CS8491 COMPUTER ARCHITECTURE

L	T	P	C
3	0	0	3

OBJECTIVES:

- To learn the basic structure and operations of a computer.
- To learn the arithmetic and logic unit and implementation of fixed-point and floating point arithmetic unit.
- To learn the basics of pipelined execution.
- To understand parallelism and multi-core processors.
- To understand the memory hierarchies, cache memories and virtual memories.
- To learn the different ways of communication with I/O devices.

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

OUTCOMES:

On Completion of the course, the students should be able to:

- Understand the basics structure of computers, operations and instructions.
- Design arithmetic and logic unit.
- Understand pipelined execution and design control unit.
- Understand parallel processing architectures.
- Understand the various memory systems and I/O communication.

TEXT BOOKS:

1. David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Fifth Edition, Morgan Kaufmann / Elsevier, 2014.
2. Carl Hamacher, Zvonko Vranesic, Safwat Zaky and Naraig Manjikian, Computer Organization and Embedded Systems, Sixth Edition, Tata McGraw Hill, 2012.

REFERENCES:

1. William Stallings, Computer Organization and Architecture – Designing for Performance, Eighth Edition, Pearson Education, 2010.
2. John P. Hayes, Computer Architecture and Organization, Third Edition, Tata McGraw Hill, 2012.
3. John L. Hennessey and David A. Patterson, Computer Architecture – A Quantitative Approachll, Morgan Kaufmann / Elsevier Publishers, Fifth Edition, 2012.

UNIT - I

Part - A

TWO MARKS WITH ANSWERS

1. List the major components of a computer system. [APR/MAY-
What are components of a computer system? [NOV/DEC-17]

* The major components of a computer systems.

1. Computer Hardware
   a) Input unit
   b) output unit
   c) processing unit (CPU)
      i) ALU ii) control unit
      iii) Registers
         * Data Registers
         * Address Registers
         * Status Registers
         * General purpose Registers
         * Program counter.
   d) Memory unit
      * Main memory
         ↳ RAM
         ↳ ROM
      * Secondary Memory

2. Computer Software
   a) System software
   b) Application software
      * General purpose Software
      * Special purpose Software

2. What are the addressing modes? [NOV/DEC-17]

* Addressing modes refers to the way in which the location of an operand is specified in an instruction is referred to as addressing modes.

* Information contained in the instruction code is the value of the operand or the address of the result/operand.

Different types of addressing modes are

1. Immediate Addressing mode
2. Register Addressing Mode
3. Register Indirect Addressing Mode
4. Direct Addressing Mode
5. Indirect Addressing Mode
6. Implied Addressing Mode
7. Relative Addressing Mode
8. Indexed Addressing Mode
9. Base Register Addressing Mode
10. Autoincrement or Autodecrement Addressing Mode.

3. State the need for indirect addressing mode. Give an Example. [APR/MAY-17]

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

* The register or memory location that contains the address of the operand is a pointer.

* when an execution take place in such mode, instruction may be told to go to a specific address.

* once it's there, instead of finding an operand, it find's an address where the operand is located.

Eg: J $S1 // S1=4008(Address)

4. what is an instruction register? [NOV/DEC-16]

* An Instruction register(IR) is the part of a CPU control unit that holds the instruction currently being executed or decoded.

* Decoding the OP-code in the instruction register includes determining the instruction, determining where its operands are in memory, retrieving the operands from memory, allocating processor resources to execute the command etc.

5. Give the formula for CPU execution time for a program. [NOV/DEC-2016]

* CPU execution time (CPU time) - Time the CPU spends working on a task.

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

6. How to represent Instruction in a computer system? [MAY/June - 2016]

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

* Since registers are referred in instructions, there must be a convention to map register names into numbers.

* Three types of instruction formats are used in MIPS They are

i) R-Type or R-Format (Register)

ii) I-Type or I-Format (Immediate)

iii) J-Type (Addressing in Branches and Jumps)

7. Distinguish between auto increment and auto decrement addressing mode. [MAY/JUNE-2016]

Auto Increment Addressing mode	Auto Decrement Addressing mode
1. The Effective address of the operand is the contents of a register in the instruction.	1. The Effective address of the operand is the content of a registers in the instruction.
2. After accessing the operand the contents of this register is automatically incremented	2. After accessing the operand the contents of this registers is automatically decremented.
3. Syntax: (Ri)+	3. Syntax, -(Ri)
4. Eg: EA = (R1)+	4. Eg: EA = -(R1)

8. What is Instruction set Architecture? [NOV/DEC-2015]

* The Instruction set Architecture (ISA) or simply architecture of a computer is the interface between the hardware and the lowest-level software.

* It includes anything Programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on.

9. How CPU execution time for a program is calculated? [NOV/DEC-2015]

* CPU execution time or simply CPU time, which recognizes this distinction, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs.

* CPU time can be further divided into

i) User CPU time - CPU time spent in the program
ii) System CPU time - CPU time spent in the operating system performing tasks on behalf of the program.

10. List the eight great ideas invented by Computer Architects. [APR/MAY-2015]

i) Design for Moore's Law
ii) Use Abstraction to Simplify Design
iii) Make the Common Case fast
iv) Performance via Parallelism
v) Performance via Pipelining
vi) Performance via Prediction
vii) Hierarchy of Memories
viii) Dependability via Redundancy.

11. Distinguish Pipelining from Parallelism.

[APR/MAY-2015]

Pipelining	Parallelism
i) Pipelining is a technique in which multiple instructions are overlapped in execution.	i) Performance can be improved by doing operations in parallel
ii) It Exploits Parallelism among the instructions in a sequential instruction stream.	ii) Parallel execution is easier when tasks are independent, but often they need to co-operate with each others.

12. State Amdahl's Law. [NOV/DEC-2014]

* Amdahl's Law is used to find the execution time of a program after making the improvement.
* It can be represented in a equation as follows:

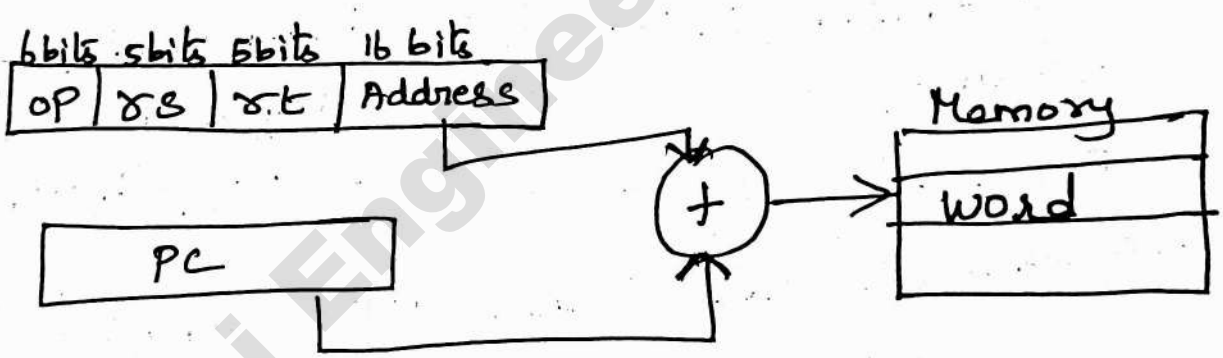
Execution time after improvement

$$\text{Speed UP} = \frac{\text{Execution time for Entire task Using Improved Machine}}{\text{Execution time for Entire task Using old original Machine}}$$

3. Brief about Relative addressing mode with an Example. [NOV/DEC-2014]

* The PC-relative addressing mode can be used to load a register with a value stored in program memory a short distance away from the current instruction.

* It can be seen as a special case of the "base + offset" addressing mode, one that selects the program counter (PC) as the "base register"



PC-relative addressing

Example:

```
bne $s0, $s1, Exit
```

(ie) #goto Exit if $s0 ≠ $s1

14) Classify the instructions based on the operation they perform and give one example to each category. [Apr/May-18] [Nov/Dec-18]

Depending on operation they perform, all instructions are divided into several groups

i) Arithmetic Instructions

Eg: ADD r1, r2, r3 (ie) $r_1 = r_2 + r_3$

ii) Data transfer Instructions

Eg: mov r1, r2 (ie) $r_1 = r_2$

iii) Logical Instructions

Eg: AND r1, r2, r3 (ie) $r_1 = r_2 \& r_3$

iv) Conditional Branch Instructions

Eg1: CMP r1, r2 (ie) Cond. flag = r1 - r2

v) Eg2: beq $s1, $s2, 25

(ie) If ($s1 = $s2) go to PC + $ + 100

v) Unconditional Branch Instructions

Eg: jr. $ra (ie) go to $ra

15 Write the equation for the dynamic power required per transistor. [Apr/May -18]

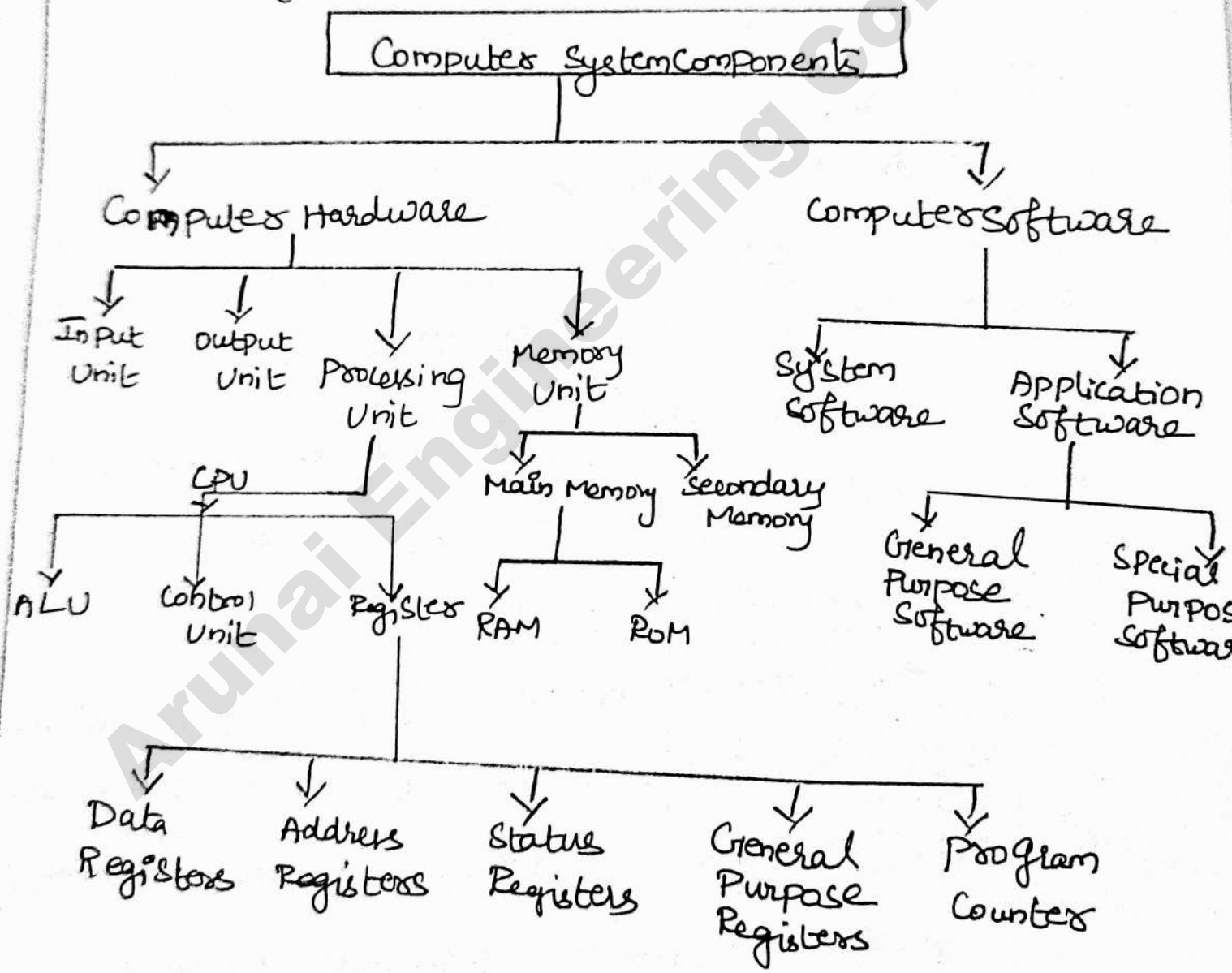
The power required per transistor is just the product of energy of a transistion and the frequency of transitions:

$$\text{Power} \propto \tfrac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

PART-B

1. Explain in detail the various components of computer system with neat diagram. [NOV/DEC-2016] [NOV/DEC-2014] [NOV/DEC-2015]

* Computer System contains many kinds of components. As a computer architect must know the components and functions of it used in Computer System



Computer Hardware

* The term hardware refers to the physical components of computer such as system unit, mouse, keyboard, monitor, printer, etc.

* Computer hardware can be classified into many units such as follows

1. Input unit 2. output unit
3. processing unit 4. Memory unit

1. Input Devices

* Input devices are used to give information to the computer. Different kinds of input devices can perform different kinds of tasks such as follows.

Mouse, keyboard, Tracker ball, Scanners, Touch pads, Light Pens, Joy sticks.

2. output Devices

* output devices are used to get the results of task performed by the computer. Different kinds of output devices used for different purpose.

* VDU or Monitor, printers, Plotters, Speakers, Speech Synthesizers

Processing Unit

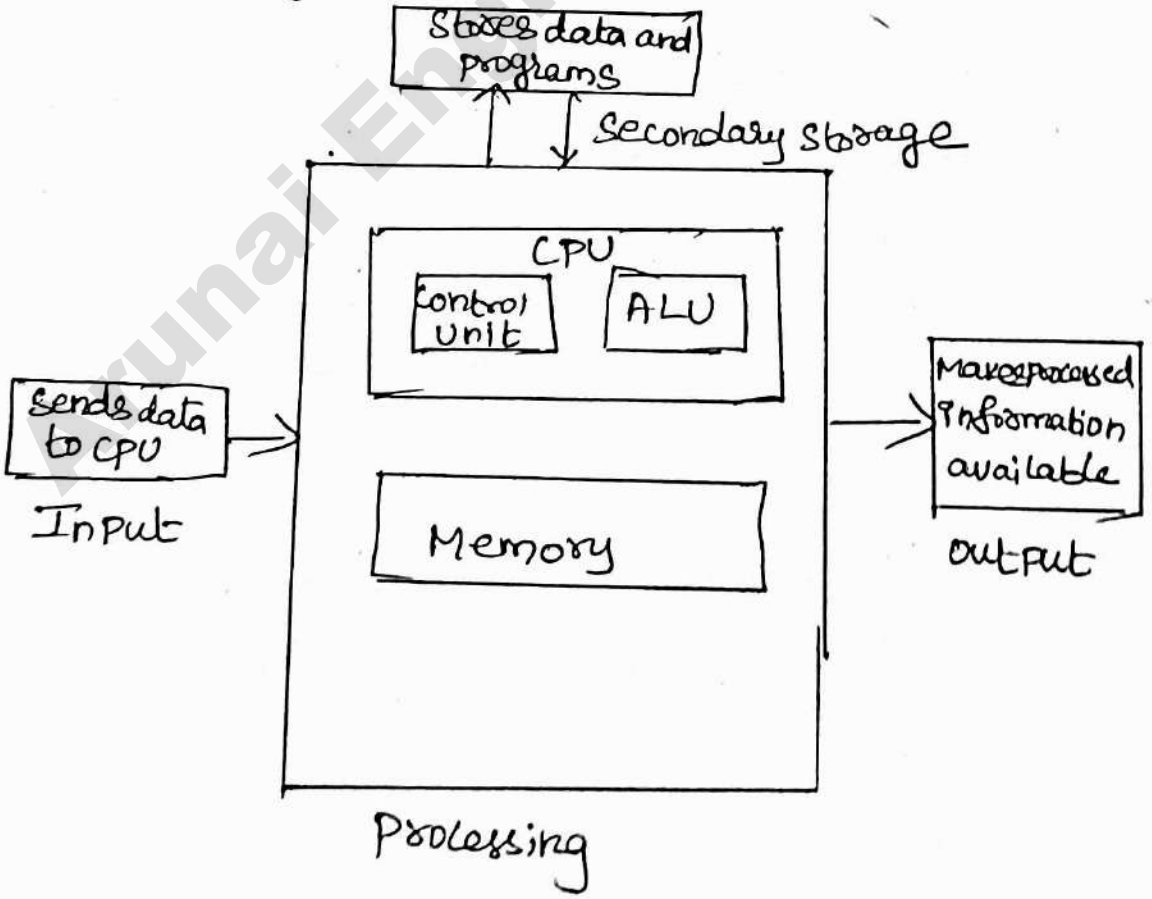
* Processing Unit is the brain of any computer System. It contains the following functional units

1. Central Processing Unit (CPU)
2. Arithmetic and Logical Unit (ALU)
3. Control Unit
4. Registers

Central Processing Unit (CPU)

* CPU is a complex set of electronic circuit which acts as a control Central for the System.

* Set of electronic circuit used to execute Stored program instructions.



CPU contains two major parts

1. Control Unit (CU)
2. Arithmetic and Logic Unit (ALU)

1. Control Unit (CU)

* Control Unit is a part of the hardware
* It directs the computer system to execute stored Program instructions and tells the ALU what operation
* Control Unit must communicate with memory and ALU
* It sends data and instructions from secondary storage to memory as needed.

2. Arithmetic and Logical Unit (ALU)

* ALU is a digital circuit used to perform arithmetic and logic operation
* ALU performs basic arithmetic and logic operations

Eg Arithmetic operations are addition, subtraction, multiplication and division.

Logic operations are comparisons of values such as NOT, AND and OR.

Registers:

* In a computer, a register is one of a small set of data holding places that are part of a computer processor.
* A register may hold a computer instruction a storage address or any kind of data.

Data registers

* It is the register of a computer's control unit that contains the data to be stored in the computer storage.

Address registers

* It is a CPU register that either stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.

Status registers

* The status register is a hardware register which contains information about the state of the processor

Program Counter

* A Program Counter is a register in a computer processor that contains the address of the instruction being executed at the current time.

General purpose registers

* General purpose registers are not used for storing any specific type of information.

* Instead operands as well as address are stored at the time of program execution.

Memory unit

The function of the memory unit is to store programs and data. There are two classes of memory.

1. Main memory (or) primary memory
2. Secondary memory

1. Main memory

* Main memory is a fast memory that operates at electronic speeds. (Expensive)

* programs must be stored in the memory while they are being executed.

* The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information

* Main Memory can be classified in to two types

1. RAM - Random Access Memory (Volatile (ie) Information is lost when you switch off PC)
2. ROM - Read only Memory (Non-volatile eg Network cards and video cards)

2. Secondary Memory

* It is used when large amount of data and many Programs have to be stored.

* It can be classifications, 1. Harddisk 2. Floppy disk 3. CD-ROM or optical disks 4. Magnetic disks 5. Tape drive 6. Flash memory

Computer Software

Software is a set of programs, which is designed to perform a well defined function. A program is a sequence of instruction written to solve a particular problem.

Types of software
1. System software
2. Application software

System software:

* System software is collection of programs designed to operate, control and extend the processing capabilities of the computer itself.

* System software is especially designed to control different operations of computer system.

Eg operating systems, Utility programs, Device drivers

Types of System software
1. System Control program
2. System Support program
3. System development programs

Application Software

* Application Software is the software that is designed to satisfy a particular need of a particular environment.

* These software are especially designed to solve the specific problems of users and it also known as software package.

Eg student record software,
Income tax software,
Financial packages

Types of application software

1. General purpose application software
2. Special purpose application software.

What is an addressing mode? Explain the various addressing modes with suitable examples. [NOV/DEC-2015]

Explain the Diffaent types of Addressing modes with suitable Example [NOV/DEC-2016]

What is the need for addressing in a computer system? Explain different addressing modes with Examples [May-2015] [APR/May-17]

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.

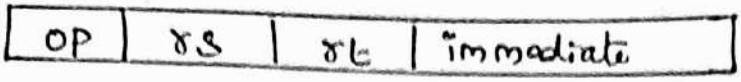
* An address computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction is known as Effective Address (EA).

* 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



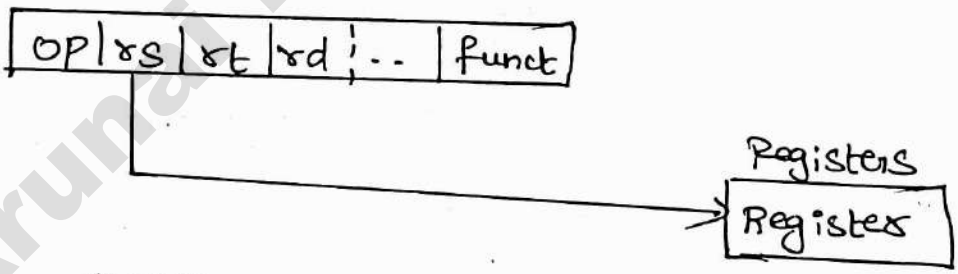
* The operand is given explicitly in the instruction.

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.

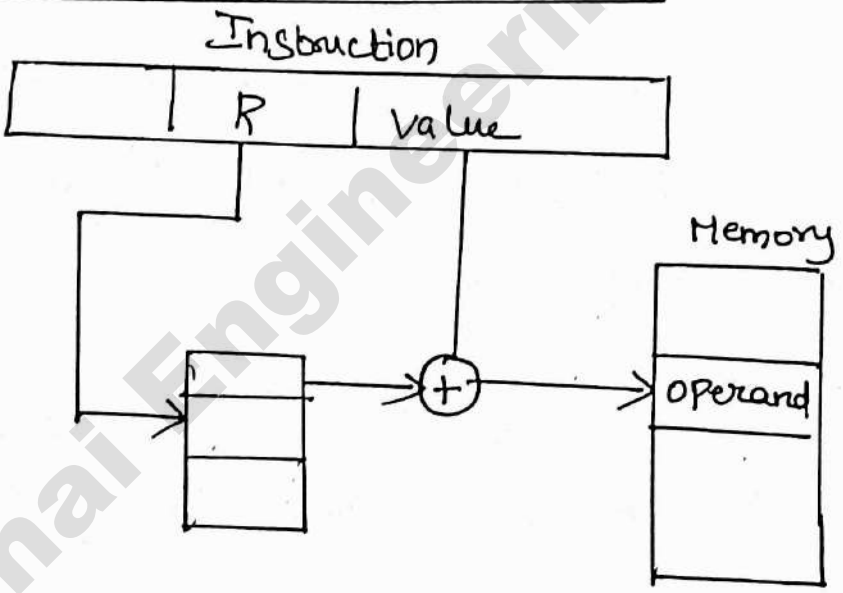
* In immediate instructions the size will be the problem for performing a load and store operations.

* 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 mode 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

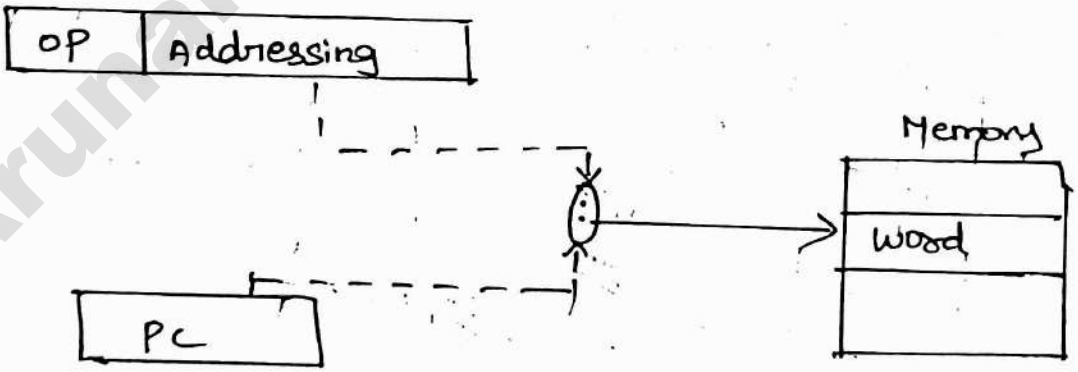
* The indexing is a technique that allows Programmer to point or refer the data (operand) stored in sequential memory locations one by one. It is an efficient mechanism for performing iterative operations.

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 p



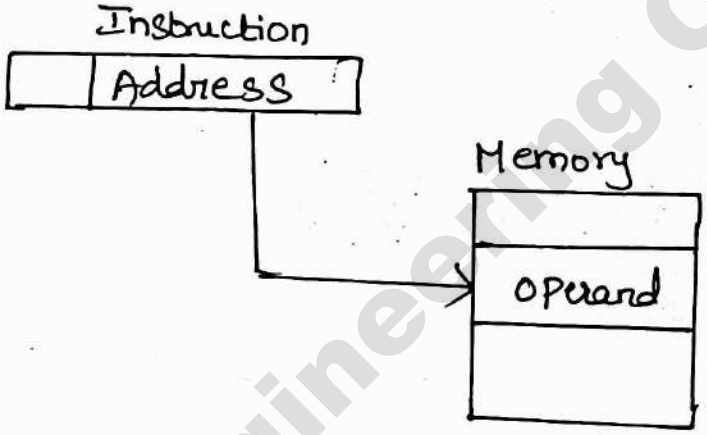
Other Addressing Modes:

6. Absolute or Direct addressing mode:

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

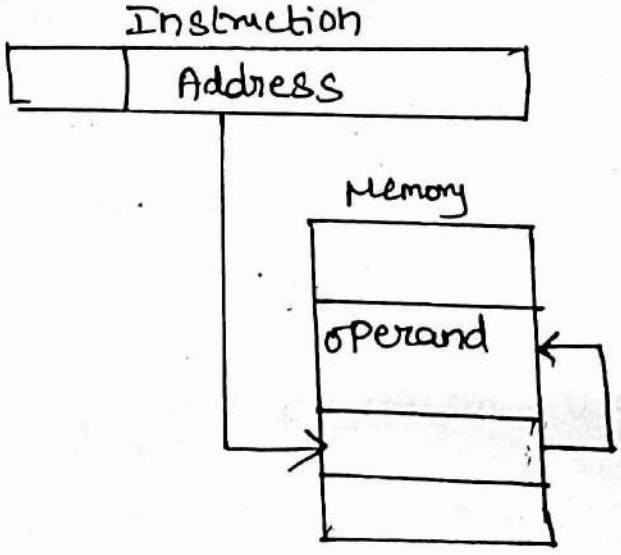
Eg MOV 2000, A

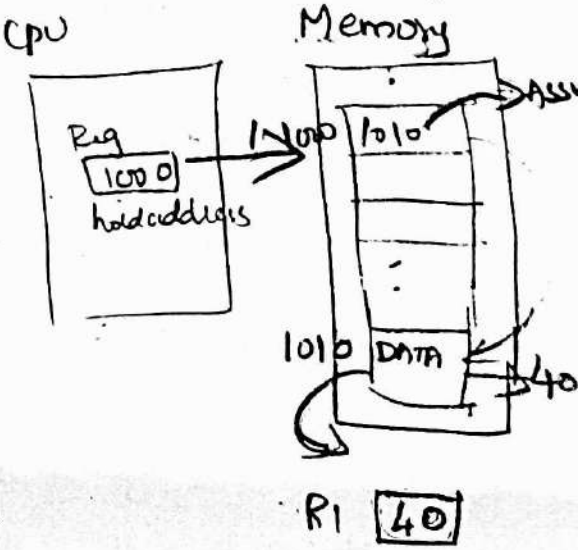
This instruction copies the contents of memory location 2000 into the A register.



7. Indirect Addressing mode:

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



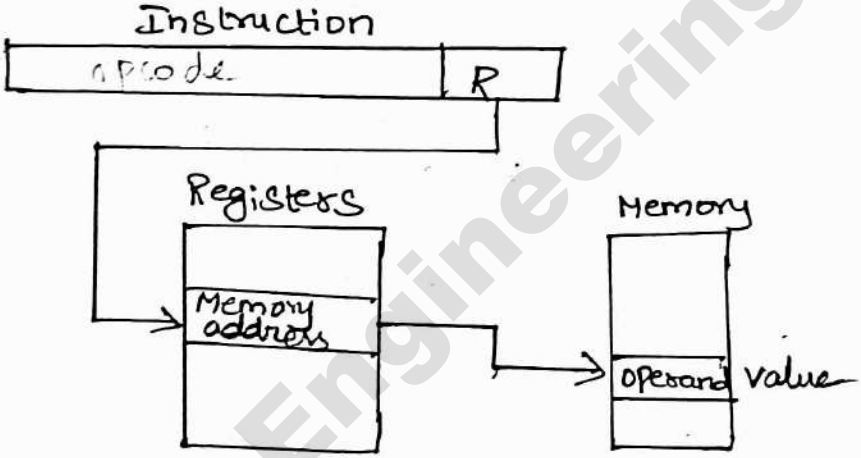


8. Register Indirect addressing mode:

* 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 location.

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.

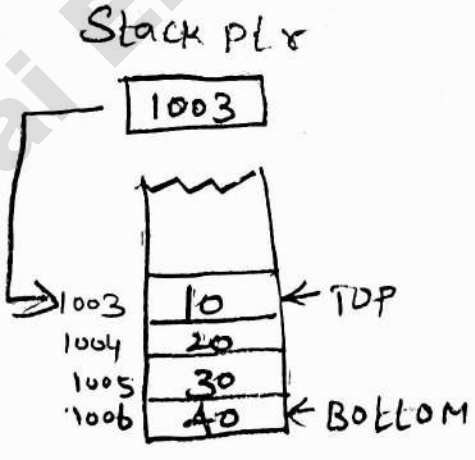
* Thus, the address of the operand (source or destination) is the contents of stack pointer.

* 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.



State the CPU performance equation and discuss the factors that affect performance

Explain the important measures of the performance of a computer and derive the basic performance equation [NOV/DEC-14] [APR/MAY-17]

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 than 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

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

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

1) 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 computers 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

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

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

CPU time = Instruction Count X CPI X Clock cycle time

* Clock rate is inverse of clock cycle time

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

Discuss about various techniques to represent instructions in a computer system [APR/MAY- 2015]
Explain various instruction formats and illustrate the same with an example [NOV/DEC-17]

Representing Instructions in a Computer System:

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

* At the lowest 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 $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 $t0, $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

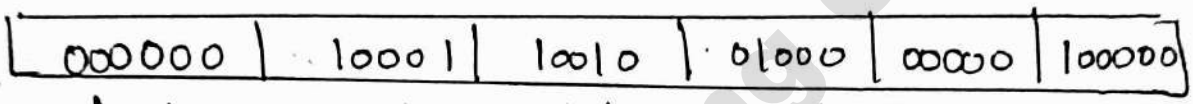
* Second field gives the number of the register that is the first source operand of the addition operation (17 $s1)

* 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



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

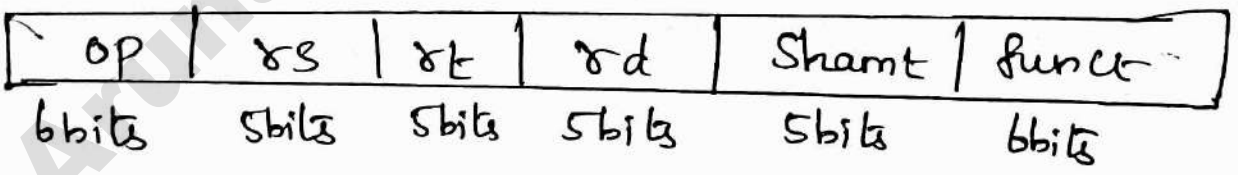
Hexadecimal	Binary	Hexadecimal	Binary
0hex	0000two	8hex	1000two
1hex	0001two	9hex	1001two
2hex	0010two	Ahex	1010two
3hex	0011two	Bhex	1011two
4hex	0100two	Chex	1100two
5hex	0101two	Dhex	1101two
6hex	0110two	Ehex	1110two
7hex	0111two	Fhex	1111two

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- basic operation of the instruction called as opcode

Opcode denotes the operation and format of an instruction

rs- the first register source operand

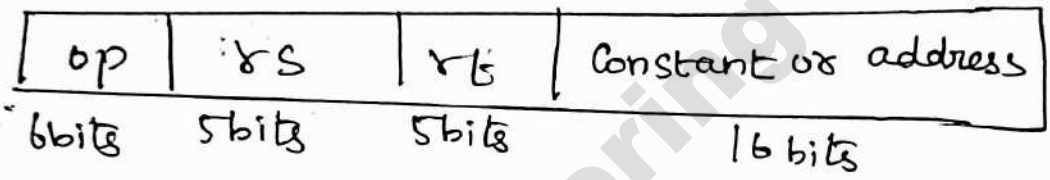
rt- the second register, source operand

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



* Used by the immediate and data transfer Instructions

Drawback of different Format

* Multiple formats complicate the hardware
* It increase the complexity

5. 1) Assume a two address format specified as source, destination Examine the following sequence of instructions and explain the addressing modes used and the operation done in every instruction. [NOV/DEC-14]

1) MOVE (R5)+, R0
2) Add (R5)+, R0
3) Move R0, (R5)
4) Move 16(R5), R3
5) Add #40, R5

Solution:

1) Move (R5)+, R0

* This instruction copies the contents of memory location whose address is specified by register R5 to register R0.

* After copying operation, then increment R5 to the next address.

2) Add (R5)+, R0

* The instruction adds the contents of memory location whose address is specified by register R5 in the content of register R0 and the

result is stored into R0, then increment the R5 address value.

3) Move (R0), (R5)

Move the value from the address R0 to the address pointed to by R5

4) Move 16(R5), R_3

* In index mode, the operand specifies both the register and an offset value x.

* The value X can be either a number, a symbol or an expression. Adding x to the value in the register gives an effective address.

* This mode can be used randomly to access elements in an array, where the register contains a base address that marks the start of the array and X is an offset from this base address.

So, " Move the value from the address pointed to by at 16 bits offset of R5 register to R3".

V) Add #40, R5

* It is immediate addressing. This instruction will store value 40 in register R5.

5) i) Consider the computer with three instruction classes and CPI measurements are given below and instruction counts for each instruction class for the same program from two different compilers are given. Assume that the computer's clock rate is 4GHz. Which code sequence will execute faster according to execution time? [Nov/Dec-14]

Code from	CPI for this Instruction class		
	A	B	C
CPI	1	2	3

Code from	Instruction Count for each class		
	A	B	C
Compiler 1	2	1	2
Compiler 2	4	1	1

Solution:

i) Instructions:

* Instruction executed by compiler 1 = 2+1+2
= 5 Instructions

* Instruction executed by compiler 2 = 4+1+1
= 6 Instructions

(ie) compiler 2 executes more number of Instructions.

ii) CPU clock cycles:

The total number of clock cycles for each sequence can be found using the following equation.

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

CPU clock cycles = Instructions executed x corresponding CPI

CPU clock cycles for compiler 1 = (2x1) + (1x2) + (2x3)
= 2 + 2 + 6
= 10 cycles

CPU clock cycles for compiler 2 = (4x1) + (1x2) + (1x3)
= 4 + 2 + 3
= 9 cycles

* It is clear that compiler 2 is faster than Compiler 1 even though it executes one extra instruction.

iii) Time for executing the instruction:

$$\text{CPU execution time} = \frac{\text{CPU Clock cycles}}{\text{Clock rate}}$$

Given:

Clock rate = 4 GHz = 4×10^9 Hz

$$\text{CPU execution time for Compiler 1} = \frac{\text{CPU Clock cycles for compiler 1}}{\text{Clock rate}}$$

$$= \frac{10}{4 \times 10^9}$$

$$= 2.5 \times 10^{-9} \text{ second}$$

$$\text{CPU execution time for Compiler 2} = \frac{\text{CPU Clock cycles for compiler 2}}{\text{Clock rate}}$$

$$= \frac{9}{4 \times 10^9}$$

$$= 2.25 \times 10^{-9} \text{ second}$$

(ie) Execution time for compiler 1 is greater than the execution time for compiler 2

So, compiler 2 will execute faster than the compiler 1

6. Explain with an example about the operations and operands of the computer Hardware? [NOV/DEC-17]

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 operations.

* The MIPS assembly language notation for performing addition operation is

```
add a, b, c
```

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

* It instructs a computer to add the two variables b and 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 no less

* 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.

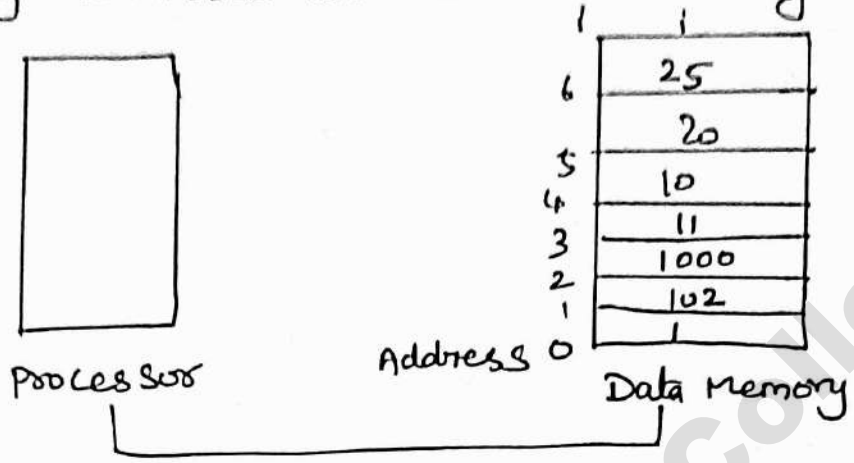
* TO access a word in memory, the instruction must supply the memory address.

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 register 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 variables 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.

7 Consider three processors P_1, P_2 and P_3 executing the same instruction set. They have clock rates of 3GHz, 2.5 GHz and 4.0 GHz respectively and CPI of 1.5, 1.0 and 2.2 respectively. i) which processor has the highest performance expressed in instructions per second?

[APR/MAY-18] [NOV/DEC-12]

Ans:.

Performance of P_1 (Instructions/sec) = 3GHz

(ie) 3×10^9

Performance of P_2 (Instructions/sec) = 2.5GHz

(ie) 2.5×10^9

performance of P_3 (Instructions/sec) = 2.2 GHz

(ie) 2.2×10^9

CPU execution time can be calculated using the following formula.

