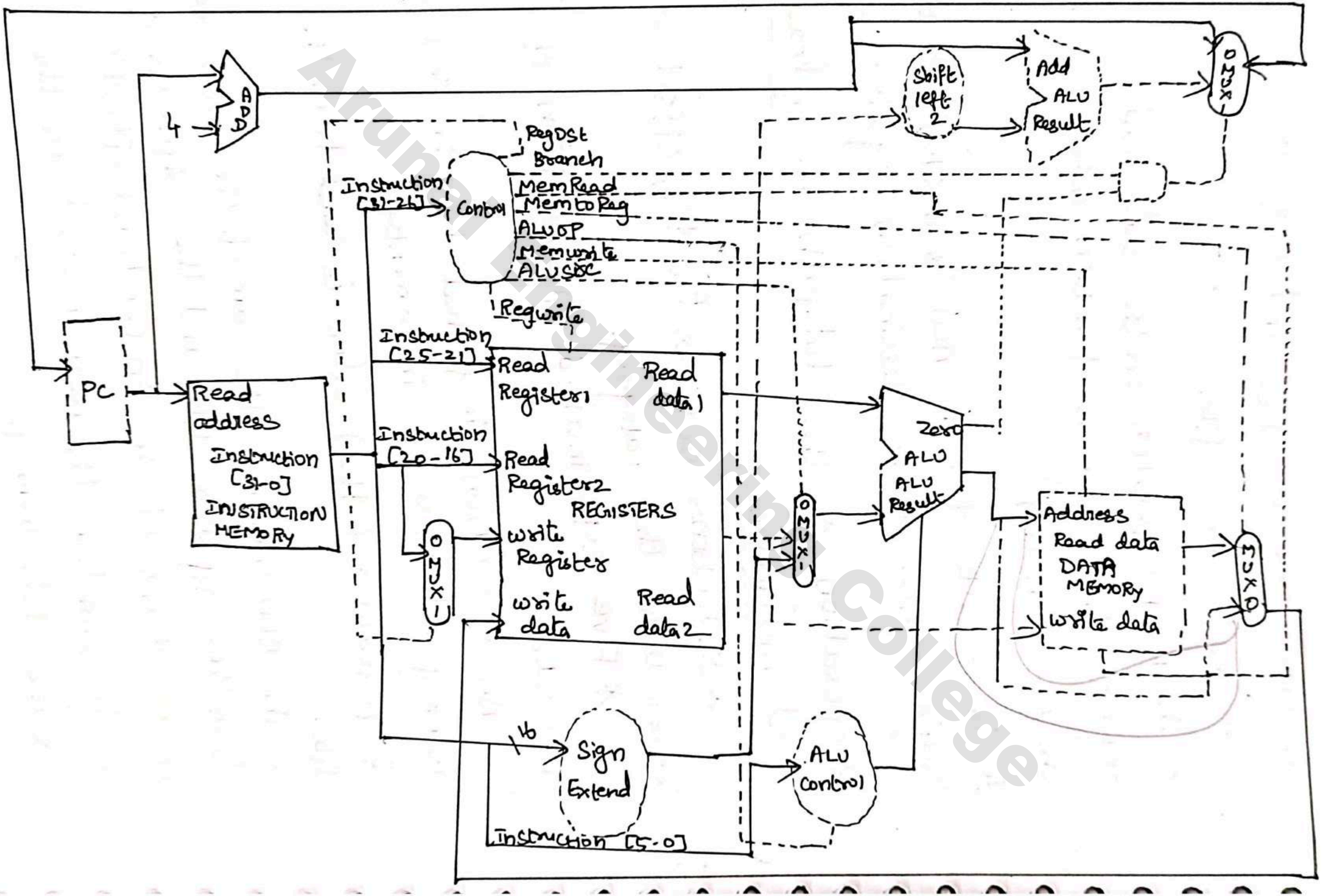
1. An instruction is fetched from the instruction memory and the PC is incremented.

2. A register (\$t2) value is read from the register file.

3. The ALU computes the sum of the value read from the register file and the sign-extended, lower 16 bits of the instruction (offset)

4. The sum from the ALU is used as the address for the data memory.

Data path operation for load instruction



5. The data from the memory unit is written into the register file in the destination register ($t1)

3. operation of the datapath for branch-on-equal instruction.

* Instructions such as beq $t1, $t2, offset comes under this category. Four steps are needed to execute the instruction. They are

1. An instruction is fetched from the instruction memory and the pc is incremented.

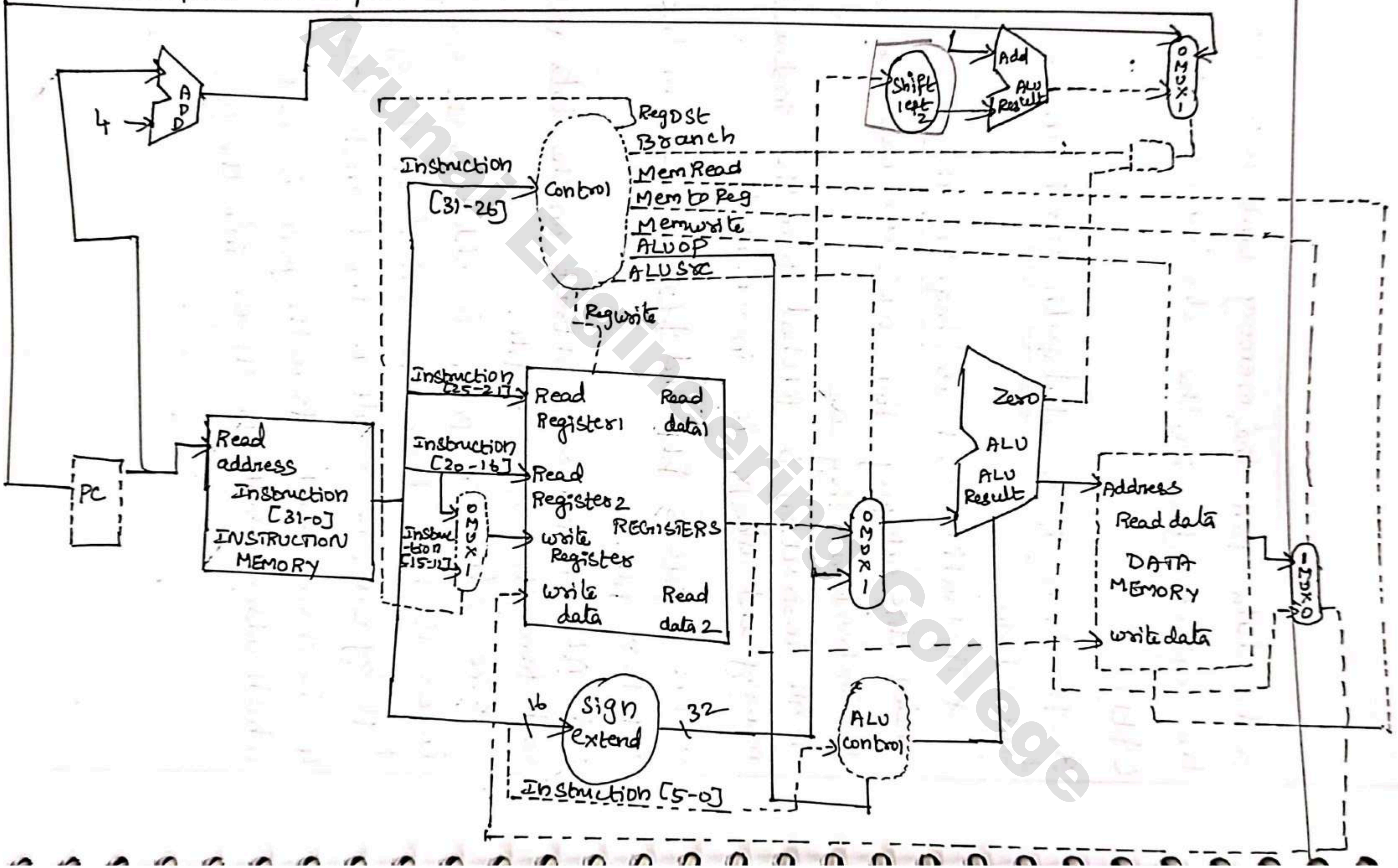
2. Two registers, $t1 and $t2 are read from the register file.

3. The ALU performs a subtract on the data values read from the register file.

* The value of PC+4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two, the result is the branch target address.

4. The zero result from the ALU is used to decide which adder result to store into the pc.

Data path in operation for branch on equal instruction



Pipelining

Pipelining:

* Pipelining is an implementation technique in which multiple instructions are overlapped in execution.

* Pipelining is most important technique used to increase the speed and Performance of the processor.

Example:

Consider the laundry process through we get the basic idea about technique.

Case1: Non-pipelined approach to laundry

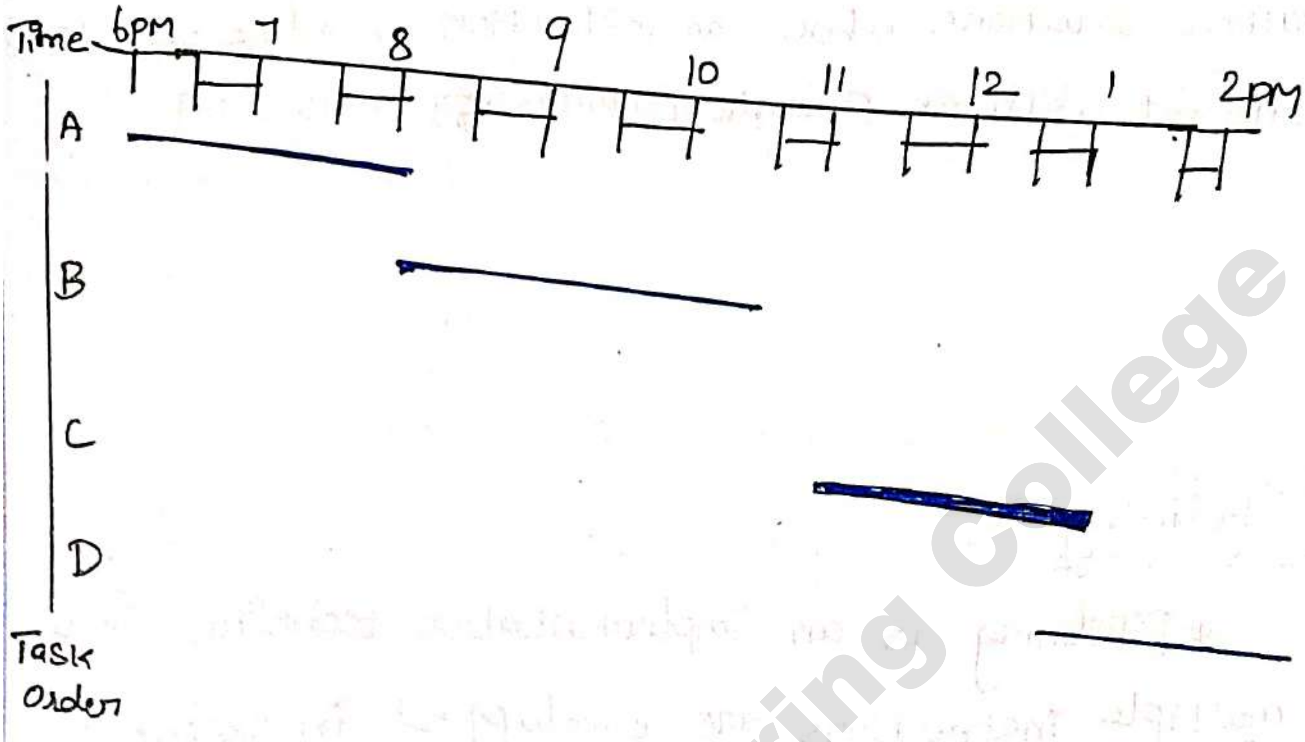
1. place one dirty load of clothes in the washer

2. when the washer is finished, place the wet load in the dryer

3. when the dryer is finished, place the dry load on a table and fold

4. when folding is finished, ask your roomate to Put the clothes away.

when your roommate is done, start over with the next dirty load



Case 2: Pipelined approach to laundry:

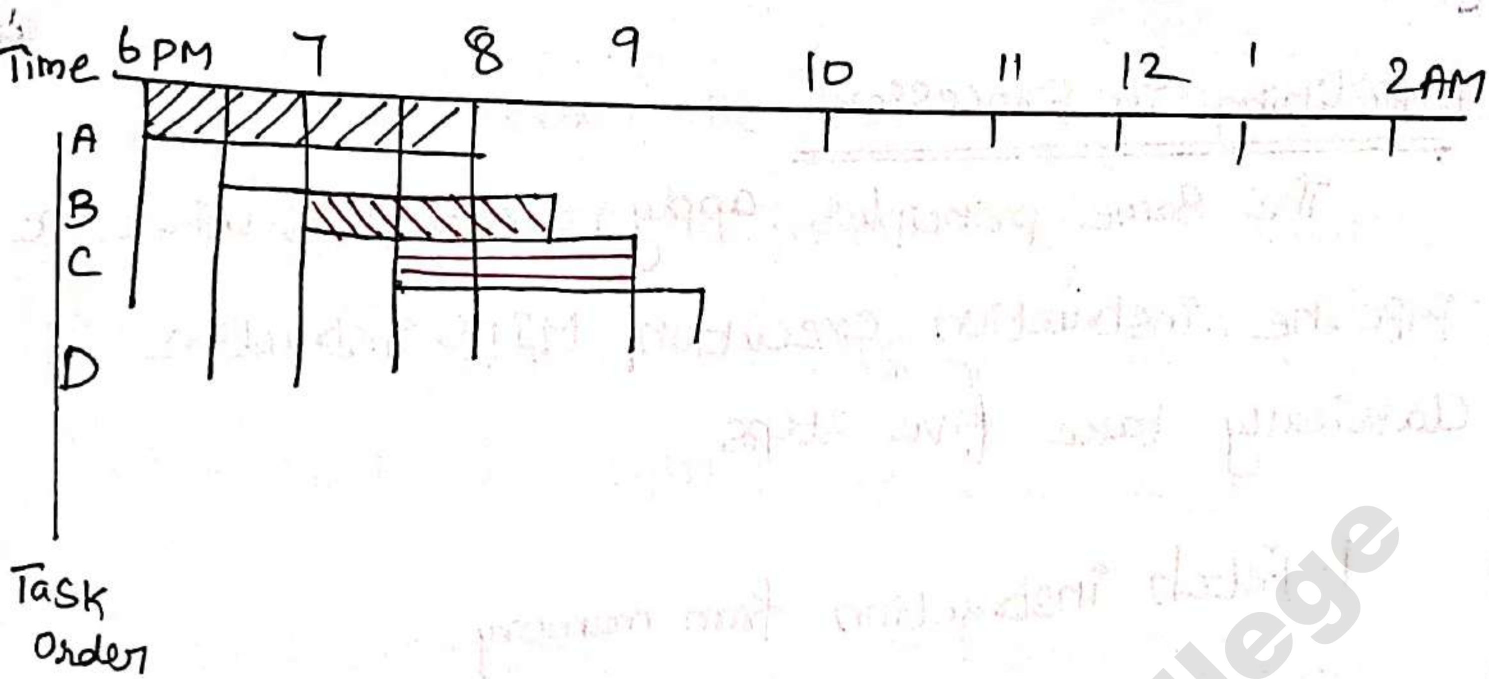
1. Place dirty load of clothes in the washer

2. As soon as the washer is finished, place in the dryer.

3. Load the washer with the second dirty load

4. First load is dry then move the wet load to the dryer and put the next dirty load into the washer.

5. Roommate put the first load away, you start folding the second load, the dryer has the third load and you put the fourth load into the washer



*These two processes is explained below

↳ A, B, C and D indicates the name of the Person.

↳ Each has dirty clothes to be washed, dried, folded and Put away.

↳ Laundry task has four stages washing, drying, folding and Putting away.

↳ Each task takes 30 minutes to complete it. So non-Pipelining laundry Process takes 8 hours for 4 loads of wash while Pipelined laundry takes just 3.5 hours

↳ Pipelined laundry is Potentially four times faster than non-pipelined laundry Process.

Pipelining in Processor:

The same principles apply to processors where we pipeline instruction execution. MIPS instructions classically take five steps

1. Fetch instruction from memory

2. Read registers while decoding the instruction. The regular format of MIPS instructions allows reading and decoding to occur simultaneously

3. Execute the operation or calculate an address

4. Access an operand in data memory.

5. write the result into a register.

Single cycle Vs pipelined Performance:

Let us assume operation times for major functional units.

↳ Memory access 200 ps

↳ ALU operation 200 ps

↳ Register file read or write 100 ps

In single clock cycle Model - Each Every instruction takes exactly in one clock cycles, so it will produce the slow speed to execute the Instruction.

↳ The time between the first and fourth Instruction in the non-pipelined design is 3x800 or 2400 PS

↳ The time between the first and fourth Instruction in the pipelined design is 3x200 or 600 PS

$$\text{Time between instruction}_{\text{pipelined}} = \frac{\text{Time between instruction}_{(\text{nonpipelined})}}{\text{Number of Pipe stages}}$$

* Pipelining improves performance by increasing instruction throughput and decreasing the execution time of an individual instruction.

Designing Instruction Sets for pipelining:

Design of the MIPS instruction set, which was designed for pipelined execution.

Some Important factor Such as

1. Length of the Instruction
2. Instruction format
3. Memory operands
4. Operand alignment

1. All MIPS instructions are the Same length. This restriction makes it much easier to fetch instructions in the first pipeline stage and to decode them in the Second stage

2. MIPS has only a few instruction formats, with the Source register fields being located in the Same place in each instruction.

Due to this Symmetry the second stage can begin reading the register file at the same time that the hardware is determining what type of instruction was fetched.

3. Memory operands only appear in loads or stores in MIPS. Due to this restriction we can use the execute stage to calculate the memory address and then access memory in the following stage.

4. Since operands are aligned in memory, data can be transferred between Processor and memory in a Single pipeline stage.

Pipelining Datapath

* There are five stages available in pipelining data path.

1. IF - Instruction fetch
2. ID - Instruction Decode and Register file Read
3. Ex - Execution or address calculation
4. MEM - Data Memory Access
5. WB - Write Back.

Handling Data Hazards And Control Hazards

Pipeline Hazards:

* There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards

Types of Hazards:

1. Structural Hazards
2. Data Hazards
3. Control Hazards

1. Structural Hazards

* The first hazard is called a structural hazard. These hazards are because of conflicts due to insufficient resources when even with all possible combination, it may not be possible to overlap the operation.

* The performance of pipelined processor depends on whether the functional units are pipelined and whether they are multiple execution units to allow all possible combination of instructions in the pipeline.

* If for some combination, pipeline has to be stalled to avoid the resource conflicts then there is a structural hazard.

* In other words, we can say that when two instructions require the use of a given hardware resource at the same time, the structural hazard occurs.

* The most common case in which this hazard may arise is in access to memory.

* one instruction may need to access memory for storage of the result while another instruction or operand needed is being fetched.

* If instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed.

* To avoid such type of structural hazards many processors use separate caches for instruction and data.

In our Laundry Example structural hazards will occur in two cases

1. If we use a washer-dryer combination instead of a separate washer and dryer.

2. If our roommates doing something else without put clothes away.

2. Data or Data dependant Hazards:

* Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.

(ie) when a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

Example:

$I_1 : A \leftarrow A+5$

$I_2 : B \leftarrow A \times 2$

Suppose $A=10$ means

* When these two operations are performed in the given order, the result is 30

* If they are performed concurrently the value of A used in computing B would be the original value 10, leading to an incorrect result.

* Data used in the I_2 depends on the result of I_1, This is called data hazard or data dependent hazard.

There are some solutions to rectify data hazards. They are

1. Forwarding or Bypassing
2. Hazard detection Unit
3. Reordering code to avoid pipeline stalls
4. Hazard detection by software.

Generally in MIPS instruction set has five stages for pipelining process.

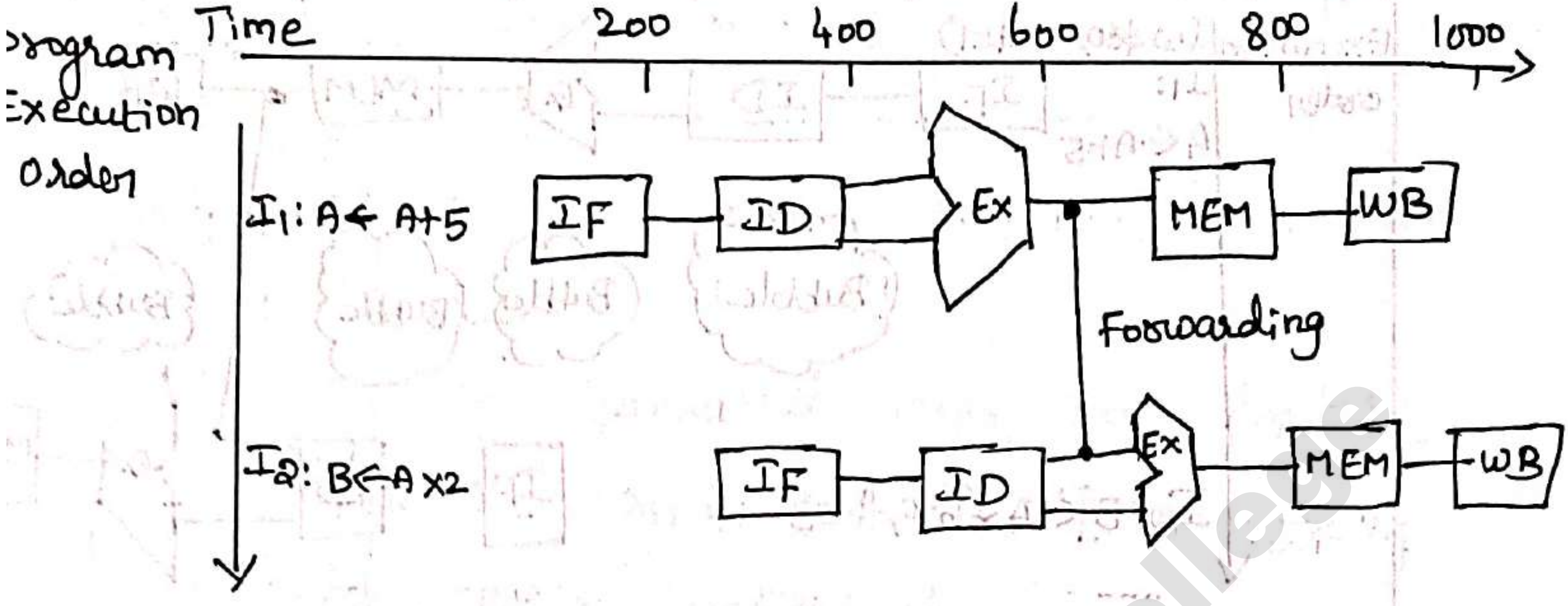
1. IF - Instruction field.
2. ID - Instruction Decode/Register file read stage
3. EXE - Execution stage
4. MEM - Memory access stage
5. WB - write Back stage.

1. Forwarding or Bypassing:

* The primary solution is based on the observation that, we donot need to wait for the instruction to complete before trying to resolve the data hazard.

* Adding extra hardware to retrive the missing item early from the internal resources is called forwarding or By passing.

The graphical representation for forwarding.



* Forwarding paths are valid only if the destination stage is later in time than source stage.

* Forwarding path cannot prevent all pipeline stalls, so to focus new data hazard is called Load use data hazard.

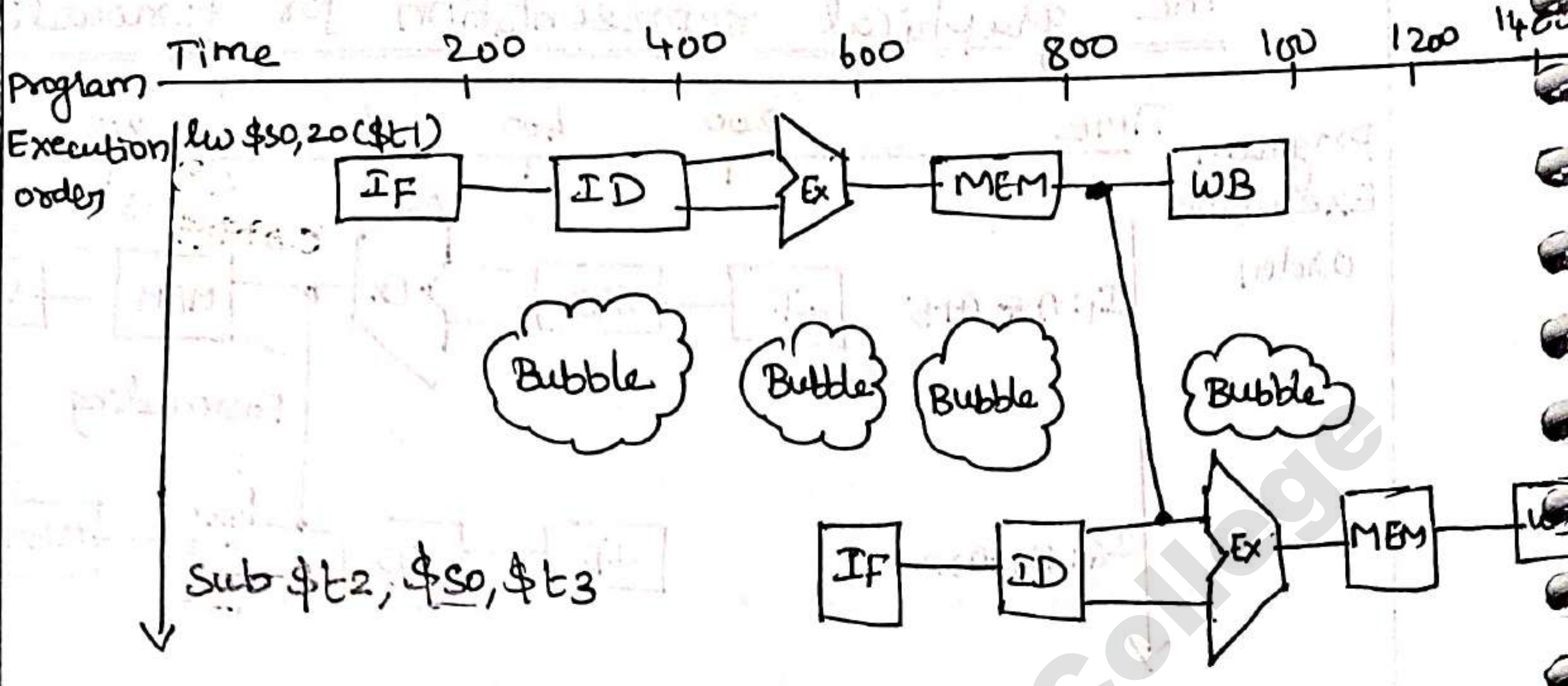
* It is a specific form of data hazards in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.

Pipeline Stall:

* It is also called bubble.