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

Let the Instruction count be 'n' and Assume it is same for all the three processors. Now calculate the CPU time for each processor.

$$\text{CPU time } P_1 = \frac{n \times 1.5}{3 \times 10^9}$$

$$= n \times 0.5 \times 10^{-9} \text{ sec}$$

$$\text{CPU time } P_2 = \frac{n \times 1.0}{2.5 \times 10^9} = n \times 0.4 \times 10^{-9} \text{ sec}$$

$$\text{CPU time } P_3 = \frac{n \times 2.2}{4 \times 10^9} = n \times 0.55 \times 10^{-9} \text{ sec}$$

CPU time for processor P_2 is smallest than other processor. Hence, P_2 produces highest performance than other processors.

ii) If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions in each processor.

Number of clock cycles

CPU time for P_1, P_2 and P_3 = 10 sec

The formula for finding CPU time can be rewritten to find the number of clock cycles.

$$\text{CPU time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

CPU Clock cycles = CPU time × Clock rate

CPU Clock cycles P_1 = 10 sec × 3×10^9 Hz = 30×10^9 cycles

CPU Clock cycles P_2 = 10sec × 2.5×10^9 Hz = 25×10^9 cycles

CPU Clock cycles P_3 = 10sec × 4×10^9 Hz = 40×10^9 cycles

8 Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. what is the MIPS assembly code corresponding to this C segment?

[Apr|May-18]

```
while (save[i] == k)
i+=1;
```

Ans:-

* The first step is to load save[i] into a temporary register. Before we can load save[i] into a temporary register, we need to have its address.

* Before we can add i to the base of array save to form the address, we must multiply the index i by 4 due to the byte addressing problem.

* We can use shift left logical, since shifting left by 2 bits multiplies by 2^2 or 4.

* We need to add the label Loop to it so that we can branch back to that instruction at the end of the loop.

Registers allocation:

↳ i $S3

↳ k $S5

↳ base of save[] $S6

```
Loop: sll $t1, $s3, 2      #temp reg $t1 = i * 4
```

To get the address of save[i], we need to add $t1 and the base of save in $s6:

```
add $t1, $t1, $s6          # $t1 = address of save[i]
```

Now we can use that address to load save[i] into a temporary register:

```
lw $t0, 0($t1)             # Temp reg $t0 = save[i]
```

The next instruction performs the loop test, exiting if save[i] ≠ k:

```
bne $t0, $s5, Exit         #go to Exit if save[i] ≠ k
```

The next instruction adds 1 to i:

```
addi $s3, $s3, 1           # i = i+1
```

The end of the loop branches back to the while test at the top of the loop.

We just add the Exit label after it

```
j Loop                     # go to Loop
Exit:
```

9 Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (Class A, B, C and D). P_1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3 and 3 respectively and P_2 with a clock rate of 3 GHz and CPIs of 2, 2, 2 and 2 respectively. Given a program with a dynamic instruction count of 1.0×10^6 instructions divided into classes as follows: 10%. Class A, 20%. Class B, 50%. Class C and 20%. Class D, which implementation is faster? what is the global CPI for each implementation? Find the clock cycles required in both cases.

Answer:

Class	A	B	C	D	
P_1	1	2	3	3	2.5 GHz
P_2	2	2	2	2	3 GHz
IC	10%.	20%.	50%.	20%.	

P1:

CPU execution time can be calculated using the following formula.

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

Class A: 1×10^5 instruction

Class B: 2×10^5 instruction

Class C: 5×10^5 instruction

Class D: 2×10^5 instruction.

$$\text{Total time}_{P_1} = \frac{(1\times10^5\times1 + 2\times10^5\times2 + 5\times10^5\times3 + 2\times10^5\times3)}{2.5\times10^9}$$

(or) $$CPU_A = \frac{(1.0\times10^5)\times1}{2.5\times10^9} = 4\times10^{-5}s$$

$$CPU_B = \frac{(2\times10^5)\times2}{2.5\times10^9} = 1.6\times10^{-4}s$$

$$CPU_C = \frac{(5\times10^5)\times3}{2.5\times10^9} = 6.0\times10^{-4}s$$

$$CPU_D = \frac{(2\times10^5)\times3}{2.5\times10^9} = 2.4\times10^{-4}s$$

Total time $= 1.04\times10^{-3}s$

Clock cycles required in Both cases:

$$\text{CPU Clock cycles} = \text{Instruction count} \times \text{CPI}$$

$$\text{Clock cycles}_A = (1.0 \times 10^5)(1) + (2 \times 10^5)(2) + (5 \times 10^5)(3) + (2 \times 10^5)(3)$$

$$\text{Clock cycles}_A = 2.6 \times 10^6 \text{ cycles}$$

$$\text{Clock cycles}_B = (1.0 \times 10^5)(2) + (2 \times 10^5)(2) + (5.0 \times 10^5)(2) + (2 \times 10^5)(2)$$

$$\text{Clock cycles}_B = 2.0 \times 10^6 \text{ cycles}$$

$$\text{Total time}_{P2} = \frac{(1\times10^5\times2 + 2\times10^5\times2 + 5\times10^5\times2 + 2\times10^5\times2)}{(3\times10^9)}$$

(ic)

$$CPU_A = \frac{(1.0\times10^5)\times2}{3\times10^9} = 6.67\times10^{-5}\,s$$

$$CPU_B = \frac{(2\times10^5)\times2}{3\times10^9} = 1.33\times10^{-4}\,s$$

$$CPU_C = \frac{(5\times10^5)(2)}{3\times10^9} = 3.33\times10^{-4}\,s$$

$$CPU_D = \frac{(2\times10^5)(2)}{3\times10^9} = 1.33\times10^{-4}\,s$$

Total time $= 6.657\times10^{-4}\,s$

P2 is the best Implementation

The Global CPI for Each Implementation

$$CPI_A = \frac{(2.5\times10^9)(1.04\times10^{-3})}{10^6} = 2.6$$

$$CPI_B = \frac{(3.0\times10^9)(6.657\times10^{-4})}{10^6} = 1.9971$$

UNIT - II

PART - A

Define Little Endian arrangement [NOV/DEC-2014]

When the lower byte addresses are used for the less significant bytes [the rightmost byte] of the word, addressing is called little-endian

---

2. What is the overflow/underflow conditions for addition and subtraction

Addition: If two operands having same sign it will cause the overflow in addition operation.

Eg first operand 32 bit word and second operand is a 32-bit word, so the result also must be 32-bit word. If the result exceeds more than 32 bit word it causes overflow

Subtraction: overflow occurs in subtraction based on two cases. [May-2015]

Case 1: If we subtract a negative number from a positive number and the result is negative, the overflow will occur.

Case 2: If we subtract a positive numbers from a negative numbers and the result is positive, then overflow

3 State the representation of double precision floating point number. [Nov/Dec-2015]

S	Exponent	Fraction
1 bit	11 bit	52 bits

*The representation of double precision floating point number take two MIPS words.

where

S is the sign of the number

4. What do you mean by subword Parallelism [May-2015] [May/June-2016]

*Suppose the size of the data item within the vector is 128-bits. with this data by partitioning the carry chains, a processor can use parallelism to perform simultaneous operations on short vectors of

1. Sixteen 8 bit operands
2. Eight 16 bit operands
3. Four 32 bit operands
4. Two 64 bit operands

*The Parallelism which occurs within a wide word, in case it extends means that is called as Subword parallelism. It is also known as data level parallelism or vector or SIMD parallelism.

5. Subtract $(11011)_2 - (10011)_2$ using 2's complement.
[Nov/Dec-2017]

$$\begin{array}{r} 11011 \\ (-)\,10011 \end{array}$$

(i) Take 2's complement for 10011

First take 1's complement

$10011 = 01100$

Now, take 2's complement $\begin{array}{r} 01100 \\ +\quad\ 1 \\ \hline 01101 \end{array}$

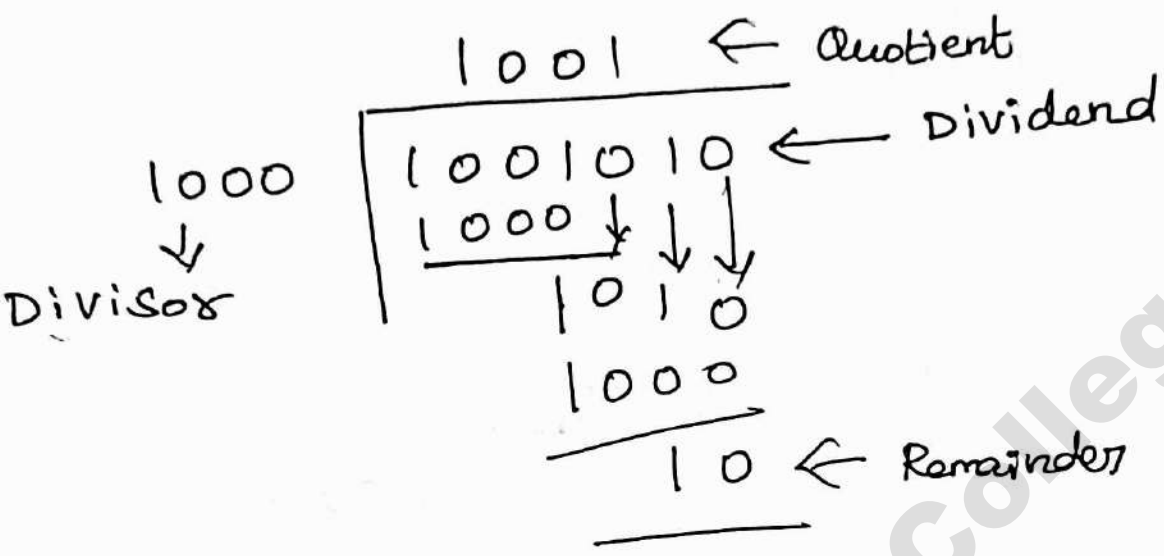
Now perform Binary Subtraction

$$\begin{array}{r} 11011 \\ 01101 \\ \hline 1\,01000 \end{array}$$

Indicate Negative Sign

Answer = $\boxed{1\,01000}$

6 Divide $(1001010)_2 \div (1000)_2$ [NOV/DEC-2017]



7. Subtract $(11010)_2 - (10000)_2$ using 1's complement and 2's complement method. [APR/MAY - 2017]

1's Complement method

i) Take 1's complement of subtrahend: 01111

ii) Add with minuend:

```
    1 1 0 1 0
  + 0 1 1 1 1
  ------------
1]  0 1 0 0 1
```

carry → 1

iii) If there is a carry add with result.

(ie)

```
    0 1 0 0 1
            1
  ------------
    0 1 0 1 0
  ------------
```

iv) The Result is 01010

2's Complement method

i) Take 2's complement of subtrahend: 10000

ii) Add with minuend:

```
    1 1 0 1 0
    1 0 0 0 0
  ------------
1]  0 1 0 1 0
```

carry → 1

iii) If there is a carry discard it: 01010

iv) The result is 01010

8 Write the rules to perform addition on floating point numbers. [APR/MAY - 2017]

Step1: Align the decimal number of the smallest Exponent number that matches the larger Exponent.

(ie) Shift the smaller number right until Exponents match.

Step2: Add/Subtract the mantissas, depending on sign

Step3: Normalize the result if necessary.

Step4: Check for overflow.

Step5: Rounding off to appropriate number of bits.

9 What is a guard bit and what are the ways to truncate the guard bits? [NOV/DEC-2016]

Guard bits:

The first of two extra bits kept on the right during intermediate calculations of a floating point format

There are several ways to truncation

1. To remove the guard bits and make no changes in the retained bits (Chooping)

2. Von Neumann rounding - If the bits to be removed are all 0s, they are simply dropped, with no change to the retained bits

3. Rounding procedure - Achieves the closest approximation the number being truncated and is an unbiased technique.

10 What is arithmetic overflow. [NOV/DEC - 2016]

* Arithmetic overflow, it is the condition that occurs when a calculation produces a result that is greater than a given register or storage location can store or represent.

* It occurs in which positive exponent becomes too large to fit in the exponent field.

---

11 Define ALU [MAY- JUNE- 2016]

* ALU - Arithmetic Logic Unit is a digital circuit used to perform arithmetic operations like, addition, multiplication and logic operations like AND, OR, NOT etc.

* It represents the fundamental building block of the central processing unit (CPU) of a computer.

12 Show the IEEE 754 binary representation of the number $(-0.75)_{10}$ in single precision [Apr/May -18]

Ans:

S	Exponent	Fraction
1bit	8bits	23 bits

where

S - Sign of the floating-point number
0 → positive, 1 → negative

Exponent - The value of the 8-bit exponent field

Fraction - 23-bit number

i) Convert decimal number $(-0.75)_{10}$ into binary

(ie)
$$0.75 \times 2$$
$$1.50 \times 2$$
$$1.0$$

The binary equivalent of

$(-0.75)_{10} = (0.11)_2$ (ie) $-0.11_2 \times 2^0$

ii) Normalize the number

(ie) $(-0.11)_2 = -1.1 \times 2^{-1}$

iii) Single precision representation

For the given number

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-127)}$

(ie) $(-1)^0 \times (1 + 0.1_2) \times 2^{(-1+127)}$

$= (-1)^0 \times (1 + 0.1_2) \times 2^{126}$

The given number can be written in IEEE single-precision format as

Sign bit	Exponent (8bit)	Fraction (23 bits)
0	0111 1110	1000000 00000000 0000 0000

13 Define a datapath in a CPU [Apr/May-18]

The 'data path' is a representation of the flow of information (data, instructions)

Data path elements are

i) Memory unit - It is used to store the instructions of a program and supply instructions given an address

ii) program Counter (PC) - It is a register that holds the address of the Current Instruction

iii) Adder - An adder is a device which is used to increment the PC to the address of the next instruction

* This adder can be built from the ALU.

14 Perform X - Y using 2's complement arithmetic for the given two 16-bit binary numbers. [Nov/Dec-18]

X = 0000 1011 1110 1111 and

Y = 1111 0010 1001 1101

X - Y (ie) 0000 1011 1110 1111
1111 0010 1001 1101

Take 2's complement for 1111 0010 1001 1101

First, take 1's complement

1111 0010 1001 1101 → 0000 1101 0110 0010

Now, take 2's complement

0000 1101 0110 0010
+ 1

Y = 0000 1101 0110 0011

Now, Perform Binary subtraction,

X = 0000 1011 1110 1111

Y = 0000 1101 0110 0011

0001 1001 0101 0010

Answer = 0001 1001 0101 0010

PART-B

Divide $(12)_{10}$ by $(3)_{10}$ Using the Restoring and non restoring division algorithm with step by step intermediate results and Explain [NOV/DEC-2014]

Discuss in detail about division algorithm in detail with diagram and examples [NOV/DEC-2015] [NOV/DEC-2016] [APR/MAY-17]

Illustrate the division algorithm with an Example [NOV/DEC-17]

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.

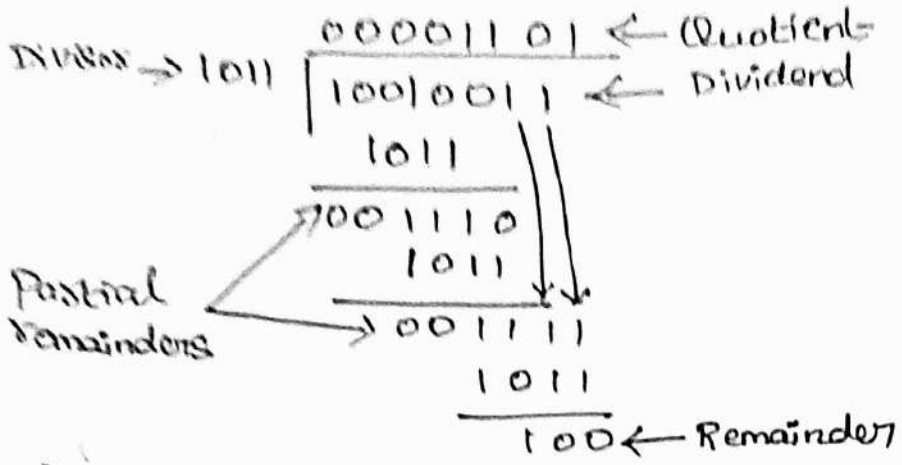
* Divide's use two operand called the dividend and divisor and produce the results called quotient and remainder.

* 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

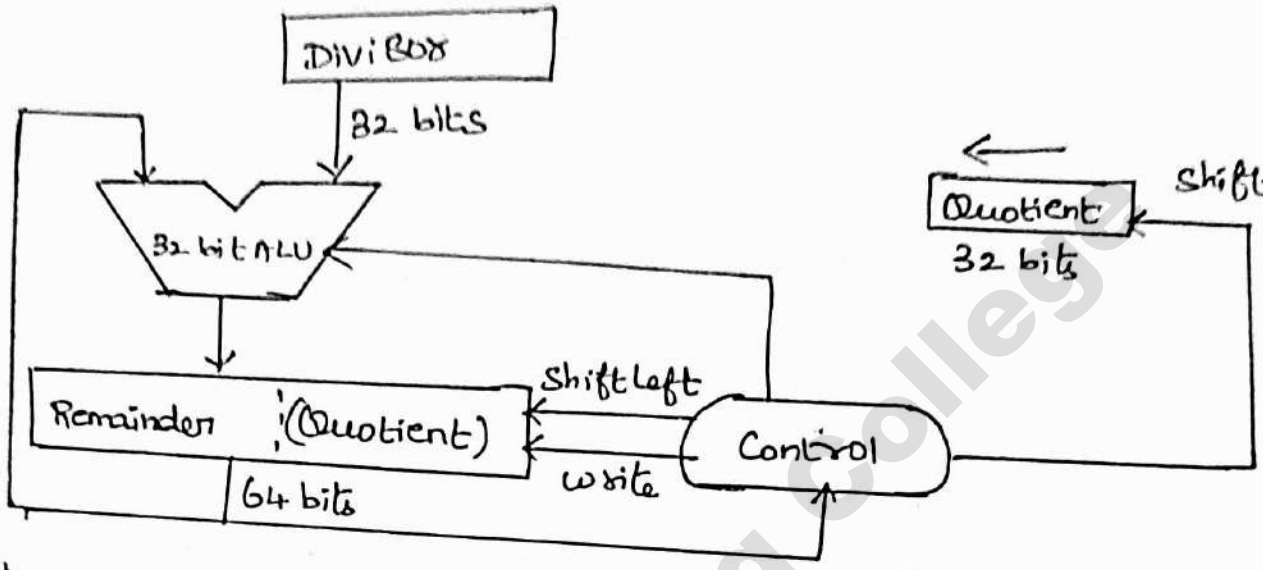
Step3: If the sign bit of Remainder register is 1, then shift Quotient register(Q) left by on binary position and set Q_0 to 0 and add divisor back to remainder register (ie restore remainder register); otherwise set Q_0 to 1.

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

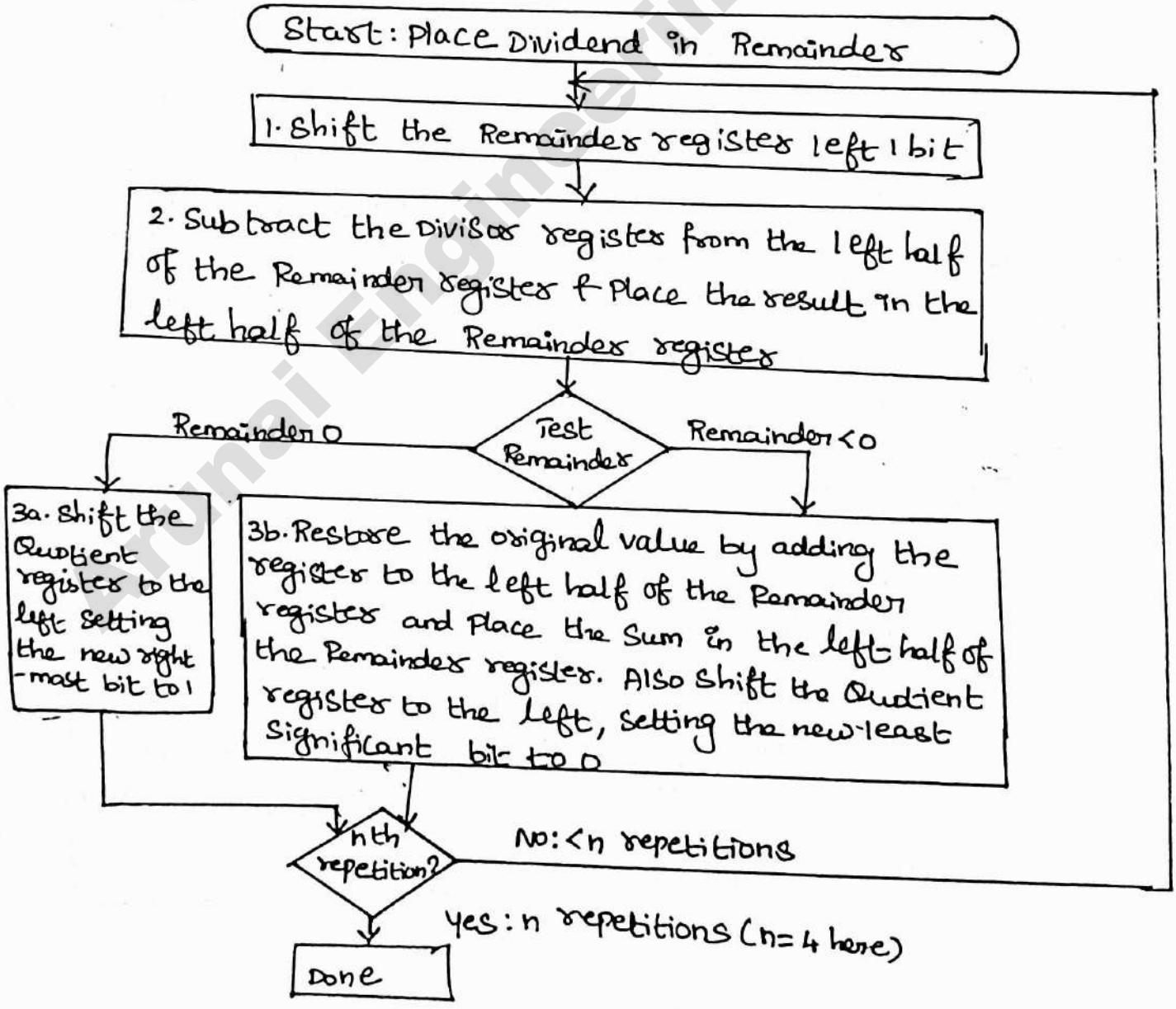
For simplicity, use 4 bit version to perform division, same concept can Expanded to 32 bit number.

Restoring Division (or) Second Version of Division Hardware of 32-bit Boolean number representation

Schematic Diagram of ALU Circuitry



Flowchart



Algorithm for Non-Restoring Division:

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

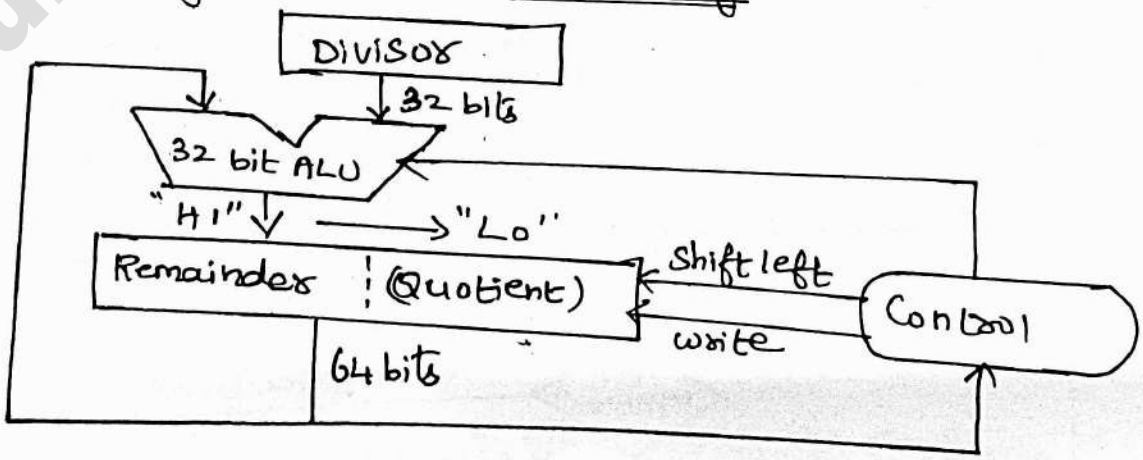
b) If the sign of RemReg is <0, set Q_0 to 0 otherwise set Q_0 to 1.

Step2: Repeat steps1 and 2 for <33 times

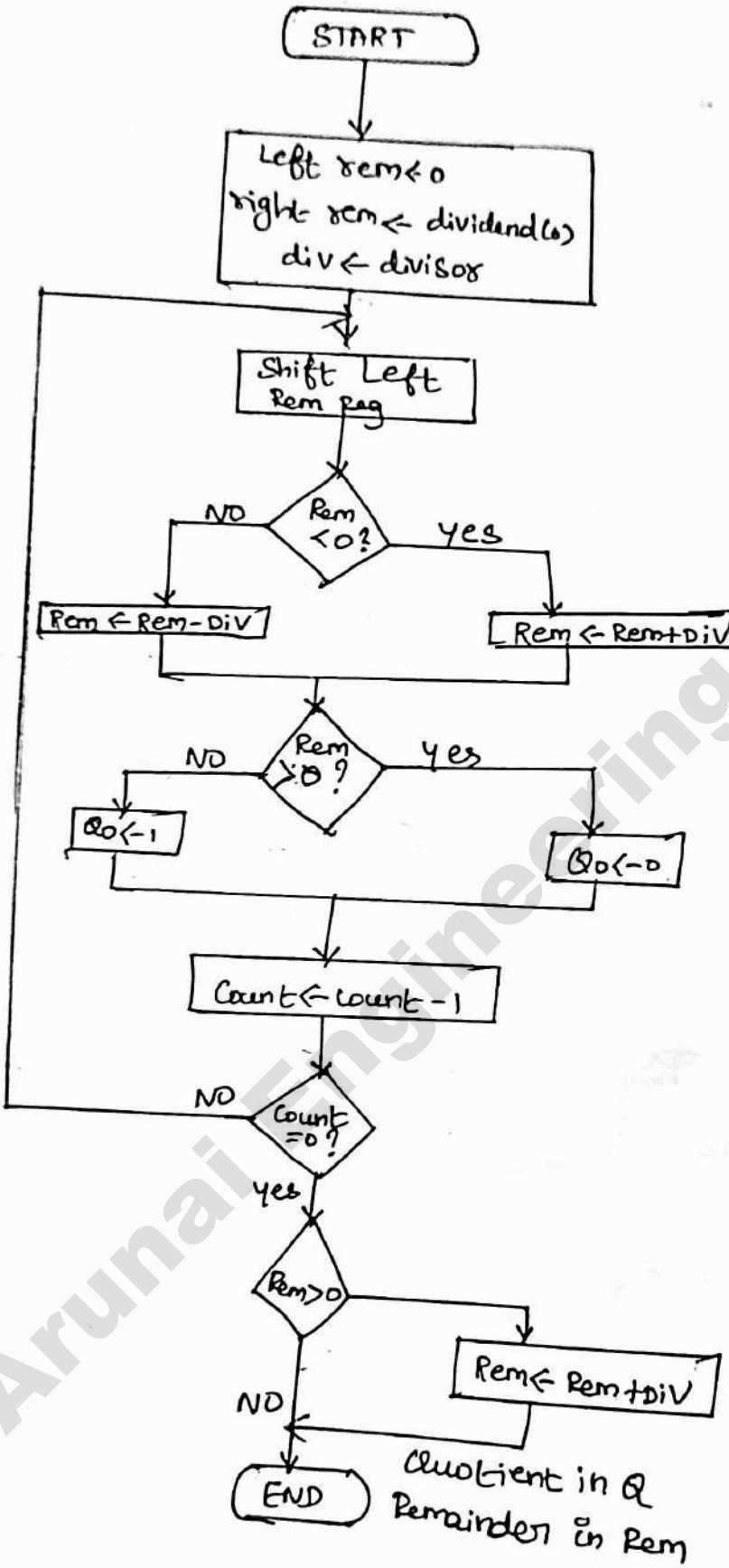
Step3: At the end of n cycle, If the sign of RemReg is 1, add divisor to RemReg. This step is required to leave the proper positive remainder in RemReg at the end of n cycles.

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

Schematic Diagram of ALU circuitry



Flowchart:



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	0010	0000	0000
	Left half Rem reg ← left Rem - Divisor		0011	
			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	Right 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	$001Q_1$
	Left half Rem reg ← left Rem − Divisor	0011	
		1101	
	Rem < 0, $Q_0 = 0$	1101	0010
4	Sll Rem Reg	1010	$0100Q_0$
	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 $(4)_{10}$

2. Explain Booth's Algorithm for the multiplication of signed two's complement numbers. [NOV/DEC-16]

Define Booth Multiplication algorithm with suitable example [MAY/June-2016]

Multiply the following pair of signed numbers using Booth's bit-pair recoding of the multiplier. A=+13 (multiplicand) and B=-6 (multiplier) [NOV/DEC-2014]

Booth's Algorithm:

* Booth's Algorithm is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly.

* In this algorithm, multiplier bits are encoded from right to left, before the bits are used for getting partial products.

* To encode the multiplier bits, two adjacent bits b_i (current bit) and b_{i-1} (previous bit) is examined as follows.

1. If i^{th} bit b_i is 0 and $(i-1)^{th}$ bit b_{i-1} is 1, then take b_i as +1 (i.e. for (0,1) pair, Multiplicant is added to the partial product

2. If i^{th} bit b_i is 1 and $(i-1)^{th}$ bit b_{i-1} is 0, then take b_i as -1 (i.e., for (1,0) pair, Multiplicant is subtracted to the partial product

3. If i^{th} bit b_i is 0 and $(i-1)^{th}$ bit b_{i-1} is 0, then take b_i as 0

4. If i^{th} bit b_i is 1 and $(i-1)^{th}$ bit b_{i-1} is 1, then take b_i as 0 (i.e.) For (0, 0) and (1, 1) pair, then neither addition nor subtraction is performed.

5. when least significant bit $b_0 = 1(0)$, assume that it had b_{i-1} as 0 (implied zero) thus take $b_0 = -1(0)$

Multiplier		Version of Multiplicand Selected by bit i
Bit i	Bit i-1	
0	0	0 × M
0	1	+1 × M
1	0	−1 × M
1	1	0 × M

Booth Multiplier Recording Table

* Booth's algorithm effectively skips over sequence of 1's and 0's in multiplier. As a result, the total number of addition/subtraction steps required to multiply two numbers decrease

* The process of inspecting the multiplier bits required by booth's algorithm can be viewed as encoding the multiplier using three digits 0, +1 and -1 where 0 means shift the Partial product to the right (ie no addition or subtraction is Performed)

* While +1 means add multiplicand before shifting and -1 means subtract multiplicand from the Partial Product before shifting.

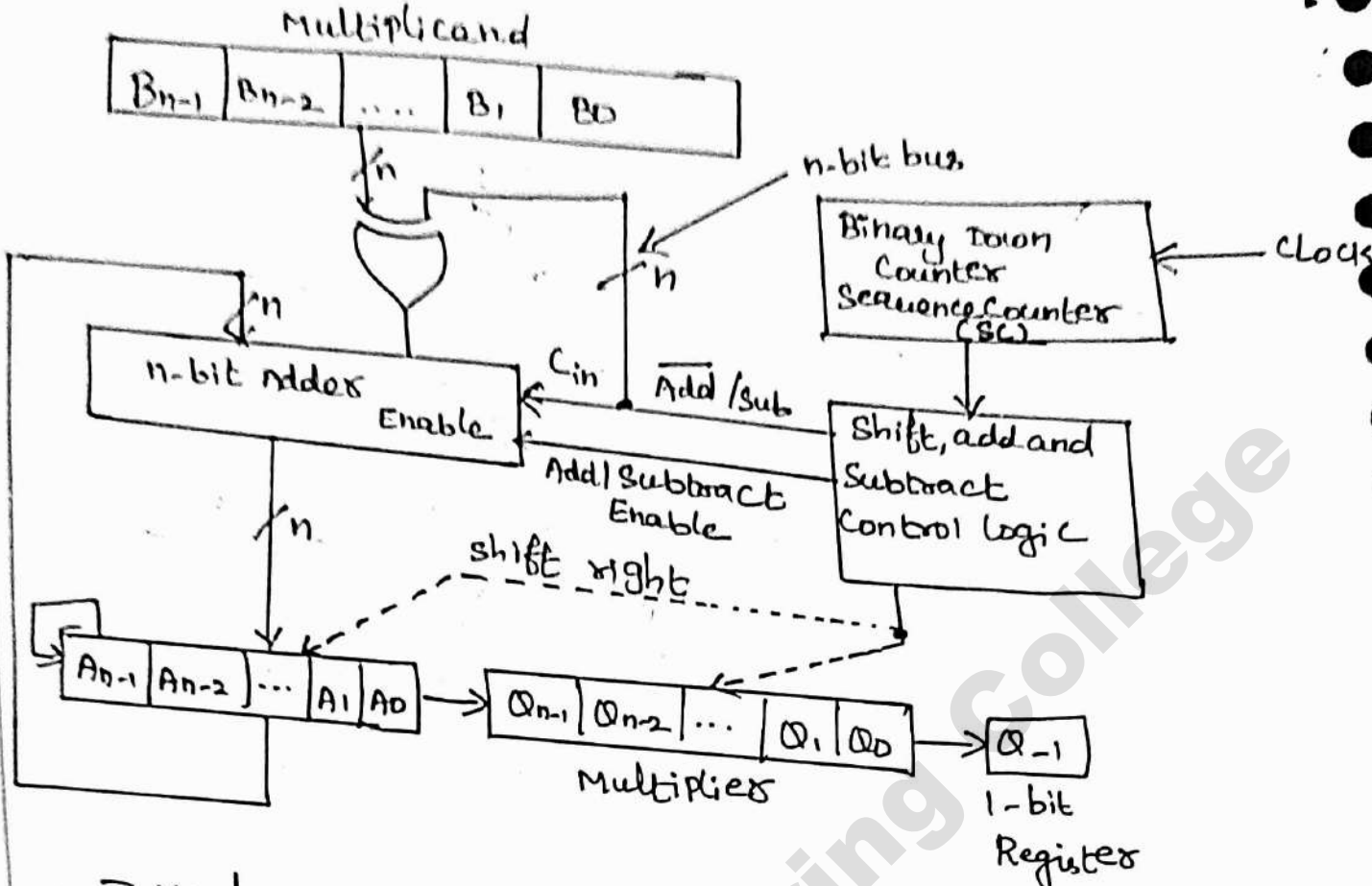
* The number thus produced is called as Signed digit number and this process of Converting a multiplier into a signed digit form is called as Multiplier recoding

Hardware Implementation of Booth's Algorithm:

* It consists of n-bit adder, shift and subtract control logic and four registers A, B, Q and Q-1

* Multiplier and Multiplicand are loaded into register Q and register B respectively and register A and Q-1 are initially set to 0.

* The Seauence Counter, Sc is set to a number n eaual to the number of bits in the multiplier.



Initial settings: $A \leftarrow 0$ and $Q_{-1} = 0$

The n-bit adder performs addition of <u>two inputs</u>.

i) A register

ii) Multiplicand.

↳ In case of addition, $\overline{Add}/Sub$ line is 0, therefore $C_{in} = 0$ and multiplicand is directly applied as a second input to the n-bit adder.

↳ In case of subtraction, $\overline{Add}/Sub$ line is 1, therefore $C_{in} = 1$ and multiplicand is complemented and then applied to the n-bit adder. As a result, the 2's complement of multiplicand is added in the A register.

↳ The shifts, add and subtract control logic scans bits Q_0 and Q_{-1} one at a time and generates the control signals.