* Pipeline stall is a stall initiated in order to resolve a hazard.

Stall even with forwarding



*Hazard Detection Unit:

* The dependence between load and the following instructions causes data hazard, it cannot be solved by forwarding.

* To solve the problem, in addition to a forwarding unit, need hazard detection unit. It operates during the ID stage so that it can insert the stall between the load and its use.

Checking for load instructions, the control for the hazard detection unit is this single condition

if(ID/Ex.MemRead and
((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)))

3. Reordering Code to Avoid Pipeline Stall:

Consider the following code segment in C

```
a = b+e;
c = b+f;
```

* Here is the generated MIPS code for this segment assuming all variables are in memory and are addressable as offsets from $t0:

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1, $t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1, $t4
sw $t5, 16($t0)
```

Solution:

* Both add instructions have a hazard because of their respective dependence on the immediately preceding lw instruction. Moving up the third lw instruction to become the third instruction eliminates hazards.

```
lw $t1, 0($t0)
lw $t2, 4($t0)
lw $t4, 8($t0)
```

```
add $t3, $t1, $t2
sw  $t3, 12($t0)
add $t5, $t1, $t4
sw  $t5, 16($t0)
```

* on a Pipelined processor with forwarding, the reordered sequence will complete in two fewer cycles, than the original version.

4. Hazard Detection By software:

* An alternative method for handling data hazard is to leave the task of detecting data dependencies to the software.

* The compiler can introduce the necessary delay needed between the two instructions by inserting NOP (NO operation) instructions.

* NOP is a case where pipeline stage executing an instruction but that does not make any changes in the state.

* NOP acts like a bubble in the Pipeline stage.

Disadvantage:

*Insertion of NOP leads to larger code size

3. Control Hazards

* The Purpose of the instruction fetch unit is to supply the execution units with a steady stream of instructions.

* This stream is interrupted when pipeline stall occurs either due to cache miss or due to branch instruction. Such situation is known as instruction hazard or control hazard or branch hazard.

Branch Penalty:

The time lost as a result of a branch instruction is often referred as the branch penalty.

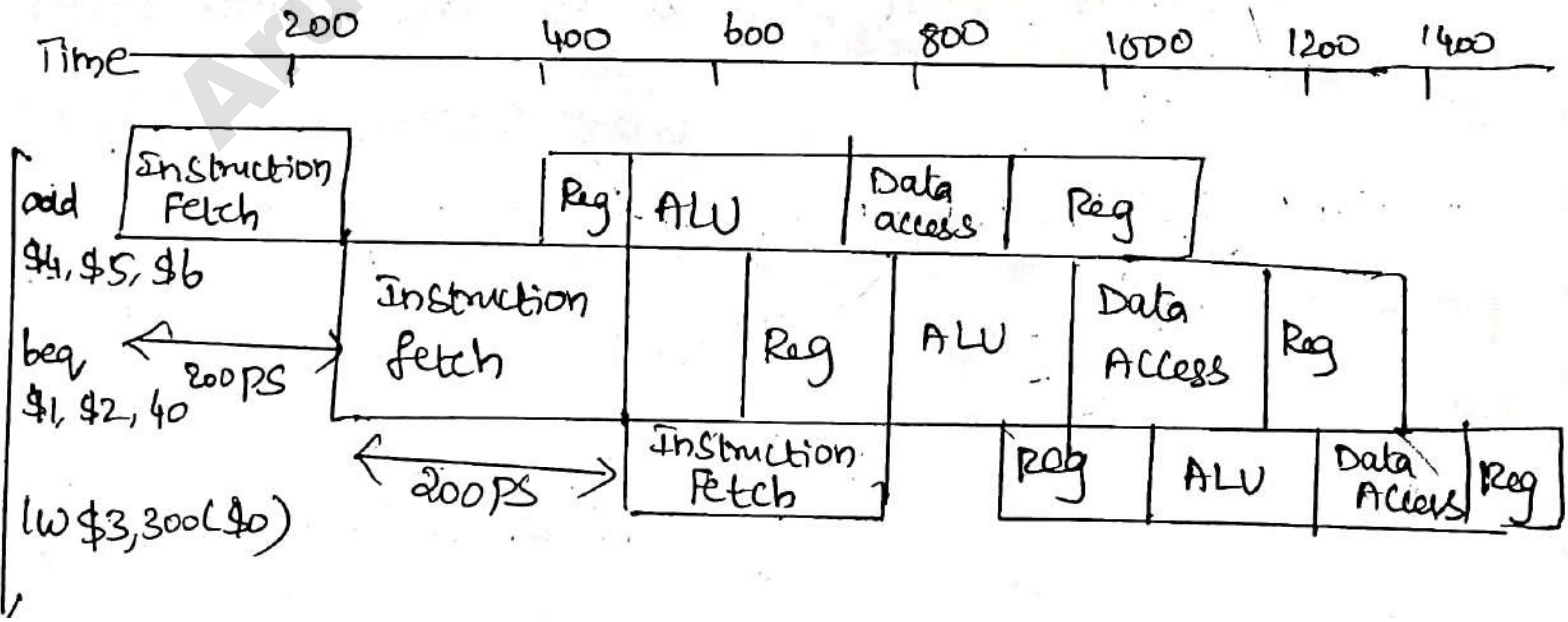
*The Pipeline cannot possibly know what the next instruction Should be, because it only received the branch instruction from memory.

*TO avoid Stall, we fetch a branch after that waiting until the pipeline determines the outcome of the branch and knows what instruction address to fetch from.

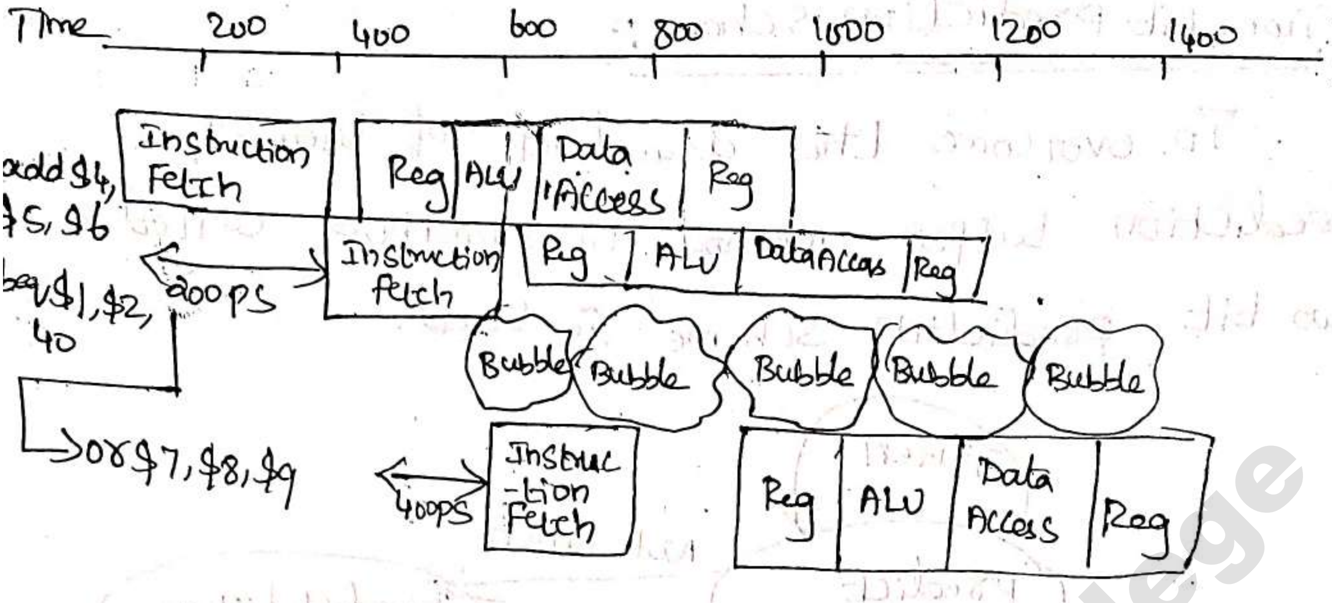
Branch prediction:

* Branch prediction is a method of resolving a branch hazard that assume a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome

Eg pipeline when the Branch is not taken



Pipeline when the branch is taken



Dynamic Hardware predictors:

* Dynamic Hardware predictors make their guesses depending on the behaviour of each branch and may change predictions for a branch over the life of a program.

* one Popular approach to dynamic prediction of branches is keeping a history for each branch as taken or Untaken

* Using the recent past behavior to predict the future behavior.

Advantages of pipeline:

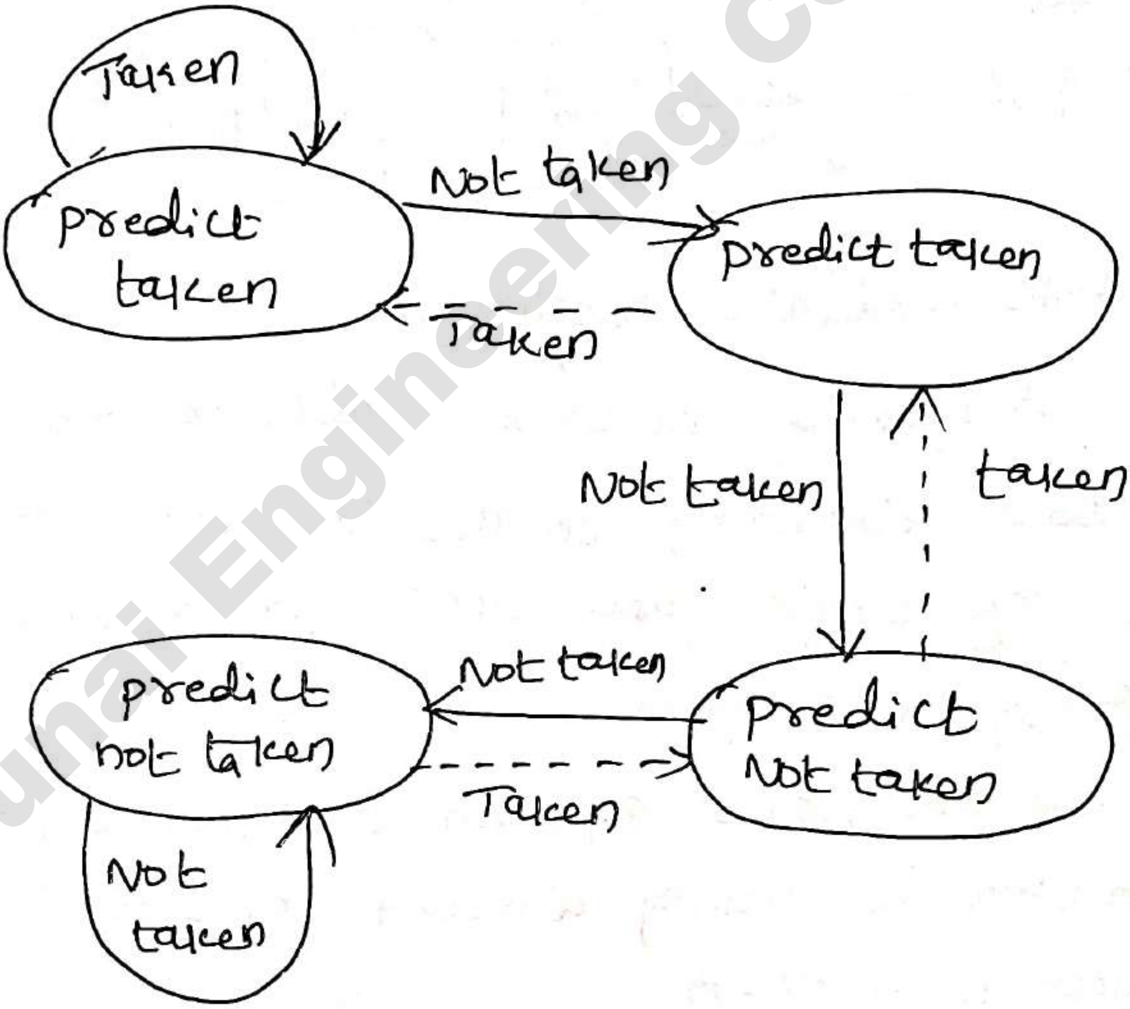
1. It increases the number of simultaneously executing instructions

* It Increases the rate at which instructions are started and completed.

* It improves instruction throughput rather than Individual Instruction Execution or latency.

Two-bit prediction scheme:-

To overcome the drawback of branch prediction buffer method, new method called two bit prediction scheme is used.



Finite state machine for 2 bit prediction scheme

Expectations

* Control is the most challenging aspect of processor design because of two reasons:

1. It is the hardest part to get right
2. It is the hardest part of make fast

* one of the hardest parts of control is implementing exceptions and interrupts.

* Handling exceptions and interrupts are more complex task than handling branches or jumps

* Exception and interrupt are initially created to handle unexpected events from within the processor.

Exception:

* Exception is an unscheduled event that disrupts program execution and used to detect overflow. It is also called Interrupt. Exception refers to any unexpected change in control flow without distinguishing whether the cause is internal or External

Interrupt: Interrupt is an exception that comes from outside of the processor.

* Interrupt refer to any unexpected change occurred only when the event is externally caused.

Type of Event	From where?	Mips Terminology
I/o device request	External	Interrupt
Arithmetic overflow	Internal	Exception
nvoke the operating ystem from user rogram	Internal	Exception
sing an undefined nstruction	Internal	Exception
ardware malfunctions	Either	Exception or Interrupt

To detect two kinds of exceptions we need to implement control.

Two kinds of Exceptions arise

1. From the portions of the instruction set

2. Implementation

* First we have to-detect the condition for exception.

* once the exception is detected next we need to take appropriate action for it.

* Detecting exceptional conditions and taking the appropriate action is on the critical timing path of a processor.

* Processor must determine the clock cycle time and thus performance. To design a control unit we must have proper attention to exceptions.

* Because exception can reduce performance and t leads complex task to getting the correct design.

xceptions in MIPS Architecture.

In MIPS architecture two kinds of exceptions occur at

1. Execution of an undefined instruction
2. An arithmetic overflow

*once exception occur then the processor must save the address of the offending instruction in he exception program counter (EPC) and then transfer ontrol to the operating system at some specified ddrees.

*After transferring control to the operating system it must take appropriate action to provide some service to the user program.

*operating system will provide the following services to user program

1. Taking some predefined action in response to an overflow.

2. Stopping the execution of the program and reporting an error.

* After performing necessary action is required because of exception, the operating system can terminate the program or may continue its instruction.

* operating system use the exception program counter (EPC) to determine where to restart the execution of the program.

* To handle the exceptions by operating system it must know the reason for the exception.

* operating system must know the reason for exception because the only it can handle properly.

* There are two main methods used to communicate the reason for an exception.

↳ MIPS architecture has the following two methods to find the reason for exception.

1. Status register
2. Vectored Interrupts

Status register:

* It is also called the cause register. It holds a field that indicates the reason for the Exception.

Vectored Interrupts:

In a vectored interrupt, the address to which control is transferred is determined by the cause of the exception.

Exception type	Exception vector address (in hex)
Undefined Instruction	8000 0000 hex
Arithmetic overflow	8000 0180 hex

* operating system can know the reason for exception by the address at which is initiated.

* The addresses are separated by 32 bytes or eight instructions.

* operating System must record the reason for the exception and may perform Some Limited processing in this sequence.

* When the exception is not vectored then a single entry point for all exception can be used.

* If single entry point is used for all exceptions then the operating system must decode the Status register to find the reason for exception.

* To handle exceptions we can add a few extra registers and control signals to their basic implementatio

Implementation of exception in MIPS architecture:

* To implement exception system in MIPS architectur ssum it has single entry point for all exception and has address value is 8000 0180

* This address value indicates it is an arithmetic overflow exception.

* To handle this exception we need to add two additional registers to our current MIPS implementation.

* Two addition registers are

i) EPC register:

A 32 bit register used to hold the address of the affected instruction This register is needed even when exceptions are vectored

ii) Cause register:

* This register is used to record the cause of the exception. In MIPS arichitecture this register is 32 bits lon and Some bits are currently unused.

* 10 bits are used to represent an undefined instruction and 12 bits used to represent arithmetic overflo

UNIT 4
PARALLELISM

Parallel Processing Challenges

* Parallel processing will increase the performance of processor and it will reduce the utilization time to execute task. But Obtaining the parallel processing is not an easy task

* The difficulty with parallelism is not in the hardware side it is in the software side.

* It is difficult to write software that uses multiple processors to complete one task faster and the problem gets worse as the number of processors increases.

* Developing the parallel processing programs are so harder than the sequential programs because of the following reasons.

1. It must get better performance or better energy efficiency from a parallel processing program on a multiprocessor.

2. If we would not have efficient energy then we have to use a sequential program on a uniprocessor because sequential programming is very simple.

* Using Uniprocessor we can solve the problem in developing parallel processing programs.

* Because uniprocessor design techniques such as super scalar and out of order Execution. take advantage of instruction Level parallelism

* Uniprocessor design techniques will reduce the demand for rewriting programs for multiprocessors.

* If number of processors level increased mean it is more difficult to write parallel processing programs.

* The following reasons we cannot get parallel processing programs as faster than sequential programs

1. Scheduling
2. Partitioning the work into parallel pieces
3. Balancing the load evenly between the workers
4. Time to synchronize
5 overhead for communication between the parties

1. Scheduling:

* Scheduling is the method by which threads, processes or data flows are given access to system resources

Eg processor time, communications bandwidth

* Scheduling is done to load balance and share system resources effectively or achieve a target quality of service.

* Scheduling can be done in various fields among that process scheduling is more important, because in parallel processing we need to schedule the process correctly

↳ Process scheduling can be done in the following ways

1. Long term scheduling
2. Medium term scheduling
3. Short term scheduling
4. Dispatcher.

2. Partitioning the work

* The task must be broken into equal number of pieces because otherwise some task may be idle while waiting for the ones with larger pieces to finish

* To obtain parallel processing task must be divided into equally to all the processor. Then only we can avoid the idle time of any processor

Balancing the Load:

* Load balancing is the process of dividing the amount of work that a computer has to do between two or more processor. So that more work gets done in the same amount of time and in generall all process get served faster.

* work load has to be distributed evenly between the processor to obtain parallel processing task.

Time to Synchronize:

* Synchronization is the most important challenge in Parallel processing. Because all the processor have equal work load so it must complete the task within specific time period.

* For parallel processing program it must have time to synchronization process, because if any process does not complete the task within specific time period then we cannot obtail parallel processing

Two methods are found to increase the scale up, Such methods are

1. Strong scaling
2. Weak scaling

1. Strong scaling:

*In this method speed up is achieved on a multiprocessor without increasing the size of the Problem.

*"Strong scaling means measuring speep up while keeping the problem size fixed".

2. Weak scaling:

*In this method speed up is achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.

*"weak scaling means the problem size grows proportionally to the increase in the number of processors".

*Let's assume that the size of the problem is M is the working set in main memory and we have P processors.

* Then the memory per processor for Strong Scaling is approximately M/P and for weak scaling it is approximately M.

* By considering the memory hierarchy it is easy to know that weak scaling being easier than Strong scaling.

*But if the weakly scaled dataset no longer fits in the last level cache of a multicore microprocessor then the resulting performance could be much worse than by using strong scaling

*Depending on the application we can select the scaling approach.

Flynn's Classification

Flynn's classification:

* Parallel processing can be classified in many ways. It can be classified according to internal organization of processors, according to interconnection structure used between processors or according to flow of information through the System.

* In 1966, Michael J. flynn, classified Processor parallelism based on the number of simultaneous instruction and data streams seen by the processor during program execution.

* Flynn's classification divides computers into four major Groups No. of I Stre No. of datastream

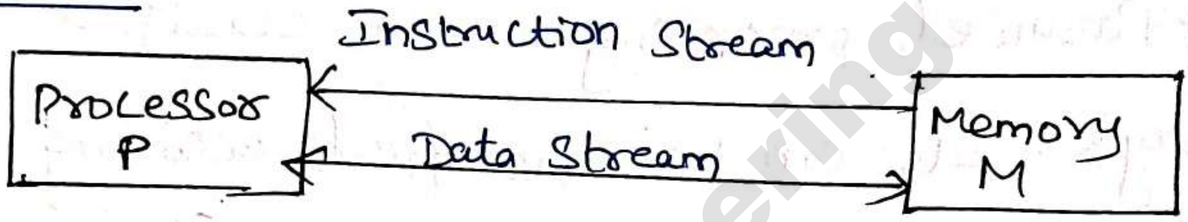
i) Single Instruction Stream - Single Data Stream (SISD)

ii) Single Instruction Stream - Multiple Data Stream (SIMD)

iii) Multiple Instruction Streams - Single Data Stream (MISD)

iv) Multiple Instruction Streams - Multiple Data Streams (MIMD)

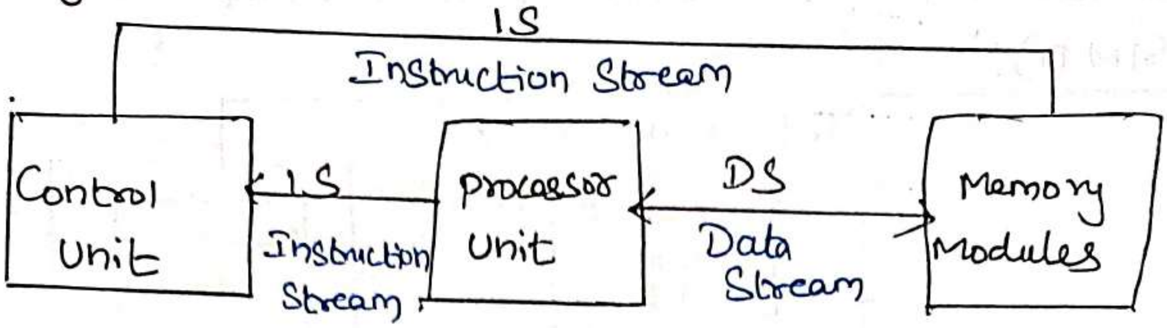
Instruction and data streams in a Sequential Computer



* A processor operates by fetching instructions and operands from main memory or cache, executing the instructions and placing the final results in memory

* The instructions form an instruction stream flowing from memory to the processor, while the operands from another stream, data stream flowing to and from the processor.

i. Single Instruction Stream Single Data Stream (SISD)



SISD Computer.

* Most conventional machines with one CPU Containing a Single arithmetic-logic unit capable only of scalar arithmetic fall into the category.

* SISD and sequential computers are thus synonymous.

* In SISD computer instructions are executed sequentially but overlap in their execution stages (pipelining).

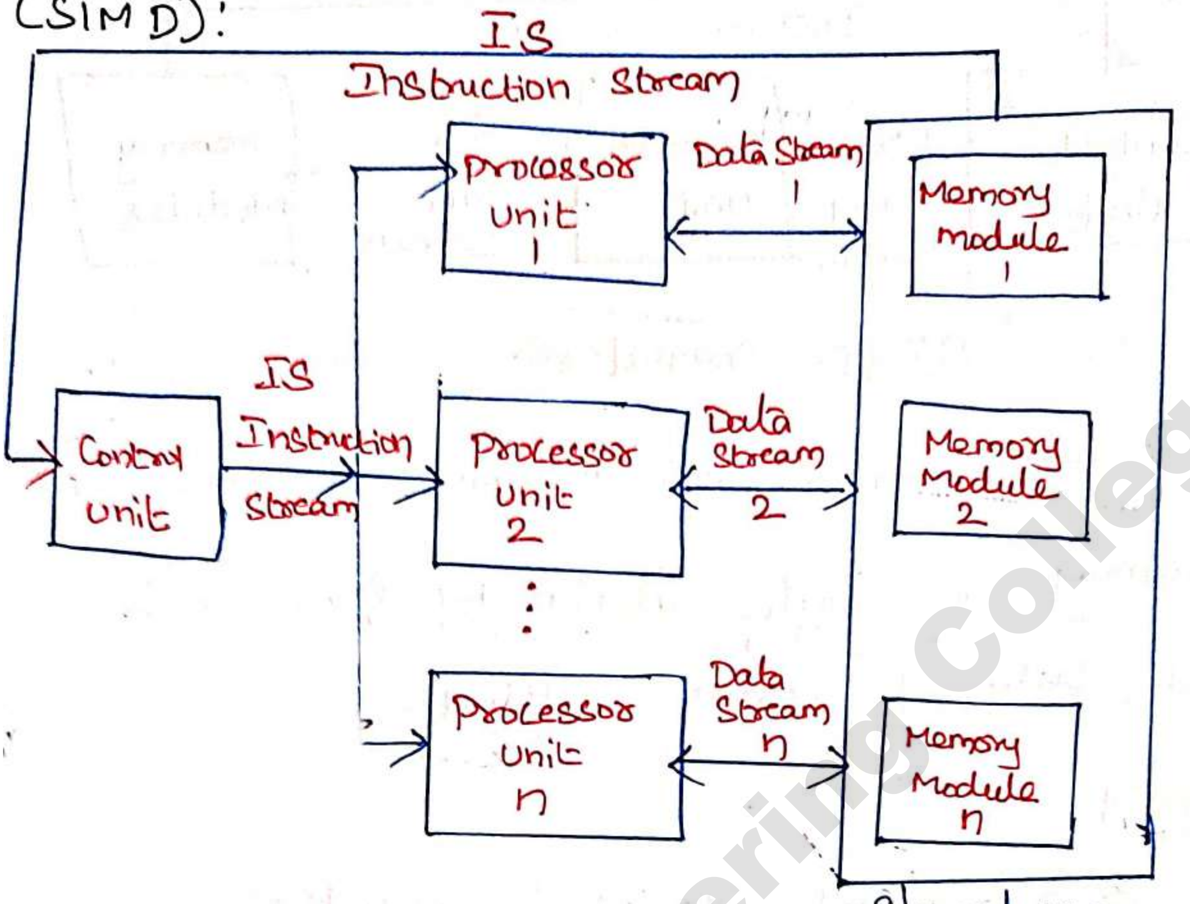
* They may have more than one functional unit, but all functional units are control by a single Control Unit.

Examples of SISD machines Includes

> CDC 6600 which is unpipelined but has multiple function units

> cray-1 which supports vector processing.

ii) Single Instruction Stream Multiple Data Streams (SIMD):



SIMD Computers

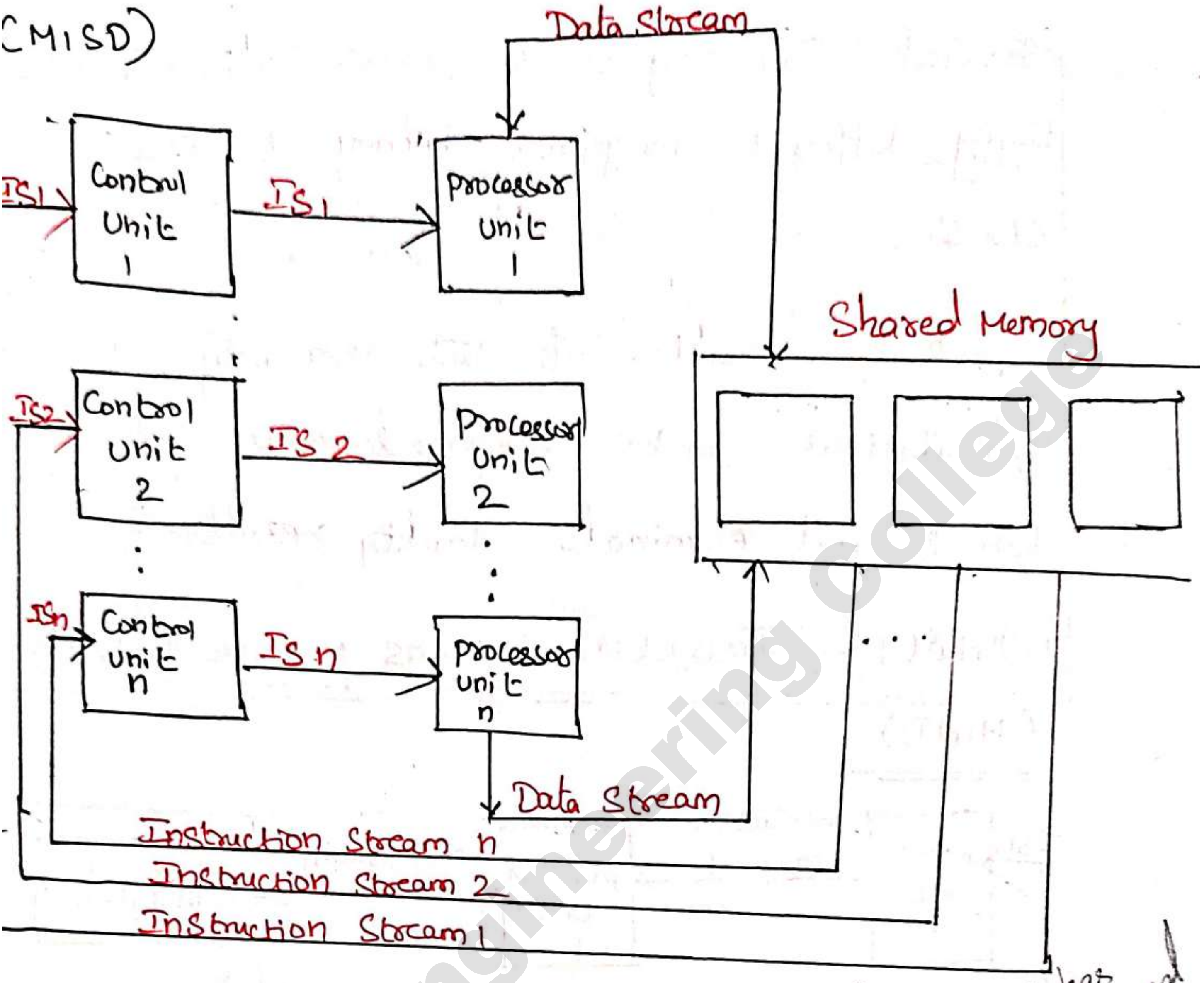
* This category corresponds to array processors. They have multiple processing/ execution units and one Control unit.

* Therefore, all processing /execution units are Supervised by the single Control Unit.

* Here all Processing elements received same instruction from control unit but operate on different data sets from distinct data streams.

* This category is also known as single program Multiple Data Stream (SPMD)

ii) Multiple Instruction Streams Single Data Stream (MISD)



MISD computer

This kind of organization has never been used

* Not many parallel processors fit well into this category.

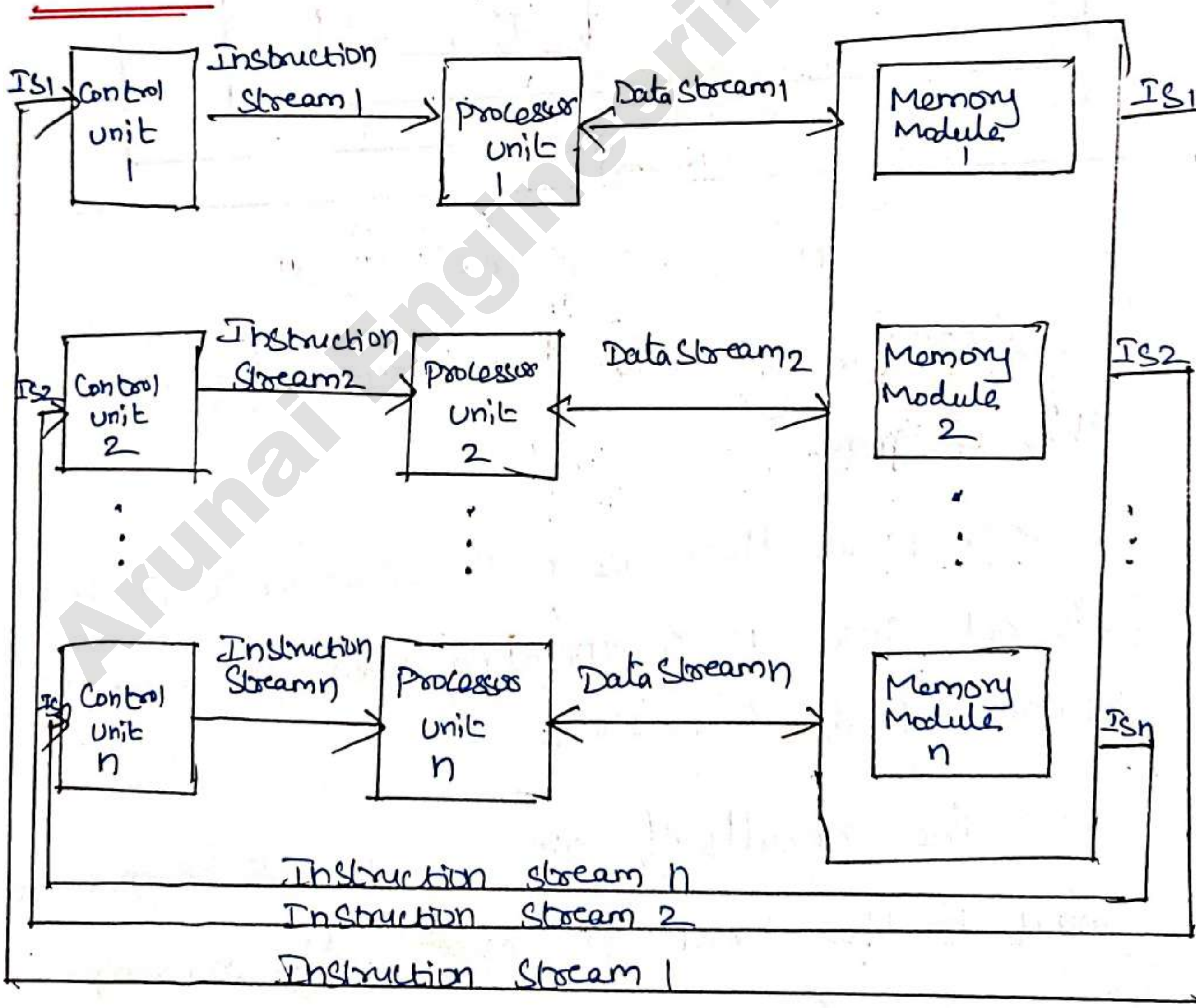
* In MISD, there are n Processor units. Each receiving distinct instructions operating over the same data stream and its derivatives.

* The results of one processors become the input to the next processor in the micropipe.

* The fault-tolerant computers where several processing units process the same data using different programs belongs to the MISD class.

* The results of such apparently redundant computations can be compared and used to detect and eliminate faulty results.

iv) Multiple Instruction Streams Multiple Data Streams (MIMD):



MIMD Computer

* Most multiprocessors system and multiple Computers system can be classified in this category.

* In MIMD, there are more than one processor Unit having the ability to execute several program simultaneously. 1) Each processor has a separate pgm. 2) An instruction stream is generated from each program

* MIMD computer implies interactions among the multiple processors because all memory streams are derived from the same data space shared by all processors. 3) Each instruction operates on diff. data

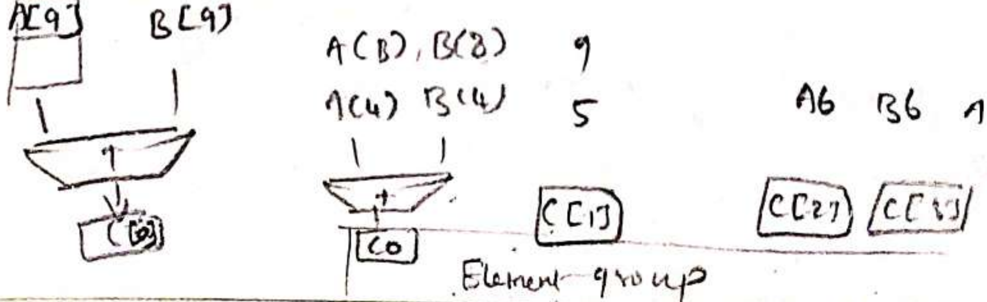
* If the n data streams are derived from disjointed sub spaces of the shared memories then we would have the so-called Multiple SISD (MSISD) operation.

scalar - single data

Vector systems:

* SIMD computers operate on vectors of data and it uses vector architecture.

* Vector architectures pipelined the ALU to get good Performance at lower cost.



* The basic philosophy of vector architecture is to collect data element from memory put them in order into a large set of registers, operate on them sequentially in registers using pipelined execution units and then write the results back to memory.

* Vector architecture has a set of 32 vector register and each with 64 bit elements

pipelining parallelism can work well

* Vector elements are independent and it can be operated on in parallel,

* All modern vector computers have vector functional units with multiple parallel pipelines called vector lanes

* Vector functions units with multiple parallel , pipelines produce two or more results per clock cycle.

Vector - Instr-set contains instruc. that operates on one-dimensional arrayof data called Vector.

Adv: Highly memory access, Reduce instruction fetch bandwidth

Dadv: Memory can easily bottleneck

Hardware Multithreading

*Hardware multithreading allows multiple threads to share the functional units of single processors in an overlapping fashion try to utilize the hardware resource efficiently.

*To permit this sharing, the processor must duplicate the independent state of each thread.

Example:

*Each thread would have a separate copy of the register file and the program counter

*The memory itself can be shared through the virtual memory mechanisms it already support multiprogramming.

* Hardware multithreading increase the Utilization of a processor by switching to another thread when one thread is stalled.

* Hardware must support the ability to change to a different thread relatively quickly.

* Thread is a light weight process and threads share a single address space but processes don't share. Thread switch is more efficient than a process switch.

* Process includes one or more threads, the address space and the operating system state.

* Process switch invokes the operating system but not a thread switch.

↳ Hardware multithreading has two main approaches such as

1. Fine grained Multithreading
2. Coarse grained Multithreading

Fine Grained Multithreading:

* Fine-grained multithreading switches between threads on each instruction, resulting in interleaved execution of multiple threads.

* This interleaving is often done in a round-robin fashion, skipping any threads that are stalled at that time.

* To make fine-grained multithreading practical, the processor must be able to switch threads on every clock cycle.

Advantage:

* It can hide the throughput losses that arise from both short and long stalls because instruction from other threads can be executed when one thread stalls

Disadvantage

↳ It slows down the execution of the individual threads because thread that is ready to execute without stalls will be delayed by instructions from other threads.

2. Coarse Grained Multithreading:

* Coarse-grained Multithreading was invented as an alternative to fine-grained multithreading

* Coarse-grained multithreading switches threads only on costly stalls, such as second-level cache misses.

* This change relieves the need to have thread switching be essentially free and is much less likely to slow down the execution of an individual thread, since instructions from other threads will only be issued when a thread encounters a costly stall.

Drawback:

* It is limited in its ability to overcome throughput losses especially from shorter stalls.

* This limitation arises from the pipeline start up costs of coarse grained multithreading issues instructions from a single thread, when a stall occurs the pipeline must be empty.

Benefit:

* The new thread will begins executing after the stall must fill the pipeline before instructions will be able to complete.

* Due to this startup overhead, coarse grained multithreading is more useful for reducing the penalty of high cost stalls.

Simultaneous Multithreading (SMT) :

* Simultaneous multithreading is a variation on hardware multithreading that uses the resources of a multiple issue, dynamically scheduled pipelined processor to exploit thread level parallelism at the same time it exploits instruction level parallelism.

* It has multiple processors and more functional unit parallelism available than most single threads can effectively use

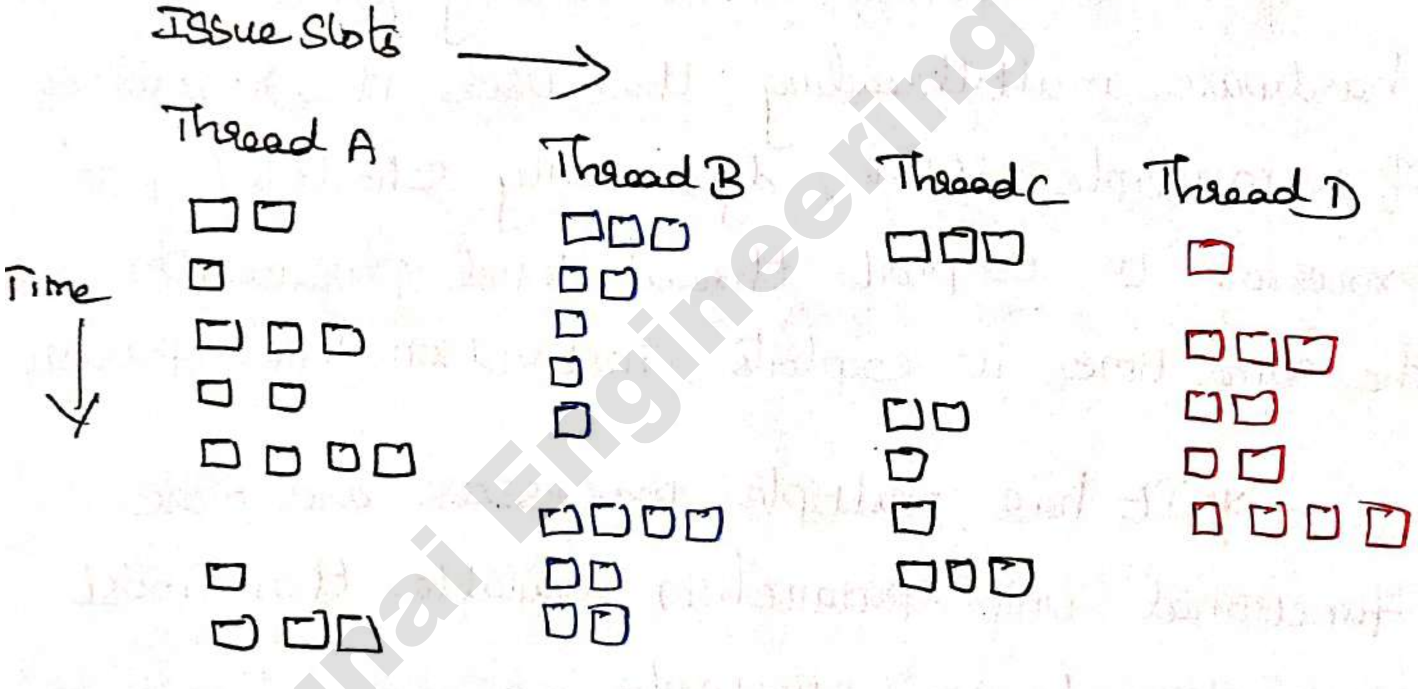
* It has register renaming and dynamic scheduling policy with these features the following task can be obtained.

* Multiple instructions from independent threads can be issued without regard to the dependences among them and the resolution of the dependences can be handled by the dynamic scheduling capability.

* SMT relies on the existing dynamic mechanisms, it does not switch resources every cycle

*SMT is always executing instructions from multiple threads, leaving it up to the hardware to associate instructions slots and renamed registers with their proper threads

Four Threads Execute independently on a superscalar with no multithreading support



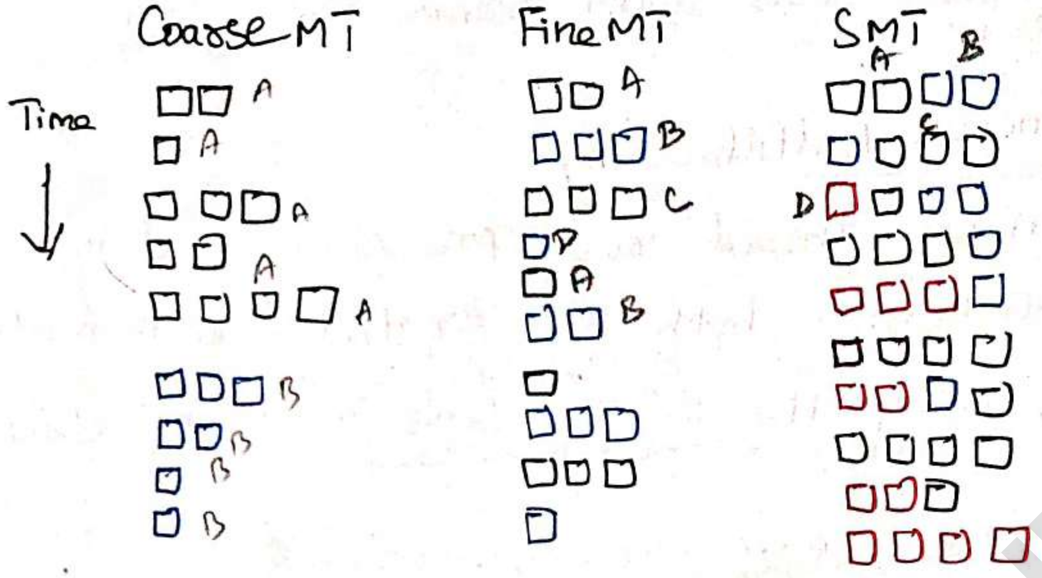
Four Threads would be combined to execute on the processor more efficiently using three multithreading options.

1. A Superscalar with coarse grained Multithreading

2. A Superscalar with fine grained Multithreading

3. A Superscalar with Simultaneous multithreading

Issue slots



* The horizontal dimension represents the instruction issue capability in each clock cycle and vertical dimension represents a sequence of clock cycle.

Coarse-grained Multithreading.

* The long stalls are partially hidden by switching to another thread that uses the resources of the processor.

* It will reduce the number of completely idle clock cycle and the pipeline start up overhead still leads to idle cycles

Fine grained Multithreading:

The interleaving of threads mostly eliminates idle clock cycles. Because only a single thread issues instructions in a given clock cycle.

* Limitations in instruction level parallelism will lead to idle slots within some clock cycles.

Simultaneous Multithreading:

* In SMT thread level parallelism and instruction level parallelism both are exploited with multiple threads using the issue slots in a single clock cycle.

* Ideally the issue slot usage is limited by imbalances in the resources needs and resource availability over multiple threads, but in practice other factors can restrict how many slots are used.

Multicore Processor

Multicore processor:

* Hardware multithreading improved the efficiency of processor but it has big challenge to deliver on the performance potential of Moore's law by efficiently programming the increasing number of processors per chip.

* Rewriting old programs to run well on parallel hardware is more difficult. computer designers must do something to overcome these difficulties.

* Solution for above program is using a single physical address space for all processors so that programs need not concern themselves with where their data is and programs may be executed in parallel.

↳ Multicore architectures are classified according to