Flowchart of Booth's Multiplication Algorithm

Algorithm:

Step 1: Load $A = 0$, $Q_{-1} = 0$

B = Multiplicand

Q = Multiplier

$SC = n$

Step 2: Check the Status of Q_0 Q_{-1}

if $Q_0 Q_{-1} = 10$ Perform $A \leftarrow A - B$

if $Q_0 Q_{-1} = 01$ perform $A \leftarrow A + B$

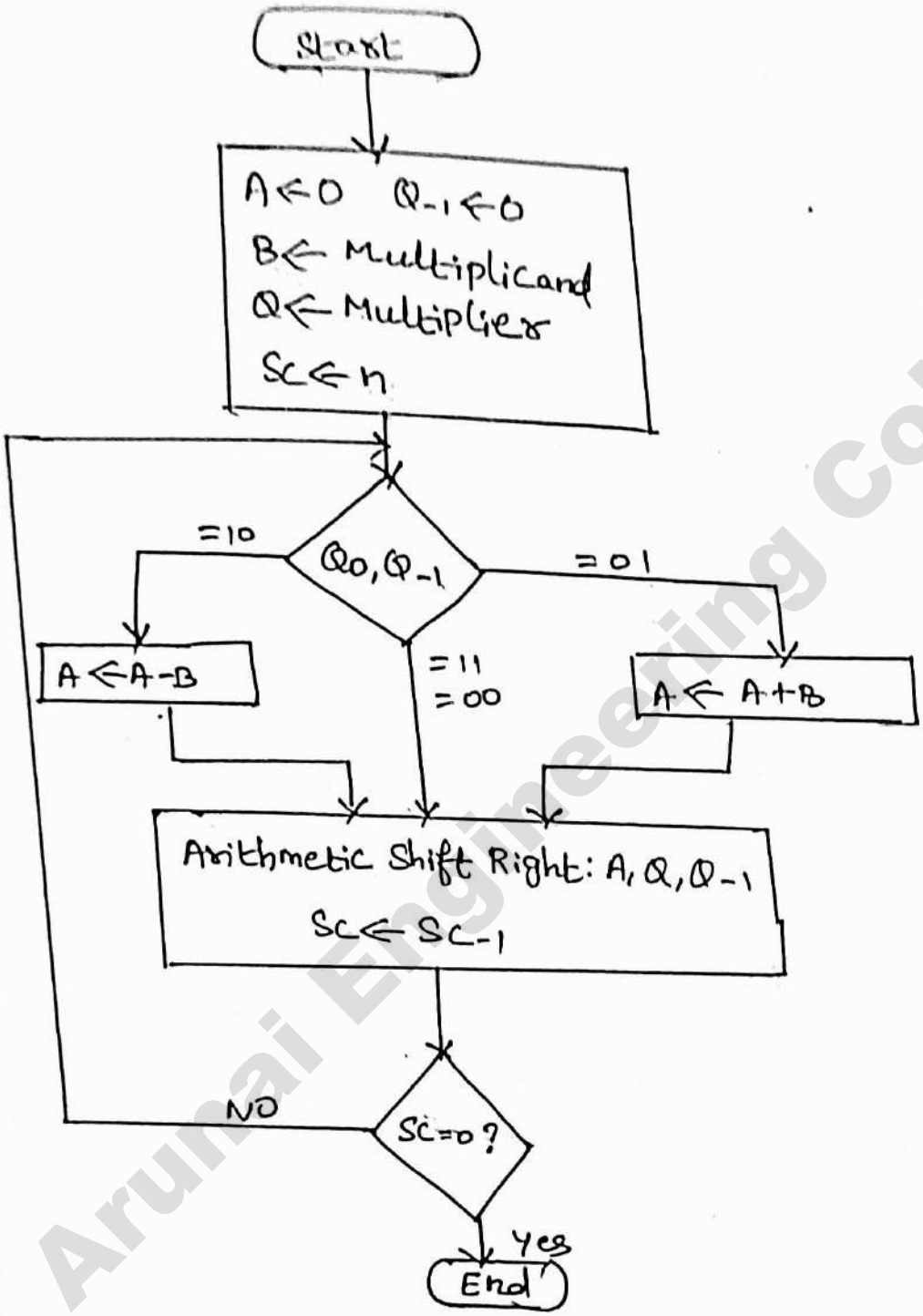
Step 3: Arithmetic Shift right : A, Q, Q_{-1}

Step 4: Decrement Sequence Counter if not zero, repeat Step 2 through 4

Step 5: Stop

Booth's Algorithm for Signed Multiplication

Flowchart:



Example:

Multiply the following pair of signed numbers using Booth's bit-pair recoding of the multiplier A = +13 (Multiplicand) and B = -6 (Multiplier)

Multiplicand:

0	1	1	0	1

+13

Multiplier:

0	0	1	1	0

6

1	1	0	1	0

-6 (2's complement form)

1	1	0	1	0	0
0	-1	+1	-1	0	

implied zero multiplier

Recoded Multiplier

					0	1	1	0	1	Multiplicand
					0	-1	+1	-1	0	Recoded Multiplier
Sign Extension →	0	0	0	0	0	0	0	0	0	
+	1	1	1	1	0	0	1	1		2's complement of multiplicand
+	0	0	0	1	1	0	1			
+	1	1	0	0	1	1				2's complement of multiplicand
+	0	0	0	0	0					
	1	1	0	1	1	0	0	1	0	(-78) in 2's complement form

3: Explain how floating point addition is carried out in a computer system. Give an Example for a binary floating point addition [APR/MAY-2015] [Apr/May-17]
Design an arithmetic element to perform the basic floating point operation [Apr/May-17]

Floating Point Addition:

Floating Point:

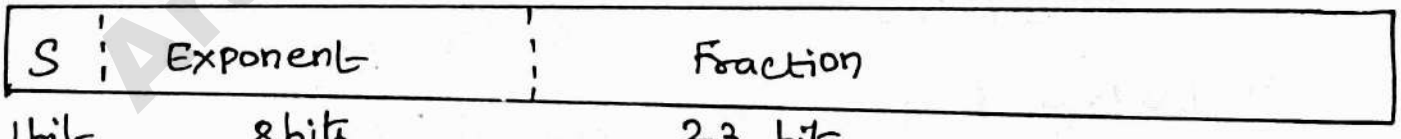
Computer arithmetic represents numbers in which the binary point is not fixed called floating point.

Floating point representation

* A designer of a floating point representation must find a compromise between the size of the fraction and the size of the exponent.

* Fraction is the value generally between 0 and 1 it is called the mantissa

* Exponent is the numerical representation system of floating point arithmetic



MIPS floating point Number representation

where S = sign of the floating point number
1 - negative, 0 - positive

General floating point numbers are of the form

$$(1)^S \; F 2^E$$

F - involves the value in the fraction field
E - involves the value in the exponent field

Overflow (Floating point)

* overflow is a situation in which a positive exponent becomes too large to fit in the exponent field.

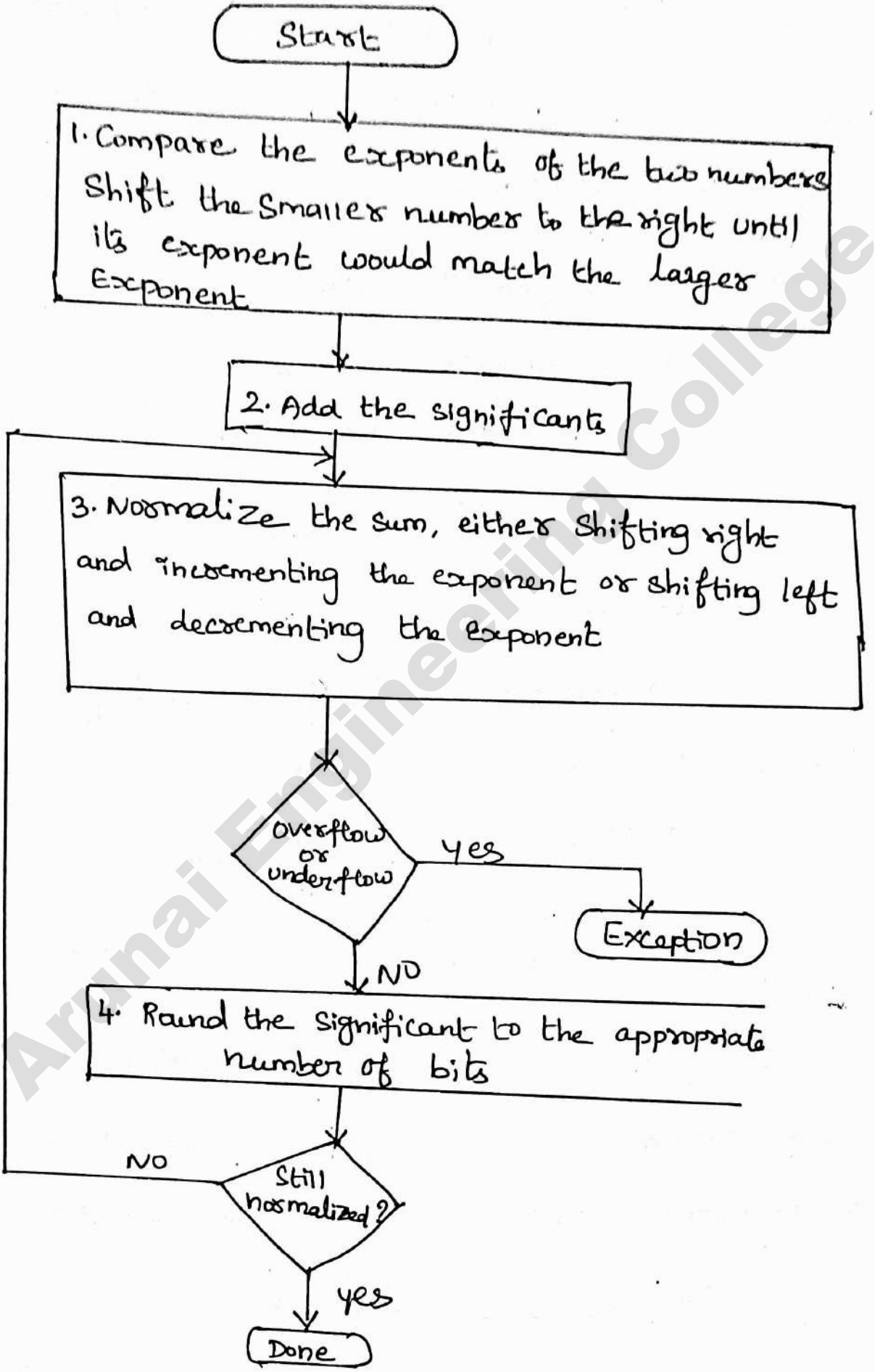
* overflow means the exponent is too large to be represented in the exponent field.

Underflow (Floating Point)

* Underflow is a situation in which a negative exponent becomes too large to fit in the exponent field.

* Underflow occurs when the negative exponent is too large to fit in the exponent field.

Flowchart for Floating Point Addition:



Binary floating point Addition:

Perform floating point addition using the numbers 0.5_{ten} and 0.4375_{ten} Use the floating point addition algorithm

Assume 4 bits of precision

First number $= 0.5_{ten}$

Second number $= 0.4375_{ten}$

Let 0.5_{ten} can also be written as

$$0.5_{ten} = 1/2_{ten} = 1/2^1_{ten}$$

$$0.1_{two} = 0.1_{two} 2^0 = 1.000_{two} 2^1$$

0.4375 Can also be written as

$$0.4375_{ten} = 7/16_{ten} = 7/2^4_{ten}$$

$$0.0111_{two} = 0.0111_{two} 2^0 = 1.110_{two} 2^2$$

Step 1:

The Significant of the number with the lesser exponent ($1.11_{two} 2^2$) is shifted right until its exponent matches the larger number

$$1.110_{two} 2^2 = 0.111_{two} 2^1$$

Step 2:

Add the Significant

$1.000_{two} 2^1 (0.111_{two} 2^1) 0.001_{two} 2^1$

Step 3:

↳ Normalize the sum, checking the overflow or Underflow

$0.001_{two} 2^1 = 0.010_{two} 2^2 = 0.100_{two} 2^3 = 1.000_{two} 2^4$

Since $-127 \leq 4 \leq 126$, there is no overflow or Underflow

Step 4:

Round the sum

$1.000_{two} 2^4$

The sum already fits exactly in 4 bits, so there is no change to the bits due to rounding

$1.000_{two} 2^4 = \underline{0.0001000}_{two} = \underline{0.0001}_{two}$

$= 1/2^4_{ten}$

$= 1/16_{ten}$

$= 0.0625_{ten}$

1.000×2^4

0.0625×2

0.1×2^1

0.01×2^2

0.001×2^3

0.0001×2^4

4: Explain the sequential version of multiplication algorithm and its hardware. [APR/MAY – 2015]

Explain in detail about the multiplication algorithm with suitable example and diagram [NOV/DEC – 2015]. [APR/MAY - 2017]

Multiplication:

* The first operand is called Multiplicand
* The second operand is called Multiplier
* Final Result is called Product

Basic rules for Binary digits multiplication

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Eg Multiplying 1010 by 1000

```
Multiplicand      1 0 1 0
Multiplier      x 1 0 0 0
                ---------
                  0 0 0 0
                0 0 0 0
              0 0 0 0
            1 0 1 0
            ---------------
Product     1 0 1 0 0 0 0 0
            ---------------
```

* 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 of 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

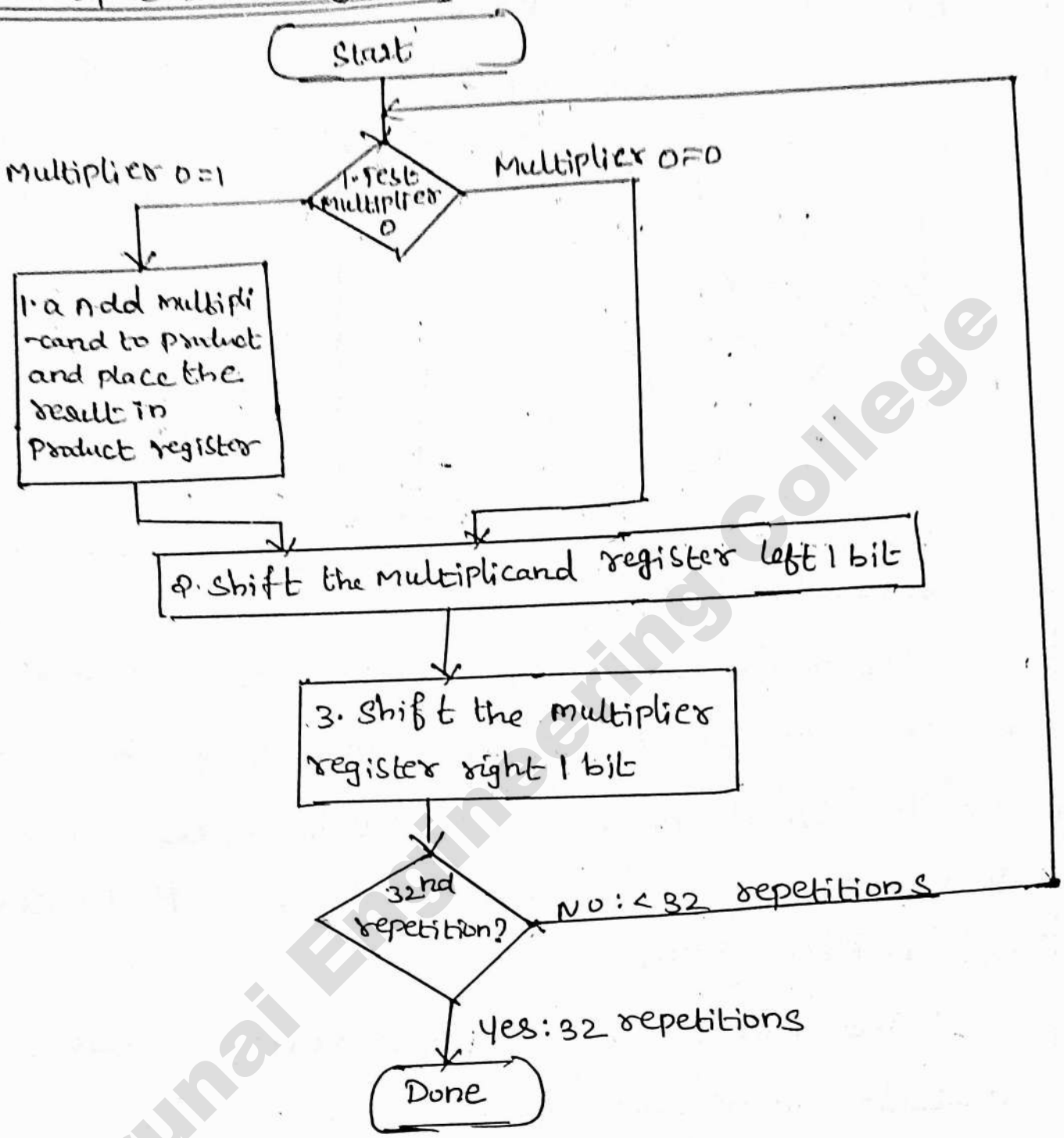
Example:

$$4 \times 8 = 32$$

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

```
  8
  8
  8
  8
 ---
 32
 ---
```

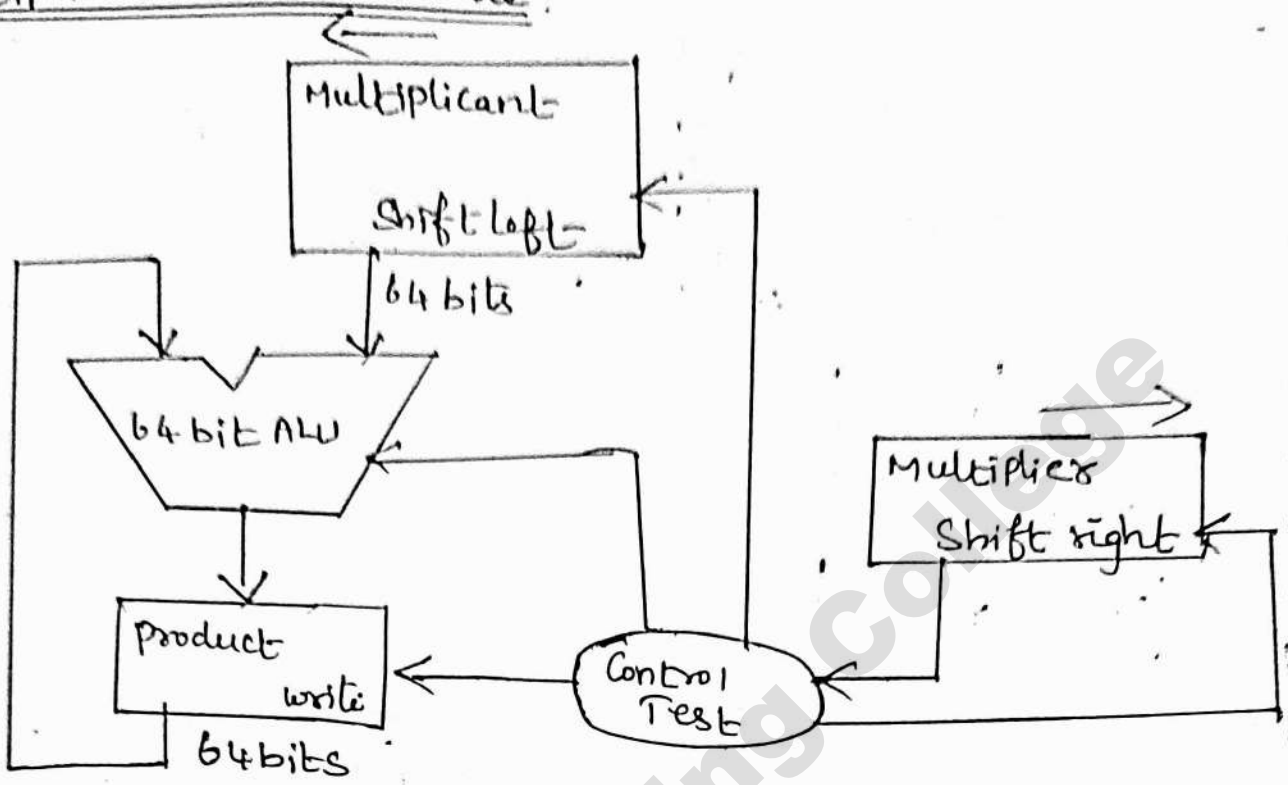
Multiplication Algorithm:



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

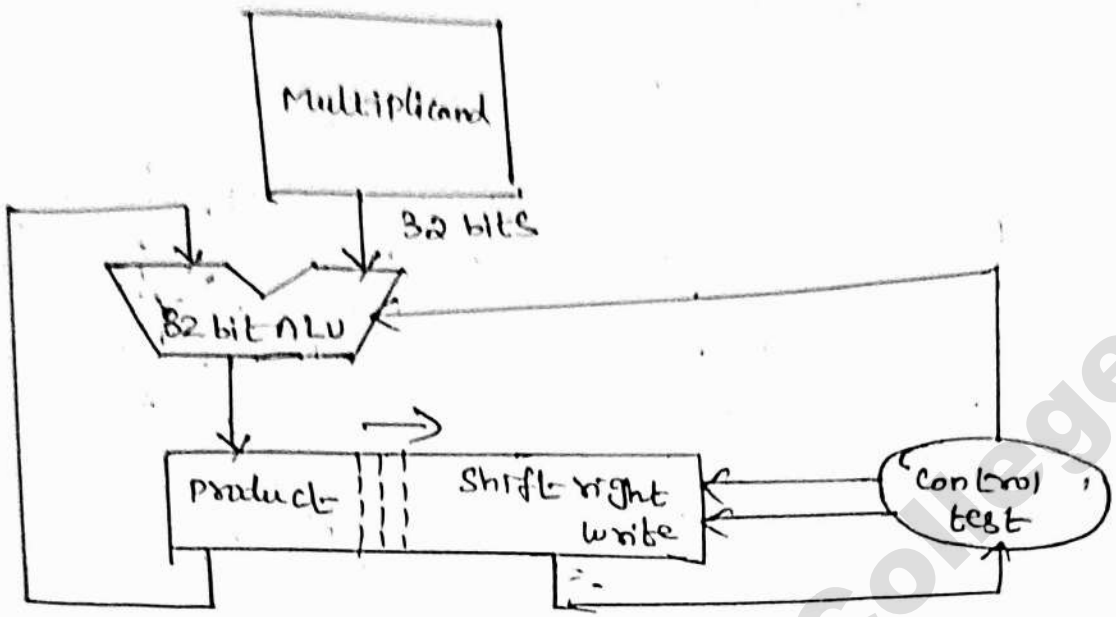
* To increase the speed of process by performing the operations in parallel manner.

* 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.

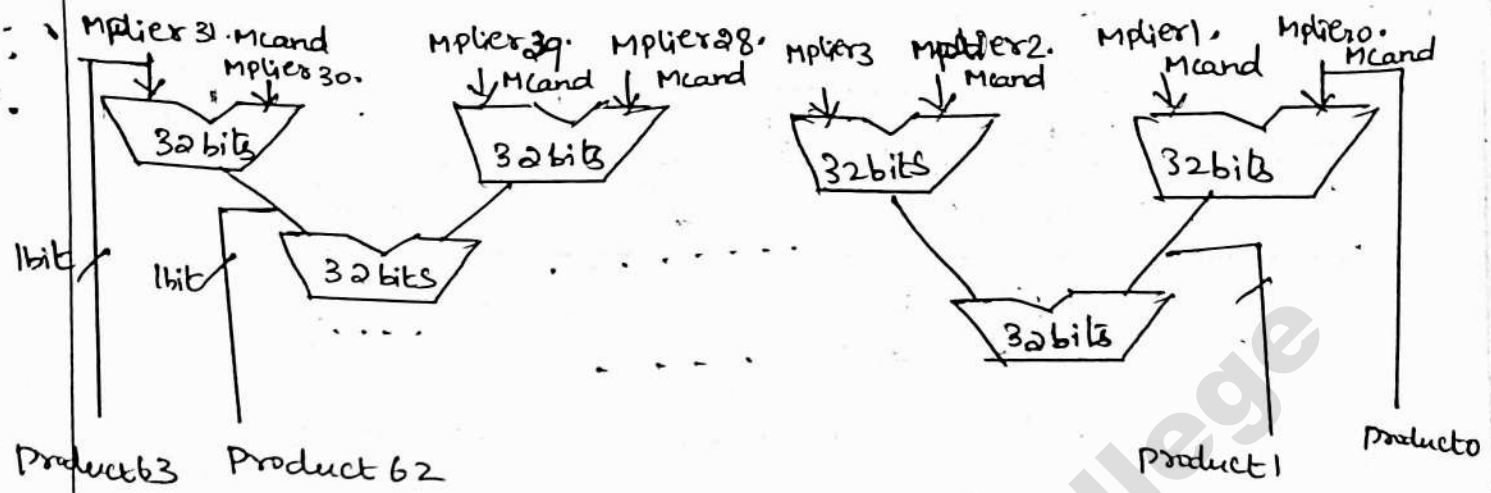
* The separate multiplier register also disappeared and the multiplier is placed instead in the right half of the Product register

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.

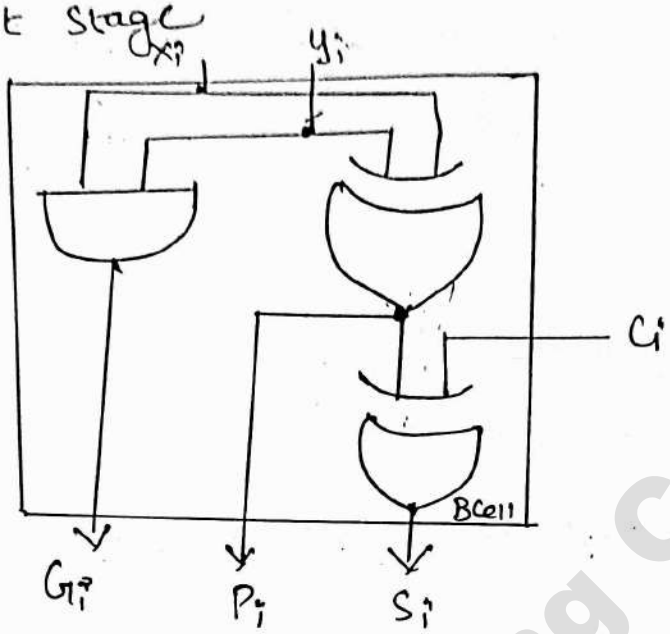
Example:

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 [1]	0000 0010	0000 0000
1	1a: 1 ⇒ prod = prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left multiplicand	0011	0000 0100	0000 0010
	3: Shift right multiplier	0 00 [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	0 000 [0]	0000 10000	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 [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

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

* The final expression for carry variable is

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

* Thus all carries can be obtained three gate delays after the input signals x, y, c_0 are applied

5 Briefly Explain Carry lookahead adder [NOV/DEC-2014]

* Carry Look ahead Adder (CLA) is a fast adder which speed up the generation of Carry signals.

* It uses two functions

i) Carry generate, G_i

ii) Carry propagate, P_i

The Logic Expression for a Sum S_i & Carry out C_{i+1} of Stage i are

$$\boxed{S_i = x_i \oplus y_i \oplus c_i}$$

$$C_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$C_{i+1} = x_i y_i + (x_i + y_i) c_i$$

$$C_{i+1} = G_i + P_i C_i$$

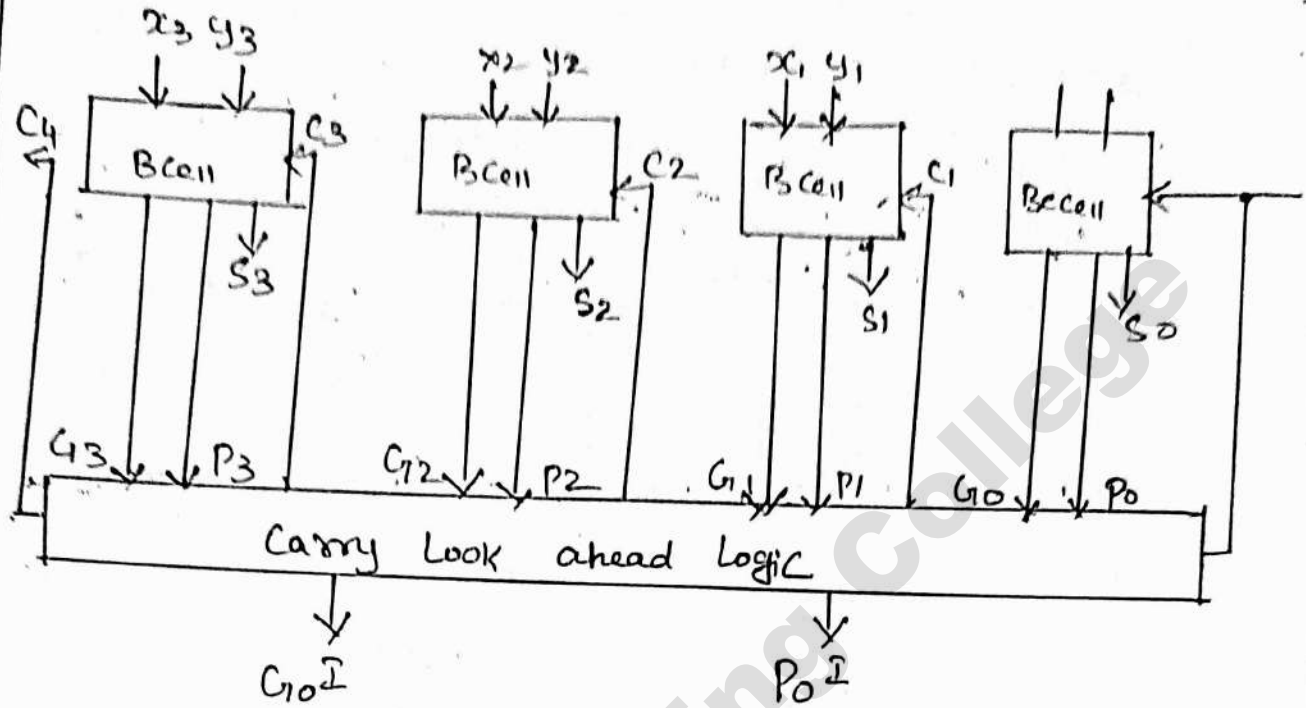
where $G_i = x_i y_i$

$P_i = x_i + y_i$

* If the generate function for Stage i is equal to 1, then $C_{i+1} = 1$

* The Propagate function means that an input Carry will produce an output Carry when either x_i or y_i is 1

Design of 4-bit adder



The carries can be implemented as

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 C_3$

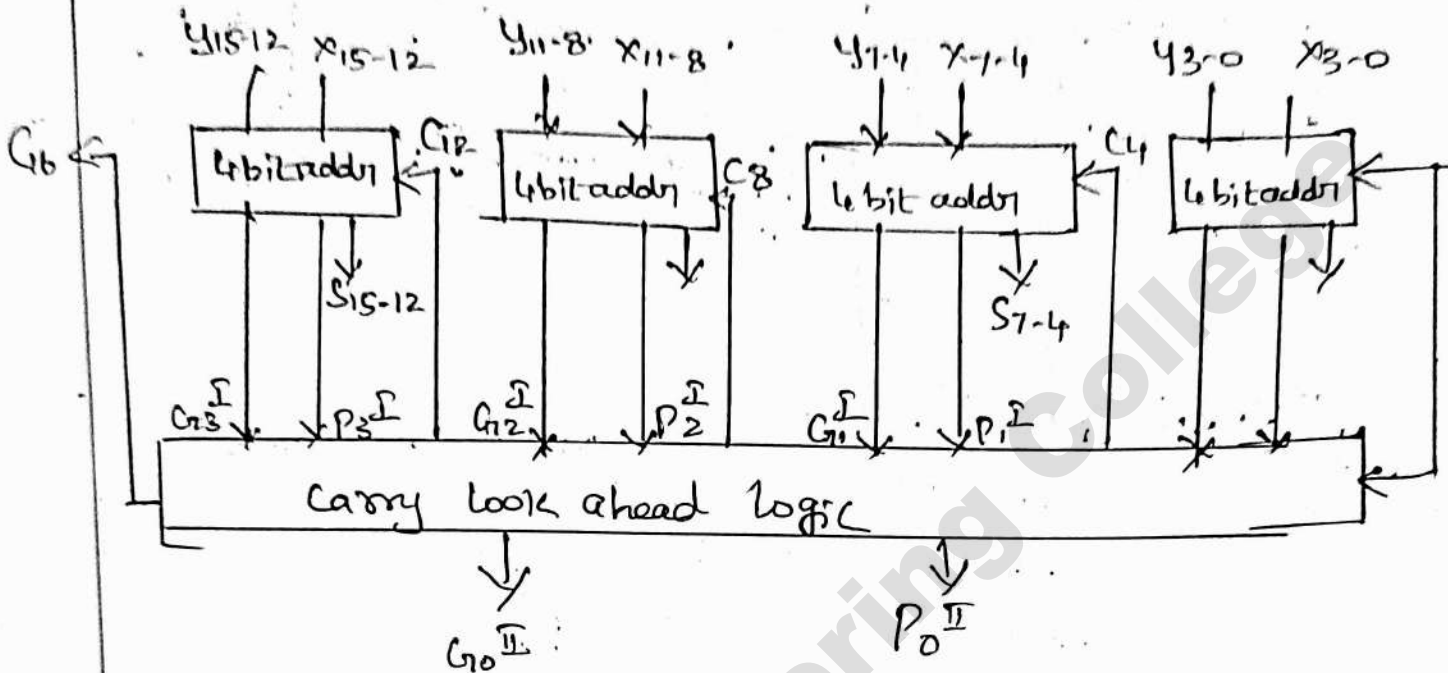
$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

* The carries are implemented in carry look ahead logic An adder of this form called carry look ahead Adder.

* Delays through the adder is 3 gate delays for all carry bits

* 4 gate delays for all sum bits which is less than Ripple Carry adder

Construction of 16 bit Adder built from 4 bit Adder



* By using look ahead logic, higher level carry look ahead circuit is developed.

* The propagate and Generate functions will be

$$P_0^I = P_3 P_2 P_1 P_0$$

$$G_0^I = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

The carry C_{16} will be

$$C_{16} = G_3^I + P_3^I G_2^I + P_3^I P_2^I G_1^I + P_3^I P_2^I P_1^I G_0^I + P_3^I P_2^I P_1^I P_0^I C_0$$

6. What is meant by subword Parallelism? Explain

[Apr/May-17]

* 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 whole 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 32 bits

* An instruction operates on 8 bit 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.

7 i) Add the numbers $(0.5)_{10}$ and $(0.4375)_{10}$ using the floating point addition. [Nov/Dec-17]

Solution:

Assume 4 bits of precision

First number = 0.5_{10}

Second number = 0.4375_{10}

Convert the decimal numbers into binary

$0.5_{10} = 0.1_2 \times 2^0 = 1.000_2 \times 2^{-1}$

$0.4375 = 0.0111_2 \times 2^0 = 1.110_2 \times 2^{-2}$

0.5×2

1.000×2^{-1}

$0.0111_2 \times 2^0$

0.111×2^1

1.11

$0.4375 \times 2 = 0.875$

$0.875 \times 2 = 1.75$

$0.75 \times 2 = 1.50$

$0.50 \times 2 = 1.00$

Step 1:

Equalize Exponents

* The significant of the number with the lesser exponent $(1.110_2 \times 2^{-2})$ is shifted right until its exponent matches the larger number

$1.110_2 \times 2^{-2} = 0.111_2 \times 2^{-1}$

Step 2:

Add the significant

7. i) Add the numbers $(0.5)_{10}$ and $(0.4375)_{10}$ using the floating point addition. [NOV/DEC-17]

Solution:

Assume 4 bits of precision

First number = 0.5_{10}

Second number = 0.4375_{10}

Convert the decimal numbers into binary

$0.5_{10} = 0.1_2 \times 2^0 = 1.000_2 \times 2^{-1}$

0.5×2

1.000×2^1

$0.4375 = 0.0111_2 \times 2^0 = 1.110_2 \times 2^{-2}$

$0.4375 \times 2 = 0.875$

$0.875 \times 2 = 1.75$

$0.75 \times 2 = 1.50$

$0.50 \times 2 = 1.00$

$0.0111_2 \times 2^0$

0.111×2^1

1.111

Step 1:

Equalize Exponents

* The significant of the number with the lesser exponent $(1.110_2 \times 2^{-2})$ is shifted right until its exponent matches the larger number

$1.110_2 \times 2^{-2} = 0.111_2 \times 2^{-1}$

Step 2:

Add the significant

$1.000_2 \times 2^{-1} + 0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

Step 3:

Normalization :-

1.000×2^{-1}
0.111×2^{-1}
1.111×2^{-1}

* Normalize the sum, checking the overflow or underflow.

0.1111×2^{0}

$0.001_2 \times 2^{-1} = 0.010_2 \times 2^{-2} = 0.100_2 \times 2^{-3} = 1.000_2 \times 2^{-4}$

* Since $127 \geq -4 \geq -126$, there is no overflow or underflow

Step 4:

Rounding:

↳ Round the sum

$\boxed{1.000_2 \times 2^{-4}}$

0.1111

$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$

$\frac{8+4+2+1}{16}$

$\frac{15}{16}$

↳ The sum already fits exactly in 4 bits, so there is no change to the bits due to rounding.

$1.000_2 \times 2^{-4} = 0.0001\underbrace{000}_{1000}{}_2 = 0.0001_2$ (borrowing 2^4)

0.1111×2^{0}

$= 1/2^4_{10}$

$= 1/16_{10}$

$= 0.0625_{10}$

$\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$
0 0 0

$\frac{1}{6}$

This sum is what we expect from adding 0.5_{10} and 0.4375_{10}

7

ii) Multiply the numbers $(0.5)_{10}$ and $(0.4375)_{10}$ using the floating point multiplication. [Nov/Dec -17]

Solution:

* In binary, the task is multiplying

$0.5_{10} = 0.1_2 \times 2^0 = 1.000_2 \times 2^{-1}$

$0.4375_{10} = 0.0111_2 \times 2^0 = 1.110_2 \times 2^{-2}$.

Step1: Calculate the exponent

* Adding the exponents without bias

$1 + (-2) = -1$ $\quad -1 - 2 = -3$

Step2: Multiply the significands

$1.000_2 = 0.5_{10} \rightarrow M1$

$1.110_2 = 0.4375_{10} \rightarrow M2$

$$\begin{array}{r} 1.000_2 \\ \times\ 1.110_2 \\ \hline 0000 \\ 1000 \\ 1000 \\ 1000 \\ \hline 1110000 \end{array}$$

* The product is $1.1100000_2 \times 2^{-3}$, but we need to keep it to 4 bits only, so it is $1.110_2 \times 2^{-3}$

Step3: Normalization

* Now check the product is normalized format and also check the exponent for overflow or underflow.

* The product is already normalized and there is no overflow or underflow.

$127 \geq -3 \geq -126$

Step4: Rounding

* Rounding the product makes no change.