i) Number of processor: Two processors (dule core), four processors (Quad core) and eight processors (octa-core) configuration.

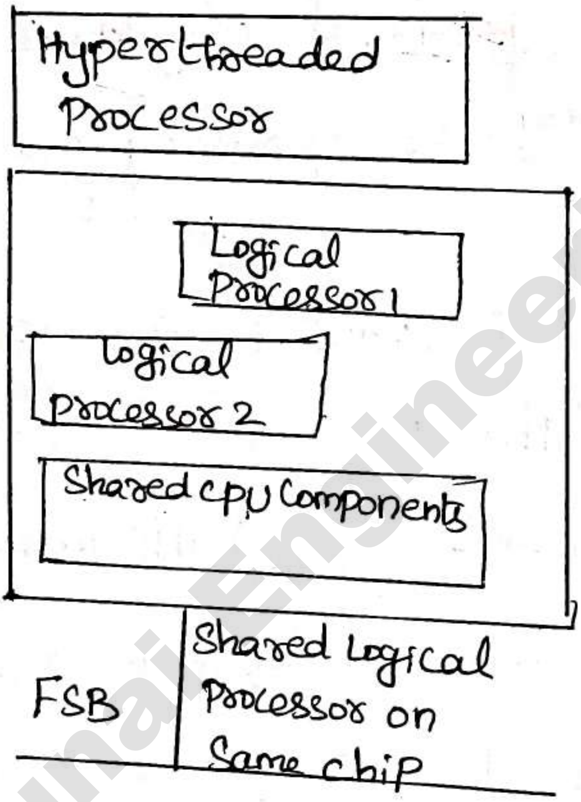
2) Approaches to processor - to - processor Communicatio

3) cache and memory implementations

4) Implementation of I/o bus and the Front Side Bus (FSB)

Three common configuration that support multiprocessing.

Type1:



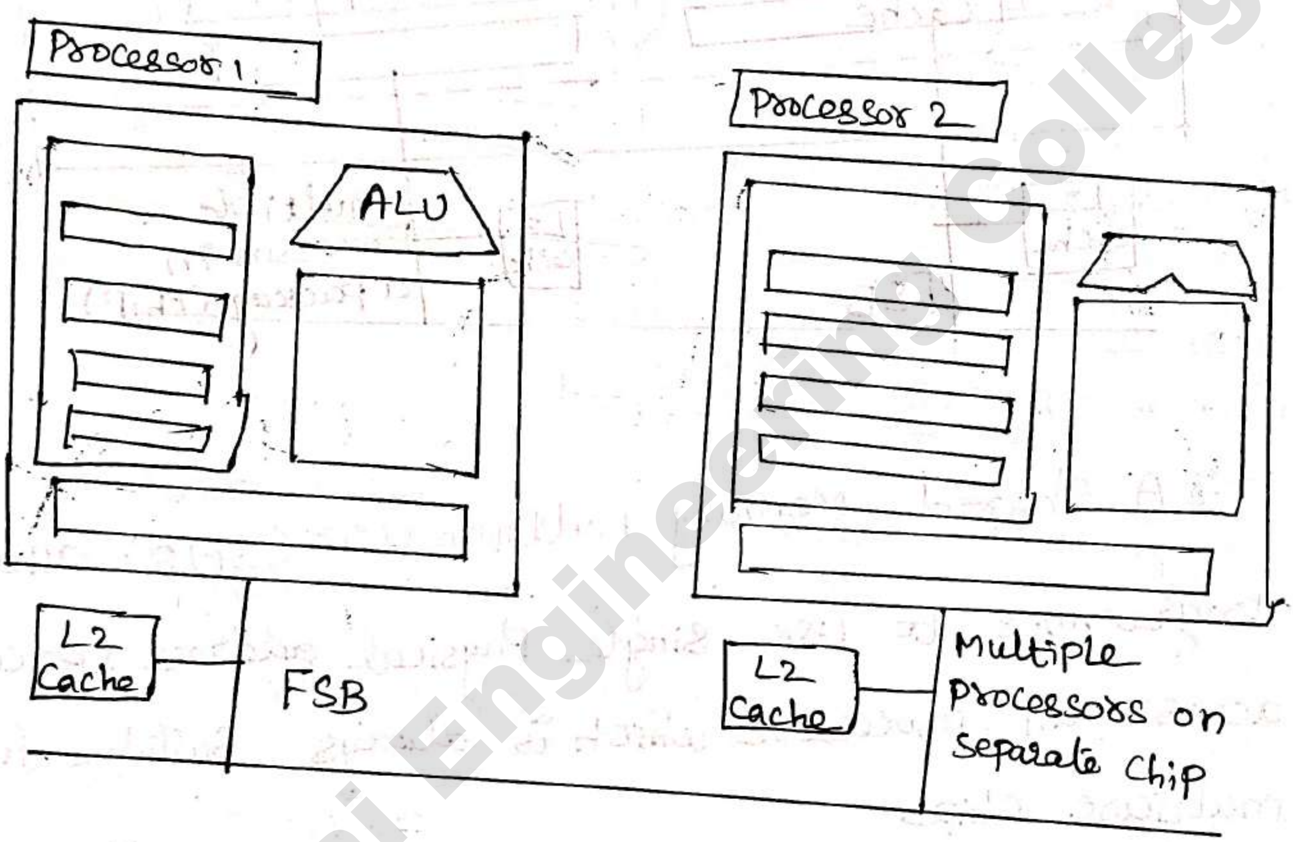
* It uses hyperthreading technology. It allows a Single processor to act like two separate processors to the operating system and the application programs that use it

* However, there is a single processor running multiple threads.

*Thus, in hyperthreaded technology the multiple processors are logical instead of physical.

*In hyperthreaded technology has some duplication of hardware but not enough to qualify a separate physical processor.

Type 2:

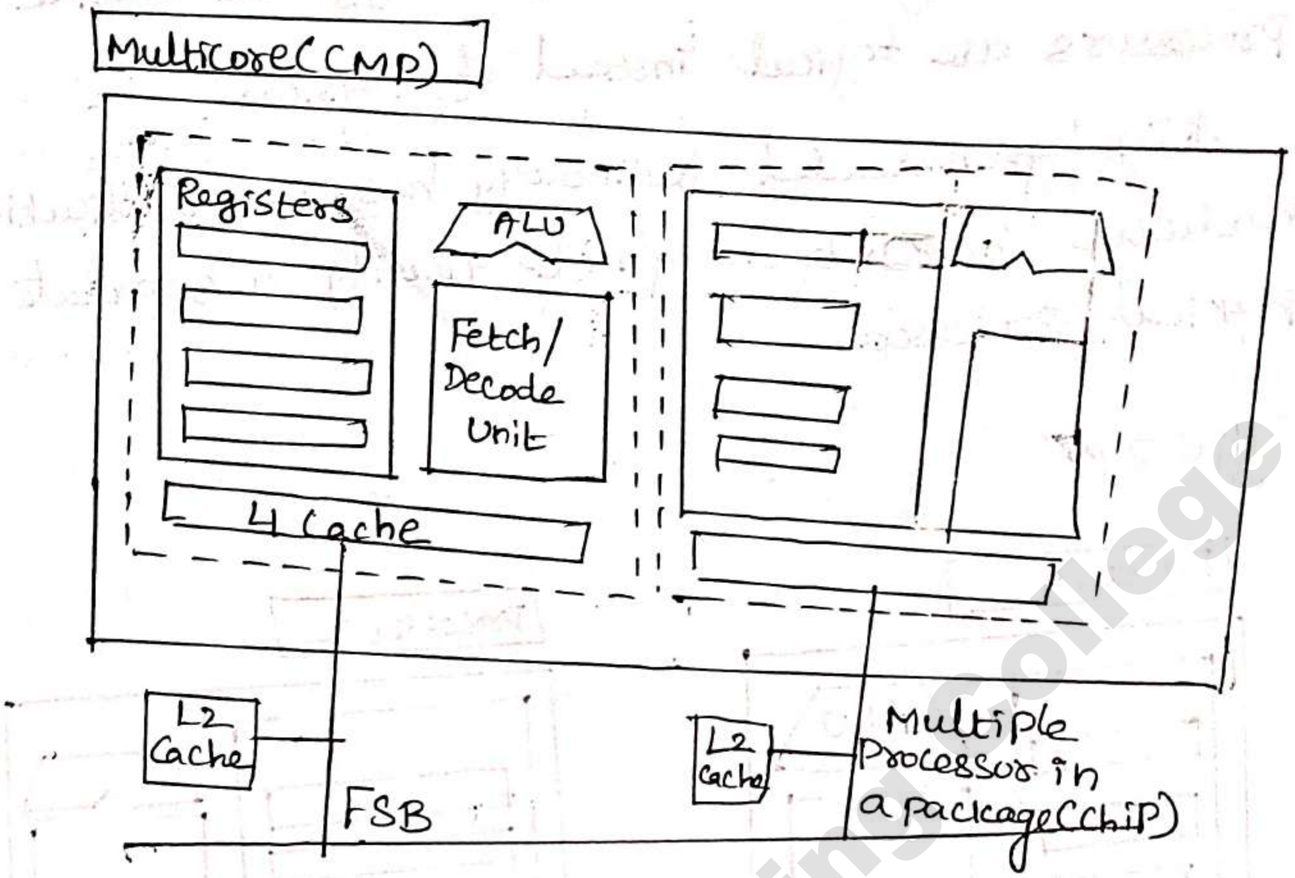


*It is the classic multiprocessor. A multiprocessor is a tightly coupled computer system having two or more processing units (Multiple processors) on a separate chip with its own hardware.

Type3:

*It represents Multicore system. It provides two or more complete processors on a single chip

Type 3:

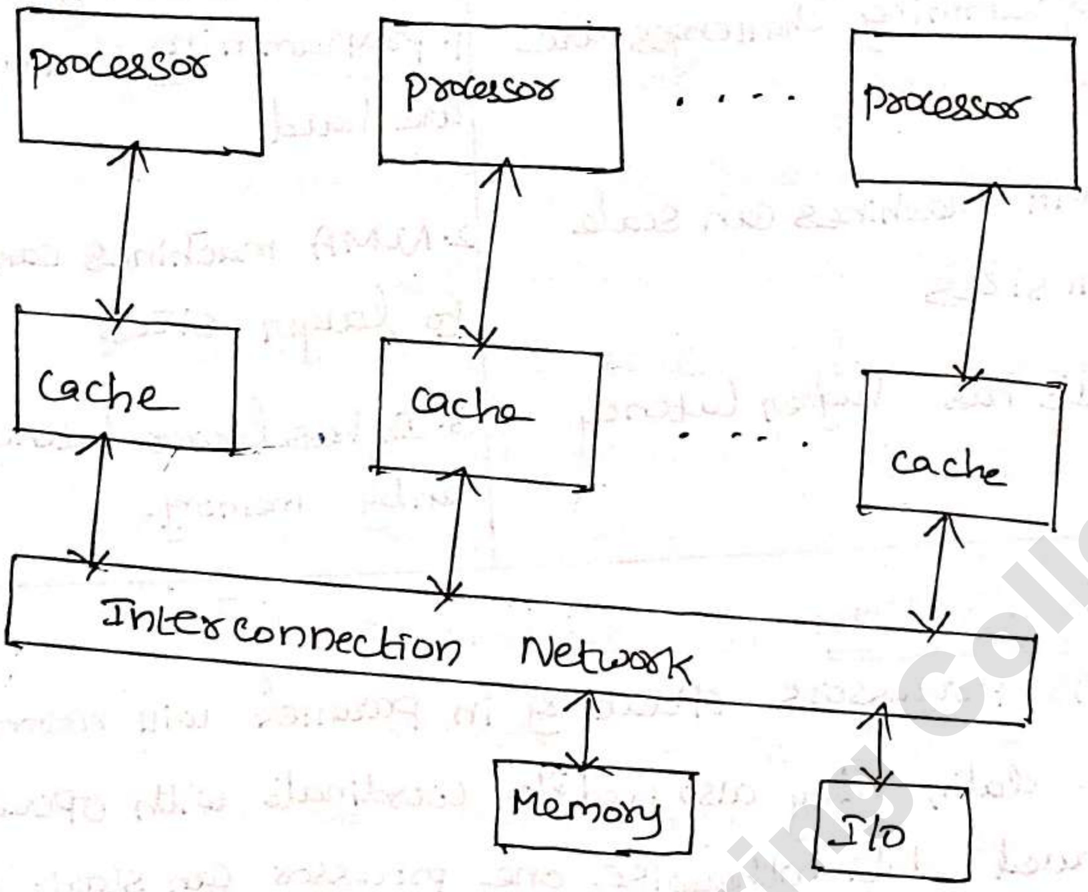


* A Shared Memory Multiprocessor (SMP) allows programmer to use single physical address space across all processors which is always suitable for multicore chips

* In shared address multiprocessor processors can communicate through variables in memory with all processors capable of accessing any memory location via loads and - stores.

* The classic organization of a shared memory multiprocessor. It can run independent jobs in their own virtual address spaces even if they all share a physical address space.

Types of single address



Classic organization ~~of~~ a shared memory Multiprocessor

Single address space Multiprocessor comes in two styles such as

1. Uniform Memory Access (UMA)
2. Non uniform Memory Access (NUMA)

* UMA is a multiprocessor in which latency to any word in main-memory is same no matter which processor requests the access.

* NUMA is a type of single address space multiprocessor in which some memory access are much faster than others depending on which processor asks for which word

Uniform Memory Access	Non uniform Memory Access
1. programming challenges are easy	1. programming challenges are hard
2. UMA machines can scale small sizes	2. NUMA machines can scale to larger sizes
3. It has higher latency	3. It has lower latency to nearby memory.

Synchronization:

* As processors operating in parallel will normally share data, they also need to coordinate with operating on shared data otherwise one processor can start working on data before another is finished with it. This coordination is called Synchronization.

* Synchronization is the process of coordinating the behavior of two or more processes which may be running on different processors

* when sharing is supported with a single address space there must be a separate mechanism for synchronization such as called Lock

* Lock is a synchronization device that allows access to data to only one processor at a time and other processors interested in shared data must wait until the original processor unlocks the variable.

-Introduction to Graphics Processing Unit

* The original justification for adding SIMD instructions to existing architectures was that many microprocessors were connected to graphics displays in pcs and workstations, so an increasing fraction of processing time was used for graphics.

* As Moore's Law increased the number of transistors available to microprocessors, it therefore made sense to improve graphics processing

* A major driving force for improving graphics processing was the computer game industry, both on PCs and in dedicated game consoles such as the Sony play station.

* The rapidly growing game market encouraged many companies to make increasing investments in developing faster graphics hardware, and this positive feedback loop led graphics processing to improve at a faster rate than general- purpose processing in mainstream microprocessors.

* Given that the graphics and game community had different goals than the microprocessor developmen community, it evolved its own style of processing and terminology.

* As the graphics processors increased in power, they earned the name Graphics processing Units or GPUs to distinguish themselves from CPUs.

Some of the key characteristics as to how GPUs vary from CPUs:

i) GPUs are accelerators that supplement a CPU, so they do not need be able to perform all the tasks of a CPU.

↳ This role allows them to dedicate all their resources to graphics.

↳ It's fine for GPUs to perform some tasks poorly or not at all, given that in a system with both a CPU and a GPU, the CPU can do them if needed.

* The GPU problems-sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes.

These differences led to different styles of architecture:

* Perhaps the biggest difference is that GPUs do not rely on multilevel caches to overcome the long latency to memory, as do CPUS.

↳ Instead, GPUs rely on having enough threads to hide the latency to memory.

↳ That is, between the time of a memory request and the time that data arrives, the GPU executes hundreds or thousands of threads that are independent of that request.

* The GPU main memory is thus oriented toward bandwidth rather than latency.

↳ There are even separate DRAM chips for GPUs that are wider and have higher bandwidth than DRAM chips for CPUS.

↳ In addition, GPU memories have traditionally had smaller main memories than conventional microprocessors.

↳ In 2008, GPUs typically have 1 GiB or less, while CPUs have 2 to 32 GiB

↳ In 2013, GPUs typically have 4 to 6 GiB or less while CPUs have 32 to 256 GiB.

* Finally, keep in mind that for general-purpose Computation, must include the time to transfer the data between CPU memory and GPU memory, Since the GPU is a coprocessor.

* Given the reliance on many threads to deliver good memory bandwidth, GPUs can accommodate many parallel processors (MIMD) as well as many threads.

↳ Hence, each GPU processor is more highly multithreaded than a typical CPU, plus they have more processors.

Warehouse - Scale Computers

* A cluster is a collection of desktop Computers or servers connected together by a local area network to act as a single larger computer.

* A warehouse-scale computer (WSC) is a cluster comprised of tens of thousands of servers

* The cost may be on the order of $150 M for the building, electrical and cooling infrastructure, the servers and the networking equipment that houses 50,000 to 100,000 servers.

* A WSC can be used to provide internet services like search, social networking, online maps, video sharing, online shopping, email, Cloud Computing.

* Clusters emphasize thread-level parallelism, whereas WSCs emphasize request-level parallelism.

* Data centers consolidate different machines and software into one location.

* WSC present a unified software model.

*Data centers emphasizes virtual machines and hardware heterogeneity in order to serve varied Customers. WSC emphasize homogeneity.

Clusters and Other Message Passing Multiprocessor

* The Alternative approach to sharing an address space is for the processors to have their own private physical address space.

* The classic organization of a multiprocessor with multiple private address space.

* In this, each processor have their own private physical address space.

* In this approach, the multiprocessor communicate via explicit message passing scheme, hence the name message passing multiprocessors.

* Communications between multiple processors is done by explicitly sending and receiving information.

* There are routines to send and receive messages

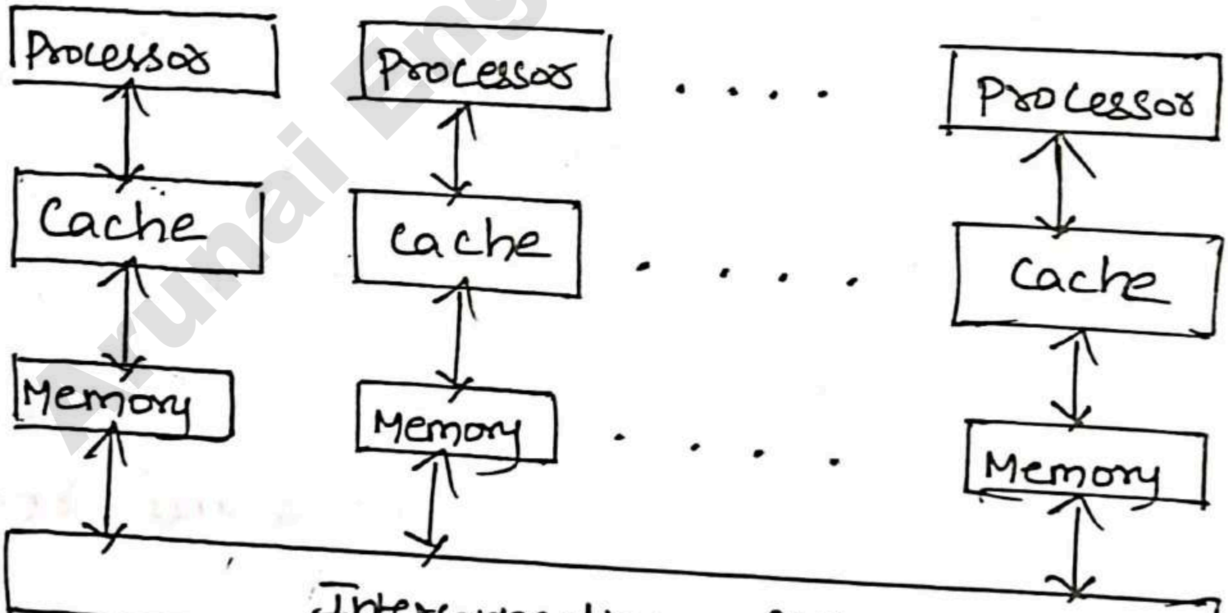
* Send message routine is used to send message to other processors in machines with private memories.

* Receive message routine is used to receive messages from another processors in machines with private memories.

* Indeed mechanisms are available to maintain when a message is sent to another processor and when a message is received from a processor.

* If a sending processor needs confirmation that the message has reached the receiver.

* Then the receiving processor can also send an acknowledgement message back to the sender.



organization of Message-Passing Multiprocessor

* Clusters are generally collection of computer connected to each other over their I/o interconnect via standard network switches and cables.

* Clusters are the best example of message-passing parallel computer.

Drawbacks of Clusters:

1. The cost of administering a cluster of n machines is about the same as the cost of administering n independent machines.

* But, the cost of administering a shared memory multiprocessor with n processors is about the same as administering a single machine

2. The processors in a cluster are connected using the I/o interconnect of each computer.

* But the cores in a multiprocessor are usually connected on the memory interconnect of the computer.

* The memory interconnect has higher bandwidth and lower latency which leads to better communication performance.

3. The overhead in the division of memory.

A cluster of n machines has n independent memories and n copies of the operating system are needed.

* But a shared memory multiprocessor allows a single program to use almost all memory in the computer and it only needs a single copy of the OS.

UNIT 5
MEMORY AND I/O SYSTEMS

Memory Hierarchy

* The aim of the memory hierarchy is to match the processor speed with the rate of information transfer at a lowest level and at a minimum possible cost.

* The memory unit that communicate directly with the CPU is called the main memory.

* programs and data currently needed by the processor reside in main memory.

* Devices that provide backup storage are called auxiliary memory.

* Memories in a hierarchy can be classified based on several attributes.

a) Accessing Method.

RAM access time of word is independent of its location (any order). RAM comes in two varieties

1) Static RAM and

2) Dynamic RAM

b) speed

1. High level

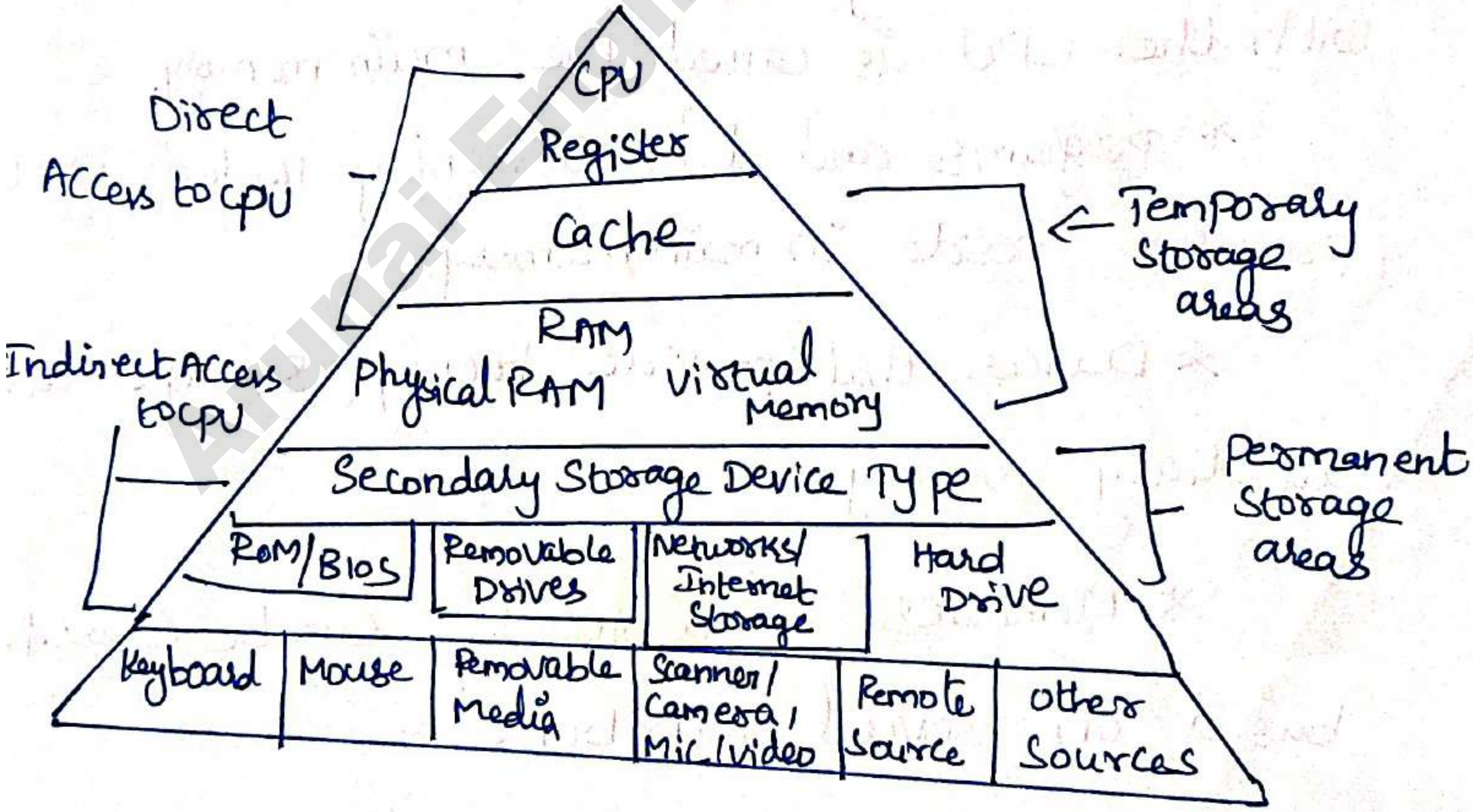
2. Lowest Level

c) Access time:

1. Primary Memory - It is directly accessed by CPU
Ex: RAM

2. Secondary Memory - It cannot be directly accessed by CPU. It is accessed by input-output channel
Ex: SAM

Memory Hierarchy



Memory Technologies

Memory Technologies

There are four primary Technologies used in memory hierarchies such as

1. SRAM technologies
2. DRAM technologies
3. Flash Memory technologies
4. Disk memory technologies

1. SRAM Technology.

* Static Random Access memory is a Part of the Random Access Memory (RAM). It is Main memory and located very close to the processor.

* SRAM'S are simply integrated circuits that are memory array with a single access port and it can provide either a read or a write.

* It has a fixed access time to any data through the read and write access times may differ.

* SRAM'S don't need to refresh so the access time is very close to the cycle time.

* Transistor hold the data as long as the power supply is not cut off

* It typically use six to eight transistors per bit to prevent the information from being disturbed when read.

* It need only minimal power to obtain the charge in standby mode.

* Most PCs and server systems used separate SRAM chips for either their primary, secondary or even caches

2. DRAM Technology:-

* In a SRAM, as long as power is applied the value can be kept indefinitely.

* In Dynamic RAM (DRAM) the value kept in a cell is stored as a charge in a capacitor

* DRAM has single transistor used to access this stored charge either to read the value or to overwrite the charge stored there.

* Because it uses only a single transistor per bit of storage they are much denser and cheaper than SRAM.

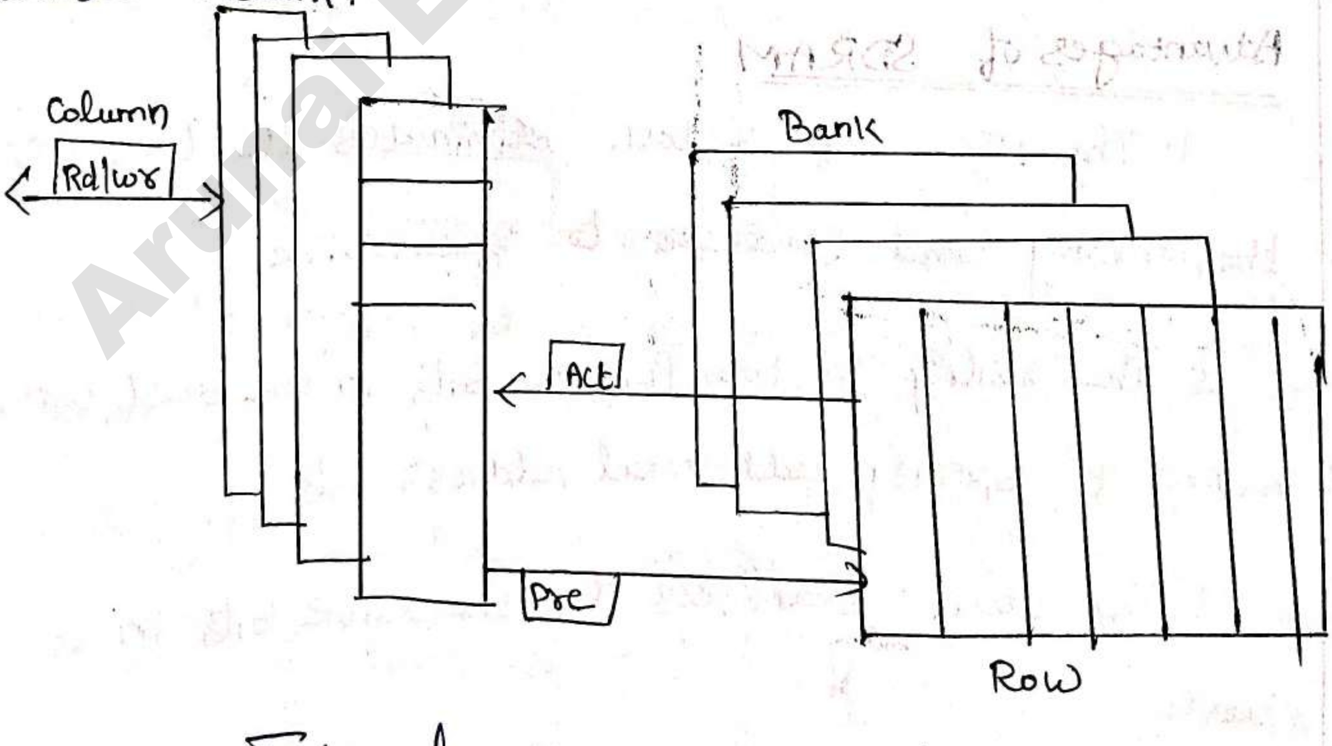
* DRAM'S Store all the charge on a capacitor so it cannot be kept indefinitely and must be Periodically refreshed.

* If every bit had to be read out of the DRAM and then written back individually, so we constantly be refreshing the DRAM by leaving no time for accessing it.

* DRAM uses a two level decoding structure and it will allow US to refresh the entire row with a read cycle followed immediately by a write cycle

* DRAM's are organized in banks, typically four for DDR3. Each bank consists of a series of rows.

* sending a PRE (Pre change) command opens or close a bank.



Internal organization of a DRAM

* A row address is sent with an Act (activate) which causes the row to transfer to a buffer, when the row is in the buffer it can be transferred by successive column address at whatever the width of the DRAM is or by specifying a block transfer and the starting addres.

* Each command as well as block transfers are synchronized with a clock.

* To improve performance DRAM'S buffer rows for repeated access.

* TO improveo the interface to processor'S DRAM'S added clocks with its and are called synchronous DRAM'S or SDRAM'S - syn. with s/y bus

Advantages of SDRAM

1. The use of a clock eliminates the time for the memory and processor to synchronize

2. The ability to transfer the bits in the burst without having to specify additional address bits

3. The clock transfers the successive bits in a burst

DDR- Double Data Rate

* The fastest version of SDRAM is called Double Data Rate.

* It means data transfers on both the rising and falling edge of the clock, thereby getting twice as much bandwidth as we expected based on the clock rate and the data width.

* The latest version of this technology is called DDR4.

* A DDR4-3200 DRAM can do 3200 millions transfers per second which means it has a 1600 MHz clock

* Personal mobile devices like the ipad use individual DRAM'S and memory for servers are commonly sold on small boards called dual inline memory modules (DIMM'S)

* DIMM contains 4-16 DRAM and they are normally organized to be 8 bytes wide for server system.

* DIMM can have so many DRAM chips and only a portion of them are used for a particular transfer.

Flash memory Technology:

* Flash memory is, a type of Electrically Erasable programmable Read only Memory (EEPROM)

* The EEPROM technologies the writes can wear out flash memory bits. Most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodelen blocks.

* This technique is called wear leveling

* In Digital cameras, flash memory is used to store picture image data.

* In MP3 players, flash memory is used to store the sound data

* Flash memory are available in modules. These modules are implemented in two types

i) Flash cards

ii) Flash Drives

4. Disk Memory Technology:

* To read and write Information on a hard disk, a movable arm containing a small electromagnetic coil called a read-write head is located just above each surface.

* The entire drive is permanently sealed to control the environment inside the drive, which in turn allows the disk head to be much closer to the drive surface.

* Each disk surface is divided into concentric circle called tracks.

* Each track is in turn divided into sector that contains the information.

* Each track may have thousands of sectors and it has 512 to 4096 bytes in size.

* Sector is one of the segments that make up a track on a magnetic disk and it is the smallest amount of information that is read or written on a disk

↳ To access the data, the OS must direct the disk through a three stage process

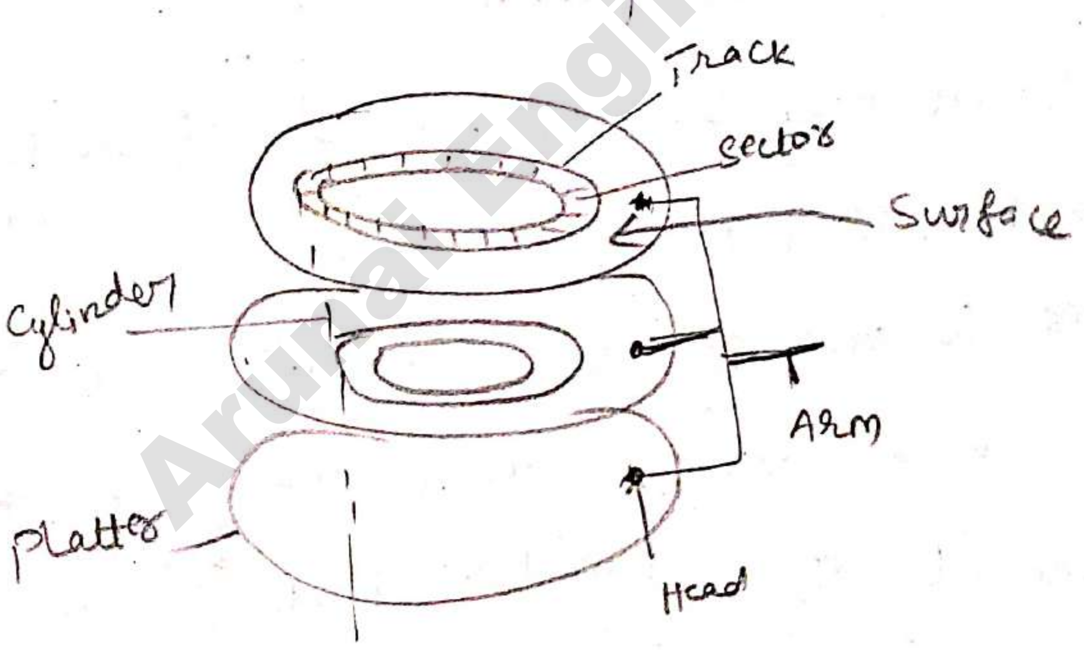
1. To position the head over the proper track. This operation is called seek and time to move the head to the desired track is called seek time

2. once the head has reached the correct track, we must wait for the desired sector to rotate

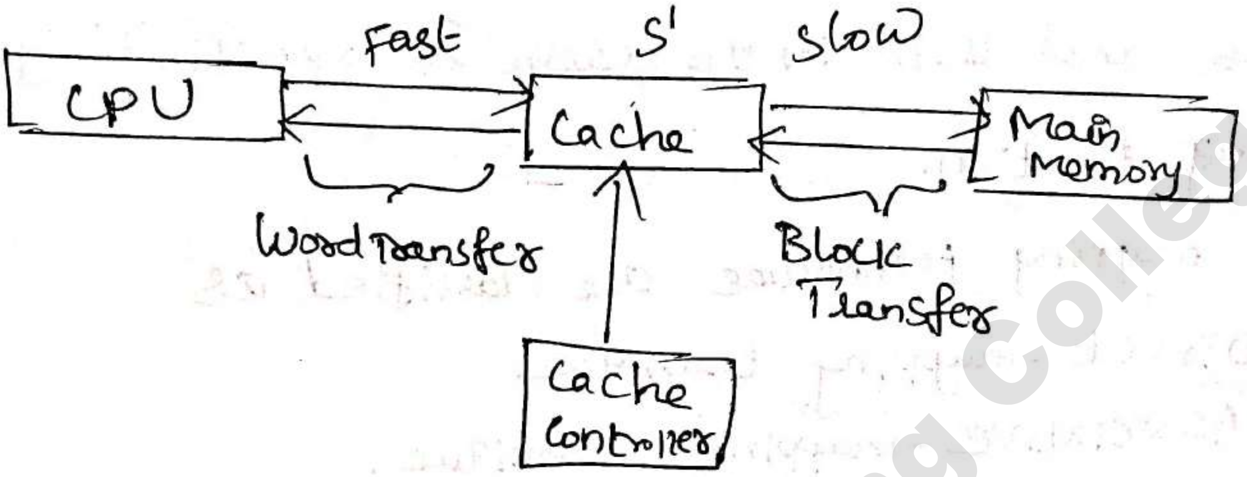
under the read/write head. This time is caued the rotational latency or rotational delay

3. Disk access is transfer time. It is the time to transfer a block of bits.

The transfer time is a function of the sector size, the rotation speed and the recording density of a track.



Cache Memory



Cache memory system

* The part of program (code) and data that work at a particular time is usually accessed from the SRAM memory. This is accomplished by loading the active part of code and data from main memory to SRAM memory.

* This small section of SRAM memory added between processor and main memory to speed up execution process is called cache memory

Mapping Techniques:

* Usually, the cache memory can store a reasonable number of memory blocks at any given time, but this number is small compared to the total number of blocks in the main memory.

* The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

The mapping techniques are classified as

1. Direct-mapping technique
2. Associative-mapping technique.
   - > fully-associative
   - > Set-associative techniques

To discuss these techniques of cache mapping we consider a cache consists of 128 blocks of 16 words each, for a total of 2048 (2k) words and assume that the main memory has 64k words. This 64k words of main memory is addressable by a 16-bit address and it can be viewed as 4k blocks of 16 words each. The group of 128 blocks of 16 words each in main memory form a page.

1: Direct-Mapping:

* It is the simplest mapping techniques.

* In this technique, each block from the main memory has only one possible location in the cache organization.

Eg: The block i of the main memory maps onto the block j (j=i modulo 128) of the cache.

∴ one of the main memory blocks 0, 128, 256, ... is loaded in the cache, it is stored in cache block 0.

Blocks 1, 129, 257, ... are stored in cache block 1

In general, the mapping Expression is

$$j = i \text{ modulo } m$$

where i = Main memory block number

j = Cache block (line) number

m = Number of blocks (lines) in the cache

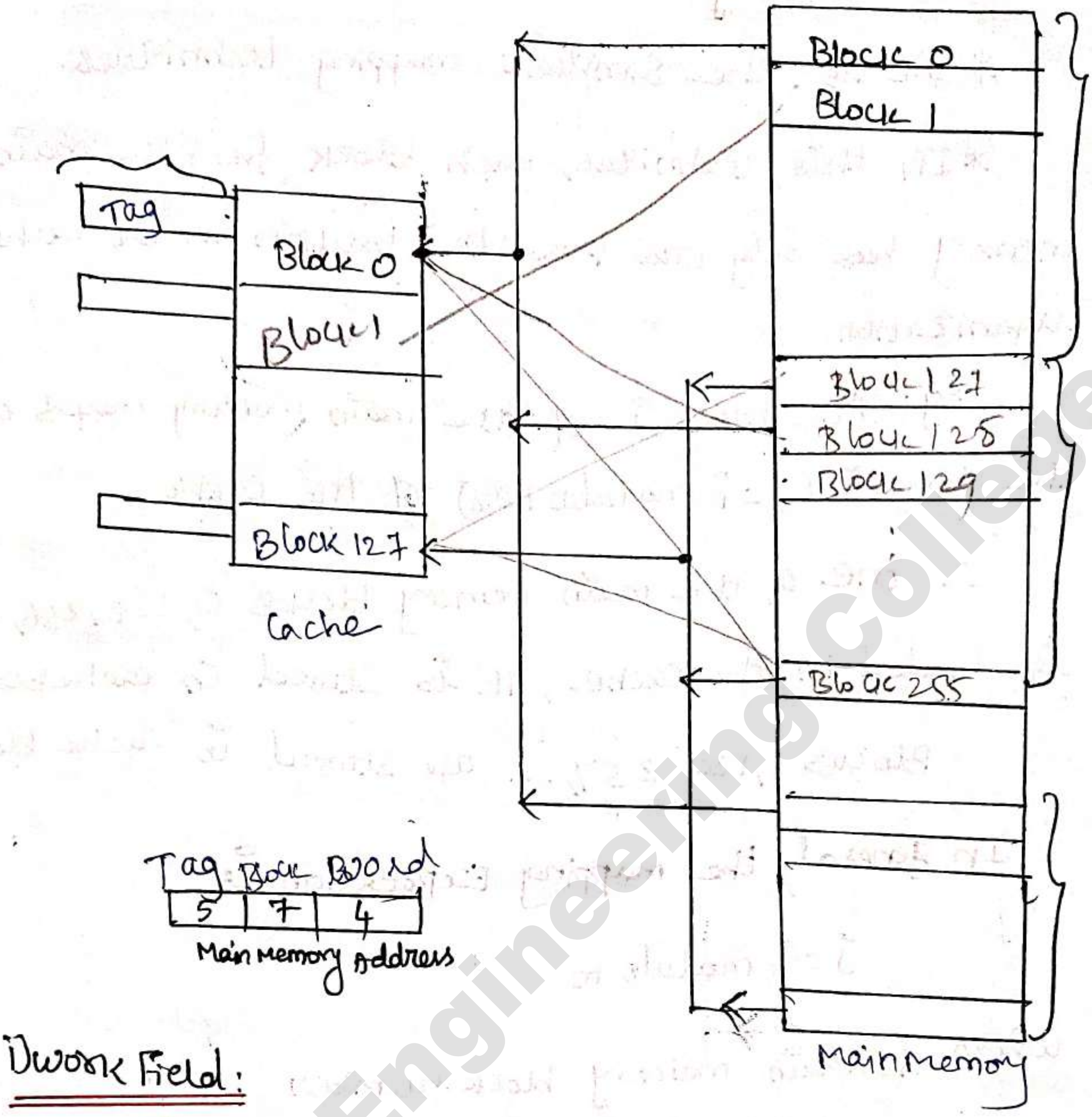
The address is divided into three fields

i) Word field

ii) Block field

iii) Tag field

Direct-Mapped cache



i) Word Field:

The lower order 4-bits select one of the 16 words in a block. This field is known as word field

ii) Block Field:

* The second field known as block field is used to distinguish a block from other blocks.

* Its length is 7-bits since $2^7 = 128$

* when a new block enters the cache, the 7-bit Cache block field determines the cache Position in which this block must be stored.

Tag field:

* It is used to store the high-order 5-bits of memory address of the block. These 5-bit (tag bits) are used to identify which of the 32 blocks (pages) that are mapped into the cache.

↳ When memory is accessed, the 7-bit cache block field of each address generated by CPU points to a particular block location in the cache.

↳ The high-order 5-bits of the address are compared with the tag bits associated with that cache location.

↳ If they match, then the desired word is in that block of the cache.

↳ If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache.

2. Associative Mapping (Fully - Associative Mapping)

* In this technique, a main memory block can be placed into any cache block position.

* There is no fix block, the memory address has only two fields

i) Word   ii) Tag

* This techniques is also referred to as fully-associative cache.

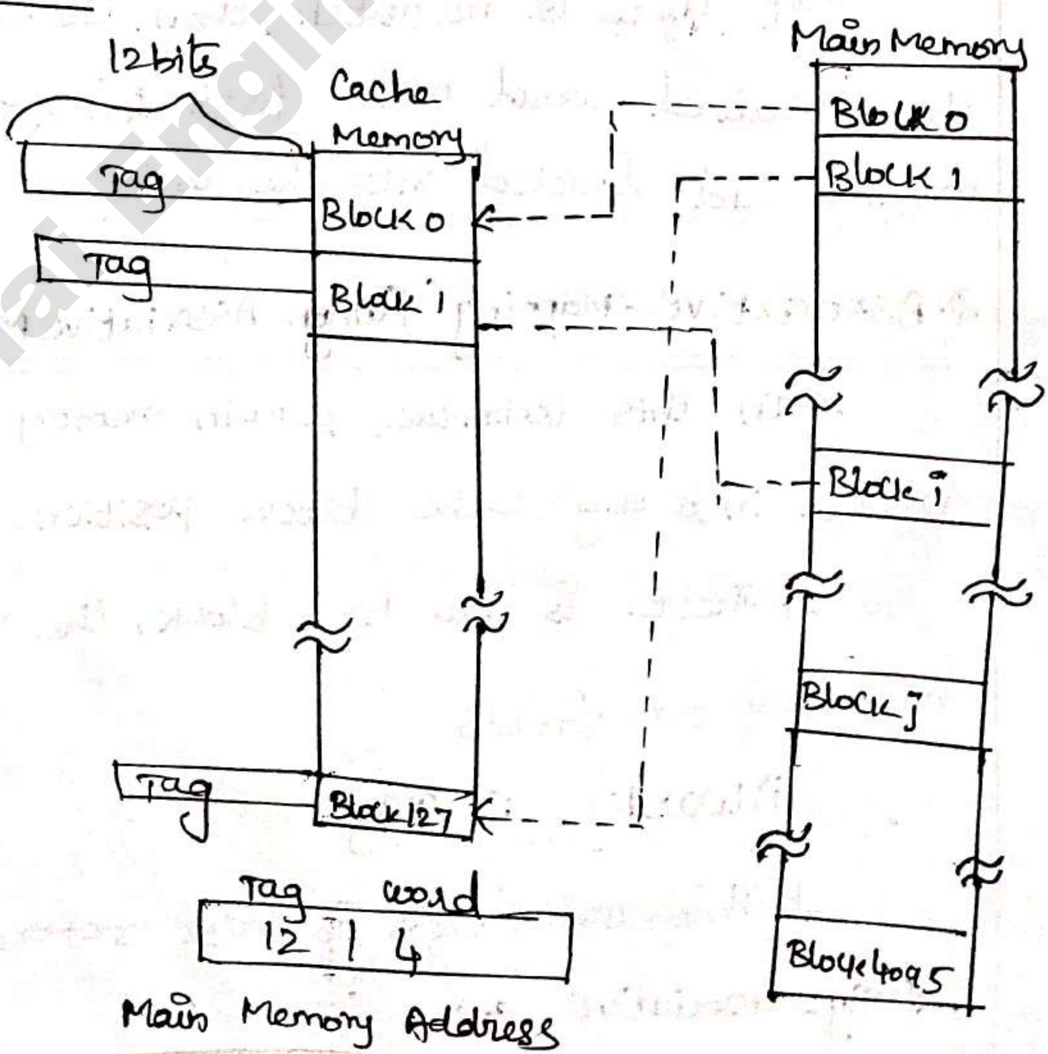
* The 12-tag bits are required to identify a memory block when it is resident in the cache.

* The high-order 12-bits of an address received from the CPU are compared to the tag bits of each block Of the cache to see if the desired block is present

* once the desired block is Present, the 4-bit word is used to identify the necessary word from the cache.

* This technique gives complete freedom in choosing the cache location in which to Place the memory block. Thus, the memory Space in the cache Can be used more efficiently

* A new block that has to be loaded into the Cache has to replace (remove) an existing block only if the cache is full.



Disadvantage:

↳ In associative-mapped cache, it is necessary to compare the high-order bits of address of the main memory with all 128 tag corresponding to each block to determine whether a given block is in the cache.

Set-Associative Mapping:-

* The Set-associative mapping is a both direct and associative mapping.

* It contains several groups of direct-mapped blocks that operate as several direct-mapped caches in Parallel.

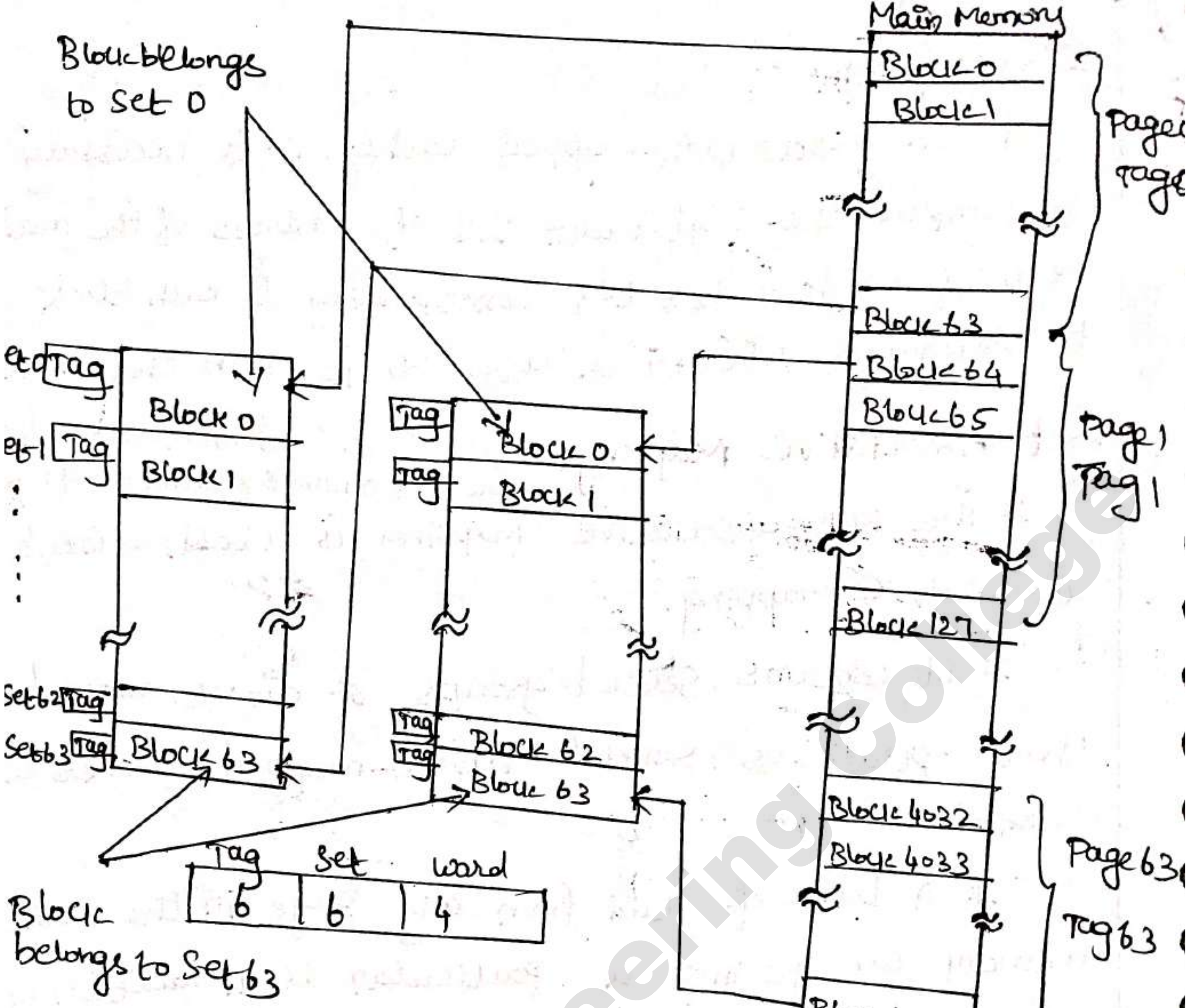
* A block of data from any Page in the main memory can go into a Particular block location of any direct-mapped cache.

* The required address comparisons depend on the number of direct-mapped caches in the cache systems.

Two-way Set-associative Cache

* Each page in the main memory is organized in such a way that the size of each page is same as the size of one directly mapped cache.

* It is called two-way set-associative cache because each block from main memory has two choices for block placement.



* In this technique, block 0, 64, 128, ... 4032 of main memory can map into any of the two (block 0) blocks of set 0, block 1, 65, 129, ... 4033 of main memory can map into any of the two (block 1) blocks of Set 1 and so on.

Memory address

Tag	set	word
6	6	4

Tag: 2^6 = 64 different memory lines in each of the $2^6 = 64$ differen set values. When the number of lines per set is n, the mapping is called n-way associative.

word (4-bits):- The lower order 4-bits are used to select one of the 16 words in a block.

Set (6-bits):- The 6-bits set field determines which set the cache contain the desired block.

Direct Mapping:

Advantage:

* It is simple and easy to implement

Disadvantage:

* It is not flexible

* Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.

Associative Mapping:

Advantages:

* It gives complete freedom in choosing a cache location to load a memory block in cache.

* space in the cache can be used efficiently.

* A new block replaces an existing block only if the cache is full.

Dis advantages:

* cost of associative mapping cache is higher than the cost of direct mapping cache.

* All the 128 tags are searched to find whether a given block is in the cache or not

Set - Associative Mapping:

Advantages:

* The contention method of direct mapping technique is solved by having a few choices for block replacement.

* The hardware cost is reduced by decreasing the size of associative search.

* It is simple to implement.

Measuring And Improving Cache Performance

* The performance of a memory system depends on the following factors.

↳ Address reference statistics - The order and frequency of the logical address, created by programs that use the memory hierarchy

↳ Access Time (t_A): Access time of each memory level

↳ Storage capacity: Memory level's capacity of storage.

↳ Block size: The size of groups/blocks transferred between levels.

Cache performance can be improved by reducing

i) The miss rate and

ii) Miss Penalty.

CPU time can be divided into two types. They are

i) The clock cycles that the CPU spends executing the program and

ii) The clock cycles that the CPU sends waiting for the memory system.

$$\text{CPU time} = \left[\begin{array}{l}\text{CPU execution clock cycles} \\ + \text{Memory-stall clock cycles}\end{array}\right] \times \text{Clock cycle time}$$

Memory-stall clock cycles can be written as the sum of the stall cycles

i) Coming from reads and

ii) coming from writes

Memory-Stall Clock cycles = Read-Stall cycles + write-Stall cy

(ie)

$$\text{Read-Stall cycles} = \frac{\text{Reads}}{\text{Programs}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$\text{write-Stall cycles} = \left[\frac{\text{writes}}{\text{program}} \times \text{write missrate} \times \text{write miss penalty}\right] + \text{write buffer stalls}$$

$$\text{Memory-Stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss Penalty}$$

Improving Cache Performance:

1. Reducing miss rates
   - Large block size
   - Larger cache size
   - Higher associativity
   - Victim Caches
2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers
3. Reducing miss penalty or miss rates via Parallelism
   - Non-blocking caches
   - Hardware Prefetching
   - compiler Prefetching
4. Reducing Cache hit time
   - Small and Simple caches
   - Avoiding address Translation