$1.110_2 \times 2^{-3}$

Step 5: Sign

* Signs of the original operands differ, make the sign of the product negative. Hence the product is

$1.110_2 \times 2^{-3}$

* Converting to decimal form to check our results

$1.110_2 \times 2^{-3} = 0.\overset{-3}{00}1110_2 = 0.00111_2$

1.110

$= 7/2^5_{10} = 7/32_{10}$

$= 0.21875_{10}$

* The product of 0.5_{10} and 0.4375_{10} is $= 0.21875_{10}$

0.5×0.4375

2 5

3 5

1 5

0 20

0.21875

8 i) Perform x+y and y-x using 2's complements for given the two binary numbers

x = 0000 1011 1110 1111 and y = 1111 0010 1001 1101

Ans: [Apr/May -18]

Perform x+y

x = 0000 1011 1110 1111

y = 1111 0010 1001 1101

1 1 1 1 1 1 1 0 1 0 0 0 1 1 0 0

Perform y-x

Step1: Find 2's complement of x;

Step2: Add the result with y;

i) Find 2's complement of x:

x = 0000 1011 1110 1111

1's complement of x = 1111 0100 0001 0000

2's complement + 1

x = 1111 0100 0001 0001

ii) Add with y

y = 1111 0010 1001 1101

x = 1111 0100 0001 0001

1 1110 0110 1010 1110

Discard

8ii) Multiply the following Signed 2's complement numbers using the Booth Algorithm. A = 001110 and B = 111001 where A is multiplicand and B is multiplier [APR/MAY-18]

Ans:

A = 001110 ← Multiplicand

B = 111001 ← Multiplier

Recode the multiplier using Booth Algorithm

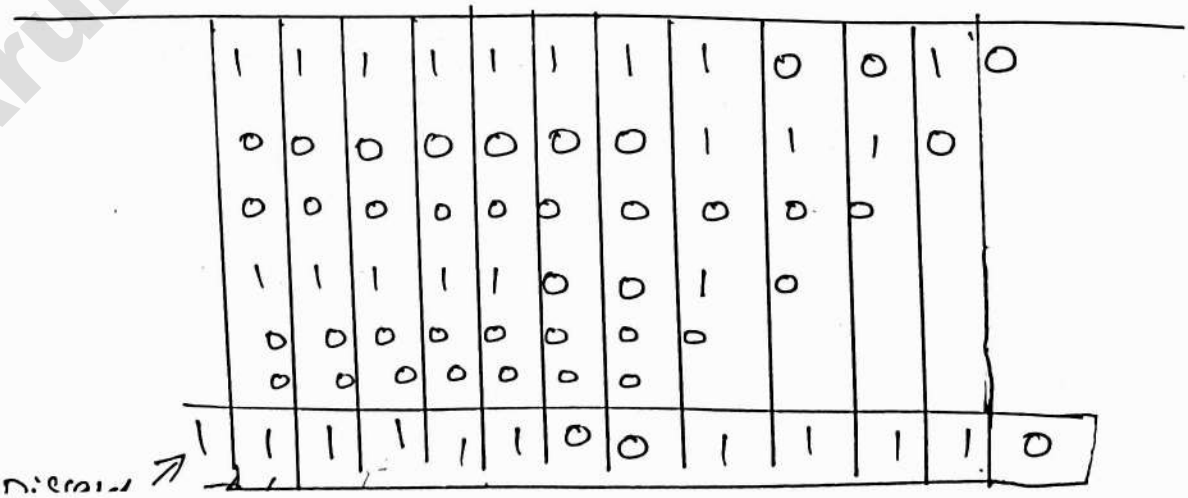
1 1 1 0 0 1 [0] ← Implied zero

1 1 1 0 0 1 0

0 0 −1 0 +1 −1 Recoded multiplier

0 0 1 1 1 0 ×

0 0 −1 0 +1 −1



Discard ↗

9 Multiply the following signed numbers using Booth algorithm. $A = (-34)_{10} = (1011110)_2$ and $B = (22)_{10} = (0\,0\,1\,0\,1\,1\,0)_2$ where B is multiplicand and A is multiplier. [NOV/DEC-18]

Ans:-

$A = (1011110)_2$ Multiplier

$B = (0010110)_2$ Multiplicand

* Recode the multiplier using Booth's algorithm
* Before recoding the multiplier add the implied zero to the immediate right of LSB.

(ie) A is multiplier

1 0 1 1 1 1 0 [0] ← Implied zero

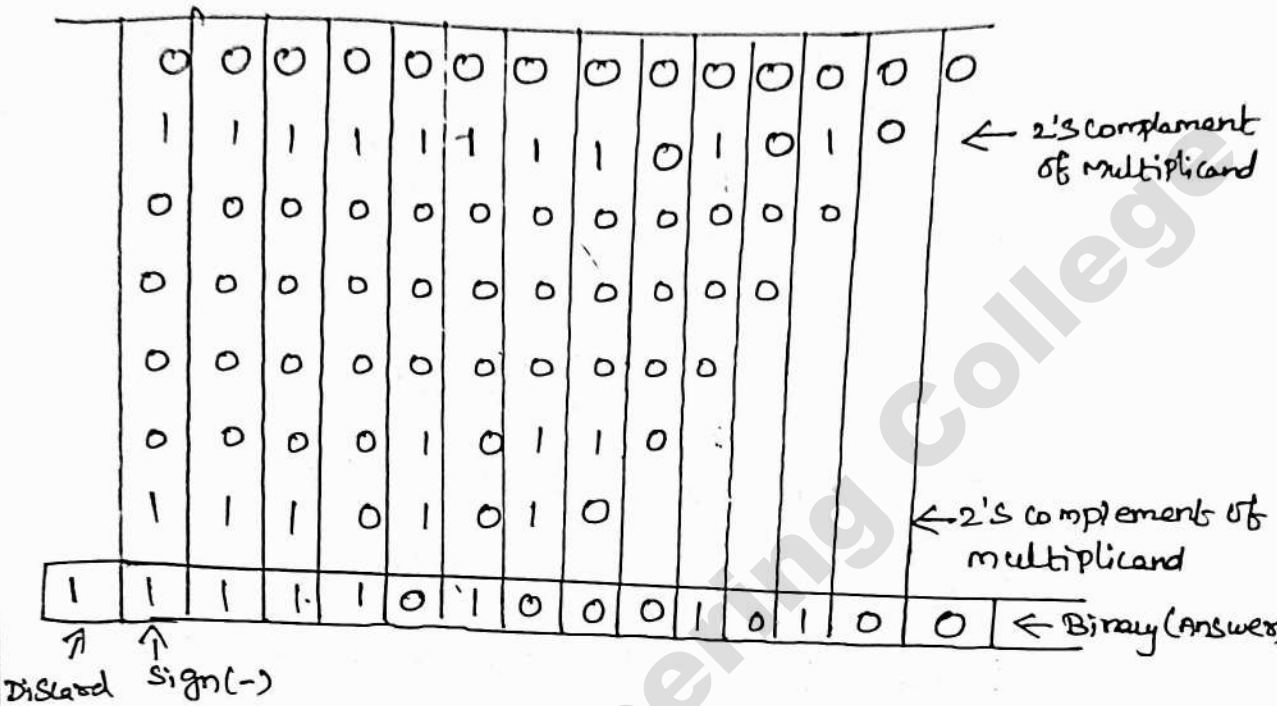
Using the recoding table we can recode the multiplier with implied zero as follows

1 0 1 1 1 1 0 0

−1 +1 0 0 0 −1 0 ← Recoded multiplier

0 0 1 0 1 1 0 ← Multiplicand

× −1 +1 0 0 0 −1 0 ← Recoded Multiplier



$= -(0100010100)_2$

↓

$-\ 548_{10}$

10 Explain floating point addition algorithm with a neat block diagram. [NOV/DEC-18]

Floating point addition:

Step1: Exponent of the two floating-point numbers must be made equal. To do this, compare the exponents of the two numbers.

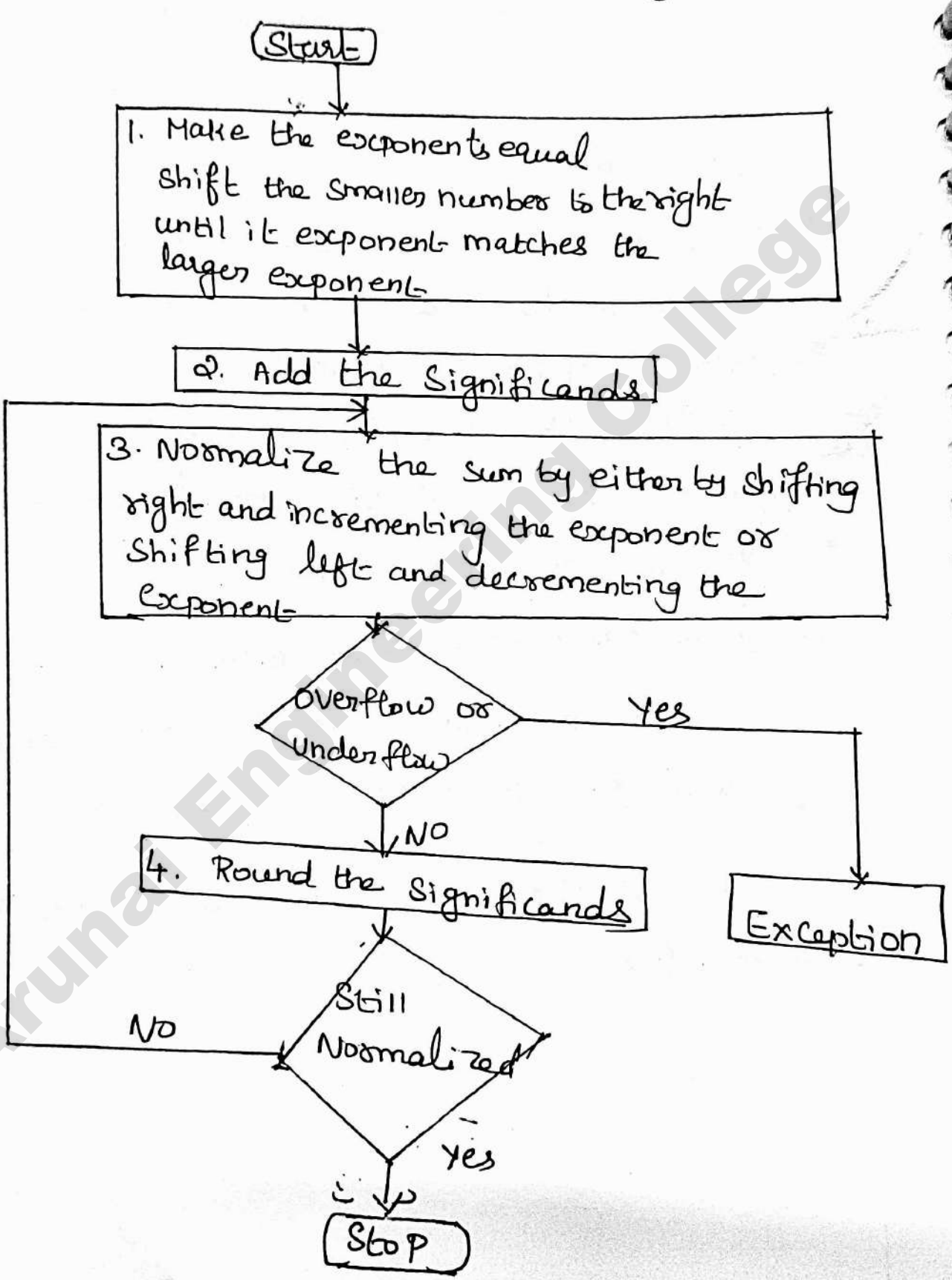
* Shift the small number to the right until its exponent is matched with the larger exponent

Step 2: Add the two significands.

Step3: Normalize the sum, either by shifting right and incrementing the exponent or shifting left and decrementing the exponent.

Step4: Round the significand to the appropriate number of bits.

Floating-Point Addition Algorithm



Perform Floating-point addition on the following two decimal numbers

$$9.999_{10} \times 10^{1} +$$
$$1.610_{10} \times 10^{-1}$$

Assume that we can store only four decimal digits of the significand and two decimal digits of the exponent.

Answer:

Step1: Equalize Exponents

* Adjust the smaller number, the second number is the smaller. Thus, the first step shifts the significand of the smaller number to the right until its corrected exponent matches that of the larger number.

* But we can represent only four decimal digits so, after shifting, the number is

$$1.610 \times 10^{-1} = 0.161 \times 10^{0} = 0.016 \times 10^{1}$$

Step 2: Add significands

The next step is addition of the significands

$$\begin{array}{r} 9.999_{10} \\ (+)\ \underline{0.016_{10}} \\ 10.015_{10} \end{array}$$

The sum is $10.015_{10} \times 10^{1}$

Step3: Normalization:

This sum is not in normalized scientific notation, so we need to adjust it.

$10.0015_{10} \times 10^1 = 1.0015_{10} \times 10^2$

Thus, after the addition we may have to shift the sum to put it into normalized form, adjusting the exponent appropriately.

Step4: Rounding

The number, $1.0015_{10} \times 10^2$ is rounded to four digits in the significand to

1.002×10^2

UNIT - III

PART - A

1. Mention the various types of pipelining [NOV/DEC-2017]

* The various types of pipelining are

1. Arithmetic pipeline - The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats.

2. Instruction pipeline - An instruction pipeline increases the performance of a processor by overlapping the processing several different instructions. This is done by dividing the the instruction execution process into several stages like fetch, decode, execute, Memory access and write back stages

---

2. Mention the various phase in executing an instruction. [NOV/DEC-2017]

The various phase in executing an instruction, they are

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.

---

3. Name the control signals required to perform arithmetic operations. [APR/MAY-2017]

* Two ALUop signals
* Seven other signals

> Reg DST - which field for write register
> Reg write - write to Register file
> ALUSrc - Source for second ALU input
> PC Src - Source for PC (PC+4 or target address)
> Mem Read - Read input address from memory
> Memwrite - write input address/data to memory
> Mem To Reg - Source of write register port data input.

* Branch control signal - Set when instruction is branch

4. Define hazard. Given an example for data hazard. [APRIMAY-2017]

Hazard:

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. Data Hazards 2. Structural Hazards
3. Control Hazards

Data Hazards:

* when a planned instruction cannot execute in the proper clock cycle because data (i.e) needed to execute the instruction is not yet available.
* This type of hazard is called Data Hazard

Eg: $A \leftarrow 3 + A$ Assume $A = 5$

$B \leftarrow 4 \times A$

5. what is meant by Pipeline bubble? [NOV/DEC-2016]

* A bubble or pipeline stall is a delay in execution of an instruction in an instruction pipeline in order to resolve a hazard.

* The values are preserved until the bubble has passed through the execution stage.

6. what is a data path? [NOV/DEC-2016]

* A datapath is a collection of functional units such as ALU, multipliers that perform data processing operations, registers and buses.

7. What is a hazard? what are its types [NOV/DEC-2015]

* In pipelining there are situations when the next instruction cannot execute in the following clock cycle. These events are called hazard

* There are three hazards present in the pipelining such as

1. Structural hazard
2. Data hazard
3. Control hazard

---

8. what is meant by branch prediction?

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

* A simple form of branch prediction is we have to assume branch is not taken.

* This assumption is only possible in five stage pipeline. for deeper pipelines this assumption not suitable, it increases branch penalty.

9. What is Exception? [Nov/Dec - 2014] [May/June-2016]

* Exceptions are internally generated unscheduled events that disrupts program execution and they are used to detect overflow.

---

10. What is a branch prediction buffer? [May/June-2015]

* Dynamic branch prediction is a prediction of branches using runtime information. To implement dynamic branch prediction method we have to use one buffer is called branch prediction buffer. It is also called branch history table.

* This buffer is small memory that is indexed by the lower portion of the address of the branch instruction.

* It also contain one or more bits indicating whether the branch was recently taken or not.

---

11. What are R-Type Instructions? (May-2015)

* The arithmetic logic instructions read operands from two registers, perform an ALU operation on the contents of the registers and write the result to a registers we call these instructions are R-type instructions

* This R-type instruction class includes add, sub, AND, OR and SLT

Eg OR $t1, $t2, $t3 reads $t2 and $t3, performs logical OR operation and saves the result in $t1

---

12 what are the advantages of Pipelining? [May/June-2016]

1. The advantage of pipelining is, it saves time in the execution of Instruction, because it uses overlapping of Instruction

2. TO Increase the speed and performance of the processor.

---

3 what is meant by 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.

---

14. How many stages available in 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.

---

15 Define Data hazard

* Data hazard occur when a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.

* Data hazard mainly occur in executing arithmetic instructions

---

16. How to resolve a hazard?

* Pipeline Stall also called bubble. A Stall Initiated in order to resolve a hazard

---

17 What is an exception? Give one example for MIPS exception. [NOV/DEC-18] [APR/MAY-18]

* Exceptions are also called as interrupt. It is an unscheduled event that disrupts program execution It is also used to detect an overflow condition.

Example:

i) Arithmetic overflow
ii) Undefined Instruction
iii) Invoke the OS from user program.

18 write the two steps that are common to implement any type of instruction. [NOV/DEC-18]

For every Instruction, the first two steps are identical

1) Send the Program Counter(PC) to the instruction memory and fetch the instruction from that memory.
2) Read one or two registers, using fields of the instruction.

19 What is the ideal CPI of a pipelined processor? [APR/May-18]

The Ideal CPI of a pipelined processor almost 1.

PART-B

Explain how the instruction pipeline works? What are the various situations where an instruction pipeline can stall? Illustrate with an Example [NOV/DEC-15] [NOV/DEC-16]

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.

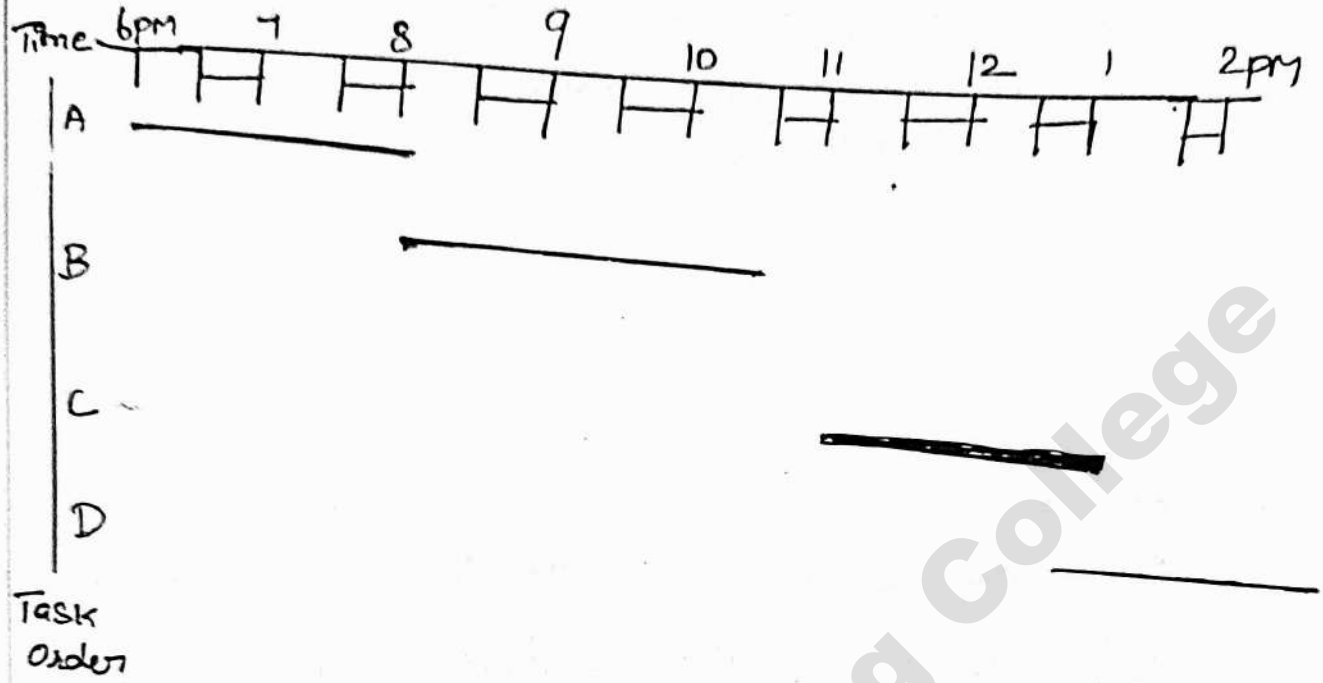
Example:

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

Case1: Non-pipelined approach to laundry

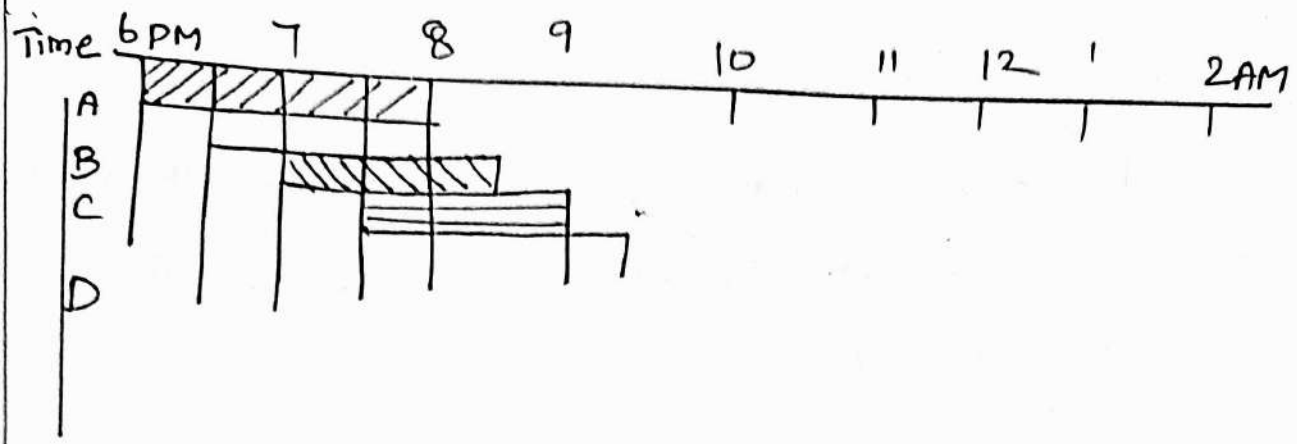
1. place one dirty load of clothes in the washer
2. when the washer is finished, place the wet load in the dryer
3. when the dryer is finished, place the dry load on a table and fold
4. when folding is finished, ask your roomate to Put the clothes away.

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 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 3x 800 or 2400 PS

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

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

* 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.

2. Explain the different types of pipeline hazards with suitable Example. [May - 2015]
Explain the pipeline hazard in detail [NOV/DEC - 2017]

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.

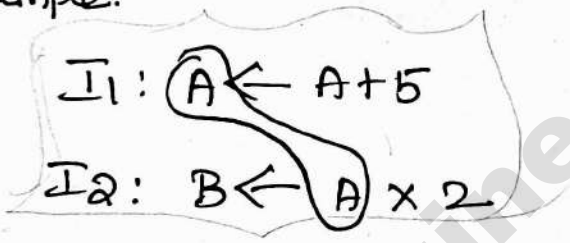
2. If our roommates doing something else without put clothes away.

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:



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.

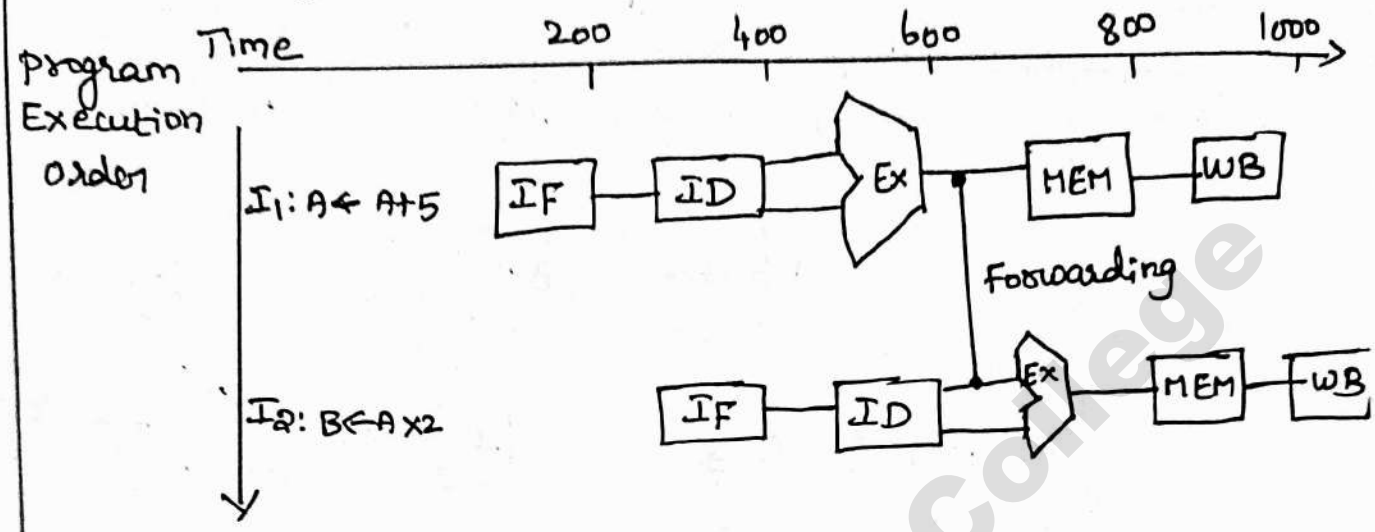
Generally in MIPS instruction set has five stages for pipelining process.

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.

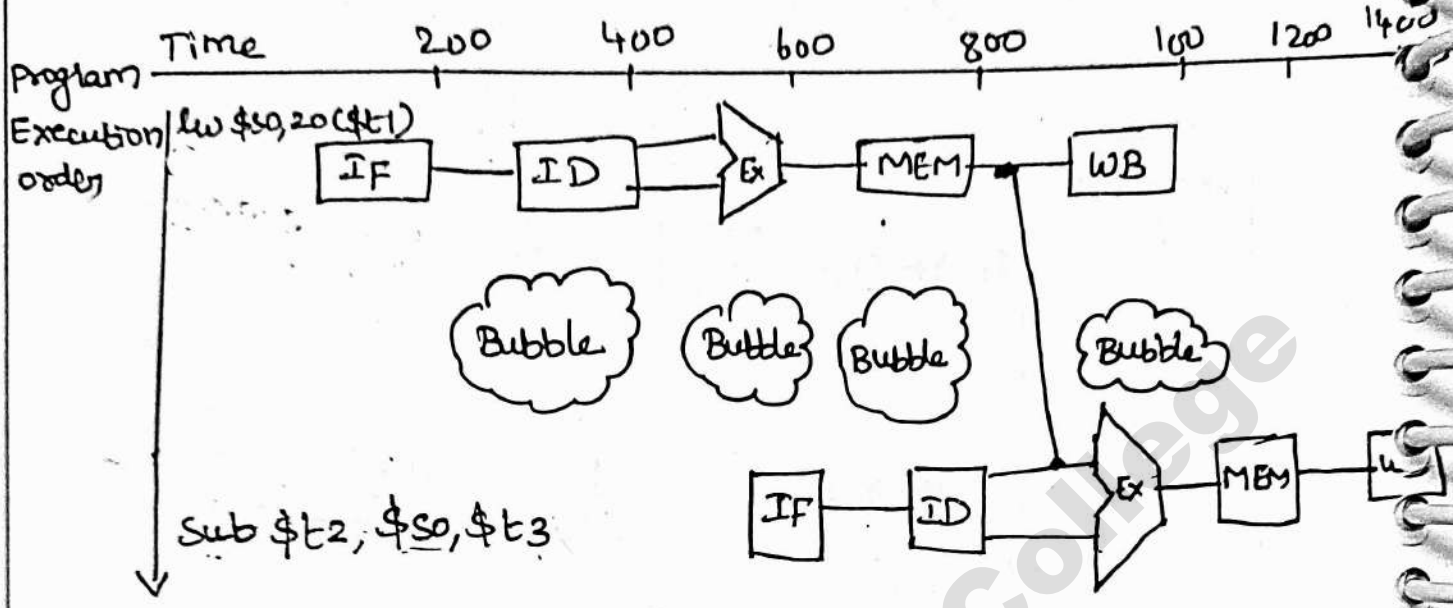
* It is a specific form of data hazards in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.

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 hazards 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.

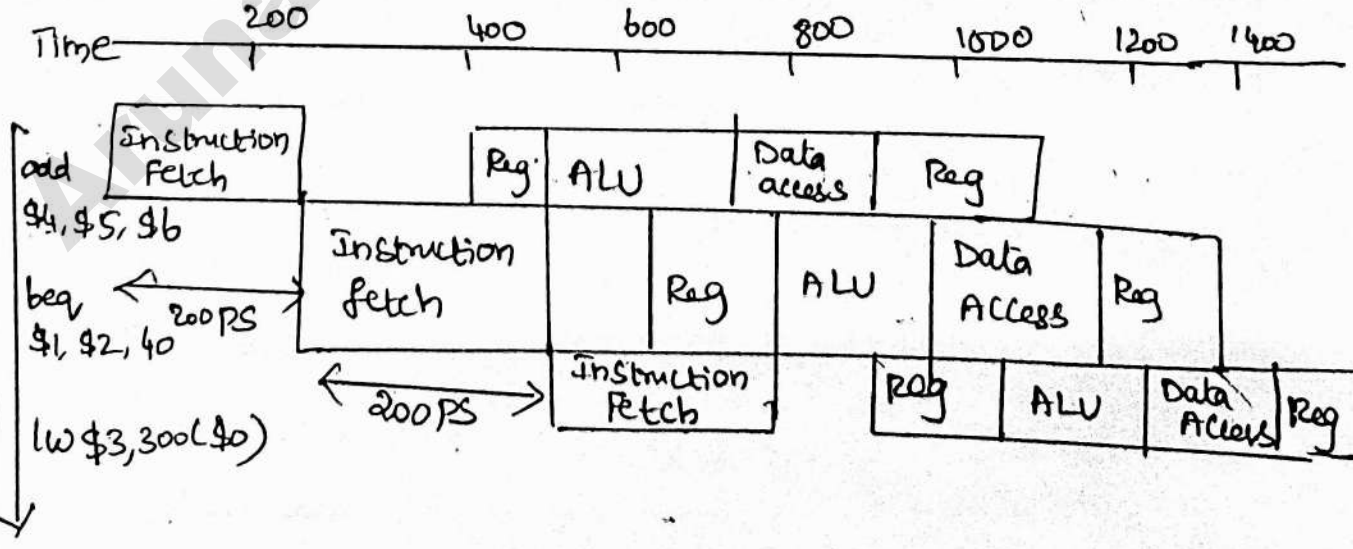
* The pipeline cannot possibly know what the next instruction should be, because it only received the branch instruction from memory.

* 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.

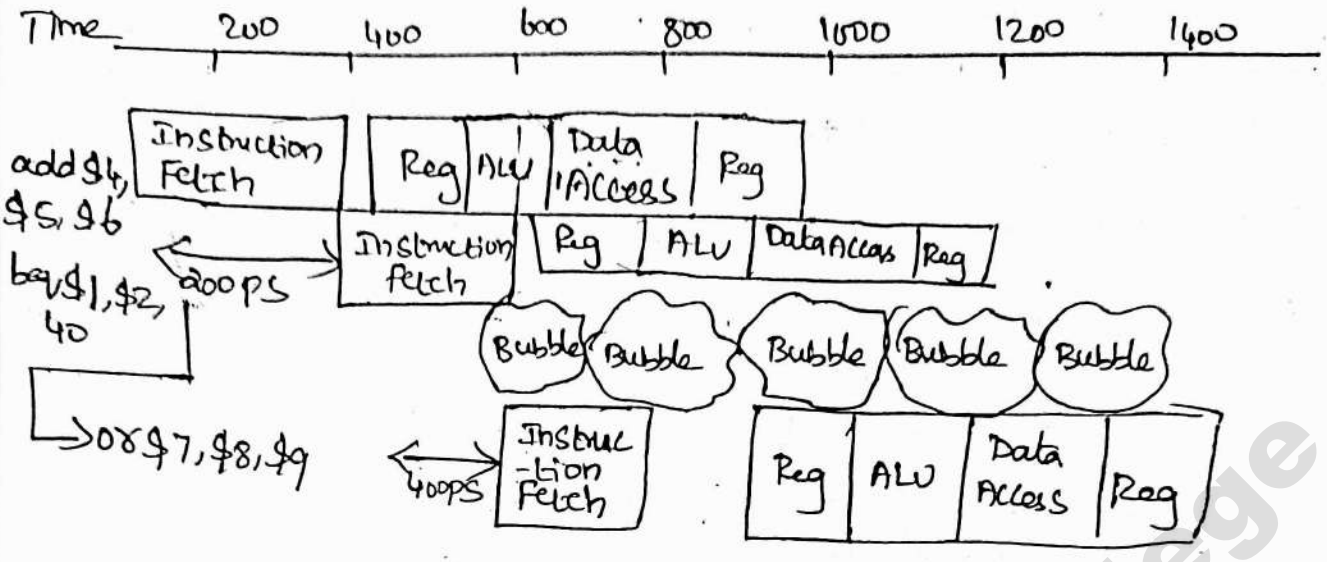
Branch prediction:

* 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:

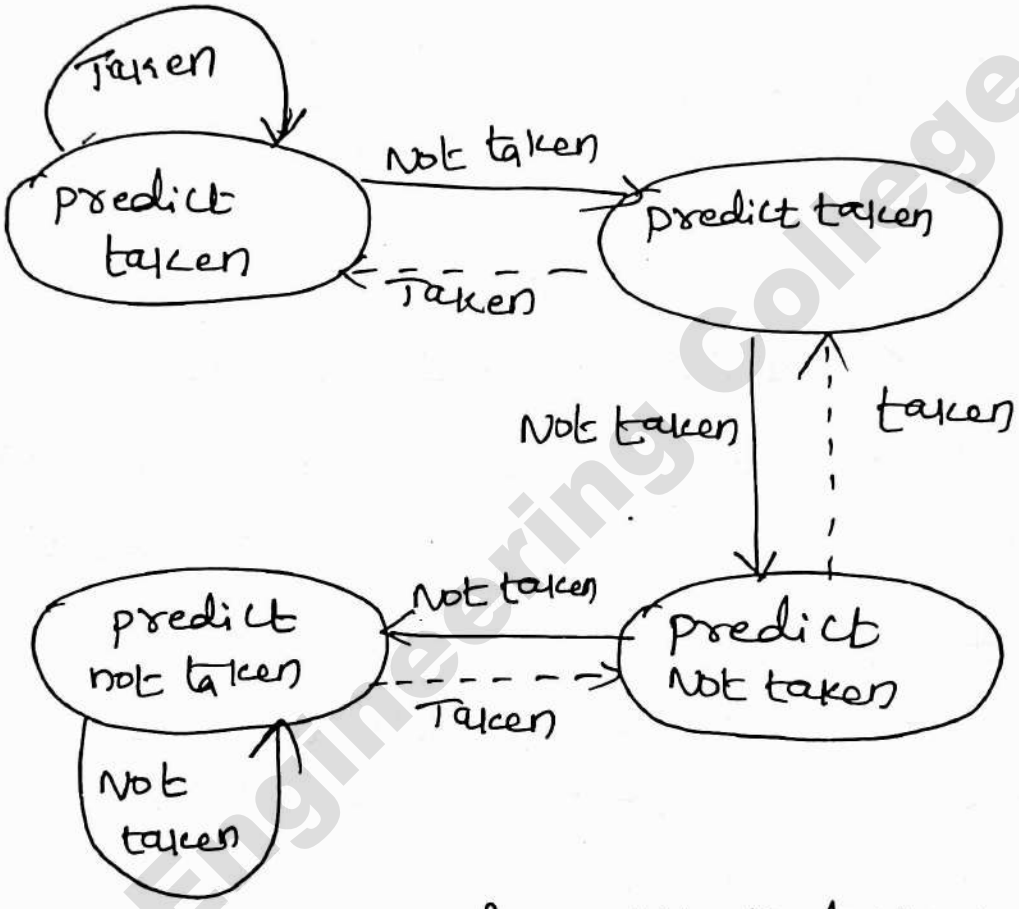
1. It increases the number of simultaneously executing instructions

* 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

1. Explain in detail how exceptions are handled in MIPS architecture? [APR/MAY-15]

* 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
Invoke the operating system from user program	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware 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 it leads complex task to getting the correct design.

Exceptions in MIPS Architecture.

In MIPS architecture two kinds of exceptions occur at

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

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

* 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 implementation.

Implementation of exception in MIPS architecture:

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

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

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

* Two addition registers are

i) EPU 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 aricitecture this register is 32 bits long and some bits are currently unused.

* 10 bits are used to represent an undefined instruction and 12 bits used to represent asithmetic overflow

5. Explain the basic MIPS Implementation with necessary multiplexers and control lines. [NOV/DEC-2015]

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 calculation
  ↳ 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 registers.

* 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

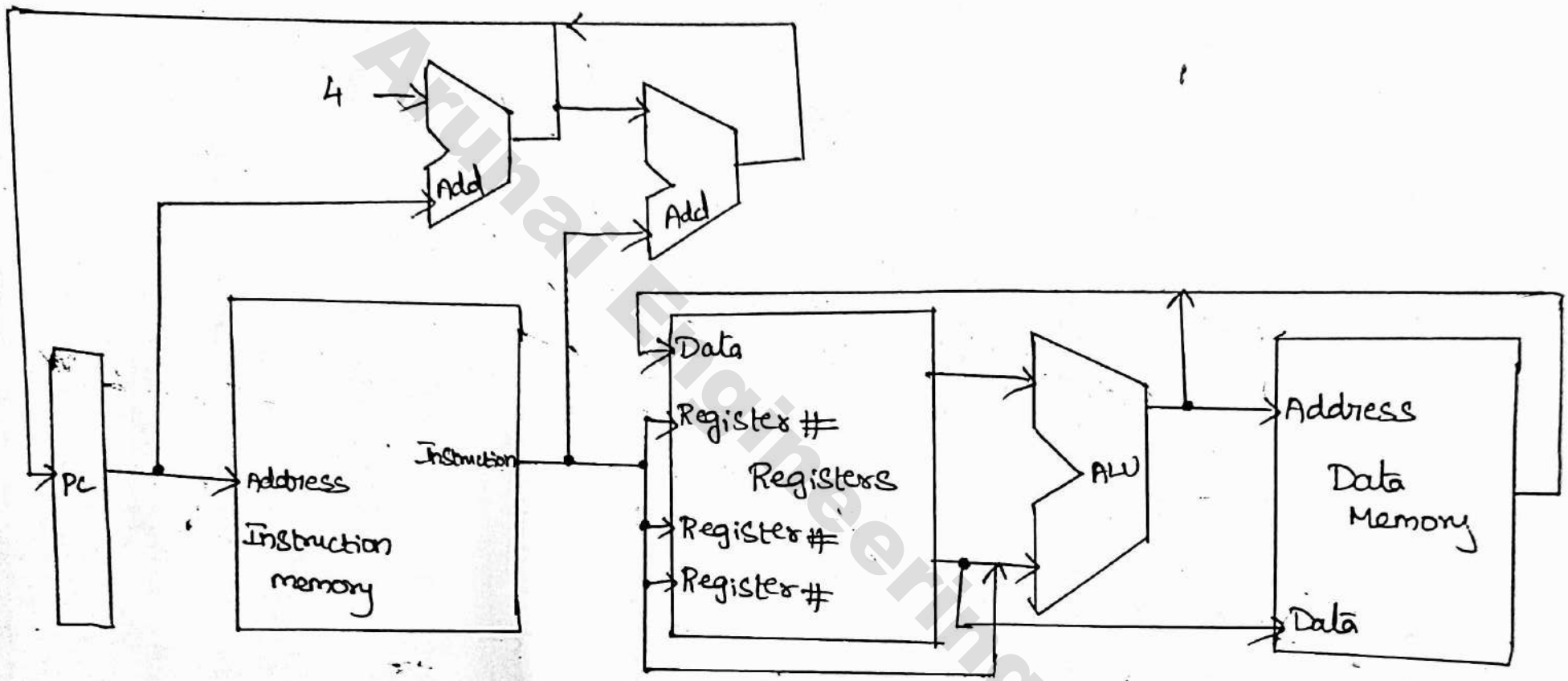
Problems in first aspect:

* 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 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

Thick lines - 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

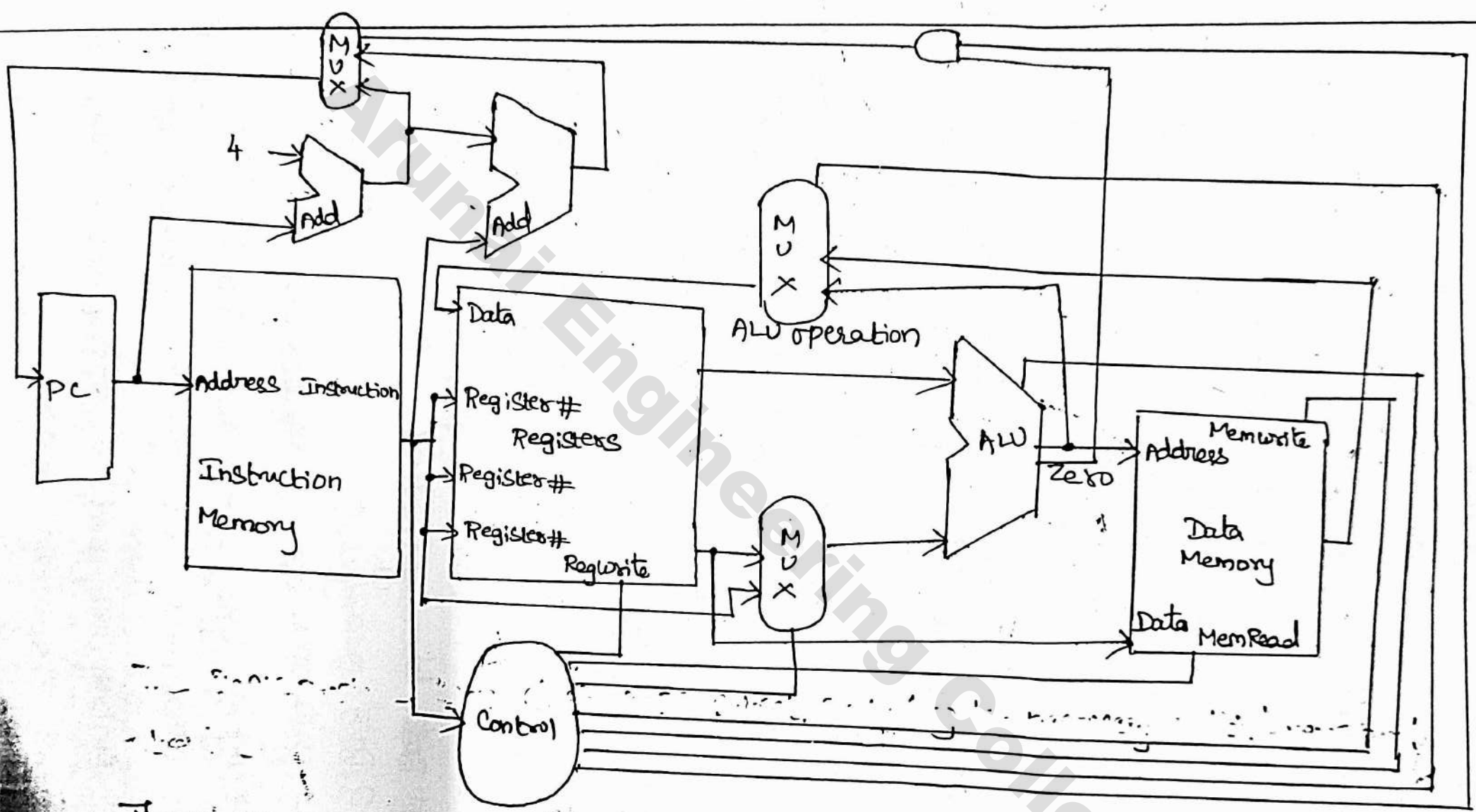
Function of third Multiplexer:

* 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 Contra 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.

6. Explain in detail the operation of the datapath.

[NOV/DEC-17]

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

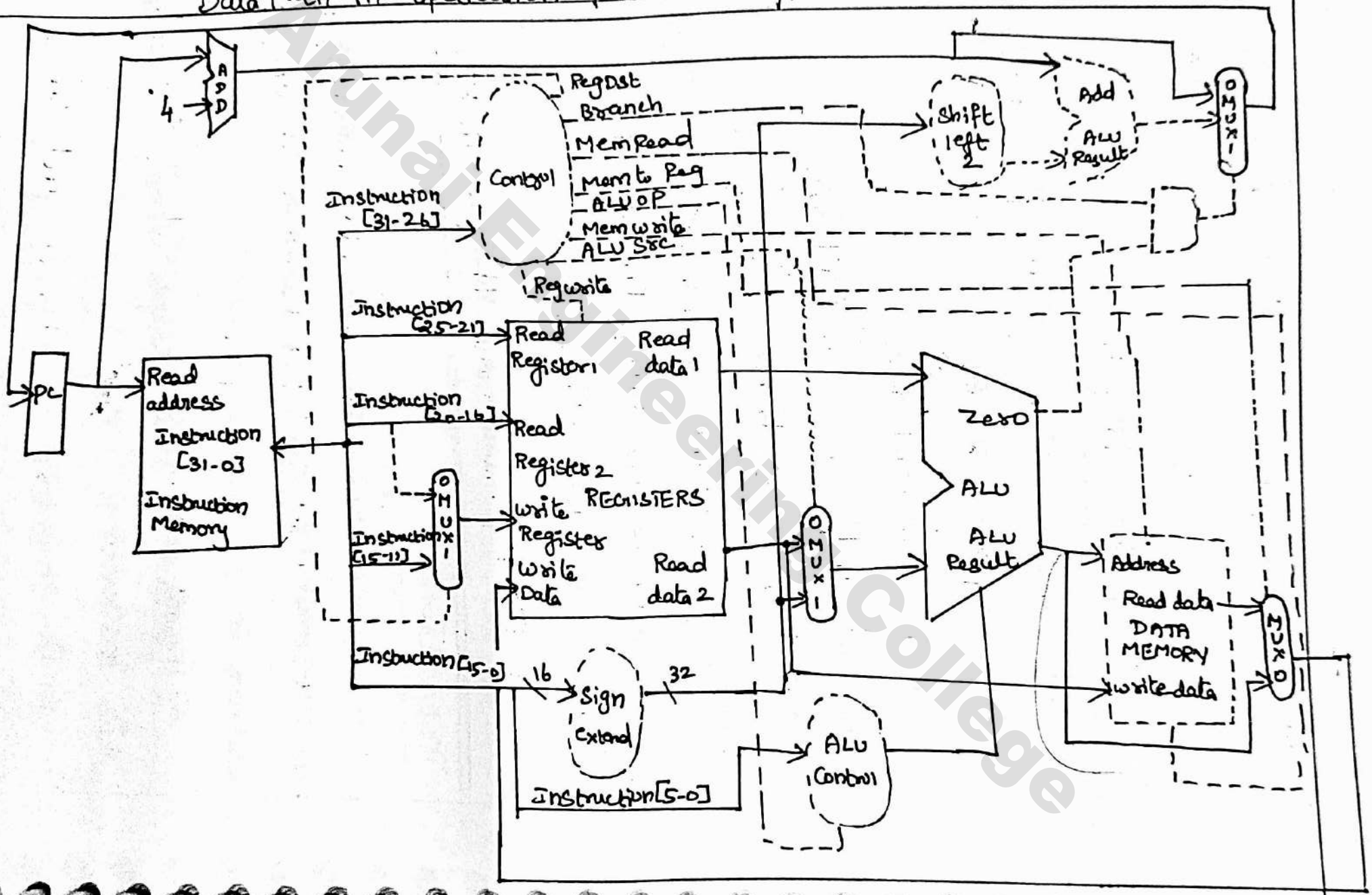
1. Operation of the datapath for an R-type instruction:

* 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.

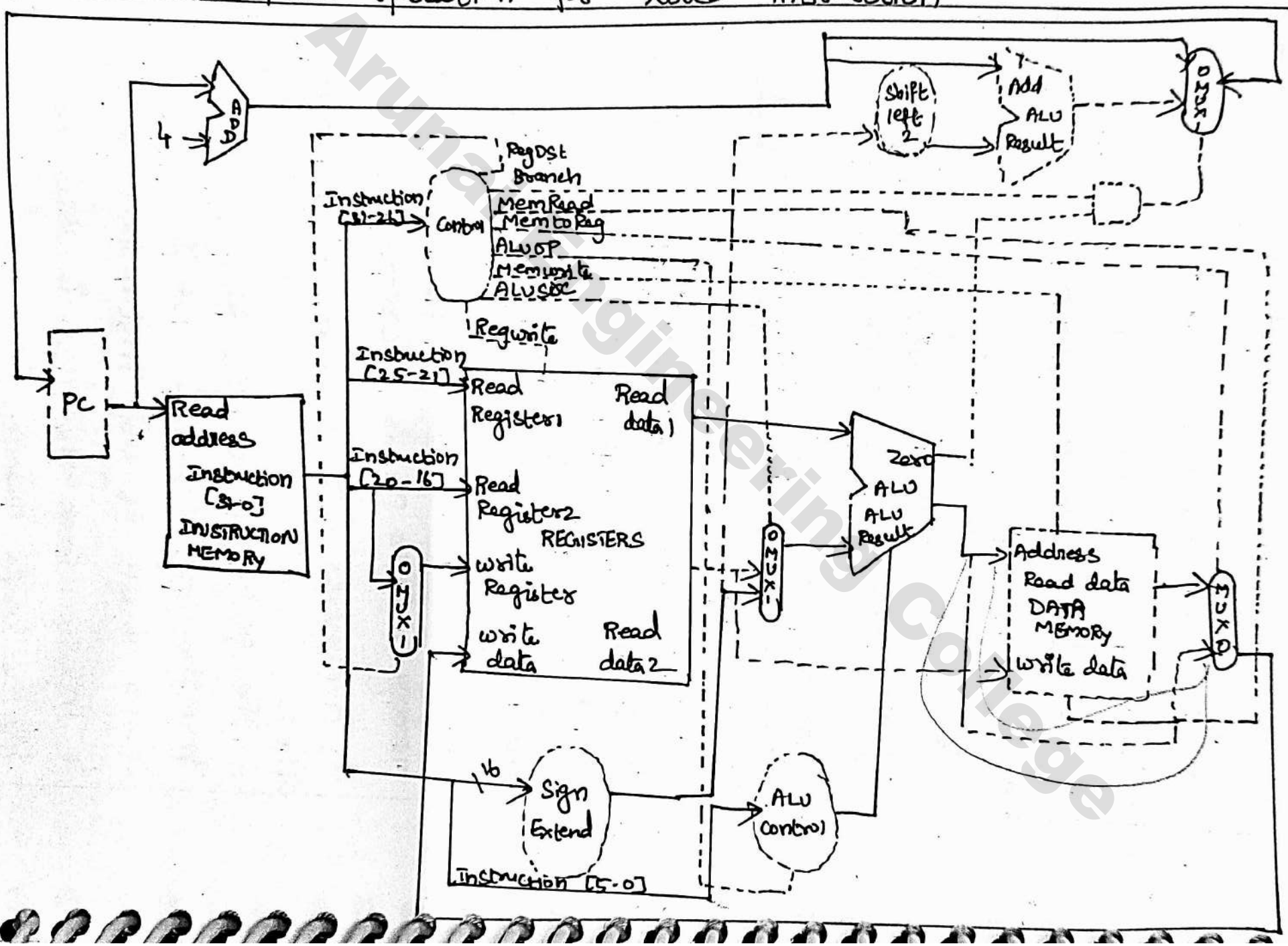
1. An instruction is fetched from the instruction memory and the PC is incremented.

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.

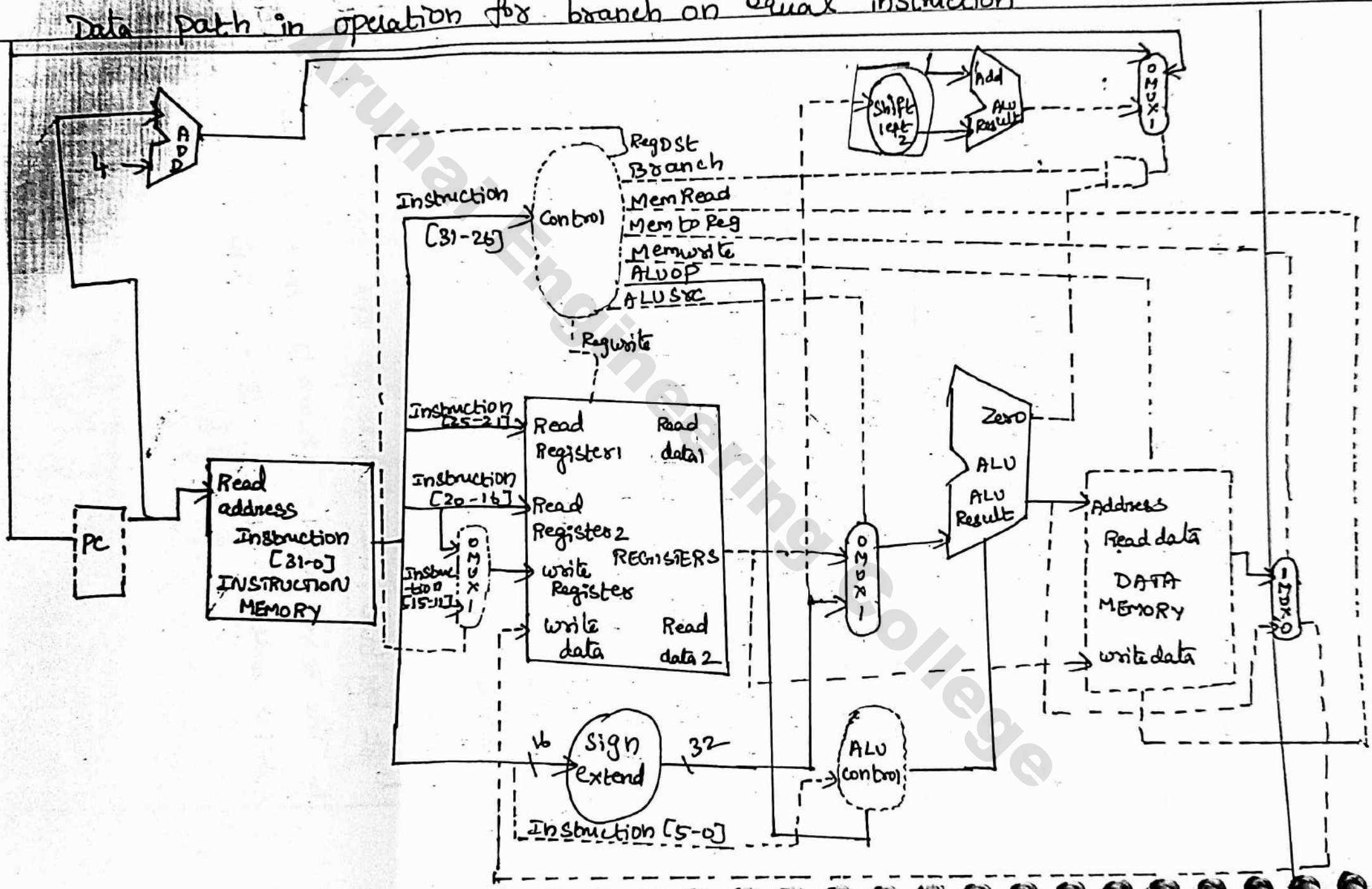
* Instructions such as `beq $t1, $t2, offset` comes under this category. Four steps are needed to execute the instruction. They are

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



7) The following sequence of instructions are executed in the basic 5-stage pipelined processor: [Apr/May-18]

OR $R1, R2, R3$

OR $R2, R1, R4$

OR $R1, R1, R2$

a) Indicate dependences and their type.

b) Assume there is no forwarding in this pipelined processor. Indicate hazards and add NOP instructions to eliminate them.

c) Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

Answer:

a) Indicate dependences and their type

Instructions:

I_1: OR $R1, R2, R3$

I_2: OR $R2, R1, R4$

I_3: OR $R1, R1, R2$

i) True Data dependency:

* Instruction I_2 and I_1 have true data dependency because I_2 can be executed only when I_1 is completed (i.e) I_2 needs $R1$ which is available only when I_1 is completed.

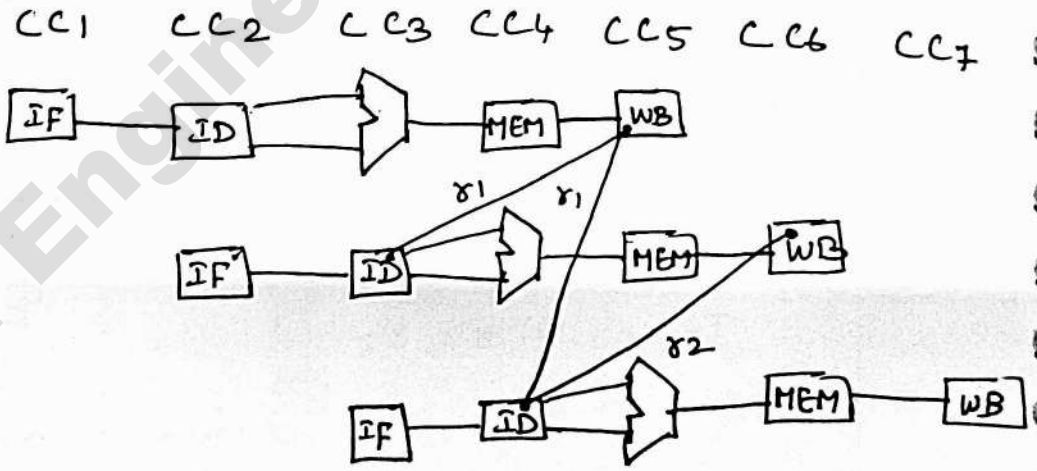
* Similar kind of data dependency exists between I_1 and I_3, I_2 and I_3.

2. output dependency:

* Instruction I_1 and I_3 are having output dependency, because I_3 cannot be executed before I_1.

* No Forwarding mean data hazard occurs.

b) Assume there is no forwarding in this pipelined processor. Indicate hazards and add Nop instructions to eliminate them.



* As in above figure, R1 is available only at CC5, but instructions I_3 need R_1 at CC3 and CC4. This is a situation of data hazard.

* Nop instruction can be inseted to avoid data hazard.

* In instruction I_2, after instruction fetch (IF) ID stage is delayed until CC5, so that $r1$ will be available at CC5.

* Similar case exist between I_2 and I_3

I_1: OR $r1, r2, r3$

I_2: NOP

I_3: NOP

I_4: OR $r2, r1, r4$

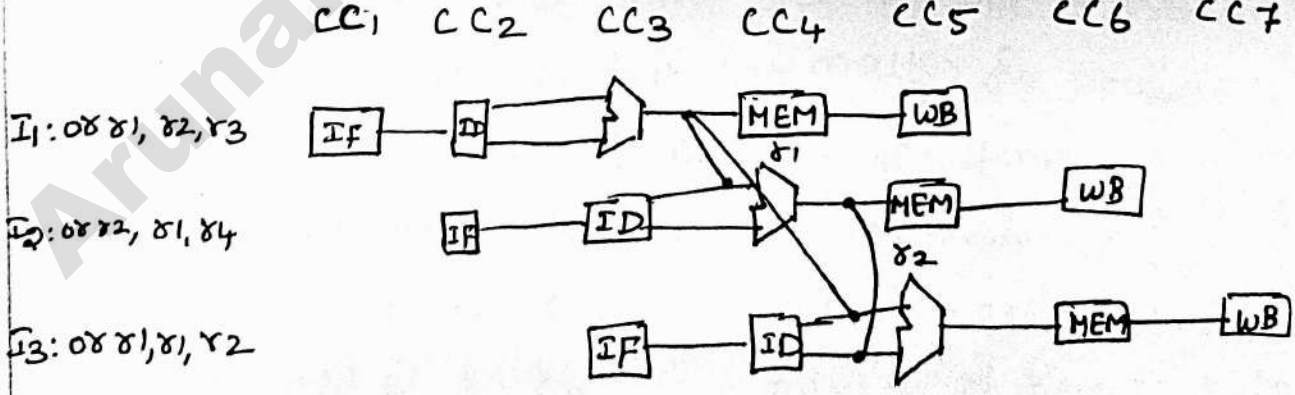
I_5: NOP

I_6: NOP

I_7: OR $r1, r2, r2$

c) Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

Forwarding:-



* when forwarding in applied, there is no need of NOP instruction.

* R_1 is forwarded to instructions I_2 and I_3 at CC_3

* Similarly, r_2 is forwarded to I_3 at CC_4.

8) The following sequence of instructions are executed in the basic 5-stage pipelined processor. [NOV/DEC-18]

```
lw  $1, 40($6)
add $6, $2, $2
sw  $6, 50($1)
```

Indicate dependences and their type Assuming there is no forwarding in this pipelined processor, indicate hazards and add Nop instructions to eliminate them.

Answer:

* Dependencies involve writing registers. There are three possibilities:

i) RAW - read after write
ii) WAR - write after Read
iii) WAW - write after write

The second and third type are not a problem for simple pipelines, but can matter for processors which permit out-of-order execution

Instruction sequence	Dependences
I_1: lw $1, 40($6)	RAW on $1 from I_1 to I_3
I_2: add $6, $2, $2	RAW on $6 from I_2 to I_3
I_3: sw $6, 50($1)	WAR on $6 from I_1 to I_2 and I_3

* In the basic five-stage pipeline WAR and WAW dependences do not cause any hazards.

* Without forwarding, any RAW dependence between an instruction and the next two instructions (if the register read happens in the second half of the clock cycle and the register write happens in the first half)

* The code that eliminates these hazards by inserting nop instruction is

Instruction sequence	
I1: lw $1, 40($6)	
I2: add $6, $2, $2	
nop	Delay I3 to avoid RAW
I3: sw $6, 50($1)	hazard on $1 from I1

A processor has five individual stages, namely IF, ID, EX, MEM and WB and their latencies are 250ps, 350ps, 150ps, 300ps and 200ps respectively. The frequency of the instructions executed by the processor are as follows; ALU: 40%, Branch 25%, Load: 20% and Store: 15%. what is the clock cycle time in a pipelined and non-pipelined processor? If you can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor? Assuming there are no stalls or hazards, what is the utilization of the data memory? Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

i)
For a pipelined processor, the clock cycle is the time of the pipeline element with the largest latency $T_{clk} = 350PS$

For a non-pipelined processor, the clock cycle is the sum of the latencies of all the pipeline elements $T_{clk} = 250+350+150+300+200$

$T_{clk} = 1250PS$

ii) Split the longest stage : ID

The next longest stage is : MEM at 300 ps.

$T_{clk} = 300PS$

UNIT - IV

PART-A

1) What is Flynn's Classification? [NOV/DEC-2014]

* Flynn's classification divides parallel hardware into four groups based on the number of instruction streams and the number of data streams

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

---

2. Brief about Multithreading (NOV/DEC-2014)

* The term multithreading implies that there are multiple threads of control in each processor.

* Multithreading offers an effective mechanism for hiding long latency in building large scale multiprocessors.

---

3. What is ILP? [NOV/DEC-2015]

Instruction level parallelism is a kind of Parallelism it executes the instruction in parallel way

---

4. Define a super scalar processor [NOV/DEC-2015]

- Super scalar processor is a dynamic multiple Issue processor. It is an advanced technique that enables the processor to execute more than one instruction per clock cycle by selecting them during Execution

---

5. What is speculation?

An approach that allows the compiler or the process to "guess" the outcome of an instruction to remove it as a dependence in executing other instruction is called speculation

---

6. what is the need for speculation [NOV/DEC-2015]

It is the most important methods for finding and exploiting more ILP

---

7. Differentiate between Strong Scaling and weak Scaling. [APR/MAY-2015]

Strong Scaling	weak Scaling
1. In this method speed up is achieved on a multiprocessor without increasing the size of the problem * It means measuring Speed Up while keeping the Problem size fixed.	1. 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.

8. Compare UMA and NUMA Multiprocessors [May/June-2015]

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

9. Define Synchronization

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

10. What is Fine grained Multithreading? [May/June-2016]

Fine grained Multithreading is a version of hardware multithreading that implies switching between threads after every instruction.

11. What is Coarse grained Multithreading?

Coarse grained Multithreading is a version of hardware multithreading that implies switching between thread only after significant events such as last level cache miss

12. What is meant by thread?

Thread is a light-weight process which includes the program counter, the register state and stack. It shares a single address space.

13. What is Multi threading ? [NOV/DEC-2016]
[NOV/DEC-2014]

* Multithreading Allows multiple threads to share the functional units of a single processor in an overlapping fashion in order to utilise the hardware resources Effectively.

14. What is Instruction level parallelism ?
[APR/MAY-2017]
[NOV/DEC-2016]
[MAY/JUNE-2016]

* Instruction level parallelism (ILP) is a measure of number of instructions that can be performed simultaneously during a single clock cycle.

* The potential overlap among instruction is called as Instruction level parallelism.

* There are two primary methods for increasing the potential amount of instruction level parallelism

1. Increasing the depth of the pipeline to overlap more instructions
2. Multiple issue.

15 Define Strong scaling and Weak scaling. [NOV/DEC-2017]

Strong scaling:

* In these methods speed up achieved on a multiprocessor without increasing the size of the problem.

" Strong scaling means measuring speed up while keeping the problem size, fixed".

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.

16 Difference between Fine-grained multithreading and coarse-graired Multithreading. [NOV/DEC-2017]

Fine-grained Multithreading	Coarse-grained Multithreading
1. Fine-grained Multithreading is a version of hardware multithreading that implies switching between threads after every instruction.	Coarse-grained Multithreading is a version of hardware multithreading that implies switching between thread only after significant events such as last-level cache miss.

17 Distinguish Implicit multithreading and Explicit Multithreading. [APR/MAY-2017]

Implicit Multithreading	Explicit Multithreading
1. Implicit Multithreading is concurrent Execution of multiple threads extracted from single sequential program.	1. Explicit Multithreading is concurrent Execution of instructions from different explicit threads.
2. Implicit threads defined statically by compiler or dynamically by hardware	2. Either by interleaving instructions from different threads on Shared Pipelines or * by parallel execution on Parallel Pipelines

18 Web server is to be enhanced with a new cpu which is 10 times faster on computation than old cpu. The original cpu spent 40% of its time processing and 60% of its time waiting for I/O. what will be the overall speed up? [NOV/DEC-18]

$$\text{Speedup} = \frac{\text{Execution time for task without enhancement}}{\text{Execution time for task with enhancement}}$$

$$\text{Speedup} = \frac{\text{Execution time old}}{\text{Execution time new}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}} \rightarrow (1)$$

$\text{Fraction}_{\text{enhanced}} = 40\% = 0.4$

$\text{Speedup}_{\text{enhanced}} = 10$

Using formula (1)

$$\frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

19 Classify shared memory multiprocessor based on the memory access latency. [NOV/DEC-18]

* Shared memory multiprocessor based on the memory access latency

i) Uniform Memory Access (UMA)

* The same time to access main memory no matter which processor requests it and no matter which word is requested, such machine are called UMA multiprocessor.

ii) Non-uniform memory Access (NUMA)

* Some memory access are much faster than others, depending on which processor asks for which word. such machines are called NUMA multiprocessors

20 Give example for each class in Flynn's Classification.
[Apr/May -18]

i) Single Instruction Single Data streams (SISD)
Eg Intel Pentium 4

ii) Single Instruction Multiple Data streams (SIMD)
Eg SSE of Instructions of X86

iii) Multiple Instruction Single Data streams (MISD)
No Examples

iv) Multiple Instruction Multiple Data streams (MIMD)
Eg: Intel Core i7

21 Protein string matching code has 4 days execution time on current machine doing integer instructions in 20% of time, doing I/O in 35% of time and other operations in the remaining time. which is the better trade off among the following two proposals? First: Compiler optimization that reduces number of integer instructions by 25% (assume each integer instruction takes the same amount of time); second: Hardware optimization that reduces the latency of each IO operations from 6us to 5us.

Ans:

Second proposal: Hardware optimization that reduces the latency of each IO operations from 6 us to 5 us is the best, because 35% of the time is consumed by I/o operations whereas integer instructions takes only 20% of the time.

Further time take for executing integer instructions is very minimum when compared to time taken for an I/o operation.

Hence, the second is the best proposal.

PART-B

Explain in detail about Flynn's classification of Parallel hardware. [Nov/Dec, 2016, 2015]

Discuss about SISD, MIMD, SIMD, SPMD and VECTOR System [May-15] [Apr/May-17]

Explain with diagrammatic illustration Flynn's classification [Nov/Dec-17]

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 Ins. No. of datastream

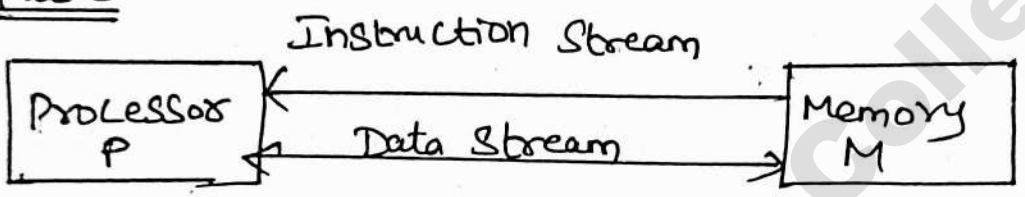
i) Single Instruction Stream - Single Data Stream (SISD)

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

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

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

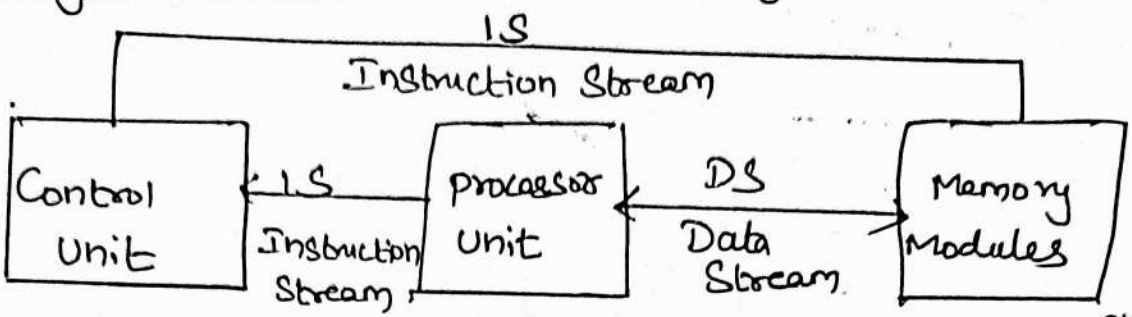
Instruction and data streams in a sequential Computer



* 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.

1. Single Instruction Stream Single Data Stream (SISD)



SISD Computer. Von Neumann computers. uniprocessor

* 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).

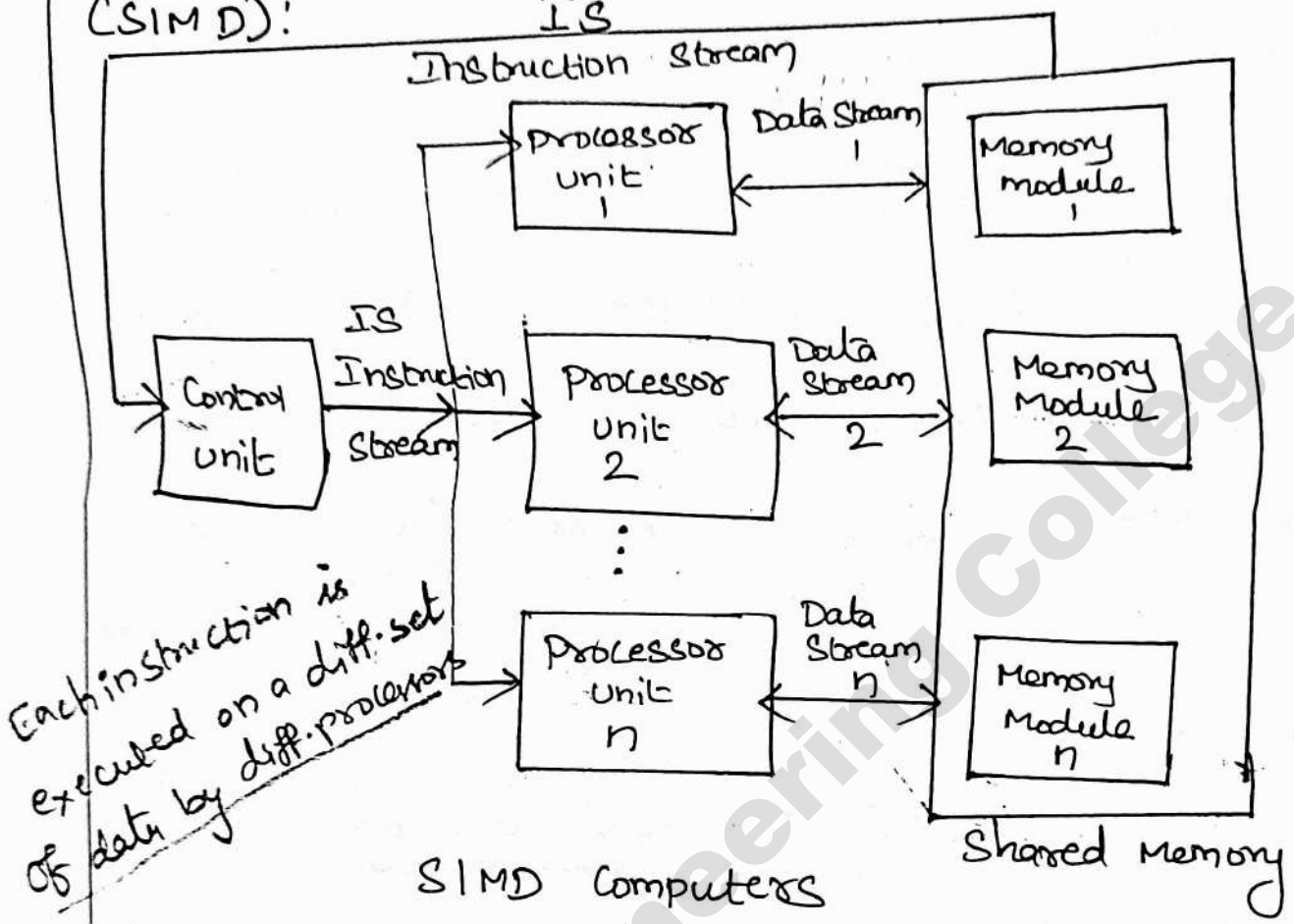
* They may have more than one functional unit, but all functional units are control by a single control unit.

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.

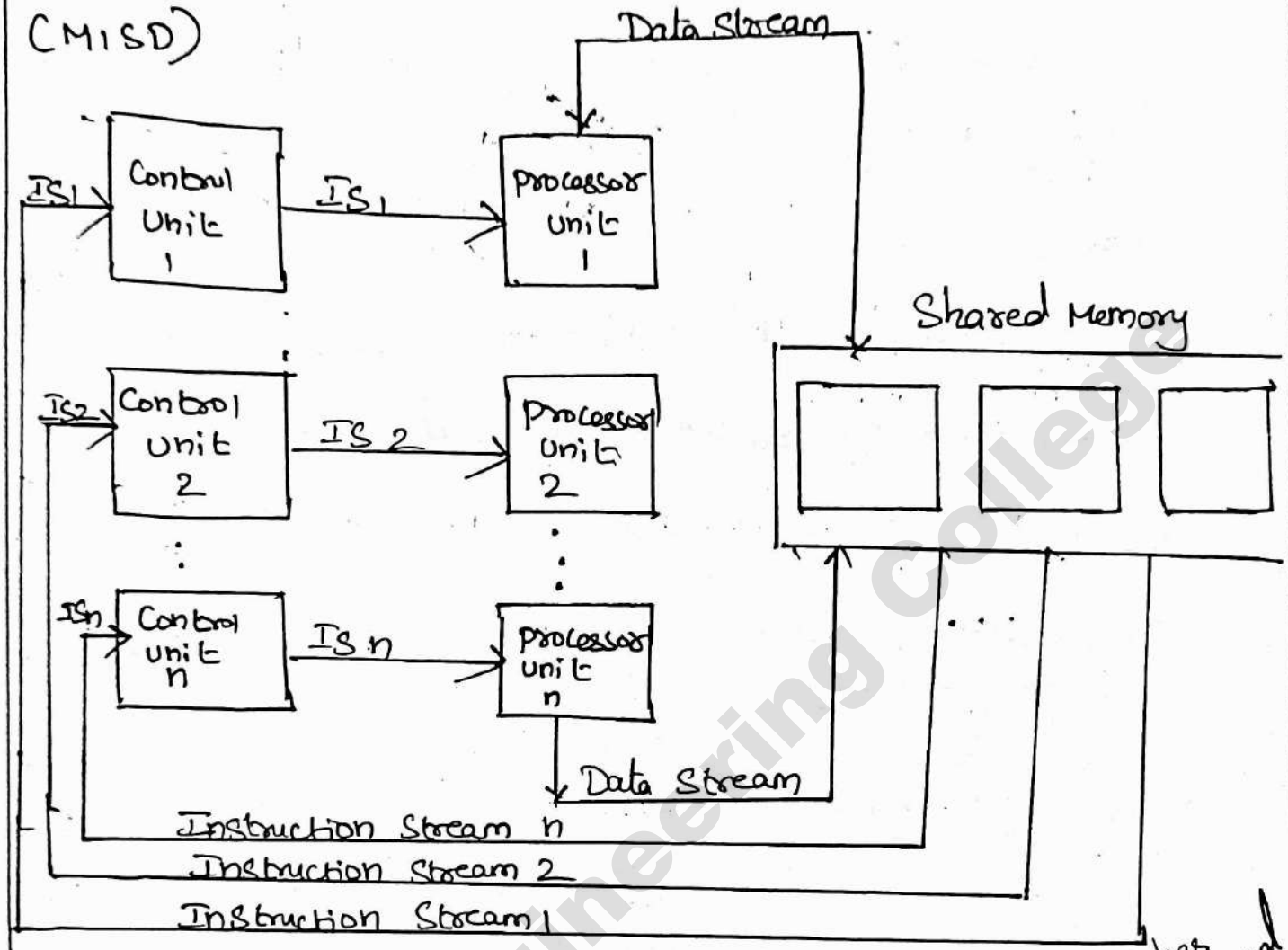
* Therefore, all processing/execution units are supervised by the single control unit.