   - Pipelined cache access
   - Trace caches

Virtual Memory

Virtual Memory.

* Virtual Memory is a technique that uses main memory as a "cache" for secondary storage.

Motivations for virtual memory are

- 1. To allow efficient and safe sharing of memory among multiple programs.

2. To remove the programming burdens of a small, limited amount of main memory.

* To allow multiple virtual machines to share the same memory we must protect the virtual machines from each other.

* we need to ensure that a program can only read and write the portions of main memory that has been assigned to virtual machines.

* Main memory need to contain only the active portions of the many virtual machines and virtual memory allow us to efficiently share the processor as well as the main memory.

* We cannot know which virtual machines will share the memory with other virtual machines whe we compile them.

* Because the virtual machines sharing the memory that can change dynamically while the virtual machines are running.

* We need to compile each program into its own address space.

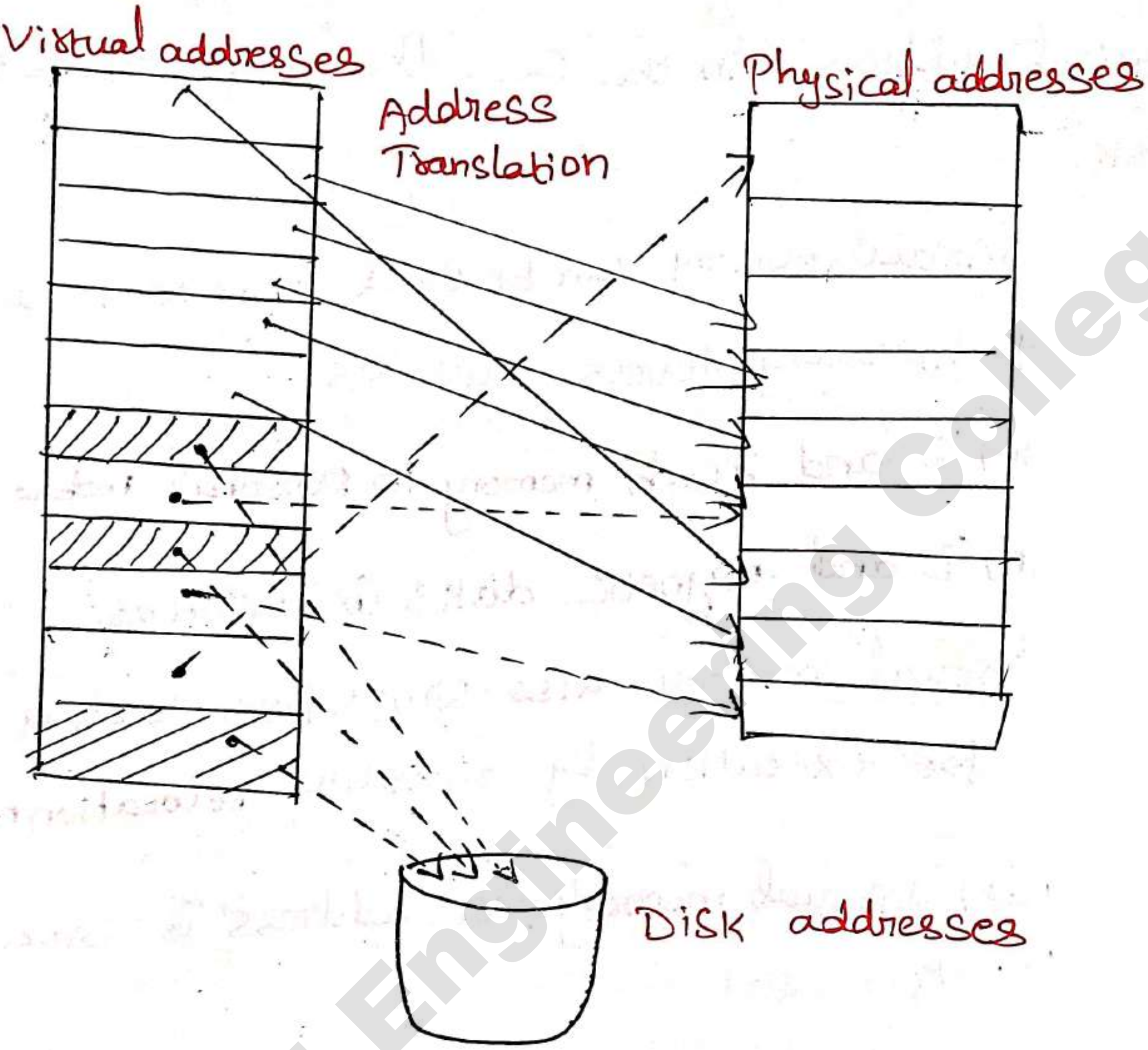
* Virtual memory implements the translation of a program address space to physical address.

* This translation process enforces protection of a program's address space from other virtual machines.

* In a virtual memory block is called as page and a virtual memory miss is called as page fault. with virtual memory, the processor produces a virtual address.

* Virtual address is an address that corresponds to a location in virtual space and is translated by address mapping to a physical address when the memory is accessed.

* The virtually addressed memory with pages mapped to main memory. This process is called address mapping or address translation.



The Block of Memory are mapped virtual addresses to physical addresses.

* The processor generates virtual addresses while the memory is accessed using physical addresses. Both virtual memory and physical memory are broken into pages.

* A virtual page is mapped to physical page. It is also possible for a virtual page to be absent from main memory and not be mapped to a physical address. in the case the page resides on disk.

* virtual memory can be used to control two memory hierarchy levels such as

1. DRAM'S and flash memory in personal mobile devices

2. DRAM'S and magnetic disks in servers.

* virtual memory also simplifies loading the program for execution by providing relocation method.

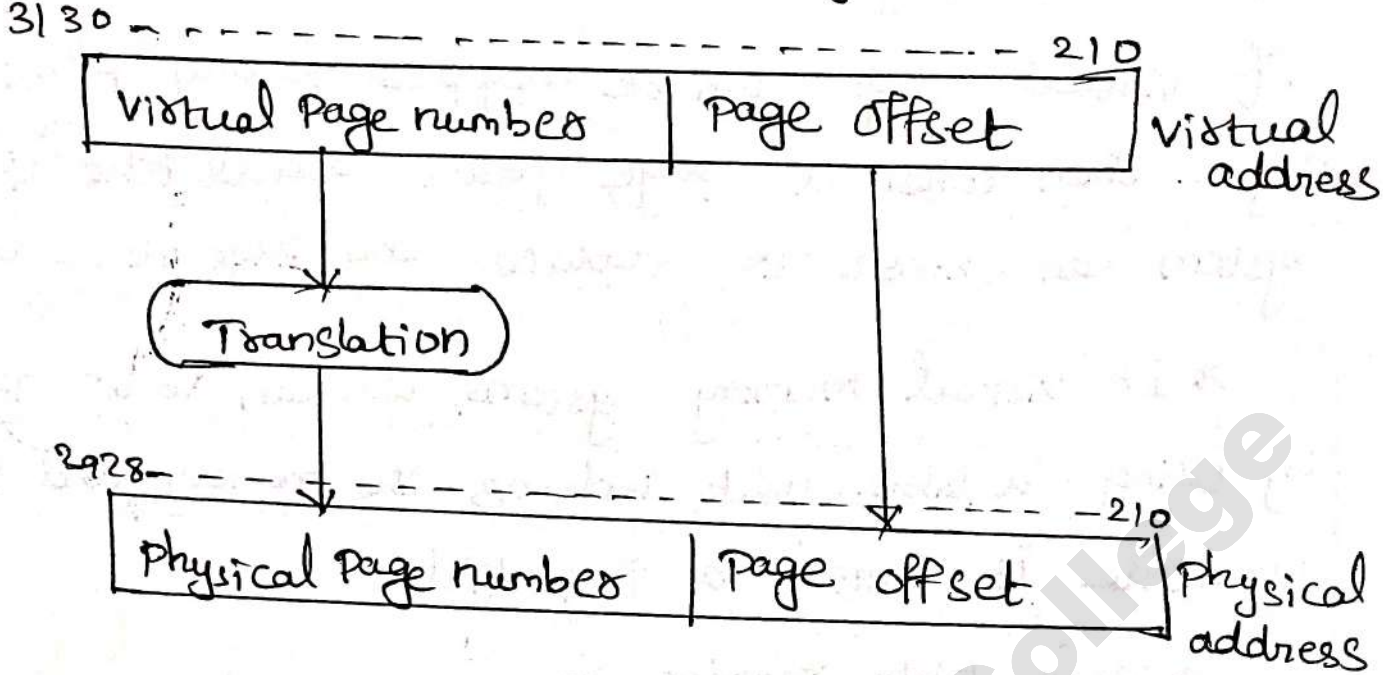
* In virtual memory, the address is broken into two parts

1) virtual page number

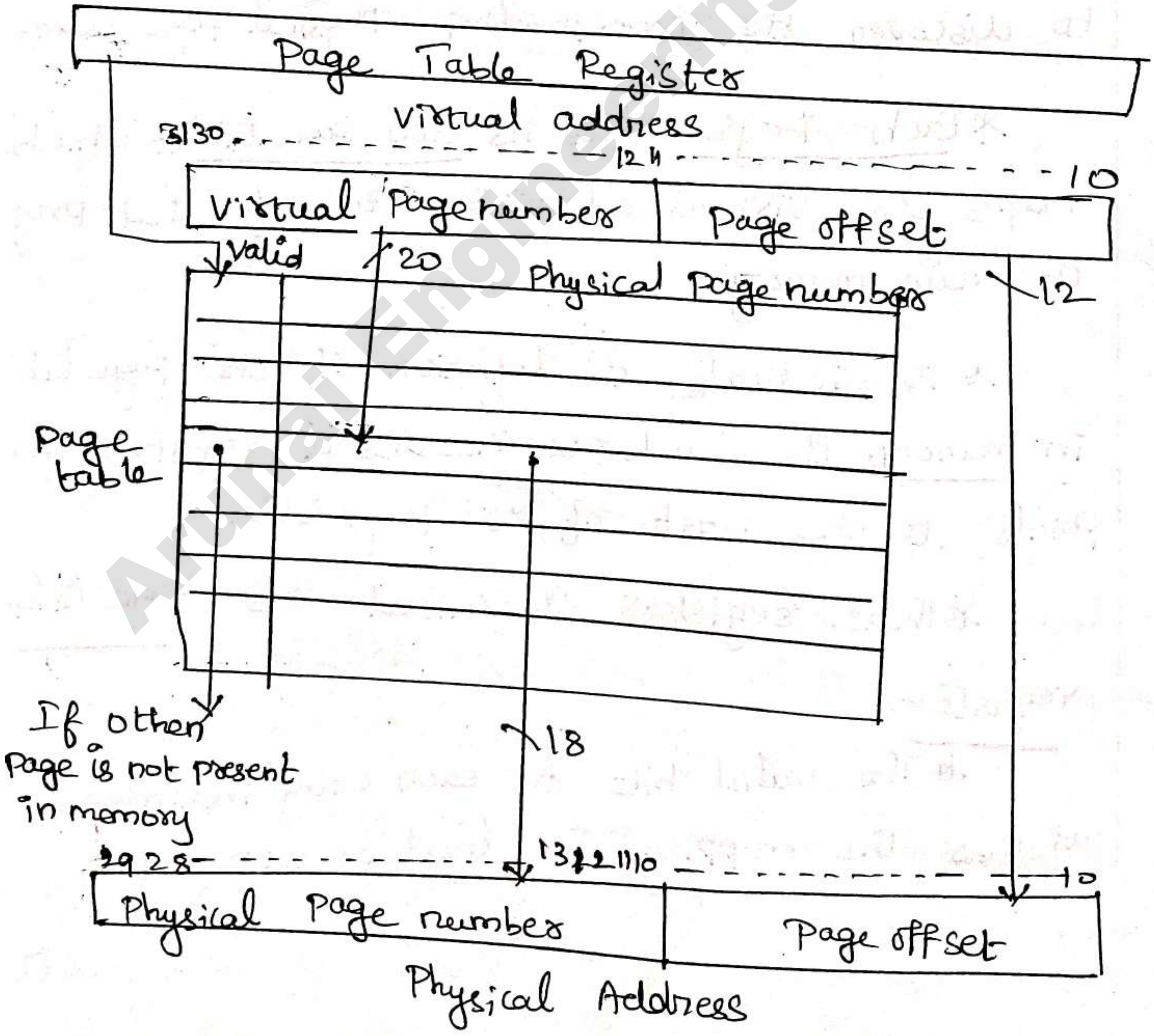
2) Page offset

* The number of bits in the page offset field determines the page size and the number of pages addressable with the virtual address need not match the number of pages addressable with the physical address.

Mapping from a virtual to Physical Address



Placing a Page:



* TO reduce the page fault the designer must, optimize page placement in more attractive way. If virtual page can be mapped to any physical page then when a page fault occurs ~~the~~ operating system can choose to replace any page it wants.

* In virtual memory systems, we can locate pages by Using a table that indexes the memory and this structure is called a page table.

* Page table resides in memory and its indexed with the page number from the virtual address to discover the corresponding physical page number.

* Each program has its own page table, which maps the virtual address space of that program to main memory.

* TO indicate the location of the page table in memory, the hardware includes a register that points to the start of the page table.

* These registers are called as page table register.

* The valid bit for each entry indicates whether the mapping is legal or not.

↳If valid bit is off, -Then the page is not Present in memory.

↳If valid bit is on - The Page is in memory.

* The Entry contains the physical page number because the page table contains a mapping for every possible virtual page.

Page Faults:

* If the valid bit for a virtual Page is off it causes a page fault. If page fault occur then the control has given to operating system.

* once the operating system gets control, it must find the Page in the next level of the hierarchy and decide where to place the requested page in main memory.

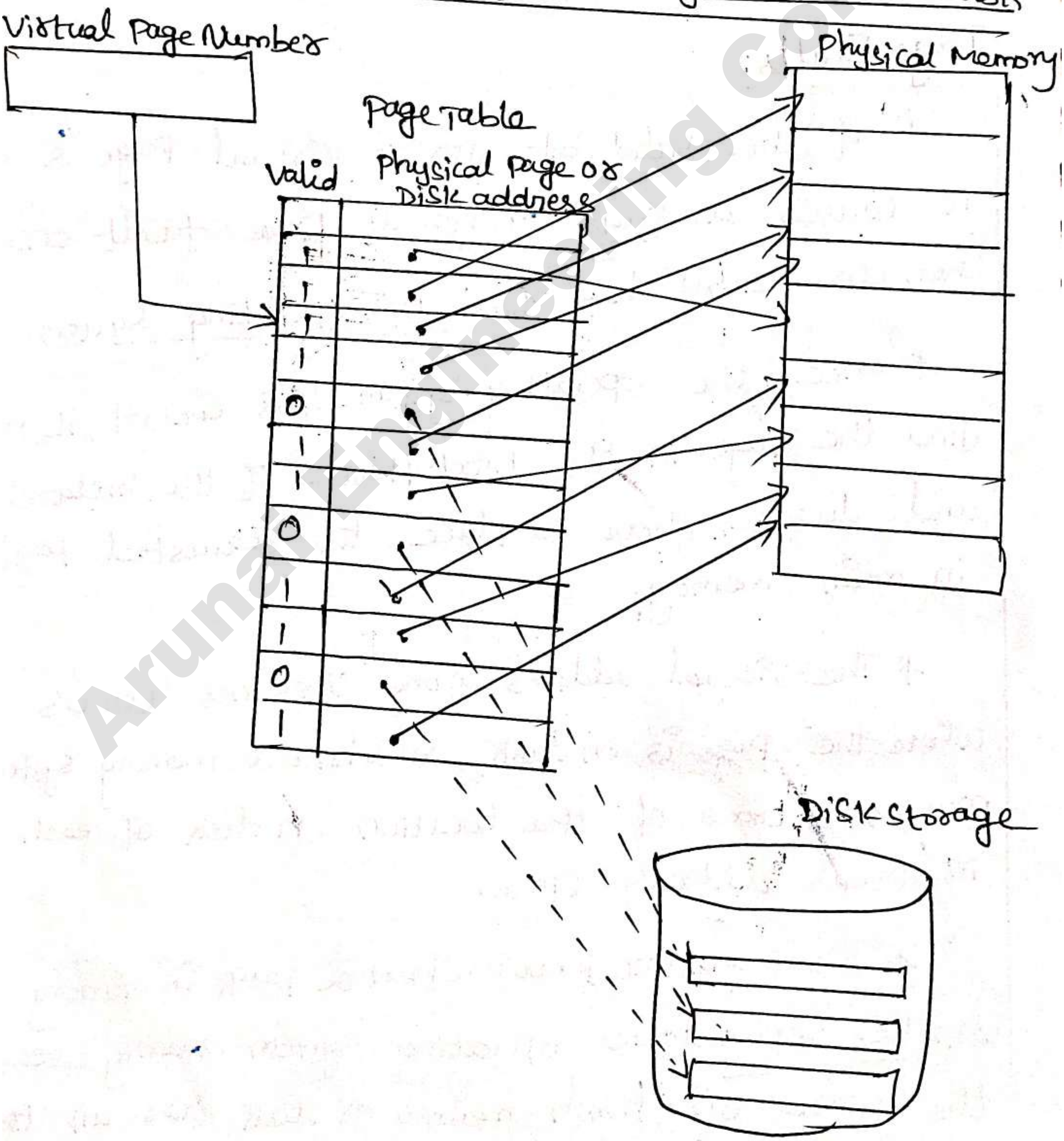
* The virtual address alone does not tell us where the page is on disk, so virtual memory system, must keep track of the location on disk of each page in virtual address space.

* We do not know when a page in memory will be replaced. So operating system usually creates the space on flash memory or disk for all the

Pages of a process when it creates the process. This space is called the swap space.

Swap space - is a space on the disk or flash memory reserved for the full virtual memory space of a process.

Page table Maps each page in virtual memory to either a page in main memory or a page stored on disk



* The virtual page number is used to index the page table. If the valid bit is on, the page table supplies the physical page number corresponding to the virtual page.

* If the valid bit is off, the page currently resides only on disk at a specified disk address. pages in the main memory and the pages on disk are the same size.

* operating system also creates a data structure that tracks which processes and which virtual addresses use each physical page.

* when a fault occurs, if all the pages in main memory are in use then the operating system must choose a page to replace.

* OS follow LRU (Least Recently used) replacement Scheme. The replaced pages are written to swap space on the disk.

Writes in virtual Memory system:

Virtual memory system must use write back it performing the individual writes into the page in memory and copying the page back to disk when it is replaced in the mory.

Advantage:

* write back scheme has major advantage in virtual memory system.

* Because the disk transfer time is small compared with its access time and copying back an entire page is much more efficient than writing individual words back to the disk.

Disadvantage:

* It is costly

* Dirty bit is set when any word in a page is written.

* If the OS chooses to replace the page, the dirty bit indicates whether the page needs to be written out before its location in memory can be given to another page.

TLB

Translation-lookaside buffer (TLB): Making Address Translation fast

* For every read and write access, the page table information is used by memory management unit. (MMU)

* so, if the page table is kept in MMU, it will be easy to access, But, the page table is too large to fit in MMU.

* It is impossible to include a complete page table as the part of processor chip.

* Therefore, the page table is kept in the main memory.

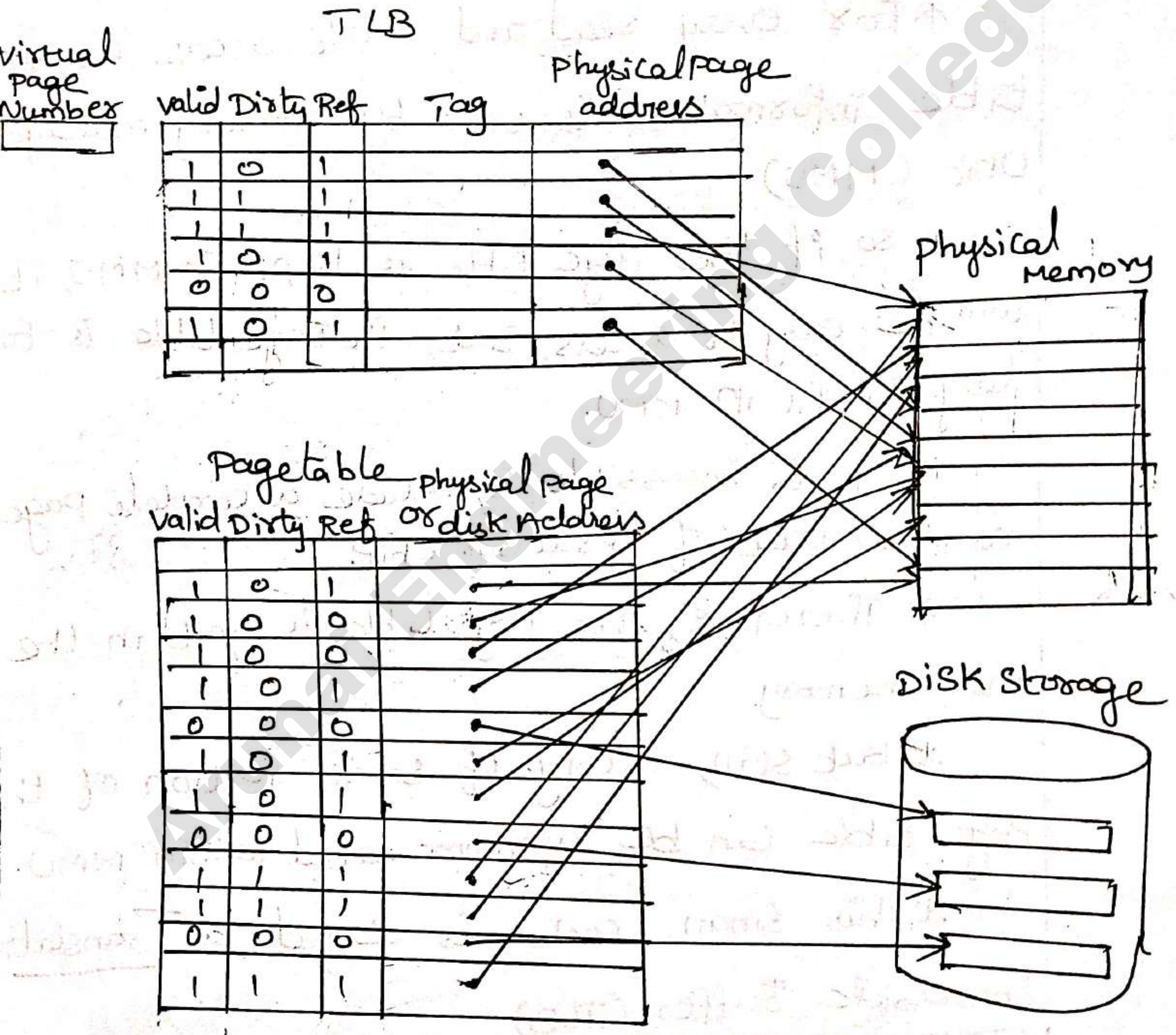
* But still, a copy of small portion of the page table can be accommodated within MMU.

* This small cache is called a Translation Lookaside Buffer (TLB).

* TLB contains the entries that correspond to most recently accessed pages.

* The structure of TLB is slightly modified from the page table in main memory.

* In addition to the information in the page table entry, TLB includes the virtual address of the entry (virtual page number)



TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1		
1	1	1		
1	1	1		
1	0	1		
0	0	0		
1	0	1		

Pagetable

Valid	Dirty	Ref	Physical page or disk Address
1	0	1	
1	0	0	
1	0	0	
1	0	1	
0	0	0	
1	0	1	
1	0	1	
0	0	0	
1	1	1	
1	1	1	
0	0	0	
1	1	1	

* Each tag entry in the TLB holds a Portion of the virtual page numbers

* Each data entry of the TLB holds a physical page number.

* Because, we access the TLB instead of the page table on every reference, the TLB will need to include other status bit, such as the dirty and the reference bits.

* on every reference, we look up the virtual page number in the TLB.

* If we get a TLB hit, the Physical page number is used to form the address and the corresponding reference bit is turned on.

* If the processor is Performing a write, the dirty bit is also turned on.

* If a miss in the TLB occurs, must determine whether it is a page fault or merely a TLB miss

i) The page is present in memory, then the TLB miss indicates only that the translation is missing.

In such cases, the processor can handle the TLB miss by loading the translation from the page table into the TLB and then trying the reference again.

ii) The Page is not present in the memory, then the TLB miss indicates a true page fault.

Accessing I/O Devices

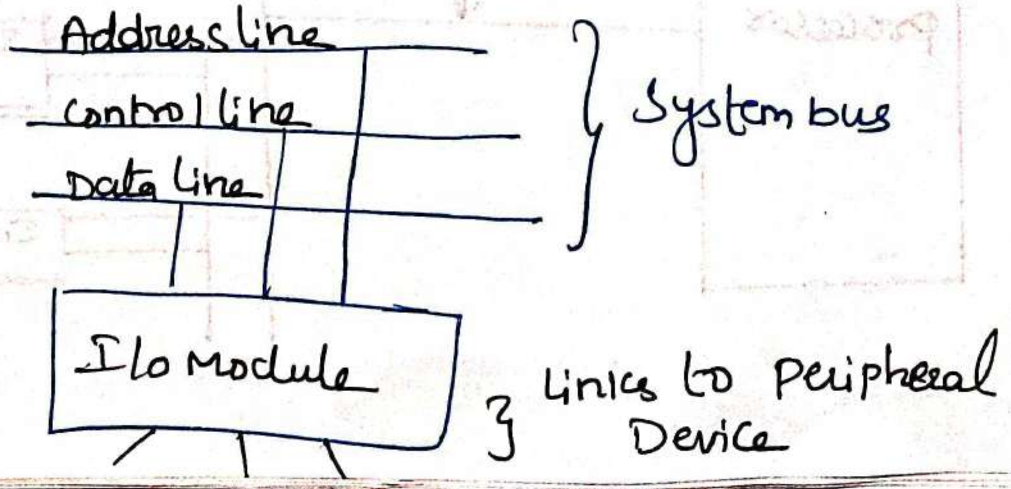
* The Input/output (I/o) device is the interface between the users and the computers, plays an important role in the performance of computer system.

* Modern computers use single bus arrangement for connecting I/o devices to cpu and memory.

* The bus enables all the devices connected to it to exchange information.

* BUS consists of 3 set of lines.

1. Address lines - processor places a particular address on address lines.
2. Control lines - Device which recognizes this address responds to the commands issued on the control lines
3. Data lines - processor requests for either Read/write. The data will be placed on Data lines



I/o Interfacing Techniques:

* I/o devices can be interfaced to a computer system I/O in two ways, which are called interfacing techniques.

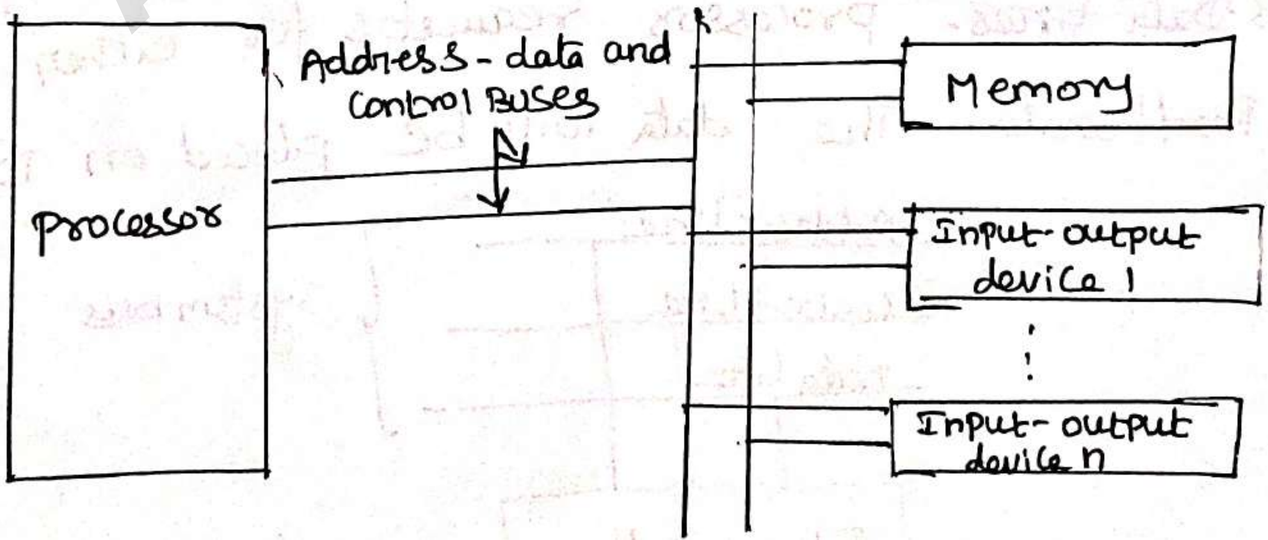
*Memory mapped I/O   * I/o mapped I/o

*Memory Mapped I/O:-

↳ Machines which are assigned a part of the main memory address space to I/o ports, this is called Memory mapped I/o.

↳ The Memory load and store instructions are used to transfer a word of data to or from an I/O port, no special I/o instructions are needed.

Memory-mapped I/o Device organization on the common Buses and Memory and I/o Interfaces



↳ I/O mapped I/O:

* In Input/output mapped I/O technique, the memory and I/O address spaces are kept separate.

* Some processors have separate control signal, which are issued when processing the instructions for I/O operations.

* A memory referenced instruction activation the READ or WRITE control line and does not affect the I/O devices.

Programmed I/O

* If IO operations are completely controlled by the CPU, the computer is said to be using Programmed IO.

* In this case, the CPU executes programs that initiate, direct and terminate the IO operations.

* This type of control can be implemented easily with less hardware.

* It is more useful in small, low-speed systems where hardware cost must be minimized.

* Data transfer is between two programmable registers. One is CPU register and the other is attached to the IO device.

* The IO device does not have direct access to main memory. A data transfer from an IO device to memory requires the CPU to execute several

Instructions. It includes an input instruction to transfer a word from the Io device to the CPU and a store instruction to transfer the word from CPU to M.

Memory-Mapped Io

* In programmed Io systems, the CPU, memory and Io devices communicate via system bus.

* The address lines of the system bus that are used to select memory locations can also be used to select Io devices.

* An Io device is connected to the bus via an Io port.

* In CPU's perspective, an Io device is an addressable data register.

* If a part of the main-memory address space is assigned to Io ports, then such systems are called as Memory-Mapped Io systems.

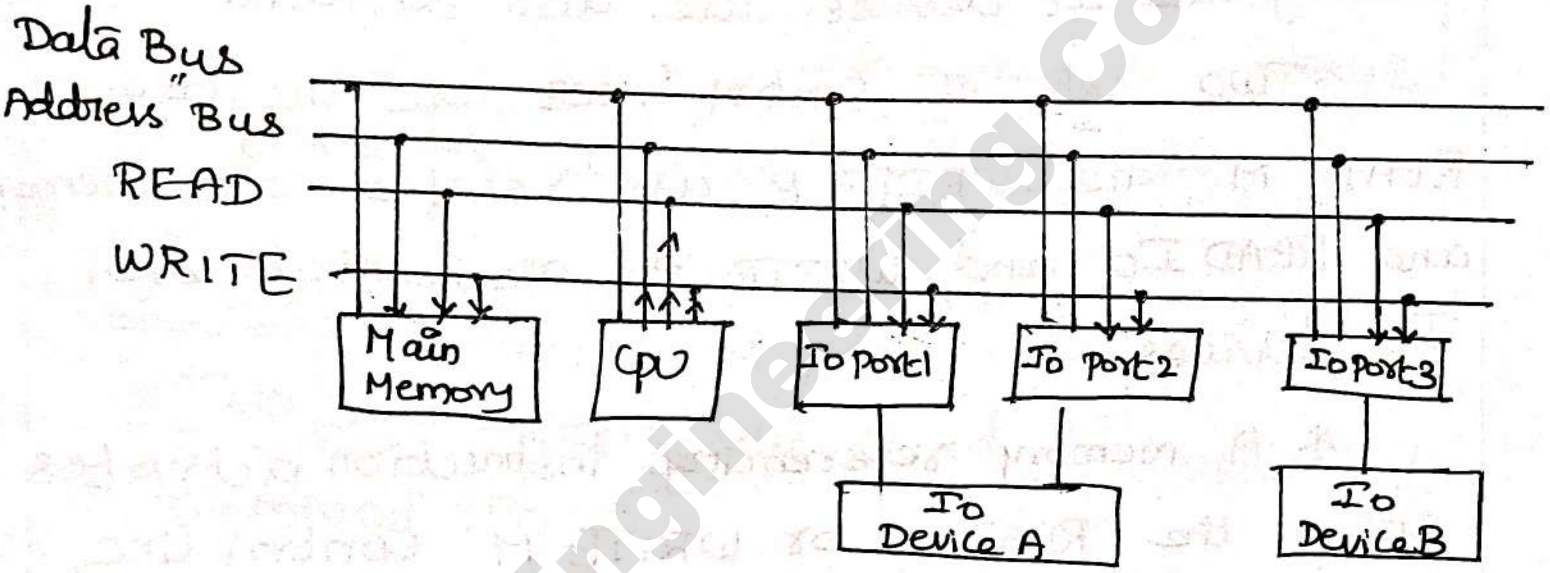
* A memory-referencing instructions becomes Io instructions automatically, if the address is address of an Io port.

* The usual memory load and store instructions are used to transfer data words to or from Io ports. so no special Io instructions are needed.

Disadvantage:

* CPU spend a lot of time in performing IO related functions.

Eg: IO -related function is testing the status of IO devices



Programmed IO with Memory - Mapped IO

* The control lines READ and WRITE are activated by the cpu for processing memory reference Instructions and IO Instructions.

* This technique is used in Motorola 680X0 series

IO-Mapped IO:

* In IO-mapped IO systems, the memory and IO address space are separate.

* Similarly, the control lines used for activating memory and IO devices are also different.

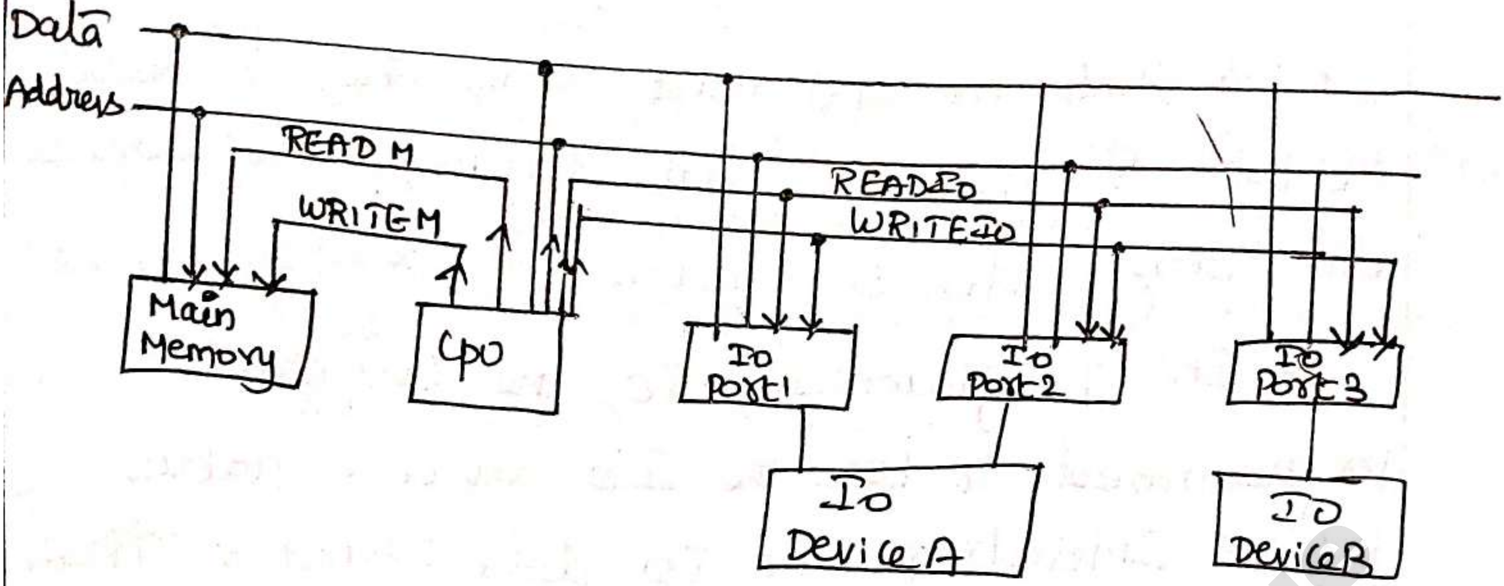
* Two sets of control lines are available READ M and WRITE M are related with memory and READ IO and WRITE IO are related with IO devices

* A memory referencing instruction activates either the READ M or WRITE M control line, but these signals do not affect the IO devices.

* The CPU executes separate IO Instructions to activate the READ IO and WRITE IO control lines.

* It causes a word to be transferred between the addressed port and the CPU.

* This technique is used in Intel 80x86 microprocessor series.



Programmed Io with Io-Mapped IO

Io Instructions:

* Two Io Instructions are used to implement Programmed Io.

*Intel 80X86 series of microprocessor series have two Io instructions called IN and OUT.

IN: The instruction IN X causes a word to be transferred from Io port X to the accumulator register A.

OUT: The instruction OUT X transfer a word from the accumulator register A to the Io Port X.

Io Data transfer:

* when the CPU executes an Io instruction such as IN or OUT, the addressed Io port must be ready to respond to the instruction

* For that, the CPU must know the IO device's Status so that the data transfer is carried out only when the device is ready.

* In programmed IO, the CPU can be programmed to test the IO device's status before Initiating the IO data transfer. The Status is specified by a single bit Information.

* The CPU must perform the following steps to determine the Status of an IO device:

1. Read the IO device's Status bit.
2. Test the Status bit to determine if the device is ready to transfer data.
3. If ready, proceed with data transfer, otherwise go to step 1.

Limitations of programmed IO:

The programmed IO method has two limitations

1. The Speed of the CPU is reduced due to low speed IO devices. The Speed with which the CPU can test and transfers data between IO devices is limited due to low transfer rate of IO devices.
2. Most of the CPU time is wasted. The time that CPU spends testing IO device Status and executing IO data transfers is too long. That can be Spent on other tasks.

Direct Memory Access

DMA controller:

↳ The Programmed I/o Method has two limitations

1. The speed with which the cpu can test and service Io device limits Io data transfer rates.

2. The time that the cpu spends testing I/o device status and executing I/o data transfers can often be better spent on other tasks.

↳ The influence of the cpu on I/o transfer rates is two-fold..

1. A delay occurs while an Io device needing service waits to be tested by the cpu. If there are many Io devices in the system then each device may be tested infrequently.

2. Programmed Io transmits data through the cpu rather than allowing it to be passed directly from main memory to the I/o device and vice versa.

*DMA and interrupt circuits use special control lines it has generic names as DMA REQUEST and INTERRUPT REQUEST, these two lines are connected the I/o devices to the CPU

*DMA and interrupt circuits increase the speed of IO operations by eliminating most of the role played by the CPU.

* Signals on these lines cause the CPU to suspend its current activities at appropriate breakpoints and attend to the DMA or interrupt request.

* These special request lines eliminate the need for the CPU to execute routines that determine I/O device status.

*DMA also allow I/O data transfers to take place without the execution of IO instructions by the CPU.

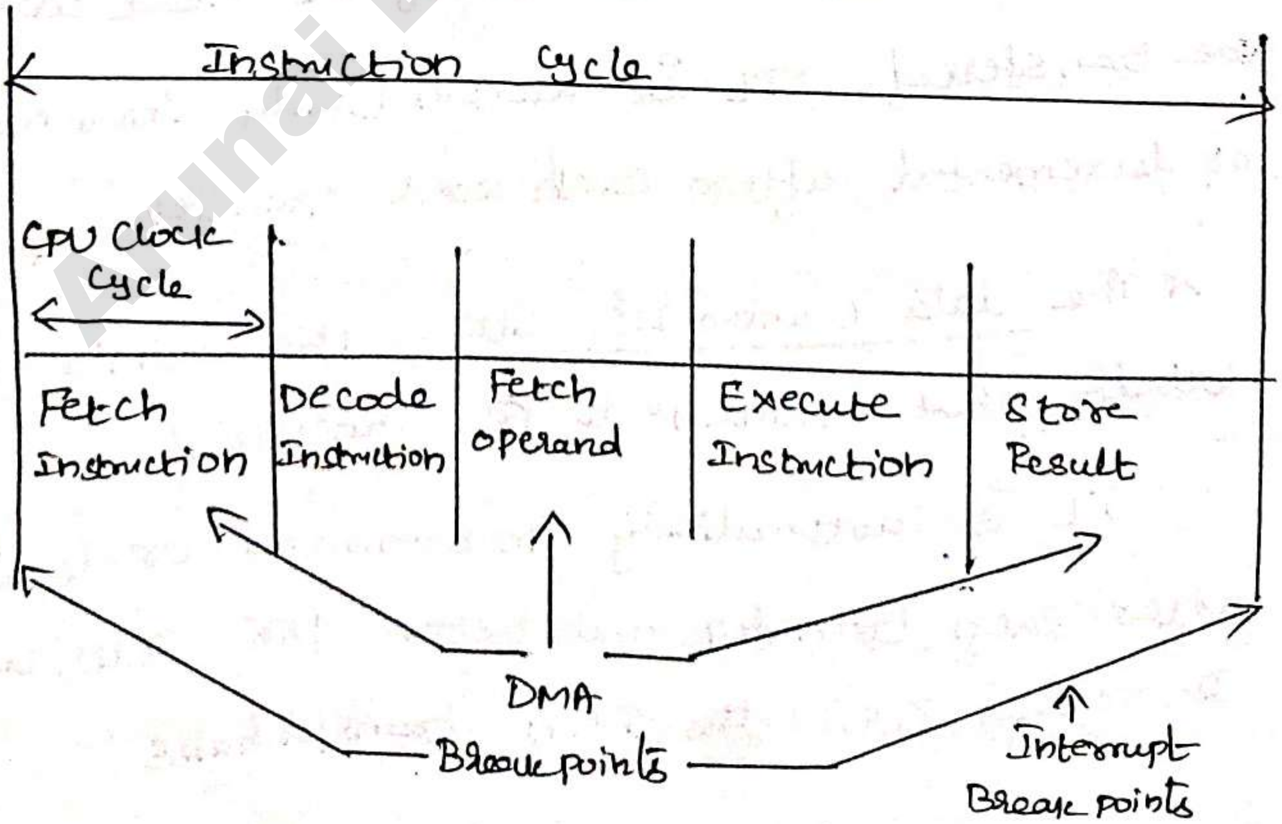
*DMA request by an Iodevice only requires the CPU to grant control of the memory (system) bus to the requesting device

The CPU can yield control at the end of any transactions involving the use of system bus.

Typical seauence of CPU actions during execution of a single Instruction.

↳ The instruction cycle is collection of number of CPU cycles and several cycles are required use of the system bus. During the instruction cycle there are five break points when the CPU can respond to a DMA request.

↳ when such a request is received by the CPU, it waits until the next breakpoint, release the system bus and signals the requesting IO device by activating a DMA ACKNOWLEDGE control line.



Direct Memory Access (DMA)

The hardware needed to implement DMA

* The circuit assumes that all access to main memory is via a shared system bus.

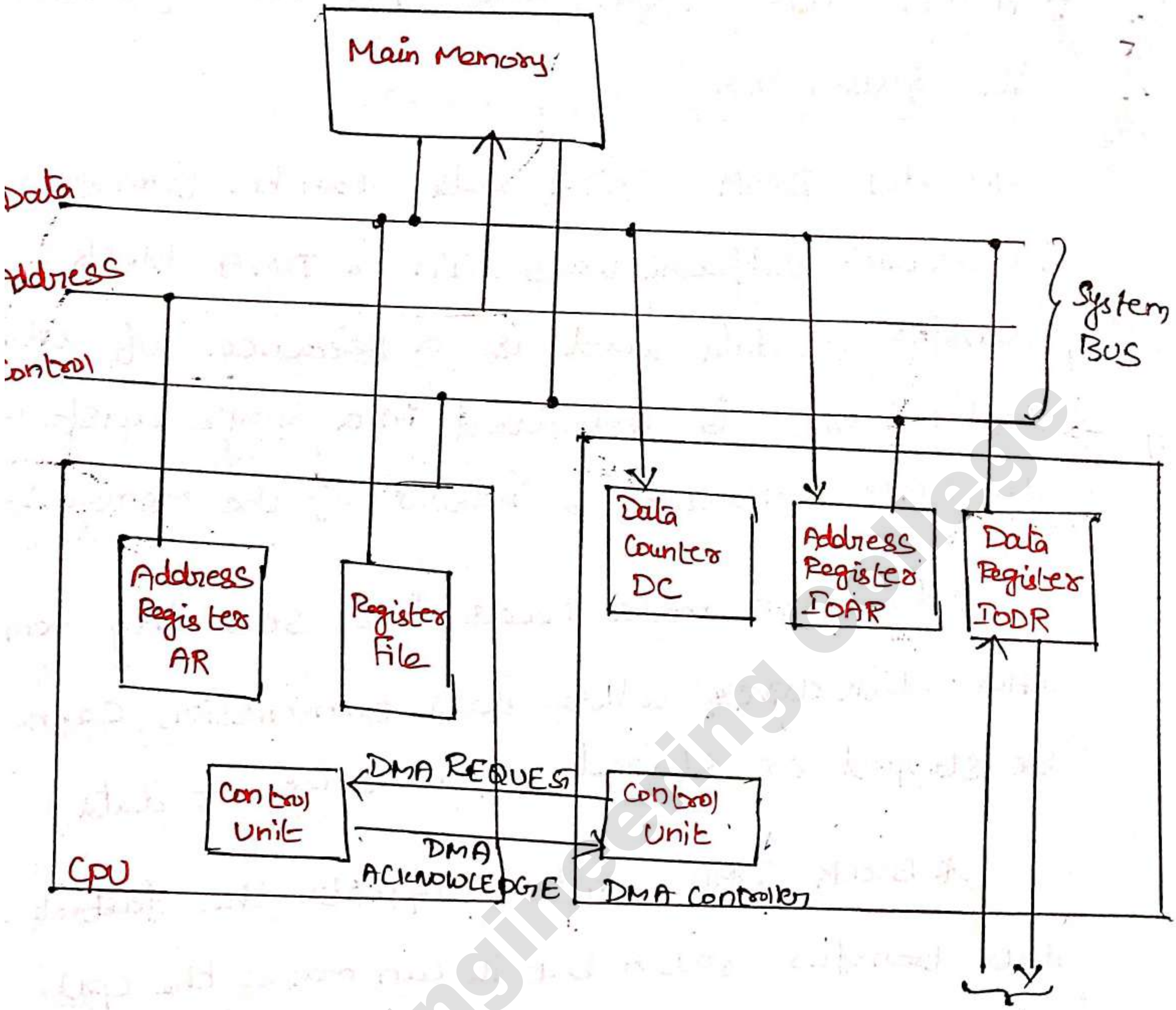
* The IO device is connected to the System bus via a Special interface circuit called DMA Controller. It contains a data buffer register IODR, it also controls an address register IOAR and data count Register DC.

* These registers enable the DMA controller to transfer data to or from a Contiguous region of memory.

* IOAR Stores the address of the next word to be transferred. It is automatically incremented or decremented after each word transfer.

* The data Counter DC Stores the number of words that remain to be transferred.

It is automatically incremented or decremented after each transfer and tested for zero, when DC reaches zero the DMA transfer halts



Circuit Required for DMA

* The DMA Controller is normally provided with interrupt capabilities to send an interrupt to the CPU to signal the end of the IO data transfer.

* The logic needed to control DMA can be placed in a single IC with other IO control circuits.

* A DMA Controller can be designed to supervise DMA transfers involving several IO devices, each

device has a different priority of access to the system bus.

* Under DMA Control data Can be transferred in Several different ways. In a DMA block transfer a data word is a Seauence of arbitrary length, that is transferred in a single burst while the DMA Controller is master of the memory bus

* This DMA mode needs the secondary memories like disk drives, where data transmission cannot be stopped or slowed without loss of data

* Block DMA transfer supports the fastest IO data transfer rates but it can make the CPU inactive for relatively long periods by tying up the system bus.

* An alternative technique called cycle stealing, it allows the DMA Controller to use the System bus to transfer one word data after it must return Control of the bus to the CPU

* cycle Stealing reduces the maximum Io transfer. rate and also reduces the interference by the DMA Controller in the CPU'S memory access.

* It is possible to eliminate this interference Completely by designing. the DMA interfaces.

* CPU is not actually Using the System bus. This is called transparent DMA.

DMA circuit has the following DMA transfer Process:

1. The CPU executes two Io instructions, which load the DMA registers IOAR and DC with their Initial values

a) IOAR register Should contain the base address of the memory region to be used in the data transfer.

b) DC register should contain the number of words to be transferred to or from the region.

2. when the DMA Controller is ready to transmit or receive data, it activates the DMA REQUEST line of the CPU.

a) The CPU waits for the next DMA breakpoint

b) DMA controller relinquishes control of the data and address lines and activates DMA ACKNOWLEDGE

c) DMA REQUEST and DMA ACKNOWLEDGE are essentially used in BUS REQUEST and BUST GRANT lines for Control of the system bus

d) DMA requests from several DMA controllers are resolved by bus priority control techniques.

3. Now DMA controller transfers data directly to or from main memory. After transferring a word, IOAR and DC registers are updated.

4. If DC has not yet reached zero but the IO device is not ready to send or receive the next batch of data then the DMA controller will release the system bus to the CPU by deactivating the DMA REQUEST line

a) The CPU responds by deactivating DMA ACKNOWLEDGE and resuming control of the system bus.

5. If DC register is decremented to zero then the DMA controller again relinquished control of the System bus. It may also send an interrupt request signal to the CPU. The CPU respondsby halting the IO device or by Initiating a new DMA transfer

* DMA can be obtained under a general method for System bus arbitration.

Interrupts

* An interrupt is an unscheduled event that disrupts program execution, used to detect overflow.

* Interrupt is an exception that comes from outside of the processor.

There are some types of interrupts. They are

1. Vectored Interrupt:

* In a vectored interrupt, the address to which control is transferred is determined by the cause of the execution.

2. Precise Interrupt:

* An interrupt or exception that is always associated with the correct instruction in pipelined computers.

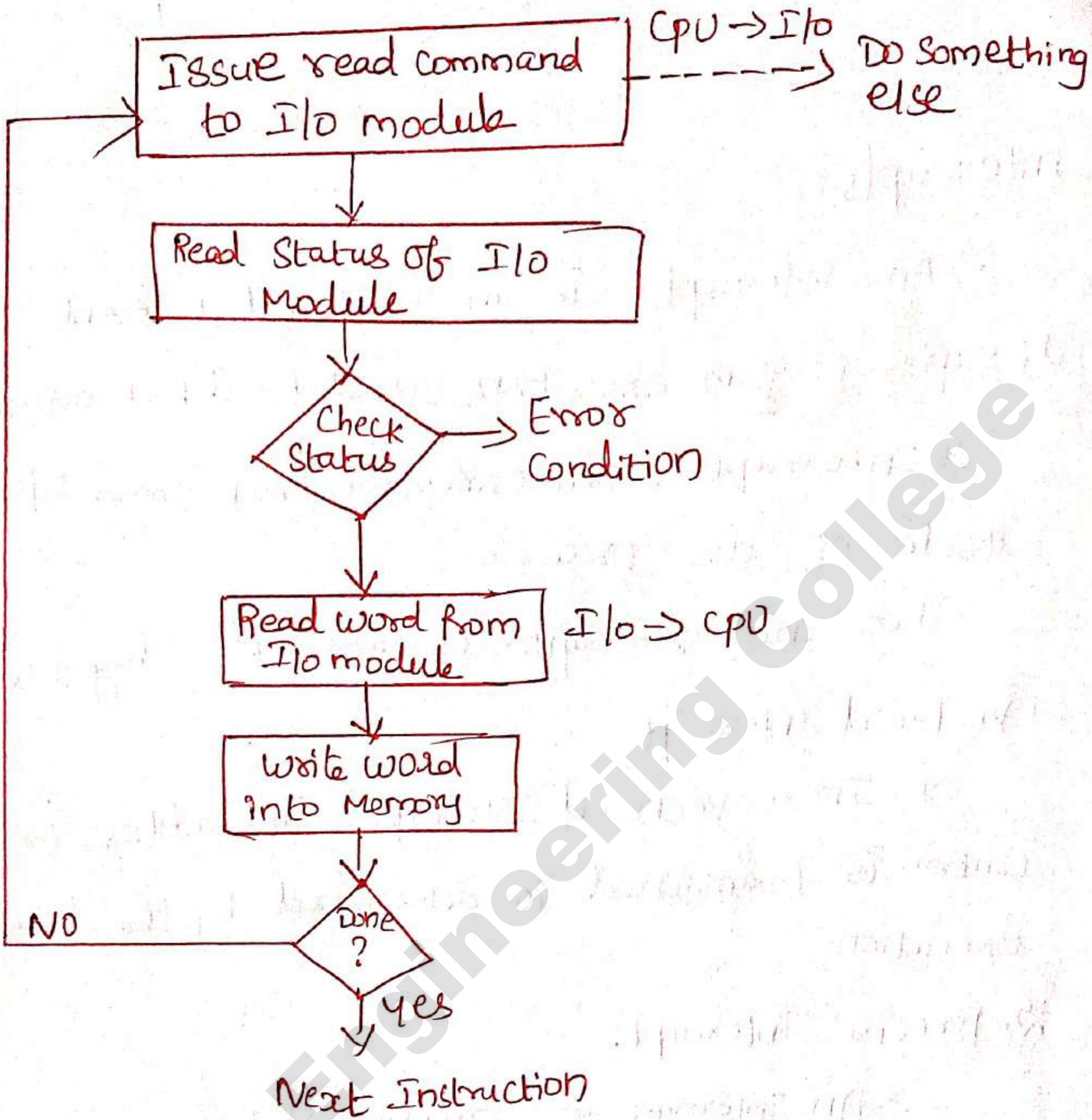
3. Imprecise Interrupt:

* An Interrupt or exception in pipelined computers that is not associated with the exact instruction, that was the cause of the interrupt or exception.

Interrupt Driven I/O

* Interrupt I/O is more efficient than programmed I/O because it eliminates needless waiting.

* Interrup I/O consumes a lot of processor time.



I/o Module View point:

1) I/O module receives a READ Command from the processor

2) I/o Module reads data from desired peripheral into data register

3) I/o Module Interrupts the processor

4) I/o module waits until data is requested by the processor

5) I/o Module places data on the data bus when requested.

Processor view point:

> The processor issue a READ command

> The processor checks for interrupts at the end of the instruction cycle.

> The processor saves the current content when interrupted ay the I/o module.

> The processor read the data from the I/o module and stores it in memory.

> The processor restores the saved content and resume execution.

Design Issues:

There are four common techniques available

1. Multiple Interrupt lines:

    * Each line may have multiple I/o modules.
    * processor picks line with highest priority

2. software polling:

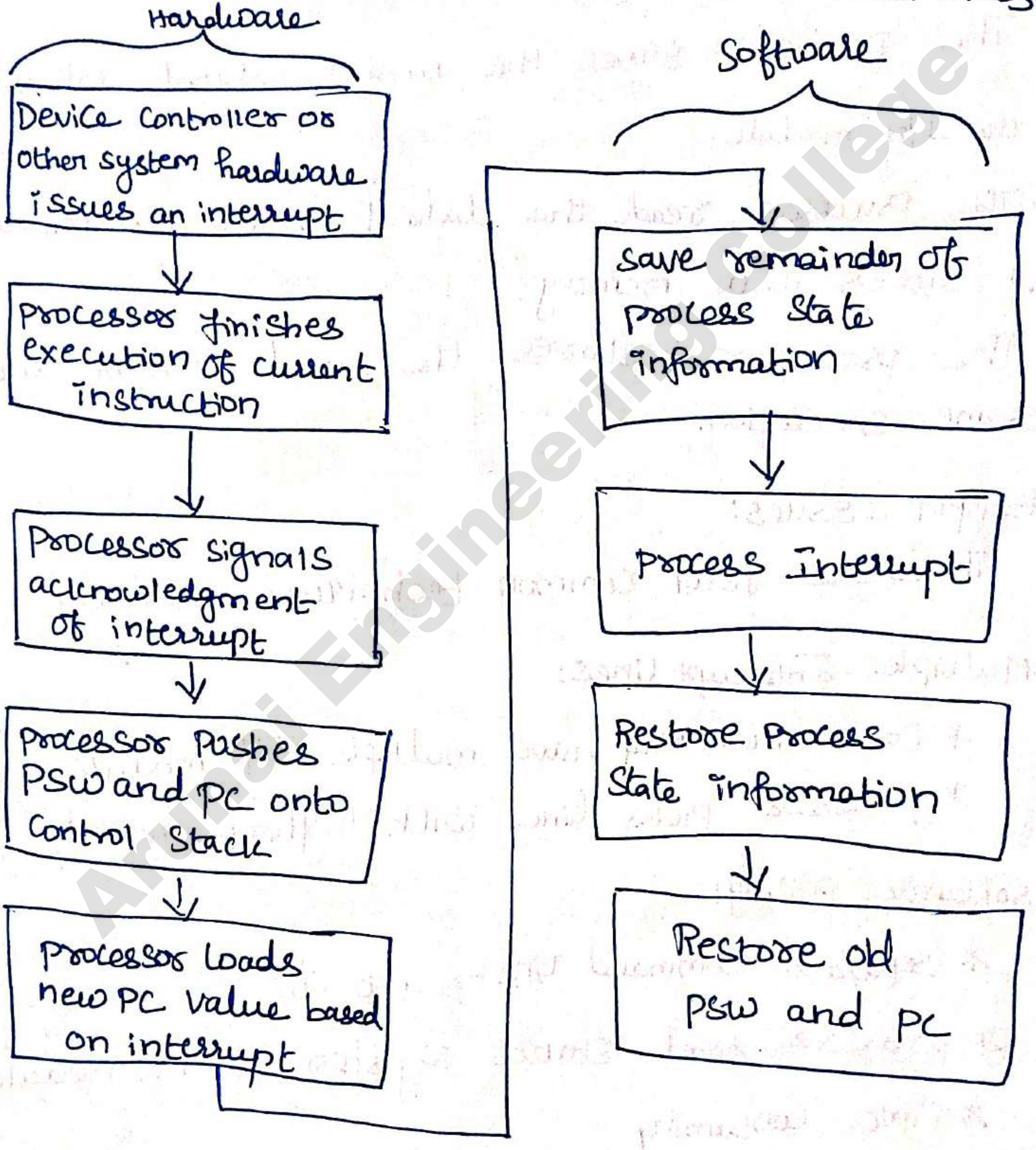
    * separate command line - Test I/o
    * processor read status register of I/o module
    * Time consuming

3. Daisy chain (Hardware poll, anchored)

    * Hardware poll
    * processor sends interrupt request
    * Requesting I/o module - Places a word of data on the data lines - "vector" that uniquely identifies the I/o module Vector Interrupt.

7. Bus arbitration:

- * I/o module first gains the Control of the bus.
- * I/o module sends interrup requests
- * The processor acknowledges the interrupt request
- * I/o module places its vector of the data lines.


Hardware
Software
Device Controller or other system hardware issues an interrupt
processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
processor pushes PSW and PC onto Control stack
processor Loads new PC value based on interrupt
save remainder of process state information
process Interrupt
Restore process state information
Restore old PSW and PC

Simple Interrupt processing

I/o System:-

* The Input/output (I/o) device is the interface between the users and the computers, plays an important role in the performance of computer system.

* Modern computers use single bus arrangement for connecting I/o devices to cpu and memory.

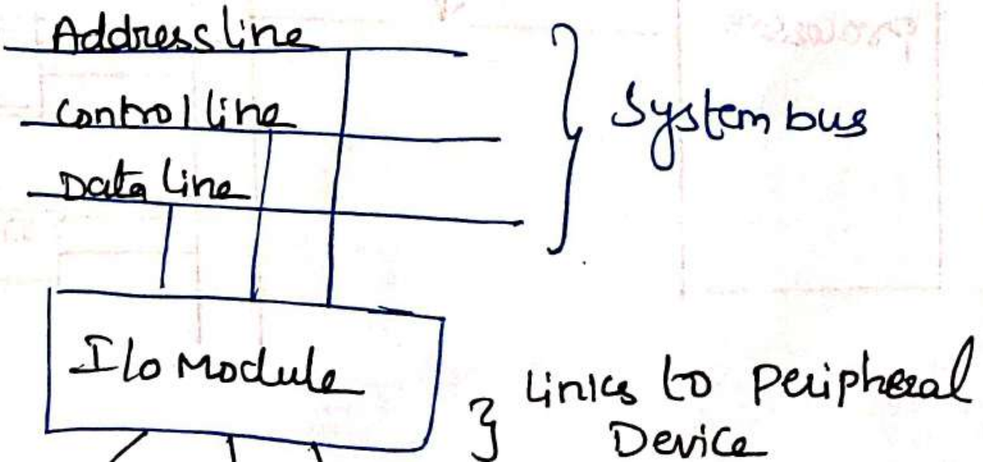
* The bus enables all the devices connected to it to exchange information.

* Bus consists of 3 set of lines.

1. Address lines- processor places a particular address on address lines.

2. Control lines- Device which recognizes this address responds to the commands issued on the control lines

3. Data lines- processor requests for either Read/write. The data will be placed on Data lines



I/o Interfacing Techniques:

* I/o devices can be Interfaced to a computer system I/O in two ways, which are called interfacing techniques.

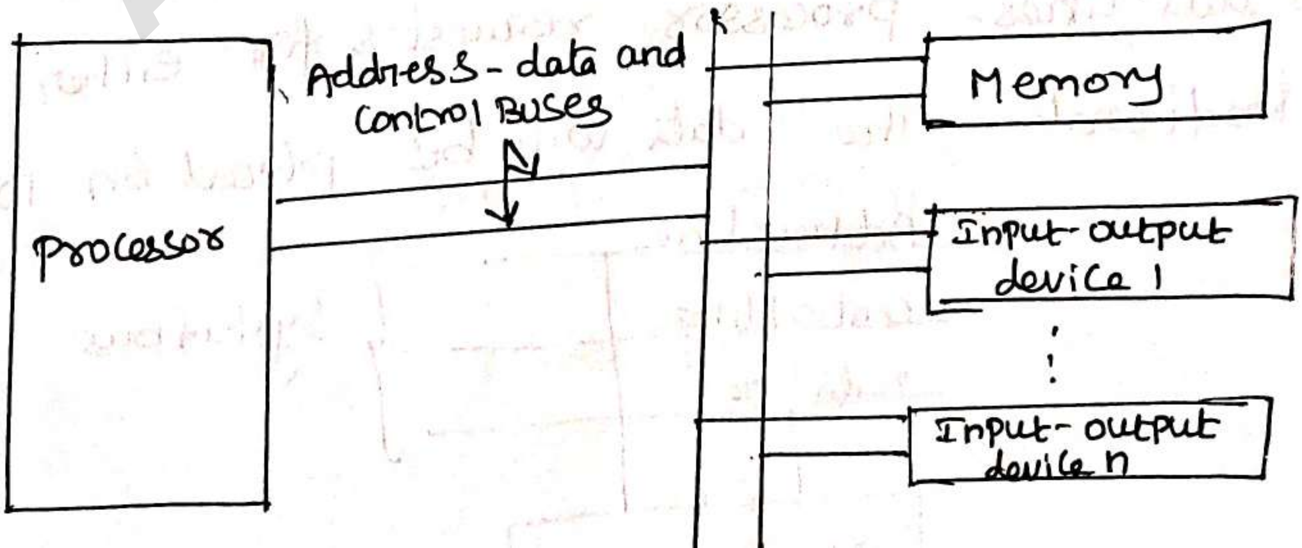
*Memory mapped I/o   * I/o mapped I/o

*Memory Mapped I/O:-

↳ Machines which are assigned a part of the main memory address space to I/o ports, this is called Memory mapped I/o.

↳ The Memory load and Store instructions are used to transfer a word of data to or from an I/O port, no special I/o instructions are needed.