* 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)

Eg. model GPU

iii) 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.

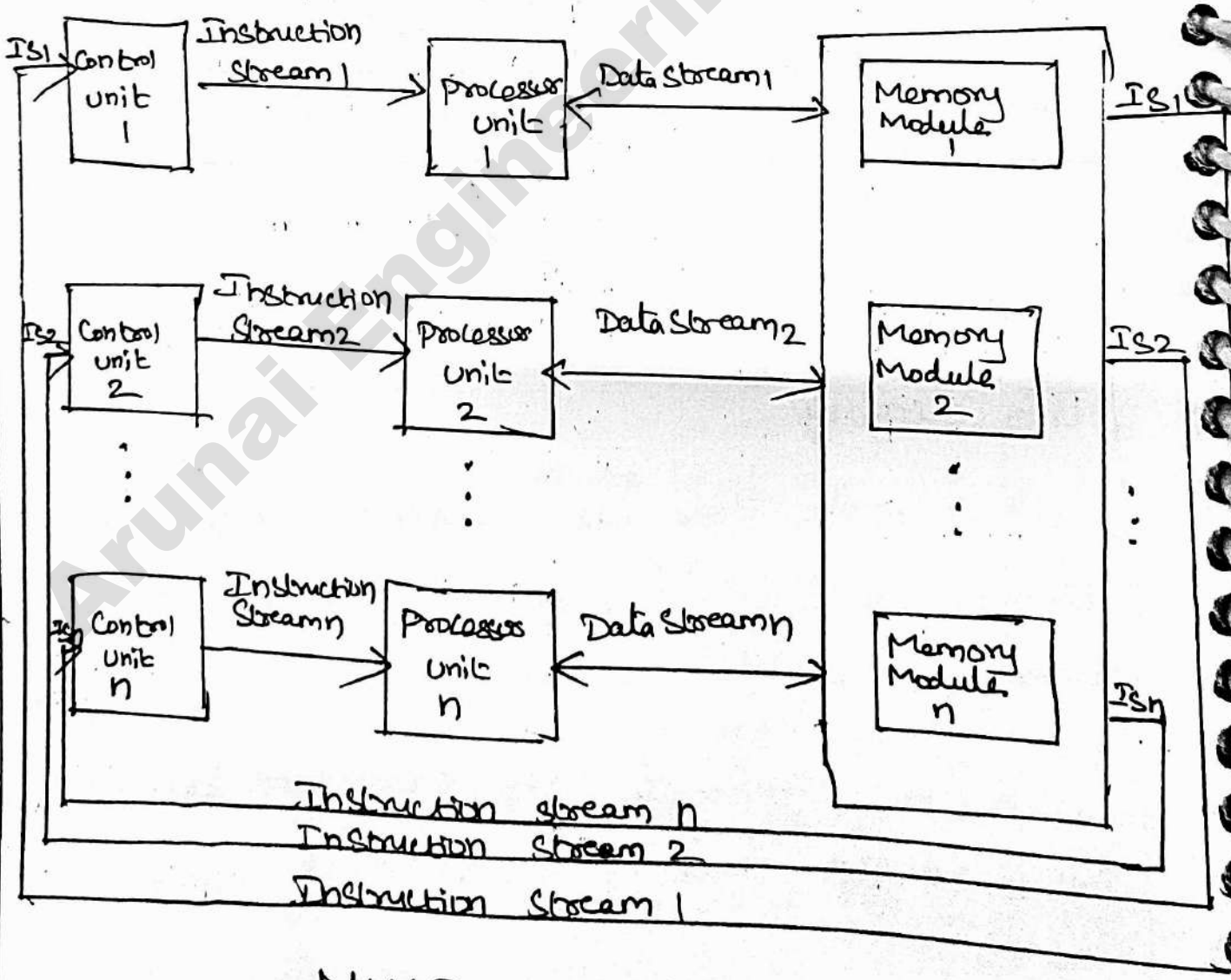
*In MISD, there are n Processor units. Each receiving distinct instructions operating over the same data Stream and its derivatives.

* 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

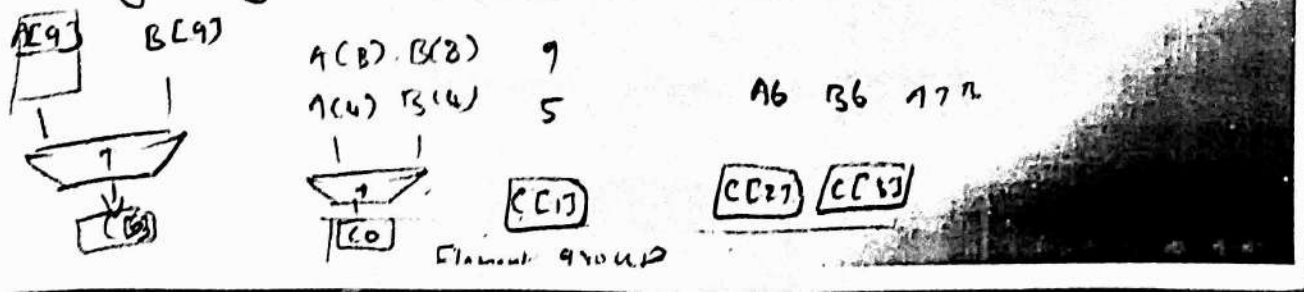
* If the n data streams are derived from disjointed sub spaces of the shared memories then we would have the so-called Multiple SISD (MSISD) operation.

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

* Vector elements are independent (pipelining parallelism can work well) 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 array of data called Vector.

Adv: Highly memory access, Reduce instruction fetch bandwidth

Disadv: memory can easily bottleneck

2. What is hardware Multithreading? Compare and Contrast Fine grained and coarse grained multithreading. [May-2015] [May-17]

Explain in detail about hardware Multithreading [Nov/Dec-2014] [Nov/Dec-2015]

Describe simultaneous Multithreading (SMT) with an example [Nov/Dec-2017]

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 could 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 Utilizati 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

1. Fine Grained Multithreading: cycle-by-cycle

* 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: - No need for dependency checking b/w instructions

* 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 : switch between cache miss → synchronous events fault

* 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.

Fine & coarse grained multithreading can start execution of instructions from only a single thread at a given

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) Instruction from multiple threads executed concurrent in same cycle

* 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

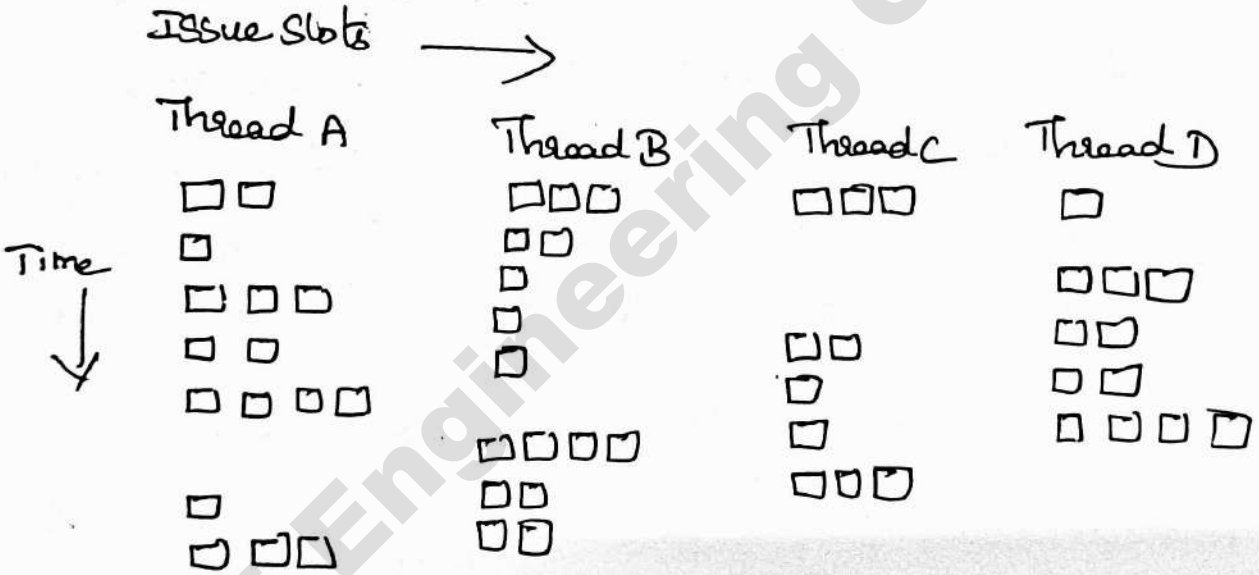
* It has register renaming and dynamic scheduling policy with these features the following tasks can be obtained.

* 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

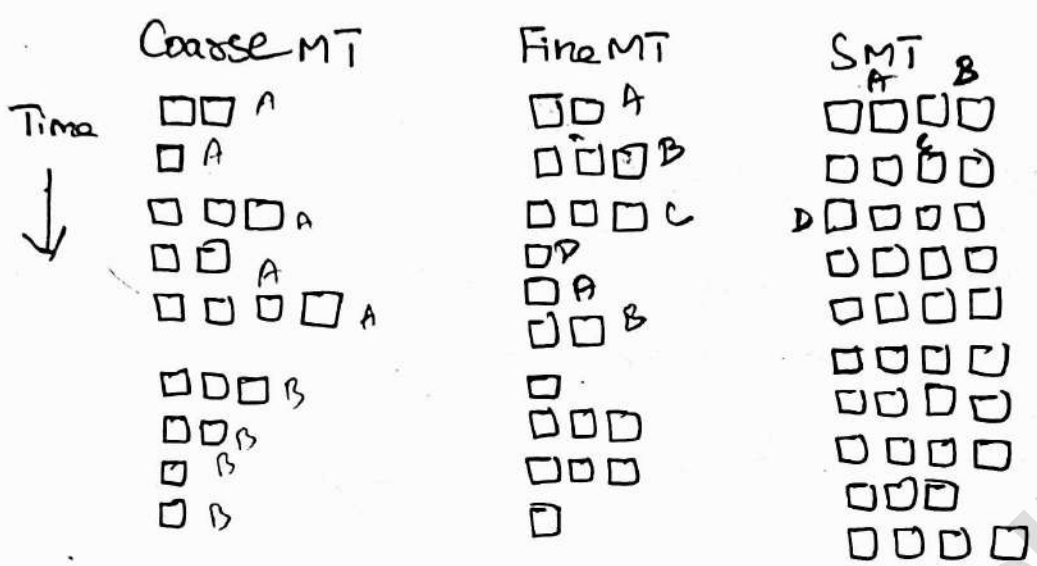
*SMT is always executing instructions from multiple threads, leaving it up to the hardware to associate instruction slots and renamed registers with their proper threads

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



* 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.

3. Explain Instruction level parallel processing. State the challenges of parallel processing [NOV/DEC-2014]

*

Instruction Level parallelism:

*pipelining is a technique that runs programs faster by overlapping the execution of instructions. This is one example of Instruction level parallelism

* Pipelining exploits the potential Parallelism among instructions. This parallelism is called Instruction-level parallelism (ILP).

* There are two primary methods for increasing the potential amount of instruction-level Parallelism

1. Increasing the depth of the pipeline to overlap more instructions.

2. Replicate the internal components of the computer.

*Another approach is to replicate the internal components of the computer so that it can launch multiple instructions in every pipeline stage.

*The general name for this technique is multiple Issue.

* Launching Multiple Instructions per Stage will allow the instruction execution rate to exceed the clock rate or the CPI to be less than 1.

* It is sometimes useful to flip the metric and use IPC or instructions per clock cycle.

* There are two major ways to implement a multiple Issue processor such as

1. Static multiple issue
2. Dynamic multiple issue

* The major differences between these two kinds of issues are the division of work between the Compiler and the hardware, because the division of work dicates whether decisions are made at Compile time or during Execution time.

* A combination of compiler based optimization and hardware techniques can be used to maximize Instruction level parallelism.

Multiple Issue pipeline:

There are two primary and distinct responsibilities that must be dealt within a multiple issue pipeline such as

1. packaging Instructions into issue slots
2. Dealing with data and control hazards

Concept of Speculation:

* one of the most important methods for finding and exploiting more instruction level parallelism is Speculation.

* It is an approach that allows the compiler or the processor to guess about the properties of an instruction so to enable execution that begin for other instruction may depend on the speculated instruction.

* Speculation is an approach whereby the compiler or process, guesses the outcome of an instruction to remove it as dependence in executing other instructions

* Speculation may be done in the compiler or by the hardware.

Eg The compiler can use speculation to reorder instructions, moving an instruction across a branch or load across a store.

Difficulty with Speculation:

* Difficulty with speculation is it may be wrong. So any speculation mechanism must include a method to check if the guess was right and a method to unroll or backout the effects of the instructions was executed speculatively. But it will add complexity

1. Packaging Instructions into issue slots:

* Issue slots are the positions from which instructions could issue in a given clock cycle and it correspond to positions at the starting blocks for a sprint

* How does the processor determine how many instructions and which instructions can be issued in a given clock cycle?

* In Static Issue processors, this process is partially handled by the compiler and in case of dynamic issue processors it is dealt with runtime by the processor

2. Dealing with data and control hazards

* In static issue processors, the compiler handled some or all of the data and control hazards statically.

* In dynamic issue processors atleast some classes of hazards using hardware techniques operating at execution time.

* The recovery mechanisms used for incorrect Speculation is done in two ways

1. Hardware speculation
2. Software speculation

1. Hardware Speculation:

* In hardware speculation the processor usually buffers the speculative results until it knows they are no longer speculative

* If the speculation is correct, the instructions are completed by allowing the contents of the buffers to be written to the registers or memory.

* If the speculation is incorrect, the hardware flushes the buffers and re-executes the correct instruction sequence.

2. Software Speculation:

* In software speculation the compiler usually inserts additional instructions that check the accuracy of the speculation and provide a fix up routine to use when the speculation is incorrect

* Speculation can improve performance when it is done properly and decrease performance when it is done carelessly

Static Multiple Issue processors:

* In static multiple issue, all processors use the compiler to assist with packaging instructions and handling hazards. It will use the Issue Packet.

* Issue Packet is set of instructions that issue together in one clock cycle and the packet may be determined statically by the compiler or dynamically by the processor.

* Static multiple issue processor usually restricts the mix of instruction that can be initiated in a given clock cycle.

* So issue packet is a single instruction that allows several operations in certain Predefined fields. This approach is called Very Long Instruction word (VLIW)

* VLIW is a style of instruction set architecture that launches many operations that are defined to be independent in a single wide instruction and it has many separate opcode fields.

* In most static issue processors compiler take some of the responsibility for handling data and control hazards

The responsibilities taken by the compiler are

1. Static branch Prediction
2. Code Scheduling to reduce or Prevent all hazards.

An Example: Static multiple Issue with MIPS ISA

Static multiple ISSue processor functions can be explained using simple two ISSues MIPS processor.

↳In two ISSue MIPS processor one of the instruction is stored in integer ALU operation or branch

↳other Instruction can be a load or store.

ISSuing two instructions Per cycle require fetching and decoding 64 bits of instructions.

*In many static multiple ISSue Processors and all VLIW processors issuing instructions simultaneously is restricted to simplify the decoding and instruction issue.

* So instruction must be paired and aligned on a 64 bit boundary with the ALU or branch portion must appear first.

Static two Issue Pipeline in operation

Instruction type	Pipe Stages							
ALU or branch Instruction	IF	ID	EX	MEM	WB			
Load or Store Instruction	IF	ID	EX	MEM	WB			
ALU or branch Instruction		IF	ID	EX	MEM	WB		
Load or Store Instruction		IF	ID	EX	MEM	WB		
ALU or branch Instruction			IF	ID	EX	MEM	WB	
Load or Store Instruction			IF	ID	EX	MEM	WB	
ALU or branch Instruction				IF	ID	EX	MEM	WB
Load or Store Instruction				IF	ID	EX	MEM	WB

*ALU and data transfer instructions are issued at the same time.

* Keeping the register writes at the end of the pipeline will simplifies, the handling of exceptions and maintenance of a precise exception model. But it is more difficult in multiple issue processors.

* In some static multiple issue processors the compiler takes full responsibility for removing all hazards, scheduling the code and inserting no-ops.

* For these kinds of design the code executes without any need for hazard detection or hardware generated stalls.

* Hardware - It can detect data hazards and generates stalls between two issue packets for that compiler avoids all dependences within an instruction pair.

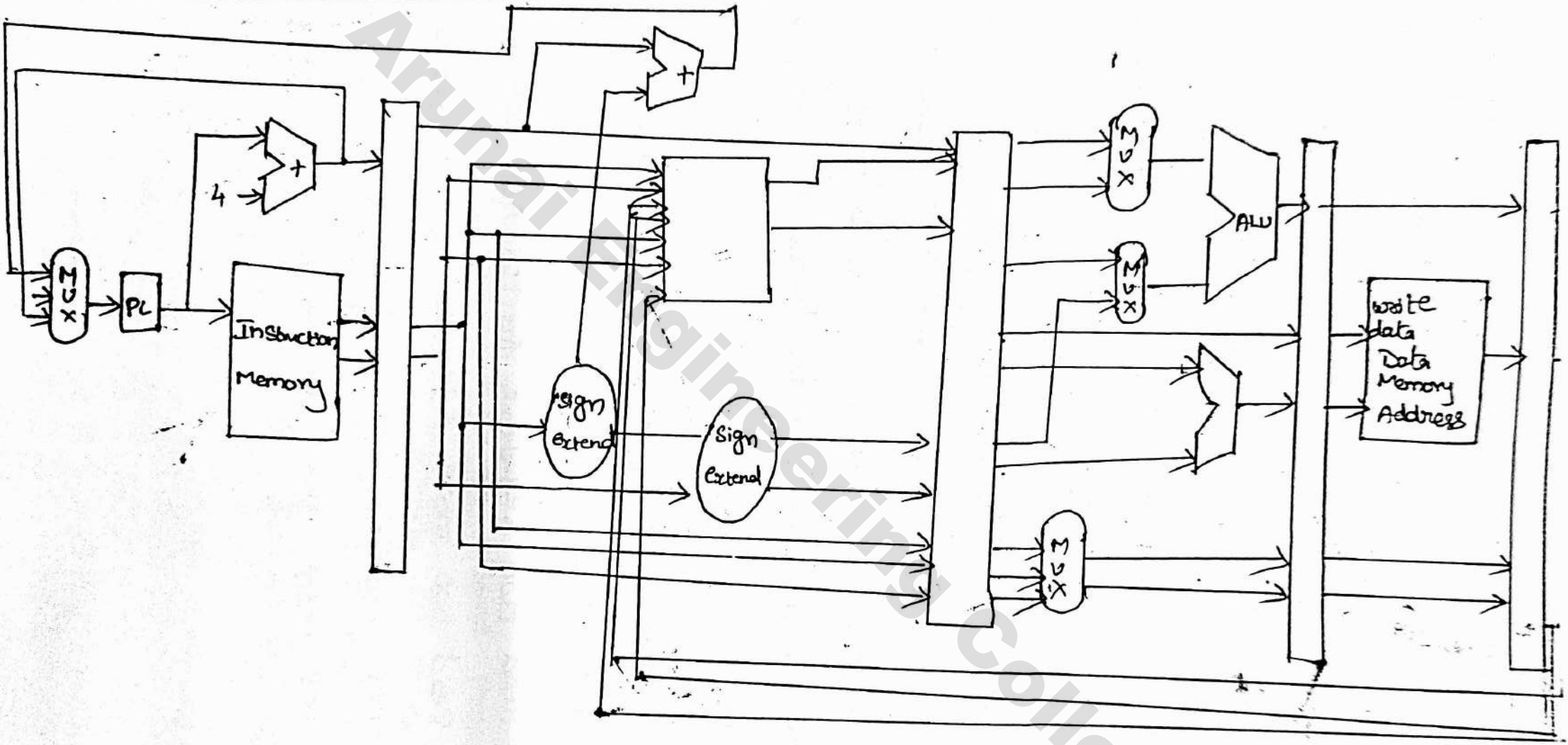
* Software - It must handle all hazards or only by to reduce the fraction of hazards between separate issue packets.

* TO perform ALU and data transfer operation in parallel, first we need an additional hardware.

* With additional hardware units some other units also necessary such as hazard detection and stall logic.

In one clock cycle we need to perform the following task

1. Read two registers for the ALU operation
2. Read two more registers for a store.
3. one register for write port for a load
4. one write port for an ALU operation



A Static Two-Issue Datapath

*TWO-issue processors can improve performance by up to a factor of 2. But many instructions are overlapped and additional overlap increases the relative performance loss from data and control hazards.

9. Dynamic Multiple Issue Processors:

*Dynamic multiple issue processors are also known as super scalar processors or simply superscalars.

* Super Scalar is an advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle by selecting them during execution.

* Achieving good performance on dynamic multiple issue processors it requires the compiler to schedule instructions to move dependences and improve the instruction issue rate.

* In super scalar processor, whether the code has scheduled or not it is given to the hardware to execute correctly

* In super scalar processors, the compiled code can run always correctly independent of the issue rate or pipeline structure of the Processor.

* Some static issue processors the code can run correctly across different implementations but it requires poorly compilation.

* Many superscalars extend the basic framework of dynamic issue decisions by including dynamic pipeline scheduling.

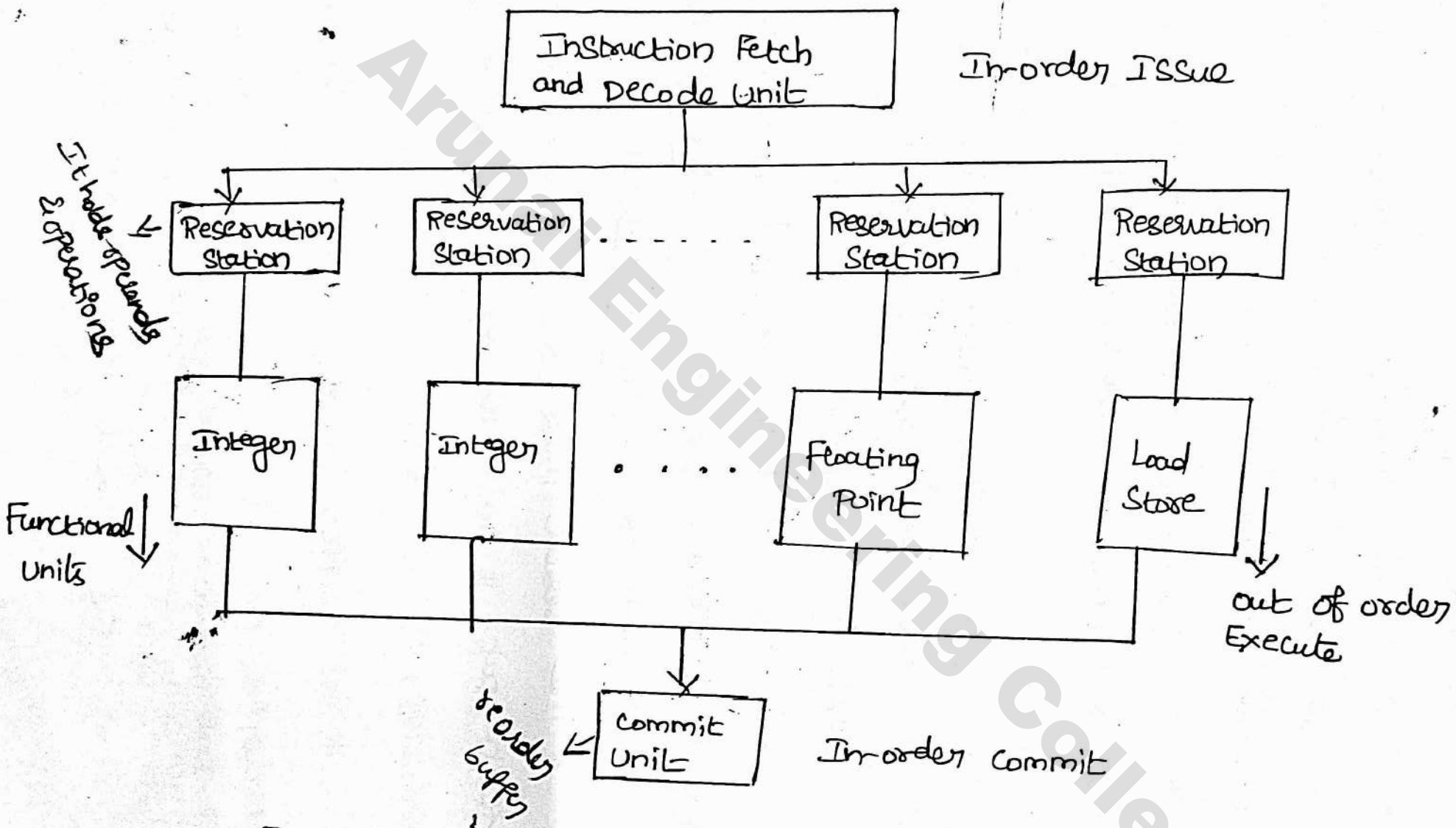
* Dynamic pipeline scheduling chooses which instructions to execute in a given clock cycle while trying to avoid hazards and stalls.

Dynamic pipeline scheduling:

* Dynamic pipeline scheduling chooses which instructions to execute next by reording the instructions it avoid stalls.

* In such processors, the pipeline is divided into three major units

1. An instruction fetch and ISSue Unit
2. Multiple functional Units
3. Commit Unit



The Three Primary Units of a dynamically Scheduled pipeline

* The final step of Updating the state is also called retirement or graduation

* The first unit fetches the instructions, decodes them and sends each instruction to a corresponding functional unit for execution

* Each functional unit has a buffer that is called reservation stations. It holds the operands and operation.

* once the buffer contains all its operands and the functional units is ready to execute then the result is calculated.

* when the result is completed, it is sent to any reservation stations waiting for this particular result as well as to the commit unit

* The result must be buffered until it is safe to put it into the register file or for a store or into memory.

* The buffer in the commit unit called the reorder buffer. It holds the results in a dynamically scheduled processor until it is safe to store the results into memory or a register

* once a register is committed to the register file it can be fetched directly from there just like a normal pipeline.

* Form of register renaming is obtained by Combination of buffering operands in the reservation stations and results in the reorder buffer

Below two steps are explained how the renaming of register is obtained.

Step 1:

* When an instruction issues, it is copied to a reservation station for the appropriate functional unit.

* If any operands are available in the register file or reorder buffer are also immediately copied into the reservation station.

* The instruction is buffered in the reservation station untill all the operands and the functional units are available.

* For issuing instruction, the register copy of the operand is no longer required and for a write to that register occured means the value could be overwritten.

Step 2:

* If an operand is not in the register file or reorder buffer, it must be waiting to be produced by a functional unit.

* The name of the functional unit will produce the result is tracked, when the functional unit eventually produces the result it is copied directly into the waiting reservation station by passing the registers from function unit

↳ These two steps use the reorder buffer and the reservation stations to implement registers renaming.

Out of order execution

* The processor executes the instruction in some order that preserves the data flow order of the program. This style of execution is called out of order execution

In-order Commit

* A commit in which the results of pipelined execution are written to the programmer visible state in the same order how the instructions are fetched

Discuss the challenges in parallel processing with necessary eg: Apr/May -17

Parallel Processing Challenges:

↑ Performance
↓ Time to execute

* Parallel processing will increase the performance of processors 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 processors. Then only we can avoid the idle time of any processor

3 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 general 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 obtain 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 speed 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.

4. Discuss Shared memory multiprocessor with a neat diagram. [NOV/DEC-2016]

Explain the term Multicore Processor [NOV/DEC-14]

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.

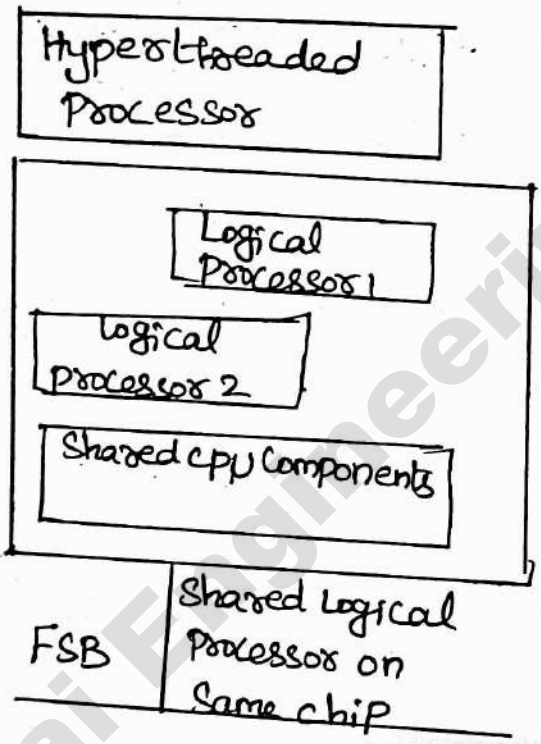
2) Approaches to processor-to-processor communication

3) cache and memory implementations

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

Three common configuration that support multiprocessing.

Type1:



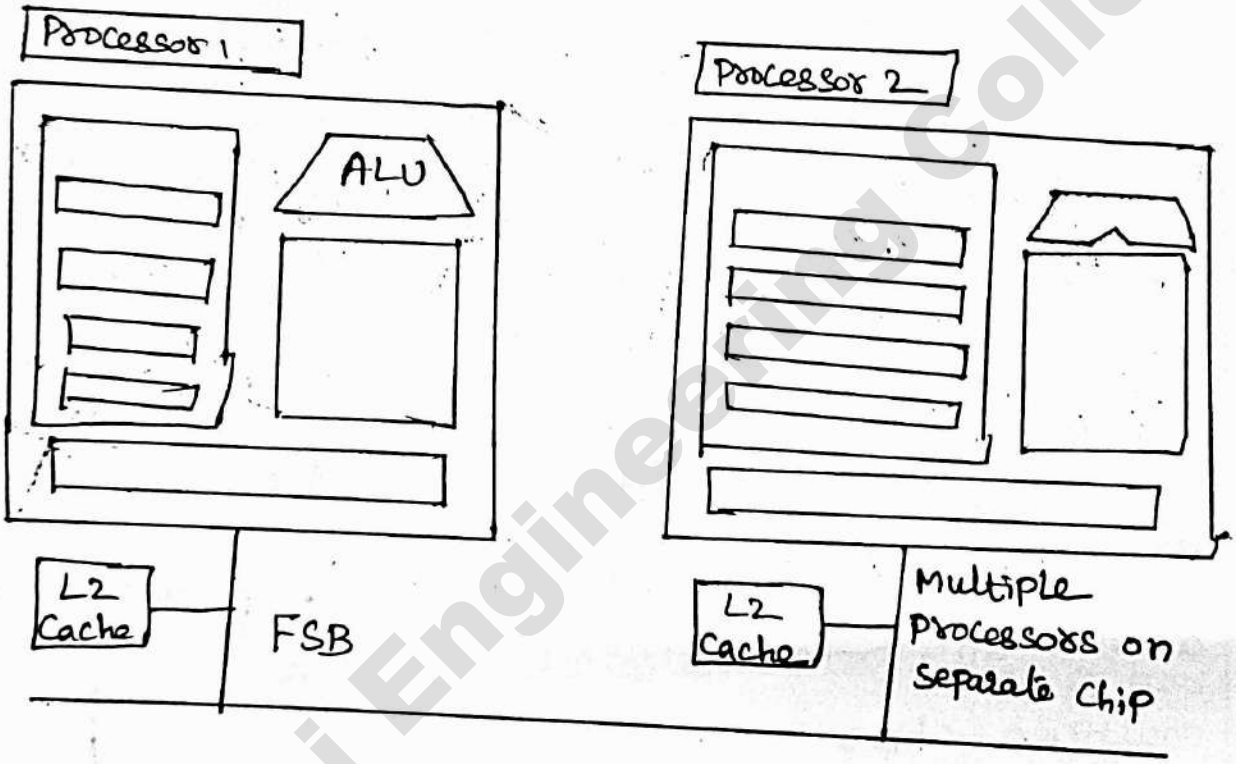
* It uses hyperthreading technology. It allows a single processor to act like two separate processors to the operating system and the application programs that use it

* 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:

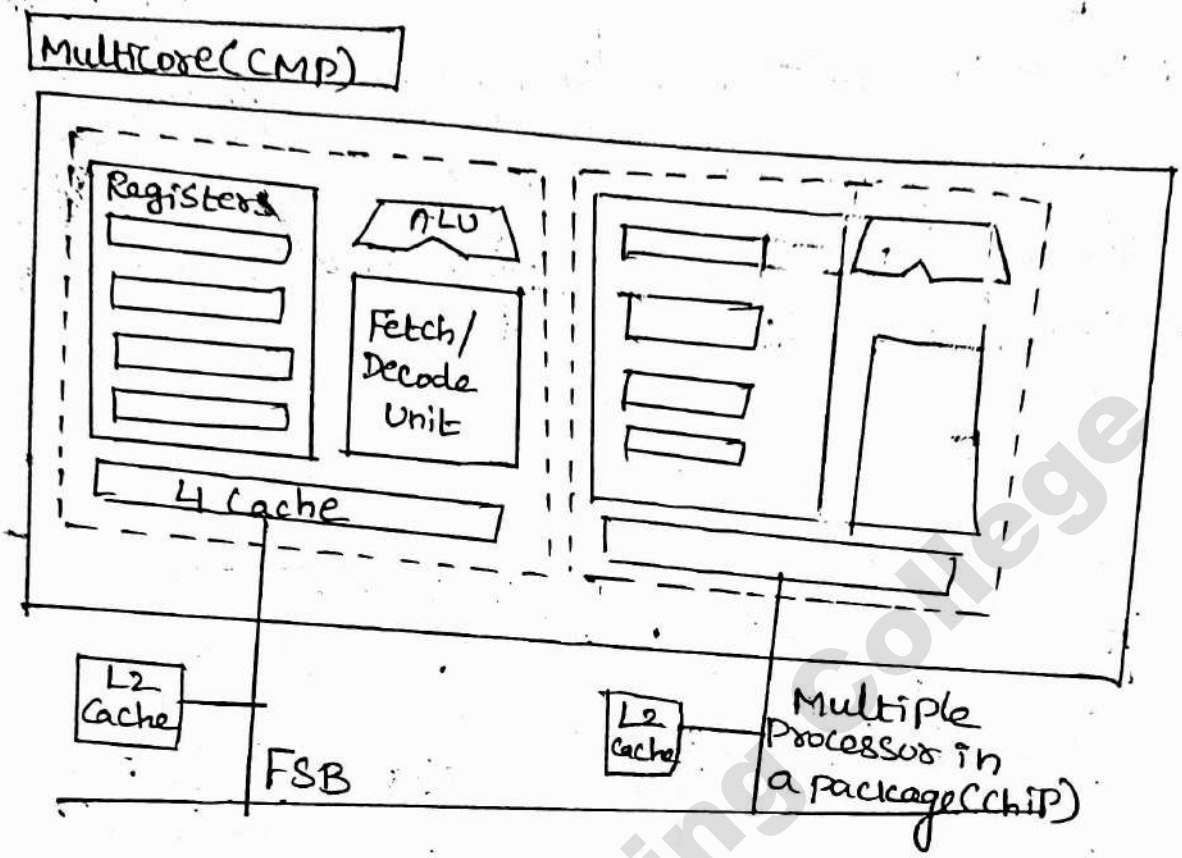


*It is the classic multiprocessor. A multiprocessor is a tightly coupled computer system having two or more processing units (Multiple processors) on a separate chip with its own hardware.

Type 3:

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

Type 3:

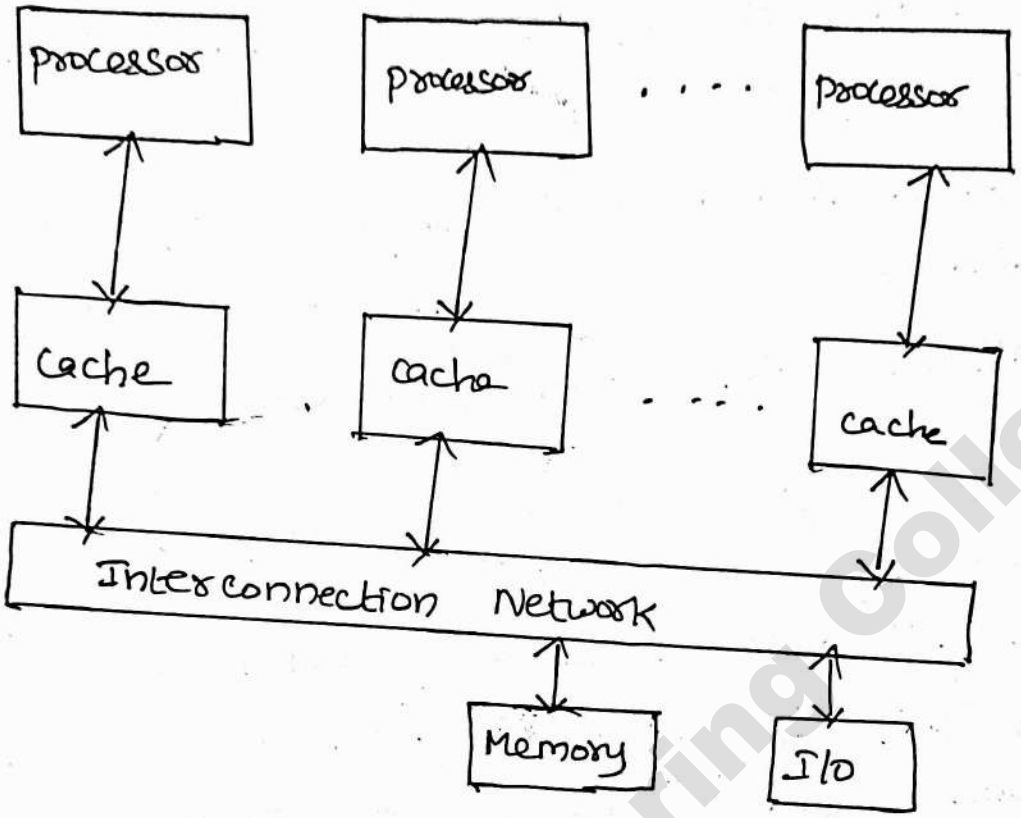


* A Shared Memory Multiprocessor (SMP) allows programmer to use single physical address space across all processors which is always suitable for multicore chips

* 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.

5. A Pipelined processor uses delayed branch technique. Recommend any one of the following possibility for the design of the processor. In the first possibility the processor has a 4-stage pipeline and one delay slot. In the second possibility, it has a 6-stage pipeline and two delay slots. Compare the performance of these two alternatives, taking only the branch penalty into account. Assume that 20% of the instructions are branch instructions and that an optimizing compiler has an 80% success rate in filling in the single delay slot. For the second alternative the compiler is able to fill the second slot 25% of the time. [APR/MAY-2017]

Solution:

$$\text{Speed up factors} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall per instruction}}$$

Pipeline stall per Instruction = Branch Instruction × Number of delays − Branch Instruction × Success rate in 1st delay − Branch Instruction × Success rate in 2nd delay

Branch Instruction = 20 % = 0.2

Success rate in 1st delay = 80 % = 0.80

Success rate in 2nd delay = 25 % = 0.25

4-Stage Pipeline:

* Pipeline depth = 4

Number of delay = 1

Pipeline Stall per instruction = $0.2 \times 1 - 0.2 \times 0.8$

$= 0.2 - 0.16$

$= 0.04$

Speed up factor = $\frac{4}{1+0.04} = \frac{4}{1.04} = 3.846$

6-Stage Pipeline:

* Pipeline depth = 6

Number of delays = 2

Pipeline Stall per instruction = $0.2 \times 2 - 0.2 \times 0.8 - 0.2 \times 0.25$

$= 0.4 - 0.16 - 0.05$

$= 0.4 - 0.21$

$= 0.19$

Speed up factor = $\frac{6}{1+0.19} = \frac{6}{1.19} = 5.042$

∴ Speed up factor of 6-stage pipeline is high, So it will be recommended.

6. i) Suppose you want to achieve a speed-up of 90 times faster with 100 processors. what percentage of the original computation can be sequential?

[Nov/Dec-17]

Solution:

$$\text{Speedup} = \frac{1}{(1-Fe) + \frac{Fe}{Se}} \rightarrow (1)$$

Where

Speedup = 90

Fe = Fraction enhanced, is always less than or equal to 1

Se = speedup, is always greater than 1

(ie) 100 (Number of processors)

Using the values in equation (1) we will get,

$$90 = \frac{1}{(1-Fe) + \frac{Fe}{100}}$$

$$90\left[(1-Fe) + \frac{Fe}{100}\right] = 1$$

$$90(1-Fe) + \frac{90Fe}{100} = 1$$

$$90 - 90Fe + 0.9Fe = 1$$

$$90 - 1 = 90Fe - 0.9Fe$$

$$89 = (90 - 0.9)Fe$$

$$89 = 89.1Fe$$

or

$$89.1Fe = 89$$

$$\therefore Fe = \frac{89}{89.1} = 0.999$$

$$\boxed{Fe = 0.999}$$

Thus to achieve a speed-up of 90 from hundred processors, the sequential percentage can only be 0.1 % (ie., 0.001)

ii) Suppose you want to perform two sums, one is a sum of 10 scalar variables, and one is a matrix sum of a pair of two-dimensional arrays, with dimesions 10 by 10. For now let's assume only the matrix sum is parallelizable; we'll seen how to parallelize scalar sums. what speed-up do you get with 10 versus 40 processors? Next, calculate the speed-ups assuming the matrices grow to 20 by 20.

[NOV/DEC-19]

Solution:

If we assume performance is a function of the time for an addition, t, then there are 10 additions that do not benefit from parallel processors and 100 additions that do. If the time for a single processor is 110 t, the execution time for 10 processor is

Execution time after improvement =

$$\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

Execution time after improvement = $\frac{100t}{10} + 10t$

$= 20t$

So, the speed-up with 10 processors is $110t/20t = 5.5$

The execution time for 40 processors is

Execution time after improvement = $\frac{100t}{40} + 10t$

$= 12.5t$

So, the speed-up with 40 processors is $= 110t/12.5t$

$= 8.8$

* Thus, for this problem size, we get about 55 % of the potential speed-up with 10 processors, but only 22 % with 40.

* Look what happens when we increase the matrix. The sequential program now takes $10t + 400t = 410t$. The execution time for 10 processors is

Execution time after improvement = $\frac{400t}{10} + 10t$

$= 50t$

* So the Speed-up with 10 processors is 41ot/5ot
= 8.2

The Execution time for 40 processor is

Execution time after improvement = $\frac{40ot}{40} + 1ot$
= 2ot

* So the Speed-up with 40 processor is = 41ot/2ot
= 20.5

* Thus, for this larger problem size, we get 82% of the potential speed-up with 10 processors and 51% with 40.

* These problem show that getting good speed-up on a multiprocessor while keeping the problem size fixed is harder than getting good speed-up by increasing the size of the problem.

UNIT-V

PART-A

1. What is DMA? [NOV/DEC-2014]

The CPU and I/O controller interact only when the CPU must yield control of the memory bus to the I/O controller in response to requests from the latter. This level of I/O control is called Direct Memory Access (DMA)

2. Differentiate programmed I/O and Interrupt I/O [NOV-2014]

Programmed I/O	Interrupt I/O
The time that the CPU spends for testing I/O device status and Executing I/O data transfer is high	The time that CPU spends for testing I/O device status and executing I/O data transfer is low

3. what is the purpose of dirty/modified bit in cache memory? (NOV-DEC 2014)

A dirty bit or modified bit is a bit that is associated with block of computer memory and it indicates whether or not the corresponding block of memory has been modified. This bit is set when the processor writes to this memory.

---

4 what is the need to implement memory as a hierarchy? (May-2015)

* Decreasing cost per bit
* Increasing capacity
* Increasing access time
* Decreasing frequency of access of the memory by the processor.

---

5 Point out how DMA can Improved I/O speed. [May/June-2015]

* The I/O device is connected to the system bus via a special interface circuit called DMA controller. In this data counter 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

6. What are the various memory technologies? (Nov/Dec-2015)

Computer memory is classified as primary memory and secondary memory.

i) primary memory technologies in use:

Static RAM, Dynamic RAM, ROM and ity types (PROM, EPROM, EEPROM) etc.

ii) Secondary memory technologies in use:

optical disk, flash-based removal memory card, hard-disk drives, magnetic tapes etc.

---

7. Define Hit ratio (Nov-Dec-2015)

The percentage of accesses where the processor finds the code or data word it needs in the cache memory is called the hit rate or hit ratio

---

8. Define Memory hierarchy (May/June-2016)

A memory hierarchy is a structure of memory that uses multiple levels of memories as the distance from the processor increases; the size of the memories and the access time both increases.

---

9. State the advantage of Virtual memory (may/June-2016)

1. Allocating memory is easy
2. Eliminates external fragmentation
3. Allows demand paging and Pocpaging
4. More efficient Swapping
5. It is the ability to load and execute a process that requires a larger amount of memory than what is available by loading the process in parts and then executing them.

---

10. Define locality of reference. what are its types?

* The program may contain a simple loop, nested loops, or a few procedures that repeatedly call each other. The point is that many instructions in localized area of the program are executed repeatedly during some time period and the remainder of the program is accessed relatively infrequently. This referred as locality of reference

---

11 Define Memory Interleaving [APR/MAY-2017]

* In Computing, interleaved memory is a design made to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory, by spreading memory addresses, evenly across memory banks

* Memory interleaving is a category of techniques for increasing memory speed.

Ex: with separate memory banks for odd and even addresses, the next byte of memory can be accessed while the current byte is being refreshed.

12. Summarize the sequence of events involved in handling (handling) an interrupt request from a single device. [APR/MAY-2017]

* An interrupt is an unscheduled event that disrupts program Execution, used to detect overflow.

↳ Processor checks for interrupts

↳ If no interrupts, fetch the next instruction for the current program.

↳ If an interrupt is pending, suspend execution of the current program, and execute the interrupthandler.

Multiple Interrupts:

Two approaches to handle multiple interrupts

1. Define priorities for interrupts. Allow interrupts to interrupt interrupts
2. Disable interrupt while an interrupt is being processed.

13 What is virtual Memory? [NOV/DEC-2017]

* 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.

---

14 How many total bits are required for a direct-mapped cache with 16KB of data and 4-word blocks, assuming a 32-bit address? [NOV/DEC-2017]

* We know that 16 KB is 4K words, which is 2^{12} words and with a block size of 4 words (2^2), 2^{10} blocks. Each block has 4×32 or 128 bits bits of data plus a tag, which is 32-10-2-2 bits, plus a valid bit.

Total cache size is

$2^{10} \times (128 + (32-10-2-2) + 1) = 2^{10} \times 147 = 147$ Kbits

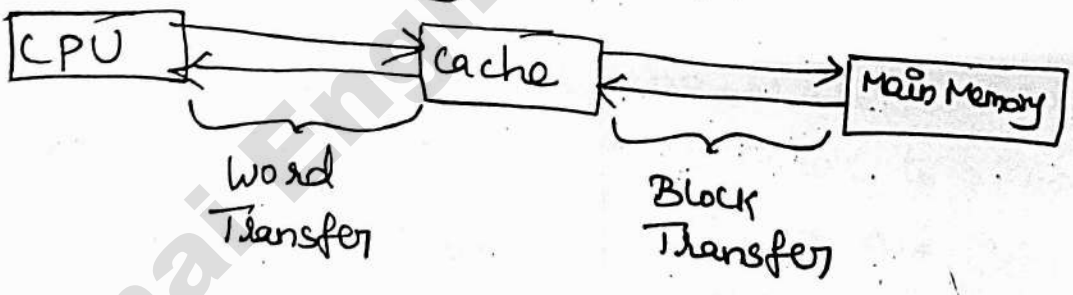
147 Kbits or 18.4 KB for a 16 KB cache.

* For this cache, the total number of bits in the cache is about 1.15 times as many as needed just for the storage of the data.

15 What is meant by address mapping? [NOV/DEC-2016]

Addressing mapping, which is defined as the smallest unit of addressed data (from the persiste-t store) that can be mapped independently to an area of the virtual address space

16. What is Cache Memory? [NOV/DEC-2016]



* Cache memory is a small-sized type of volatile Computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications and data.

17 Distinguish SRAM and DRAM. [Apr/May-18]

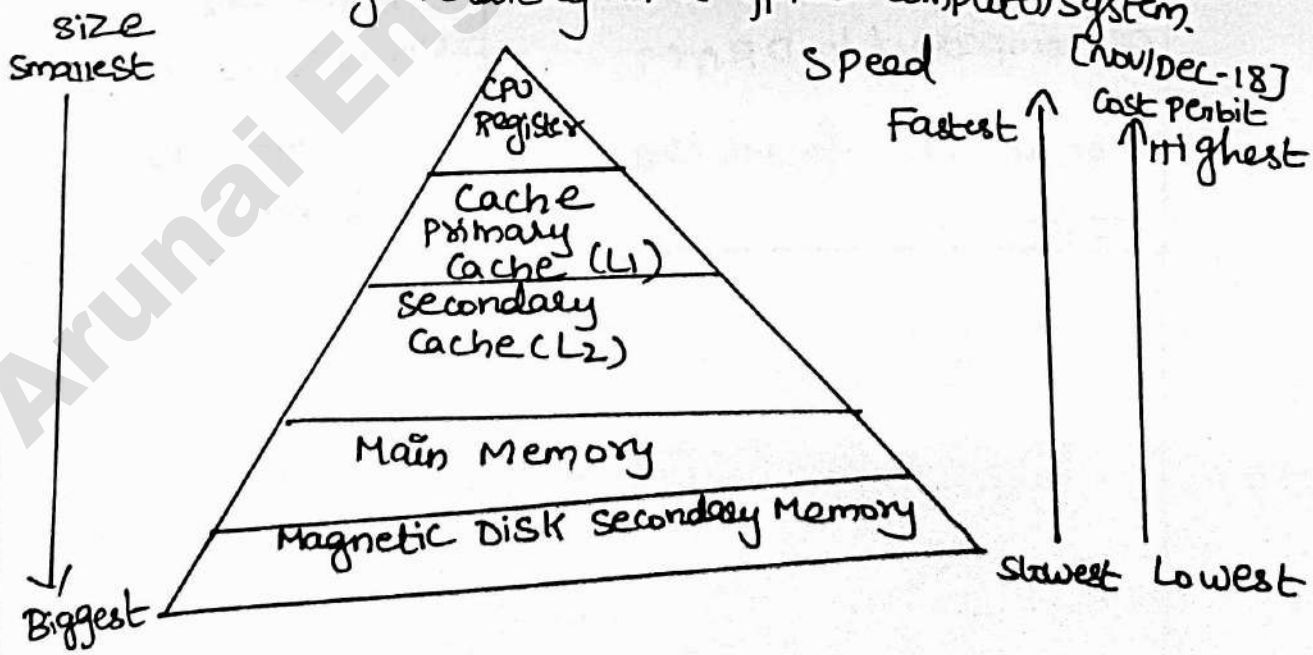
Static RAM (SRAM)	Dynamic RAM (DRAM)
* Static Random Access Memory uses transistor to stor a single bit of data.	* DRAM Dynamic Random Access memory uses a separate capacitor to store each bit of data.
* SRAM does not need periodic refreshment to maintain data.	* DRAM needs Periodic refreshment to maintain data
* SRAM are used in Cache memory	* DRAM are used in Main Memory
* SRAM are expensive as compared to DRAM	* DRAM are less expensive as compared to SRAM
* SRAM are faster than DRAM	* DRAM are Slower than SRAM

18 What is the use of DMA Controller? [Apr/May-18]

* Direct Memory Access is used to transfer large blocks of data at high speed, a special unit is provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor.

* The process is managed by a chip known as a DMA controller (DMAC)

19 Draw the memory hierarchy in a typical computer system. [Nov/Dec-18]



20 What is meant by memory-mapped I/O? [NOV/DEC-18]

* 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 instruction 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.

PART-B

7. Discuss DMA Controller with block Diagram [NOV/DEC 2016]
Draw the typical block diagram of a DMA controller and explain how it is used for direct data transfer between memory and peripherals? [NOV/DEC-2015]
Explain about DMA controller with the help of a block diagram [May/June-2016] [
What is meant by DMA? Explain the use of DMA Controllers in a computer System [May/June-17]

DMA Controller:

↳ The Programmed I/O Method has two limitations

1. The speed with which the CPU can test and service I/O device limits I/O 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 I/O device needing service waits to be tested by the CPU. If there are many I/O devices in the system then each device may be tested infrequently.

2. Programmed I/O 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.

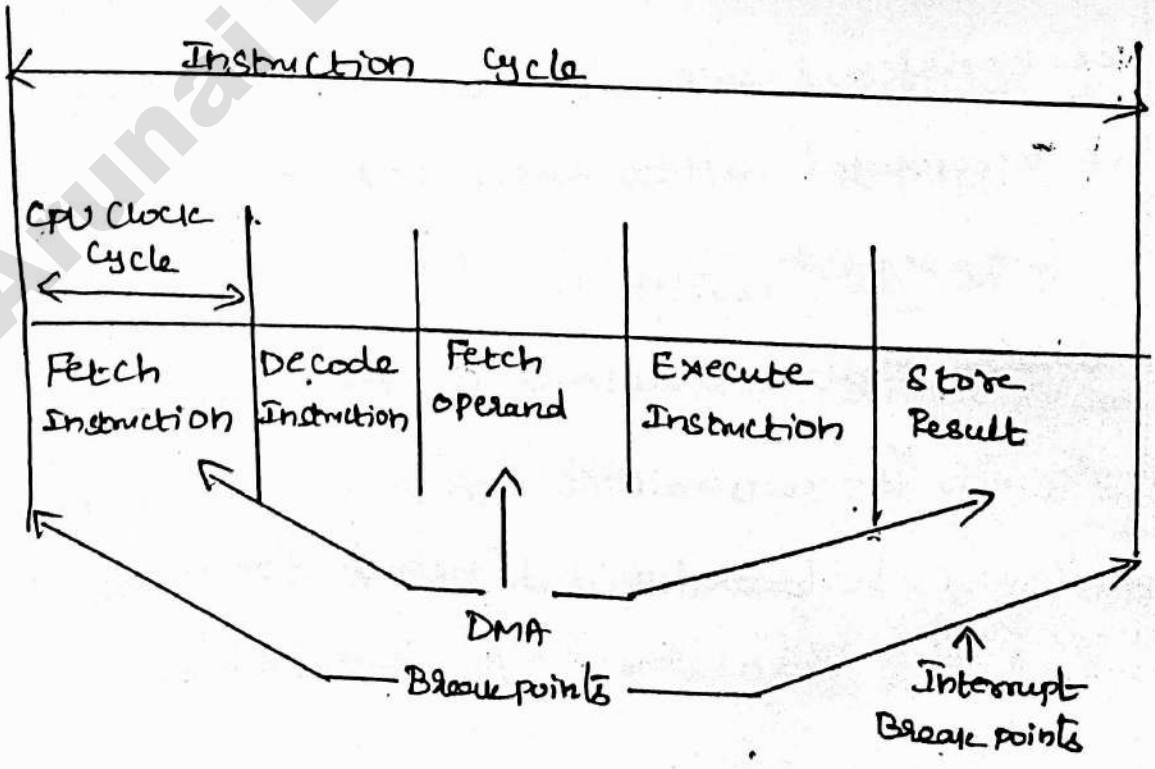
*DMA request by an Iodevice only requires the CPU to grant control of the memory (system) bus to the requesting device

The CPU can yield control at the end of any transactions involving the use of system bus.

Typical sequence of CPU actions during execution of a single Instruction.

↳ The instruction cycle is collection of numbers 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.

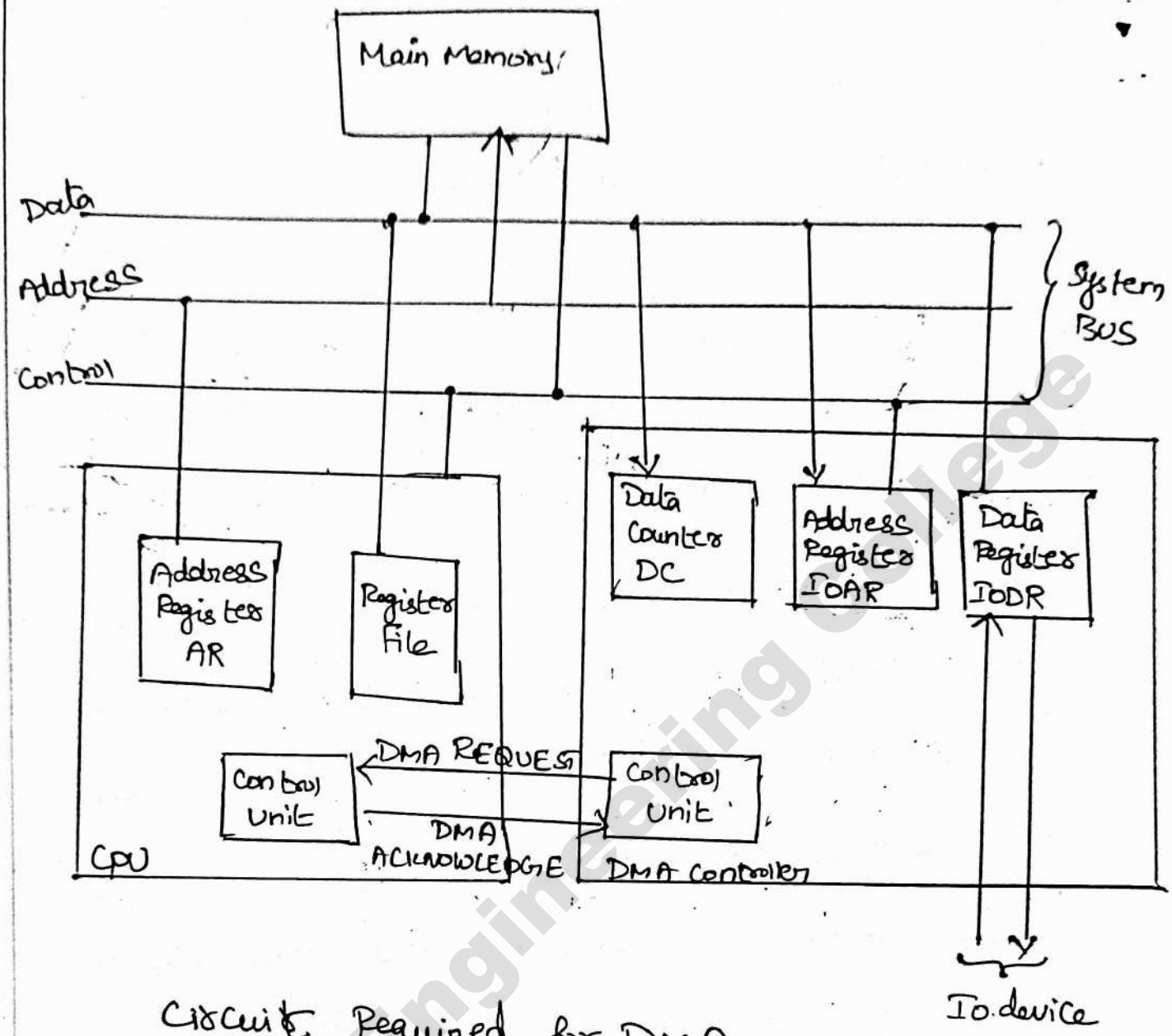


Direct Memory Access (DMA)

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 sequence 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 I/O 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 I/O 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 syste

2) What is Virtual Memory? Explain the steps involved in virtual memory address translation. [MAY-2015]
What is virtual memory? Explain in detail about how virtual memory is implemented with neat diagram [NOV/DEC-2015]

Virtual Memory: (Storage allocation scheme) [NOV/DEC-2016] [MAY-17]

* 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.

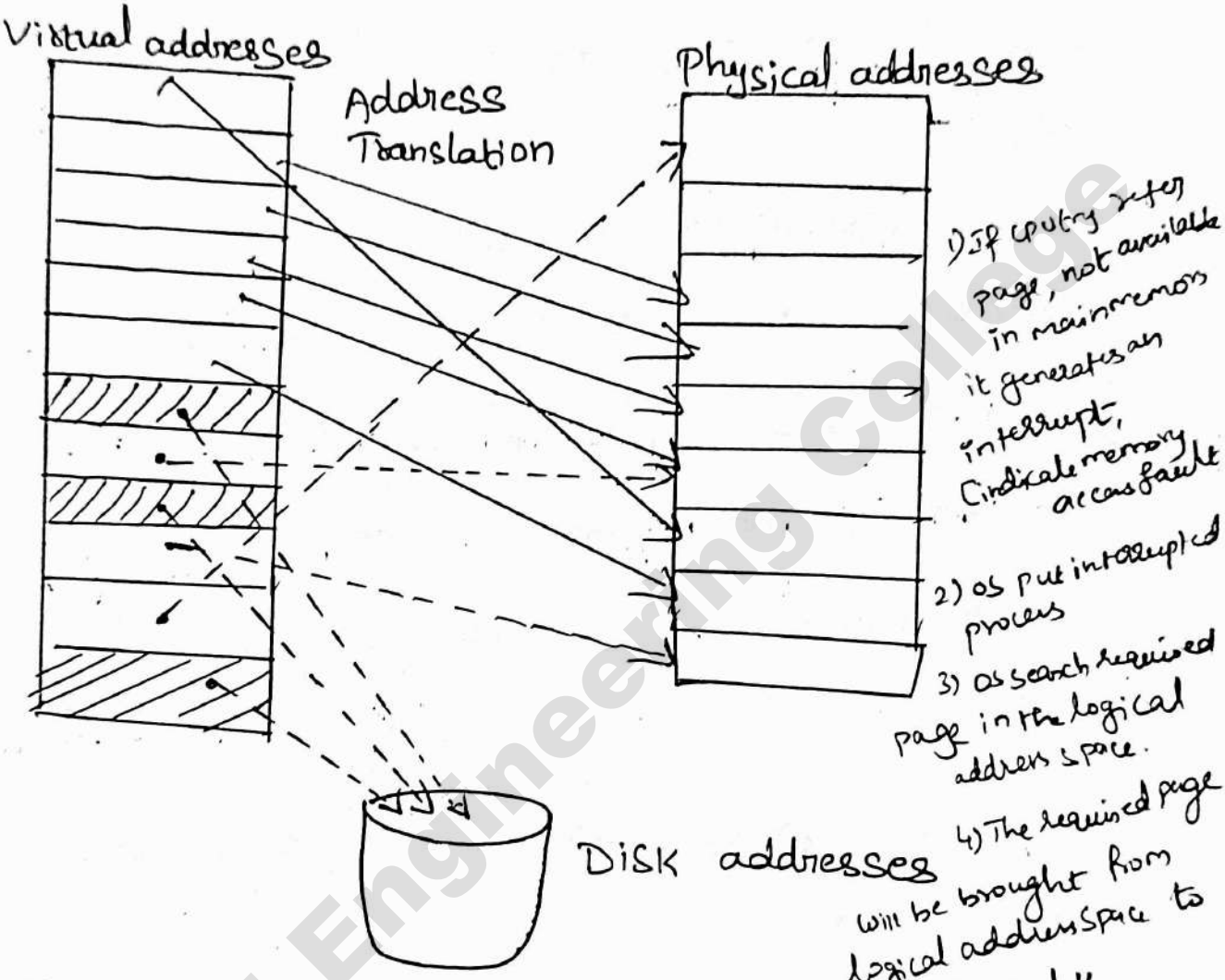
* Virtual memory implements the translation of a program address space to physical address.

* 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.

* Not necessary all pages or segments are present in main memory

Required Pages need to be loaded into memory, Virtual Memory is implemented by Demand Paging or Demand segmentation.

* 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.

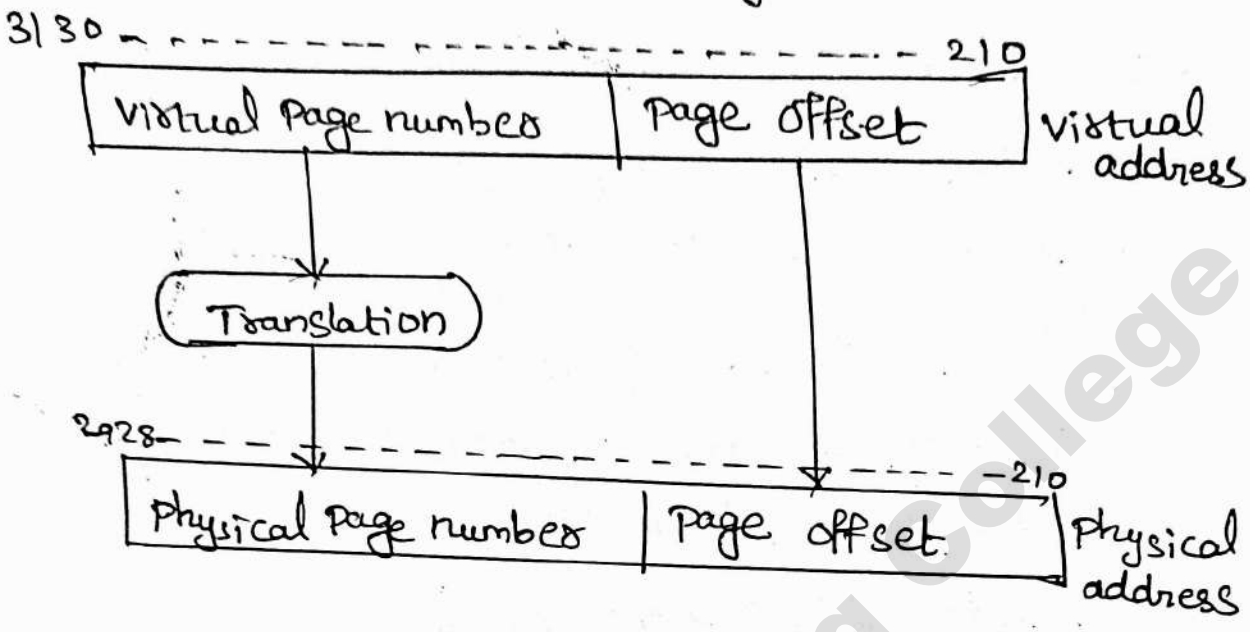
* Virtual memory also simplifies loading the program for execution by providing relocation method

* In virtual memory, the address is broken into two parts

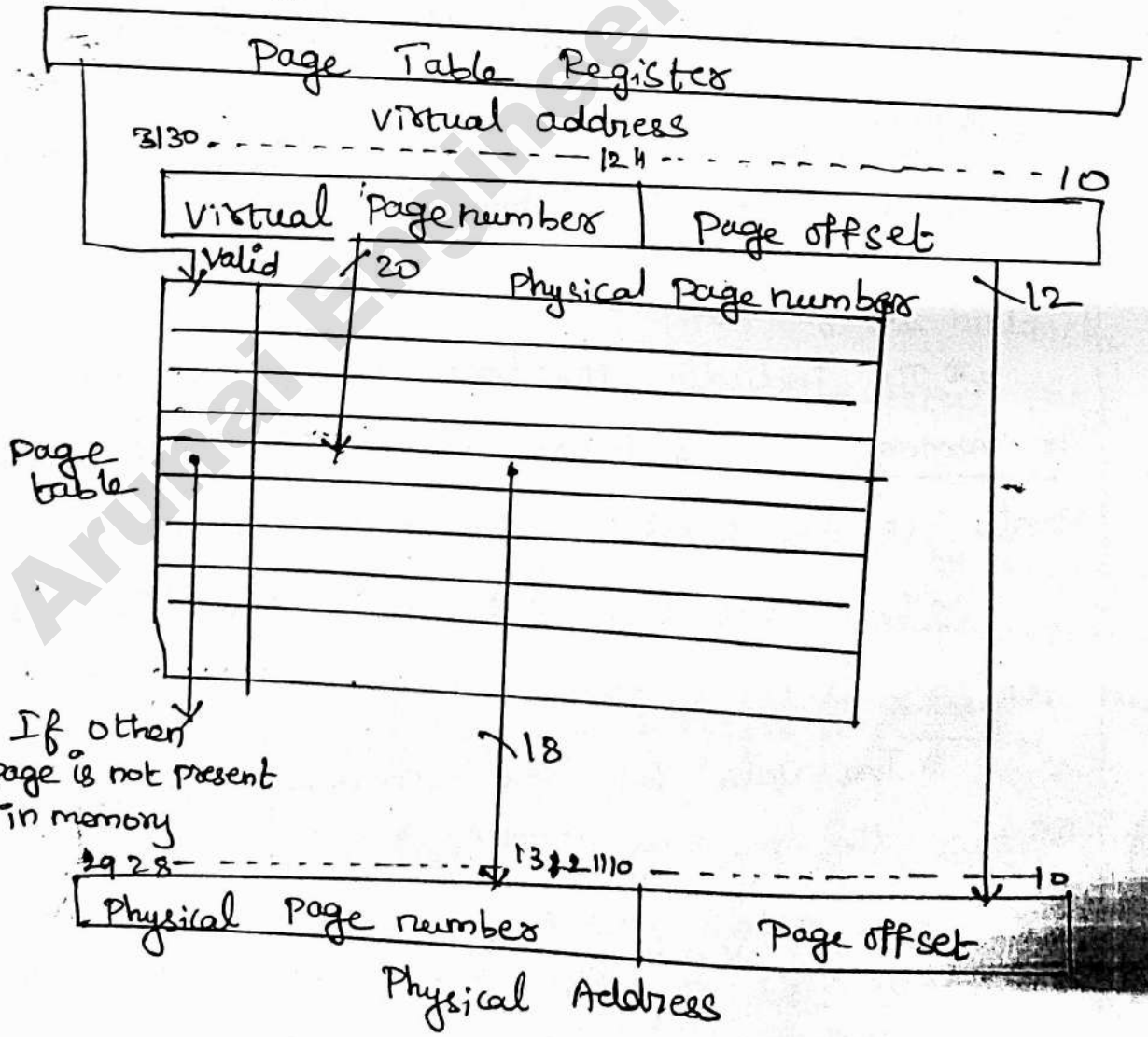
1) Virtual page number
2) Page offset

* 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 numbers.

* 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.

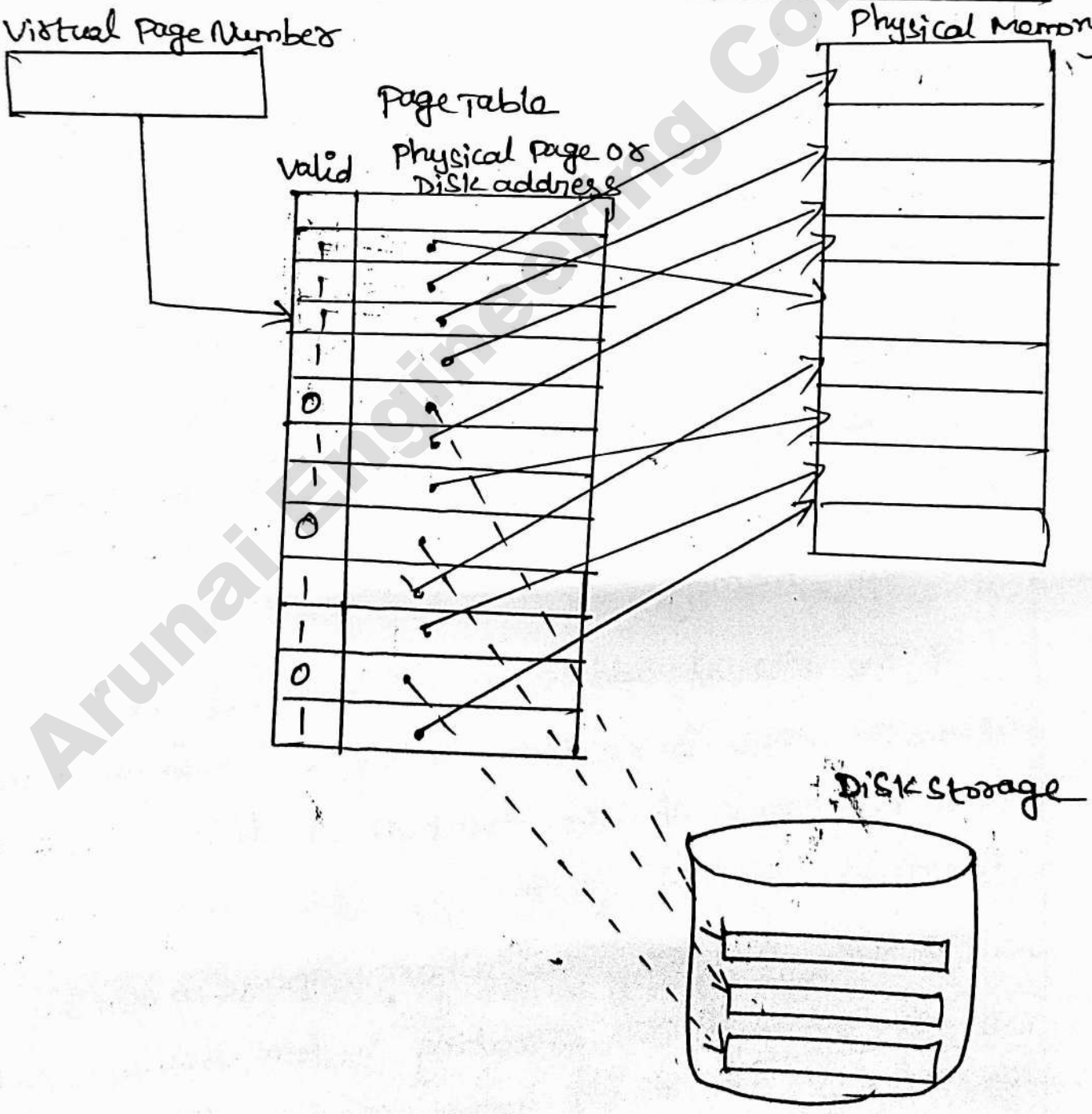
* The virtual address alone does not tell us where the page is on disk, so virtual memory system, must keep track of the location on disk of each page in virtual address space.

* 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

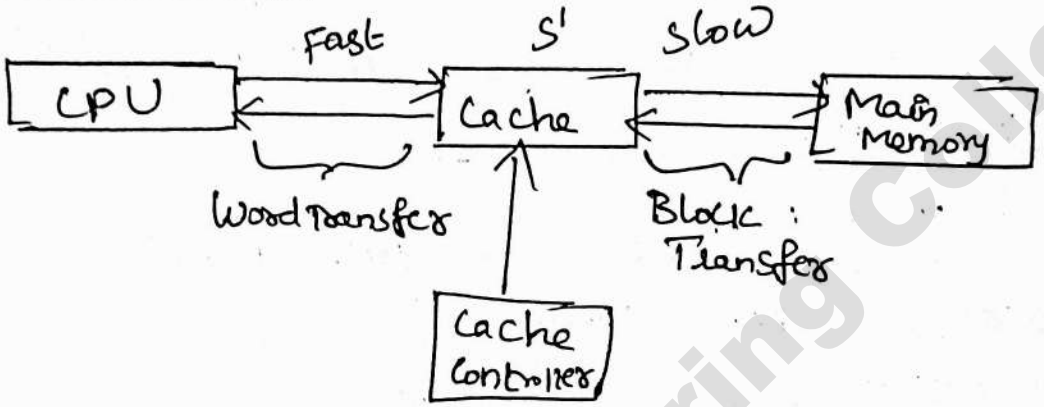
* Dirty bit is set when any word in a page is written.

* 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.