Memory-mapped I/o Device organization on the common Buses and Memory and I/o Interfaces



↳ I/O mapped I/O:

* In Input/output mapped I/O technique, the memory and I/O address spaces are kept separate.

* Some processors have separate control signal, which are issued when processing the instructions for I/O operations.

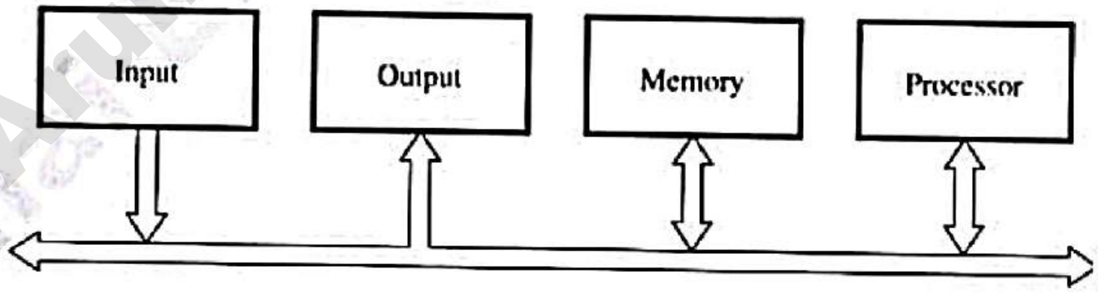
* A memory referenced instruction activation the READ or WRITE control line and does not affect the I/O devices.

Bus Structure

- A group of lines that serves as the connecting path from one device to another is called a Bus.
- A Bus may be defined as a set of communication lines/ data paths that carry the data, address or control signal among various units of a CPU.
- A memory/ system bus interconnects the processor with memory units and I/O units.
- When a word of data is transferred between units, all bits are transferred in parallel.
- The bits are transferred simultaneously over many wires, or lines, with one bit per line.
- There are 2 types of Bus structures. They are,
  - Single Bus Structure
  - Multiple Bus Structure

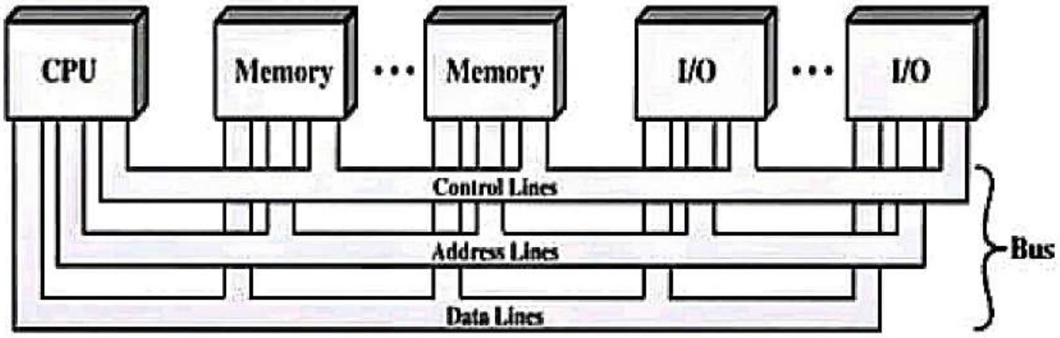
Single Bus Structure

- All the components (I/O, memory and processor) are connected to a common bus.
- It allows only one transfer at a time because only two units can actively use the bus at any given time.
- Advantage of using single bus structure is
  - Low cost
  - High flexibility in attaching peripheral devices.
- The only problem is that the performance is low due to slow data transfer.



Multiple Bus Structure

- To achieve high performance, multiple bus structures are used.
- This allows two or more transfers to be carried out at the same time, concurrently.
- This provides a better performance but at an increased cost.



- The devices connected to a bus vary widely in their speed of operation.
- Hence buffer registers are used to hold information during transfers.
- Thus, the buffer register prevents a high speed processor from being locked to a slow I/O device during data transfer.
- This allows the processor to switch rapidly from one device to another.

TECHNOLOGIES FOR BUILDING PROCESSORS AND MEMORY

- A computer architect must plan the design based on the future technology changes
- The designer must be aware of rapid changes in implementation technology.
- There are four implementation technologies to be considered. They are,
  - Integrated Circuit logic technology
  - Semiconductor DRAM
  - Magnetic disk technology
  - Network technology

Integrated circuit logic technology

- A transistor is a small electronic device made of semiconductor material that carries current and amplify.
- It was used in II generation computers and was very slow.
- Due to its low capacity, Integrated circuits(IC) were introduced.
- The IC technology emerged with the fabrication of transistors on a single chip.
- The Small Scale Integration (SSI) technology involved few transistors (< 100) on a silicon chip.
- Then emerged the Medium Scale Integration (MSI) composing hundreds of transistors on a chip.

- The III generation computers relied on SSI and MSI technology.
- The IV generation computers used Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) technology that was composed of thousands of transistors on a single chip.
- The V generation computers being dependent on the Ultra Large Scale Integration (ULSI) technology that possesses several millions of transistors on a chip.
- The transistor density increases by about 35% per year.

- The III generation computers relied on SSI and MSI technology.
- The IV generation computers used Large Scale Integration (LSI) and Very Large Scale Integration (VLSI) technology that was composed of thousands of transistors on a single chip.
- The V generation computers being dependent on the Ultra Large Scale Integration (ULSI) technology that possesses several millions of transistors on a chip.

The transistor density increases by about 35% per year.

Technology used in computers	Relative performance/unit cost
Vacuum tube	1
Transistor	35
Integrated circuit	900
Very large-scale integrated circuit	2,400,000
Ultra large-scale integrated circuit	6,200,000,000

Semiconductor DRAM (dynamic random-access memory)

- DRAM is a semiconductor memory device that stores each bit of data in a separate passive electronic component like capacitor.
- Capacity increases by about 40% per year every two years.

Magnetic disk technology

- Magnetic storage is the storage of data on a magnetized medium.
- The density increased by about 30% per year before 1990, and increased to 100% per year in 1996.
- It has again dropped back to 30% per year since 2004.
- But still, disks are still 50 – 100 times cheaper per bit than DRAM.

Network technology

- In order to provide communication from one computer to another, LAN (Local Area Network) like Ethernet, MAN (Metropolitan Area Network), WAN

Bus Arbitration

* The device that is allowed to initiate data transfers on the bus at any given time is called the busmaster.

* Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The selection of bus master is usually done on the priority basis.

Approaches to bus arbitration:

1) In Centralized bus arbitration, a single bus arbiter performs the required arbitration. The bus arbiter may be the processor or a separate controller connected to the bus.

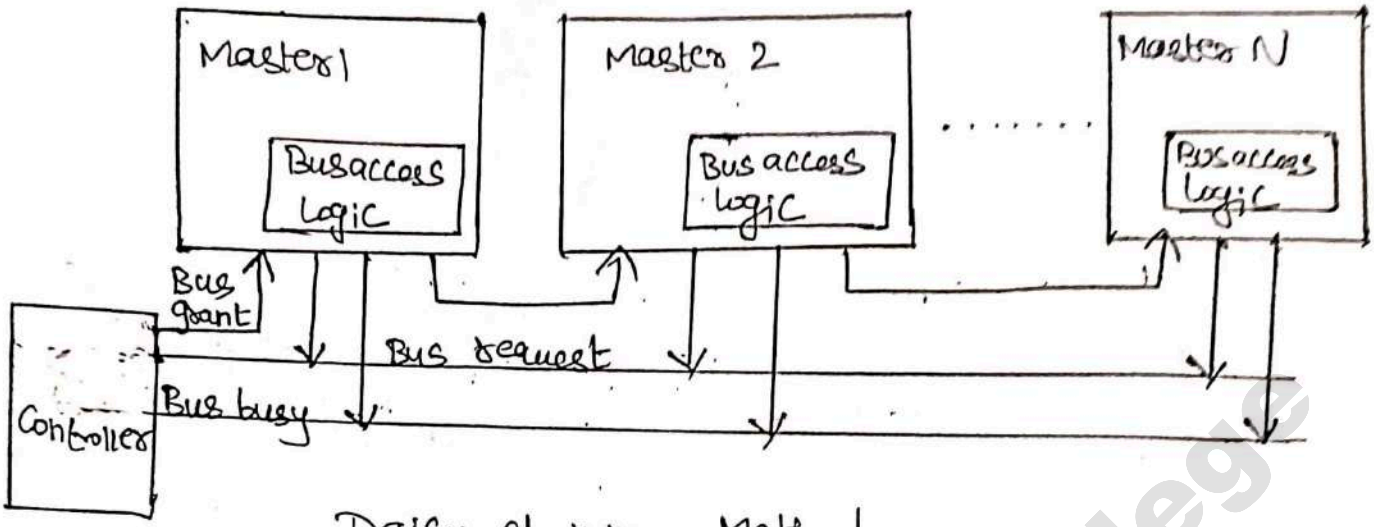
2) In Distributed bus arbitration, all devices participate in the selection of the next bus master.

1) Centralized Bus Arbitration Schemes:

There are three different arbitration schemes that use the centralized bus arbitration approach.

a) Daisy chaining b) Polling method c) Independent request

a) Daisy Chaining!



Daisy chaining Method

* It is a simple and cheaper method

* All masters make use of the same line for bus request.

* In response to a bus request the controller sends a bus grant if the bust is free.

Advantages:

1. It is a simple and cheaper method

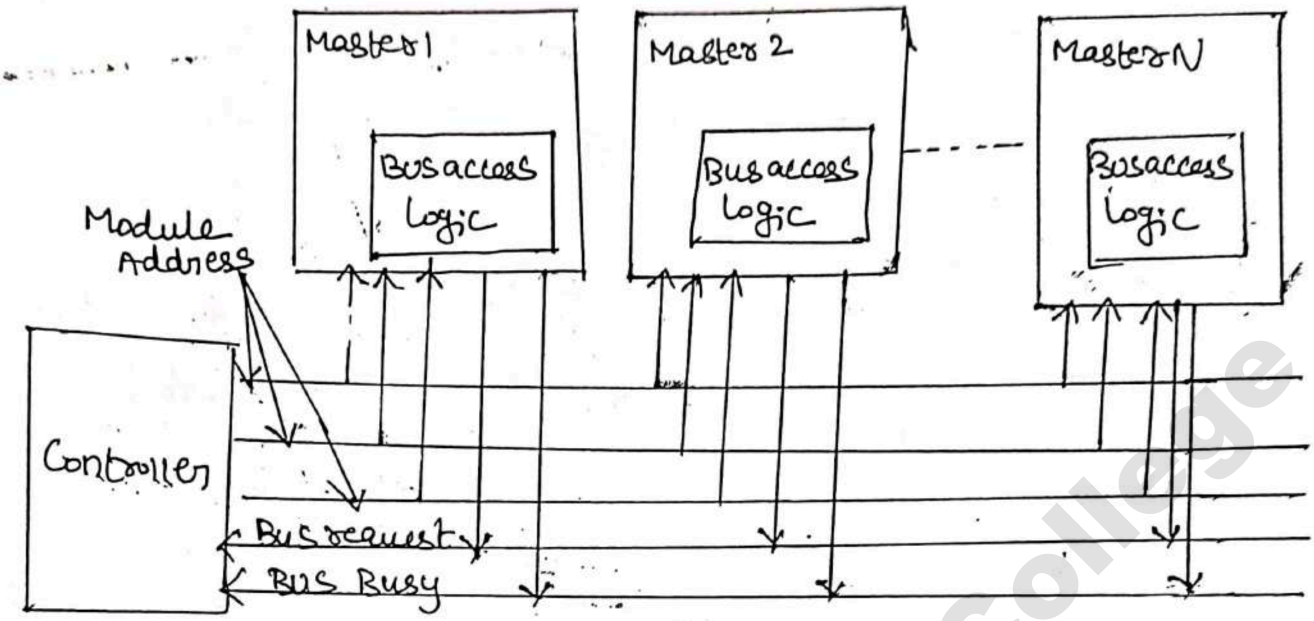
2. It requires the least number of lines and this number is independent of the number of masters in the system

Disadvantages:

1. The Priority of the master is fixed by its Physical location

2. Failure of any one master causes the whole system to fail.

b) Polling Method:



* In this the controller is used to generate the addresses for the masters.

* Number of address lines required depends on the number of masters connected in the system.

Eg 8 masters connected in the system, at least three address lines are required.

Advantages:

1. The priority can be changed by altering the polling sequence stored in the controller

2. If the one module fails entire system does not fail.

c) Independent request:

In this scheme each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.

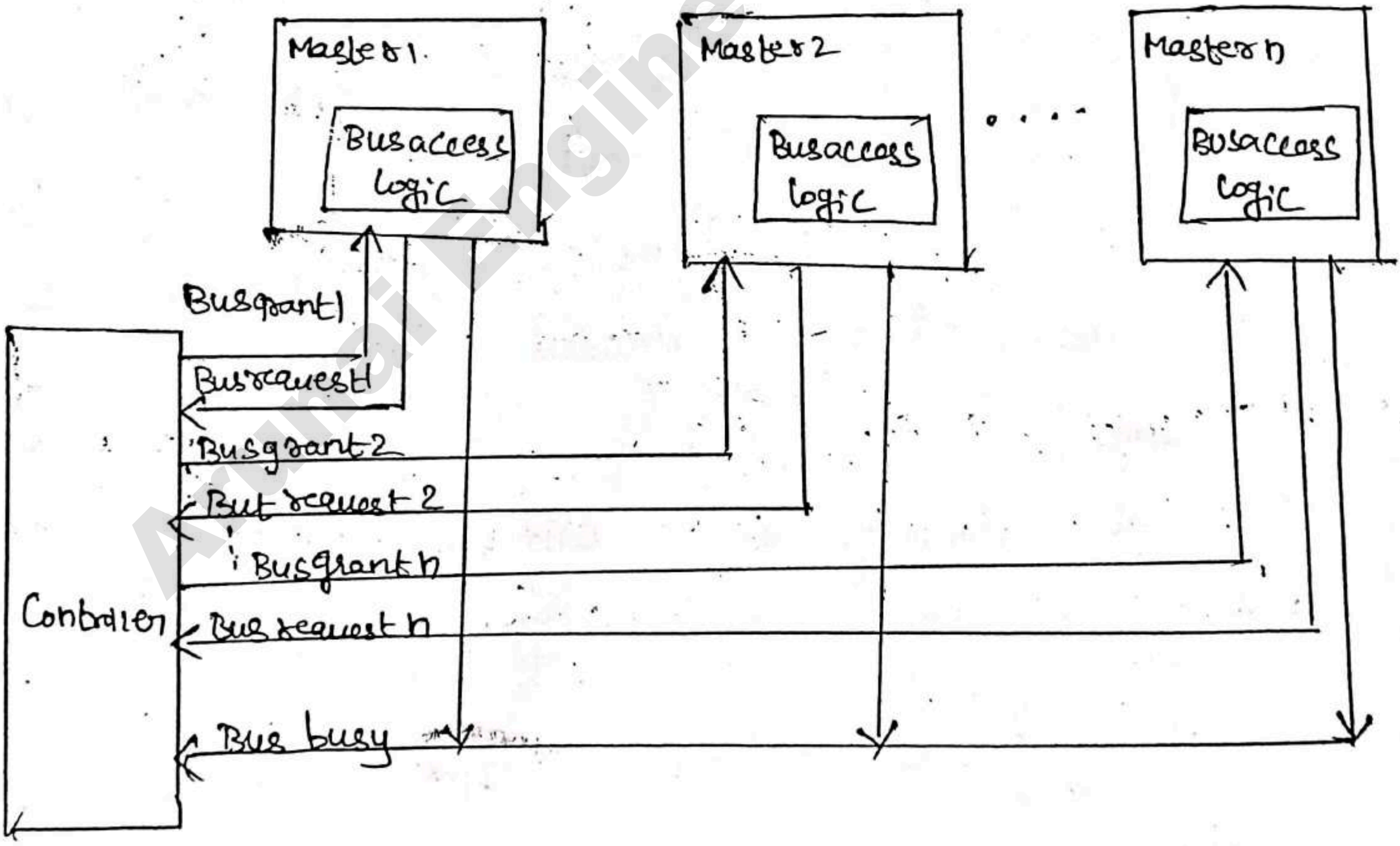
* The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

Advantage:

1. Due to separate pairs of bus request and bus grant signals, arbitration is fast and is independent of the number of masters in the system.

Disadvantage:

1. It requires more bus request and grant signals (2 x n signals for n modules.



Independent request Method

Universal Serial Bus (USB)

* A universal serial BUS (USB) is the commonly used interconnection standard.

* It enables communication between devices and a host controller such as a personal computer. It is a fast serial bus.

* It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives.

* The commercial succes of the USB is due to its simplicity and low cost.

The USB has been designed to meet the following key objectives:

↳ provide simple, low cost and easy to use interconnection system.

↳ Accommodate a wide range of I/O devices and bit rates.

↳ Enhance user friendliress through plug-and-play mode of operation.

Device characteristics:

↳ The speed, volume and timing constraints associated with data transfers to and from the devices vary significantly.

↳ USB can provide a small amount of power to the attached device through the USB cord.

↳ Devices that only need a little power can get it from the bus and do not need a separate electric power plug.

Plug-and-play:

↳ The plug-and-play feature means that when a new device is connected, the system detects its existence automatically.

↳ The USB standard defines both the USB software and hardware for communication.

↳ The software finds the type of the device connected and how to communicate with it.

↳ The USB is also called hot-pluggable, which means a device can be plugged into or remove from a USB port while power is turned on.

USB versions

Currently, five different USB standards are used.

i) USB 1.0, ii) USB1.1, iii) USB2.0,

iv) USB 3.0 v) USB 3.1

* USB 3.1 doubled the speed of 3.0

USB Transfer speeds:

USB offers five different transfer Speeds.

i) 1.5 MBit per second (called low speed)

ii) 12 MBit per second (Full speed)

iii) 480 MBit/second (Hi speed)

iv) 5 Gbit per second (super Speed),

v) 10 Gbit/second ("Super speed +")

* Hi speed is only available in USB 2.0 and later, and super speed is only available in USB 3.0

USB Connectors:

A number of different USB connectors exists, the two major types of connectors are

Type A:

The USB 2.0 standard is type A.

It has a flat rectangle interface that inserts into a hub or USB host which transmits data and supplies power.

A keyboard or mouse are common examples of a type of A USB connector.

Type B:

A type B USB Connector is square with slanted exterior corners. It is connected to an upstream port that uses a removable cable such as printer.

The type B connector also transmits data and supplies power.

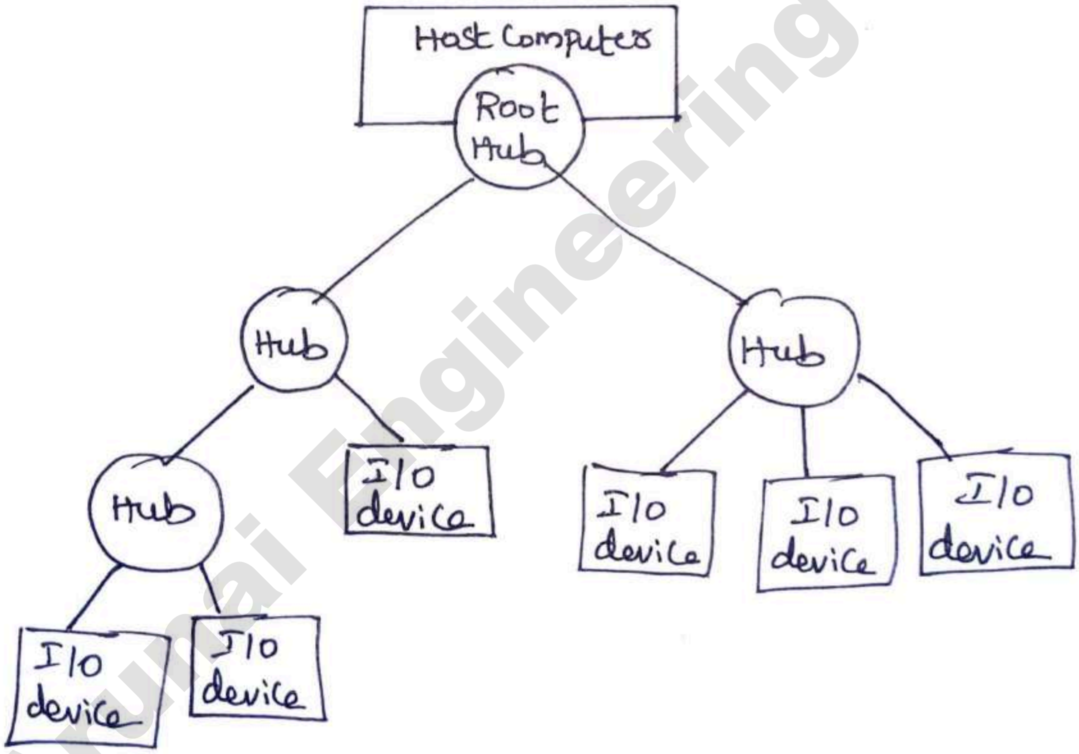
Some type B Connectors do not have a data connection and are used only as a power connection.

USB Architecture:

* The USB uses point-to-point connections and a serial transmission format.

* when multiple devices are connected, they are arranged in a tree structure.

USB Tree Structure



* Each node of the tree has a device called a hub. It acts as an intermediate transfer point between the host computer and the I/o devices.

* At the root of the tree, a root hub connects the entire tree to the host computer.

* The leaves of the tree are the I/o devices like, mouse, a keyboard, printer, camera or a speaker.

* The I/o devices are allowed to send messages at any time. To handle this, USB operates on the basis of polling.

* A device may send a message only in response to a poll message from the host processor.

* Hence no two devices can send the messages at the same time.

* Each device is assigned a 7-bit address. This address is local to the USB tree.

Isochronous Traffic on USB:

* USB supports transfer of isochronous data in a simple manner.

* Isochronous data need to be transferred at precisely timed regular intervals.

* The root hub transmits unique sequence of bits over the USB tree every millisecond.

* It is called as Start of Frame character which is used as the marker to indicate the beginning of data.

* Thus, digitized audio and video signals can be transferred in a regular and at precise time.

Electrical Characteristics:

* USB connections consist of four wires: Two wires carry power, +5V and Ground and another two wires carry data.

* The I/O devices that do not have large power requirements can be powered directly from USB.

Two methods are used to send data over USB cable.

i) Single-ended Transmission (Low-speed)

* This method is used when data is sending at low speed.

* A scheme in which a signal is injected on a wire relative to ground is called as Single-ended transmission.

* A high voltage relative to ground is transmitted on one of the two data wires to represent a 0 and on the other to represent a 1.

* The ground wire carries the return current in both the cases.

* This scheme is vulnerable to noise.

ii) Differential signaling.

* High-speed USB uses Differential signaling method.

* The data signal is injected between two data wires twisted together.

* Here, the ground wire is not involved.

* The receiver senses the voltage difference between the two signal wires directly without reference to the ground.

* Here, the noise component is called out.