3. Define Cache Memory? Explain the various Mapping Techniques associated with each memories [May/June 2016]

Explain Mapping functions in cache memory to determine how memory blocks are placed in cache [NOV/DEC-2014] [APR/May-17]

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

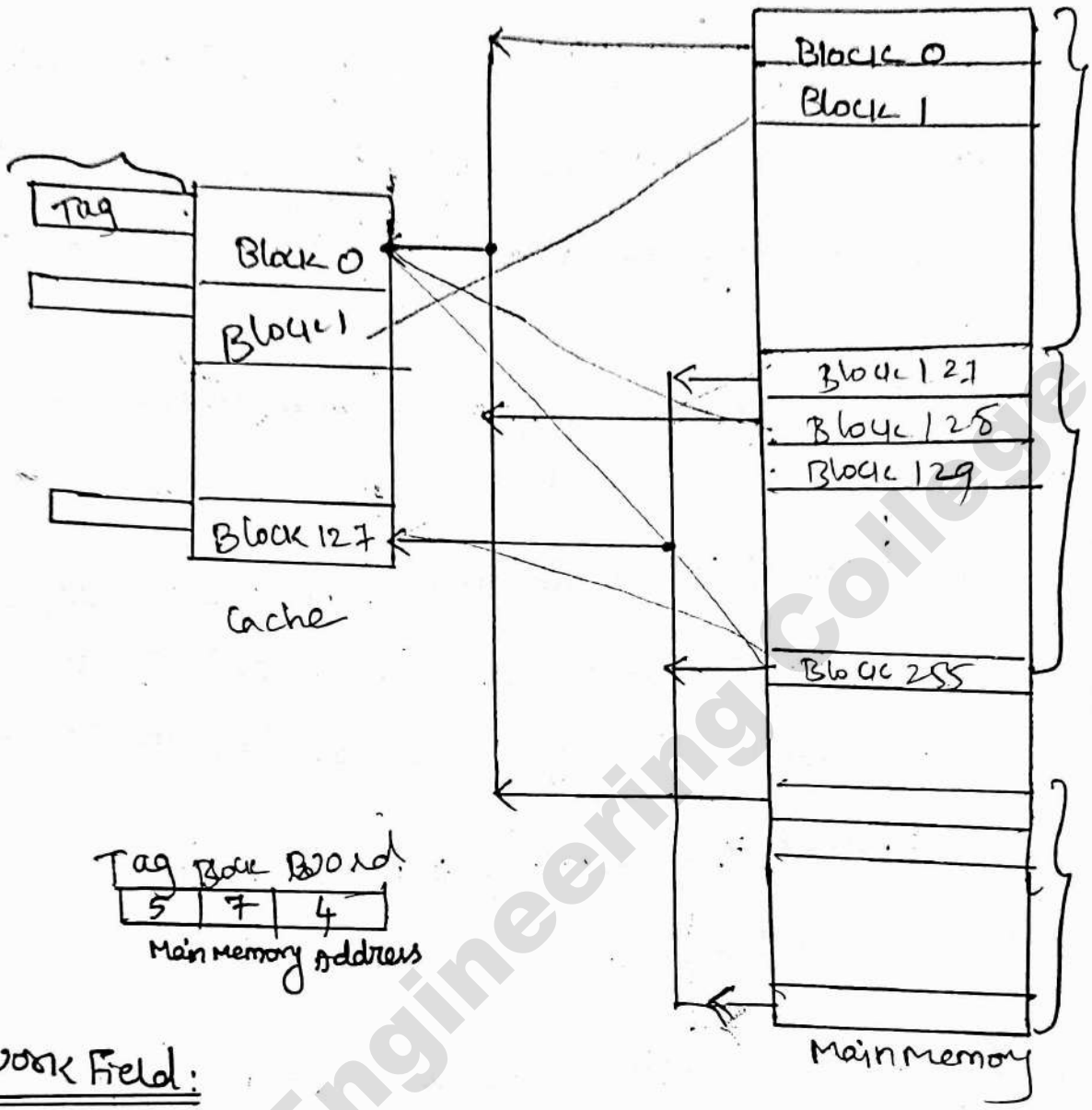
The address is divided into three fields

i) Word field

ii) Block field

iii) Tag field

Direct-Mapped cache



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: - Identifies which memory block is in the slot

* 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 cont...

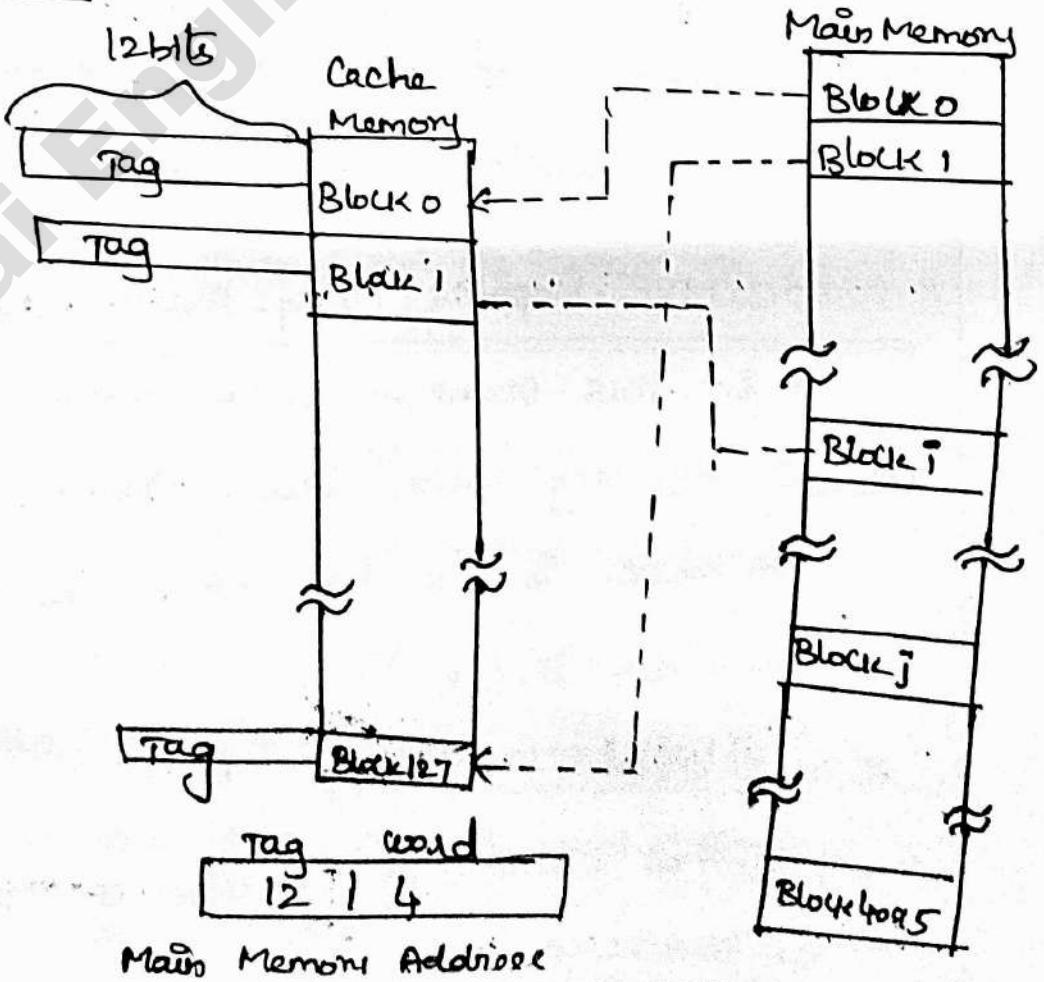
* The 12-tag bits are required to identify a memory block when it is resident in the cache.

* 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.

3) Set-Associative Mapping: - Cache is divided into number of sets Each set contains a number of lines

* 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.

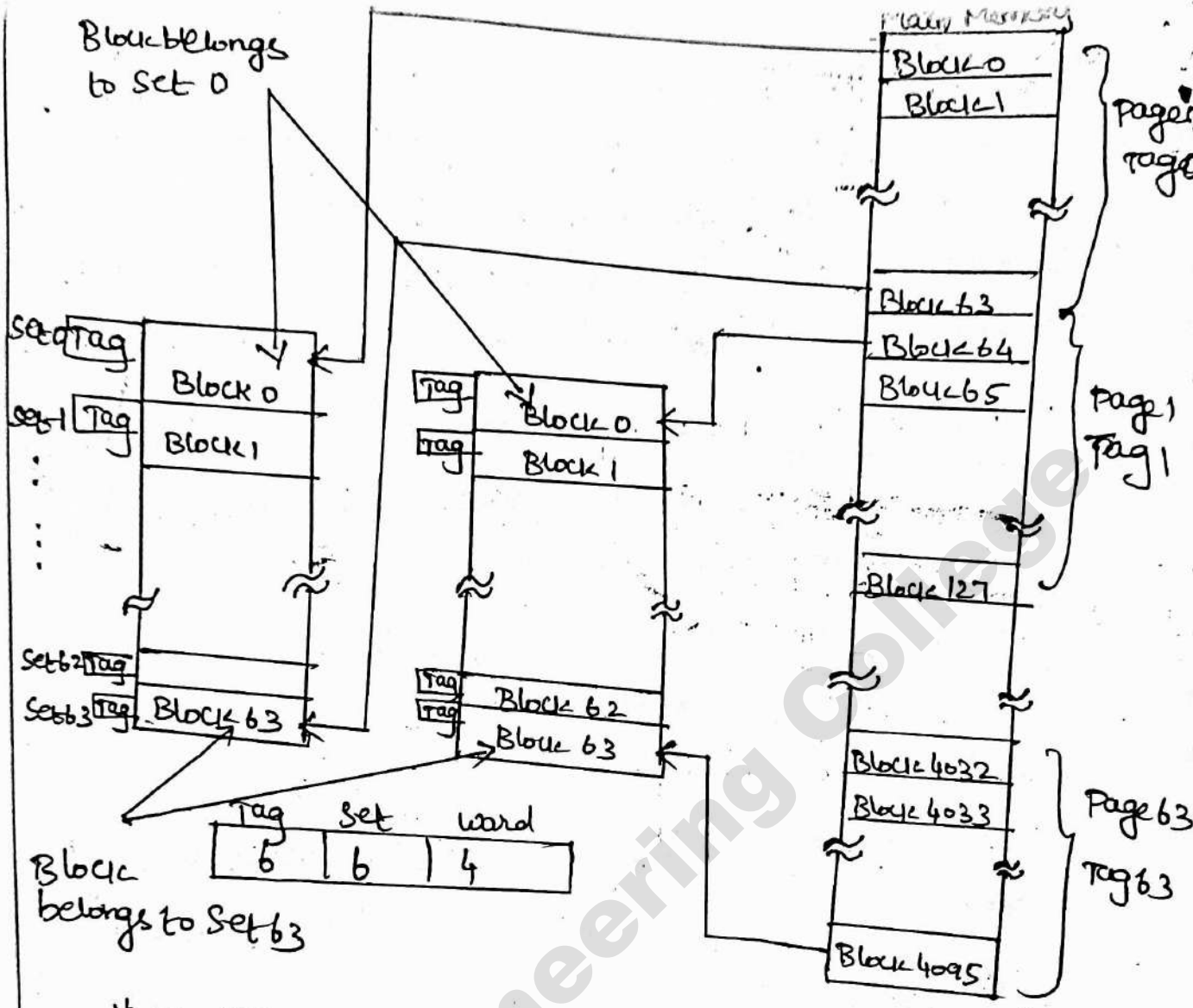
* A block of data from any page in the main memory can go into a particular block location of any direct-mapped cache.

* The required address comparisons depend on the number of direct-mapped caches in the cache systems.

Two-way set-associative cache

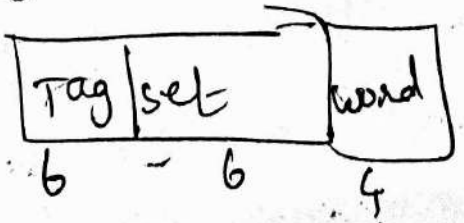
* Each page in the main memory is organized in such a way that the size of each page is same as the size of one directly mapped cache.

* It is called two-way set-associative cache because each block from main memory has two choices for block placements.



* 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:

2^6 = 64 different memory lines in each of the $2^6 = 64$ different 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.

Disadvantages:

* 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.

Explain in detail about the Bus arbitration techniques in DMA [NOV/DEC-2014]. [Apr/May-17]

A bus is comm. path way connecting two or more bus (CPU, I/O, Main memory) are interconnected by common bus ← data bus, address bus, control bus

Bus Arbitration: → Deciding which devices get to use the shared bus

*The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.

Two parts < Issuing command (Bus master), Transferring the data (Bus slave)

* 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.

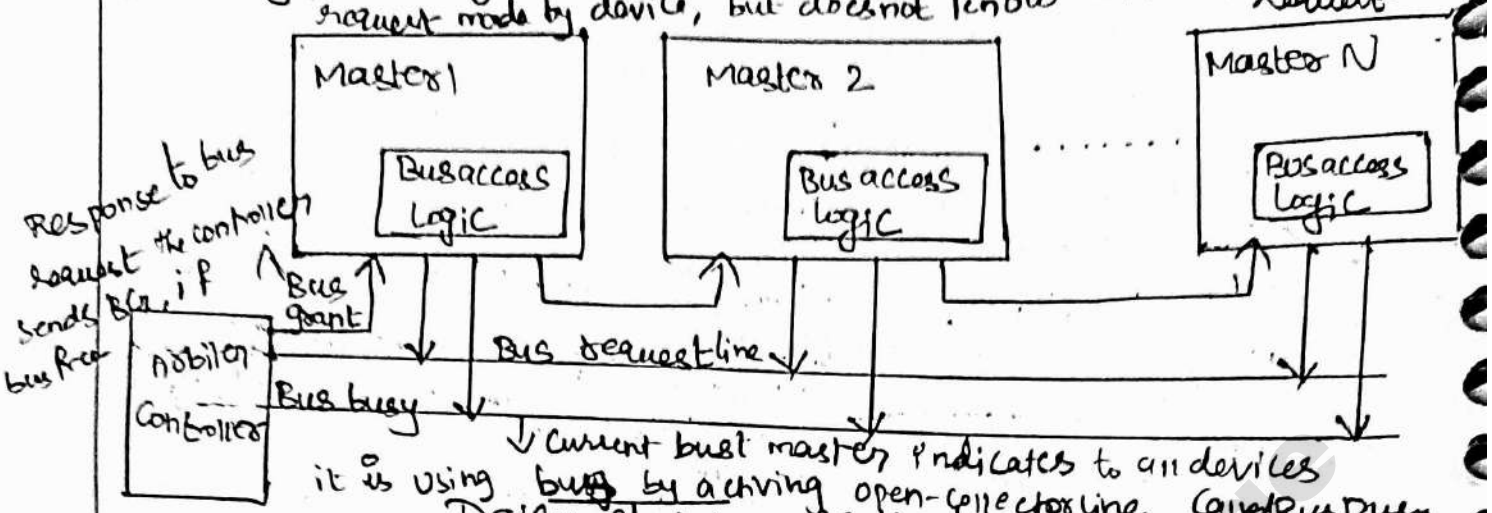
1) Centralized Bus Arbitration Schemes:

1) higher reliability 2) operation of bus is not dependent on any single device

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: BR is wired OR line, controller only request made by device, but does not know which device made the request



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.

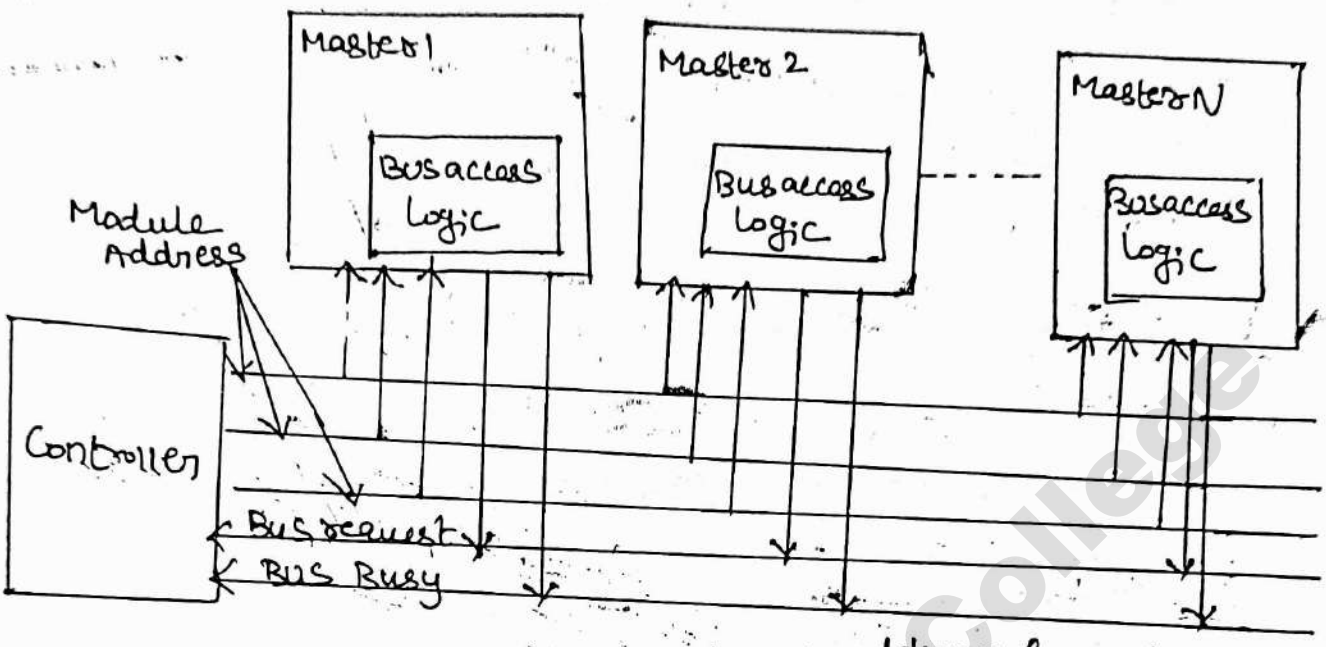
Advantages:

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: - All masters use same line for Bus request.



* In this the Controller is used to generate the addresses for the masters. (assign addresses for masters)

(All masters connected a local address bus).

* 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: * In response to bus request, the controller places the address of the bus master on address bus.

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.

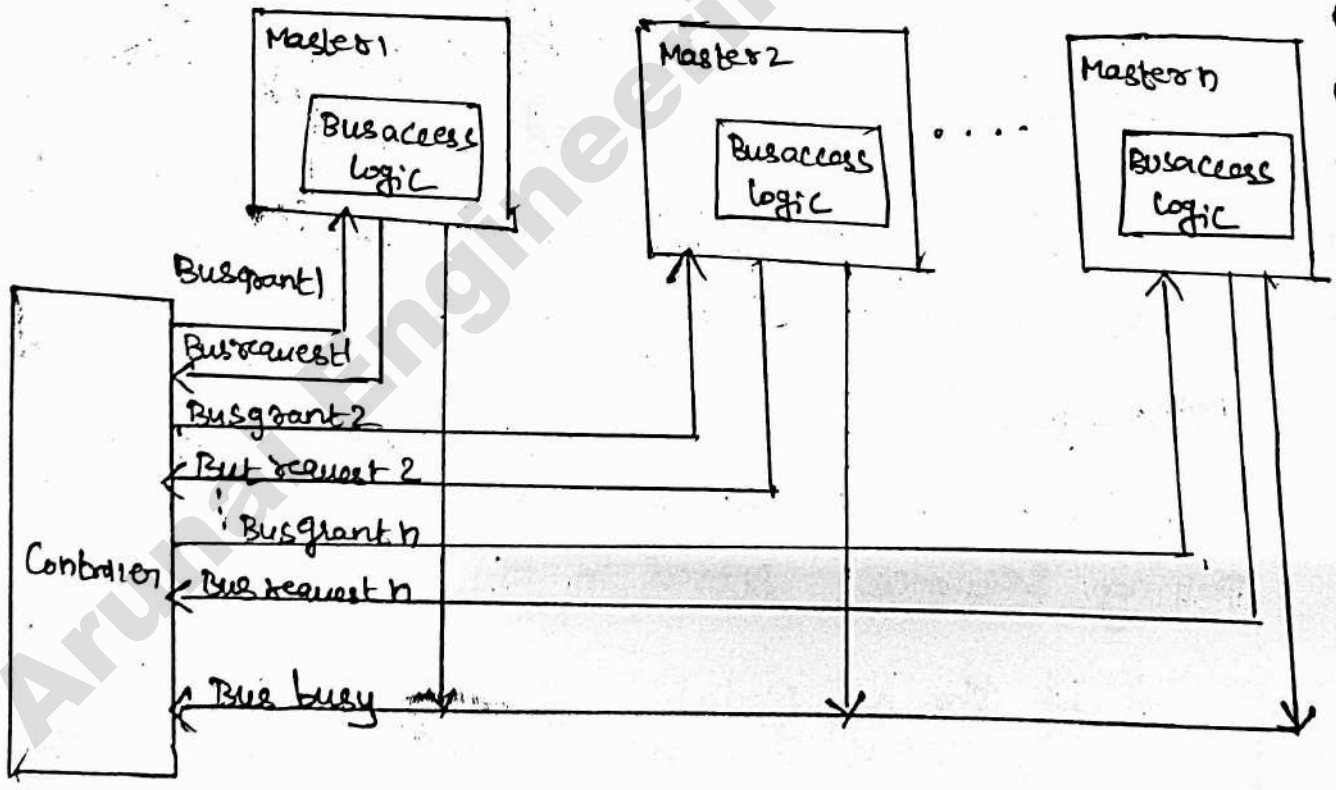
* The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

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

5 Elaborate on the various Memory technologies and its relevance. [APRIMAY-2015]

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. Low density, high power, expensive fast

* 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 hold only one bit data

* 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. * Address not divided * use for caches

* MOST PCs and server systems used separate SRAM chips for either their primary, secondary or even caches

Fast Access

2. DRAM Technology: - high density, low power, cheap, slow

* 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.

Big memory

* Because it uses only a single transistor per bit of storage they are much denser and cheaper than SRAM.

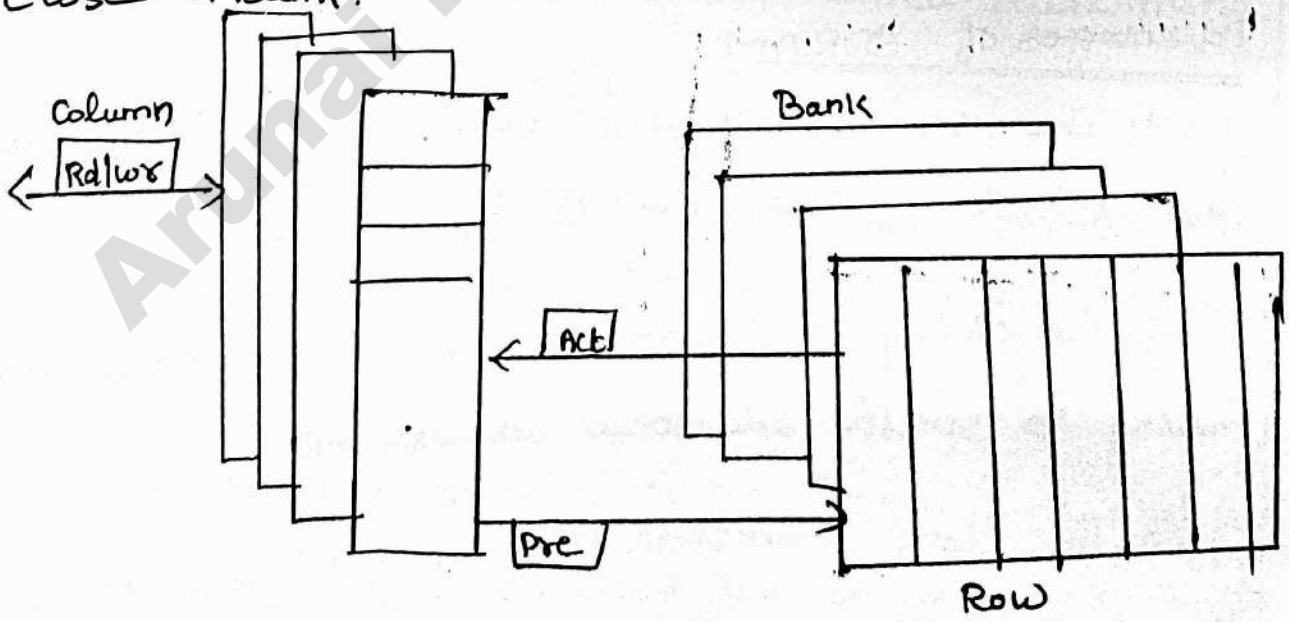
* DRAM'S Store all the charge on a capacitor So it cannot be kept indefinitely and must be periodically refreshed.

* 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. (Two halves of address Row/Colm) data periodically, to retain the data used for main memory

* 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 charge) 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 address.

* Each command as well as block transfers are synchronized with a clock.

* To improve performance DRAM's buffer rows for repeated access.

* TO improved the interface to processor's DRAM's added clocks with its and are called Synchronous DRAM's or SDRAM's - synchronously 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.

3. 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 trodden 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

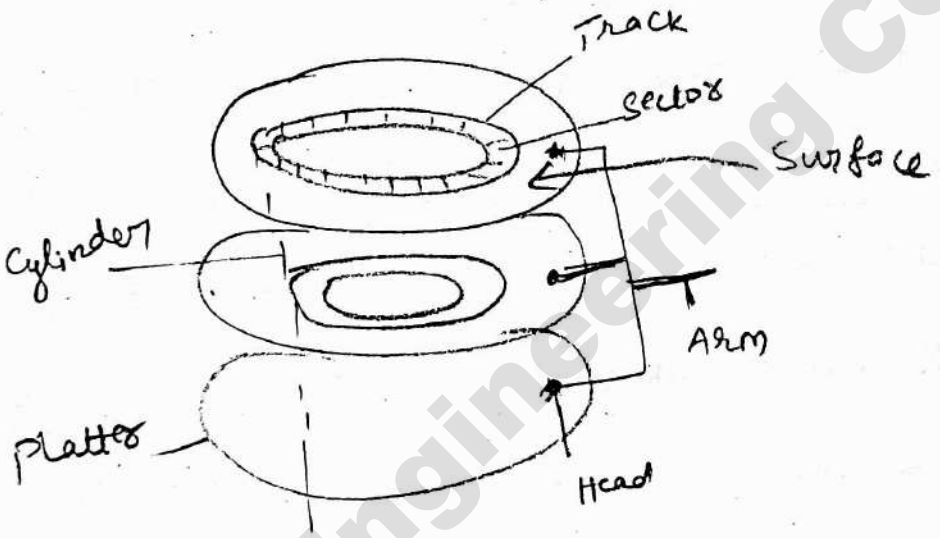
1. To position the head over the proper track. This operation is called seek and time to move the head to the desired track is called seek time

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 caped 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.



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

[NOV/DEC-17]

Solution:

* The number of memory miss cycles for instructions in terms of the instruction Count (I) is

2 %/100 = 0.02
0.02 X 100 = 2.00

Instruction miss cycles = I X 2% X 100 = 2.00 X I

* As the frequency of all loads and stores is 36%, we can find the number of memory miss cycles for data references:

36%/100 = 0.36
4%/100 = 0.04
0.36 X 0.04 = 0.0144 X 100 = 1.44

Data miss cycles = I X 36% X 4% X 100 = 1.44 X I

* The total number of memory stall cycles is 2.00I + 1.44I = 3.44I.

* This is more than three cycles of memory stall per instruction.

* Accordingly, the total CPI including memory stalls is $2 + 3.44 = 5.44$.

* Since there is no change in instruction count or clock rate, the ratio of the CPU execution time is

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

$$= \frac{CPI_{stall}}{CPI_{perfect}}$$

$$= \frac{5.44}{2}$$

* The Performance with the Perfect Cache is better by $\} = \frac{5.44}{2} = 2.72$

Suppose, speed-up the computer in the example by reducing its CPI from 2 to 1 without changing the clock rate, which might be done with an improved Pipeline.

* The System with cache misses would then have a CPI of $1 + 3.44 = 4.44$.

* The system with the perfect cache would be

$$\frac{4.44}{1} = 4.44 \text{ times as fast.}$$

* The amount of execution time spent on memory stalls would have risen from

$$\boxed{\frac{3.44}{5.44} = 63\%}$$

to

$$\frac{3.44}{4.44} = 77\%$$

Similarly, increasing the clock rate without changing the memory system also increases the performance lost due to cache misses.

7 Discuss about Programmed I/Os associated with Computers. [APR/MAY-18]

Programmed IO:

* 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.

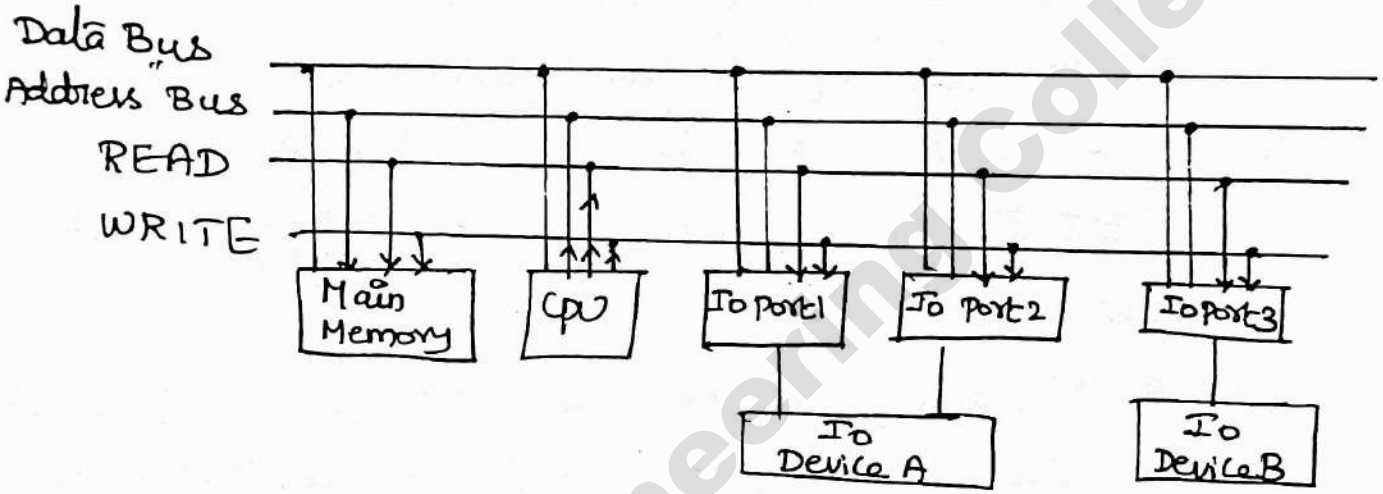
* A memory-referencing instructions becomes Io instructions automatically, if the address is address of an Io port.

* 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



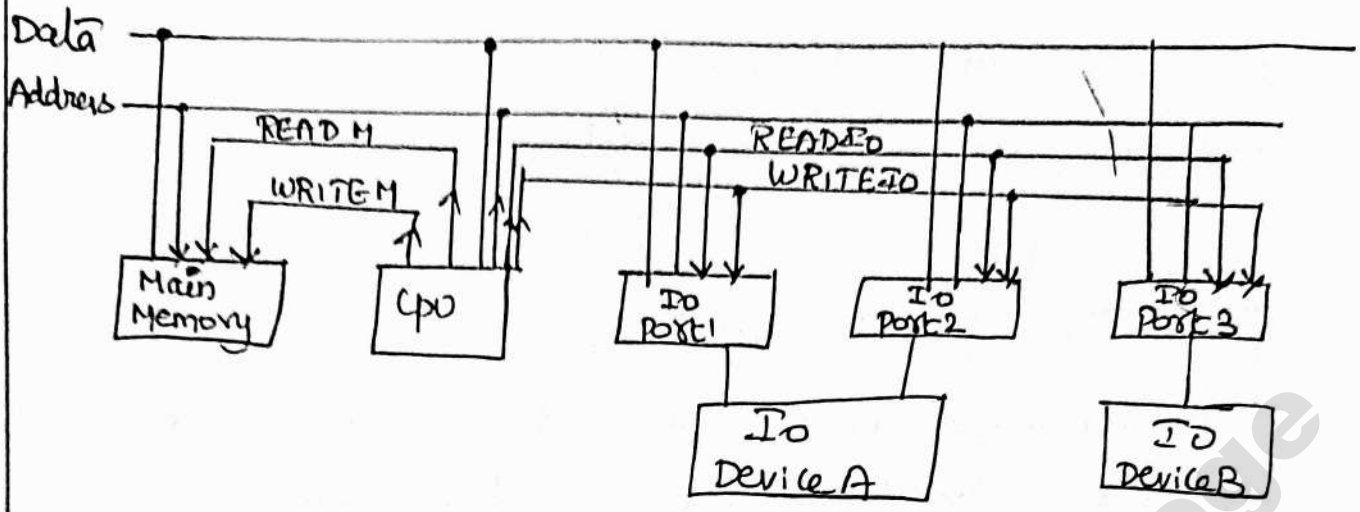
Programmed Io with Memory-Mapped Io

* 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.

8) Write the sequence of operations carried out by a processor when interrupted by a peripheral device connected to it. [Apr/May-18] [Nov/Dec-18]

Interrupts:

* An exceptional event that causes the CPU to temporarily transfer control from its current program to another program is called an Interrupt.

* Interrupts are the primary means by which IO devices obtain the services of CPU.

* In Programmed IO, the CPU is continuously checking the status of the IO device still that is ready to transfer the data.

* But, interrupt frees the CPU from the need for checking the status of its IO devices.

* There are various sources of Interrupts both Internal and external to the CPU.

* IO Interrupts are external to CPU.

* Interrupts can also be categorized as

i) Hardware Interrupts

ii) Software Interrupts

Eg: Software generated interrupts are, attempt by an instruction to divided by zero, to execute a privileged instruction.

Interrupt Handling Mechanism:

* The basic method of interrupting the CPU is activating a special control line called INTERRUPT REQUEST which connects the interrupt source to the CPU.

* An Interrupt indicator is then stored in a CPU register that can be checked by the CPU periodically.

* CPU does this checking at the end of every instruction cycle.

* If an interrupt is identified, the CPU execute a specific interrupt-handling program.

* Each Interrupt source has its own interrupt service routines.

* So, the CPU must determine the address of the interrupt program to be executed.

* If more than one interrupts requests at the same time, then priorities must be assigned and the interrupt with highest priority is selected for handling.

The following steps are taken while handling the Interrupts:

1. Io device enables the INTERRUPT REQUEST control line.

2. Interrupt indicator is enabled in the CPU register.

3. The CPU identifies the source of Interrupt.

4. The CPU obtains the memory address of the required Interrupt handler.

* Mostly this address is provided by the Interrupting device along with its interrupt request.

5. The Program Counter(PC) and other Processor registers are saved in Stack as a Subroutine call.

6. The PC is loaded with the address of the interrupt handler. Execution proceeds until return instruction is reached. Immediately, the control is transferred back to the interrupted program.

9 Define Translation Lookaside Buffer (TLB). what is its use? [NOV/DEC-18]

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.

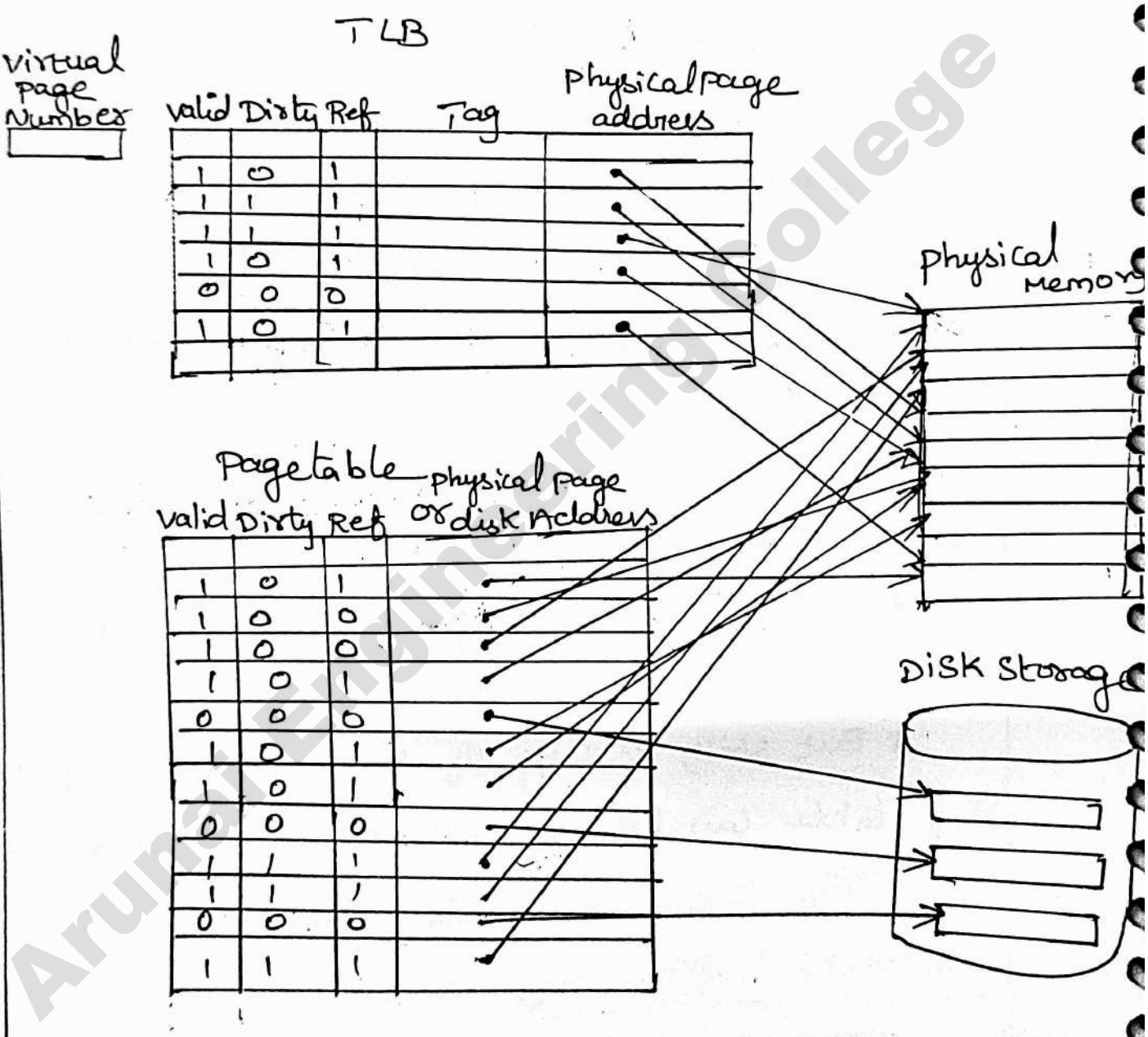
* But still, a copy of small portion of the page table can be accommodated within MMU.

* 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)



* 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.

* In this case, the processor invokes the operating system using an exception.