
	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Introduction to Unit I – Overview & Instructions		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No. 1 / 12	

1.CONTENT LIST:

Introduction to Unit I – Overview & Instructions

2. SKILLS ADDRESSED:

Listening

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basic concepts of Computer and its components.

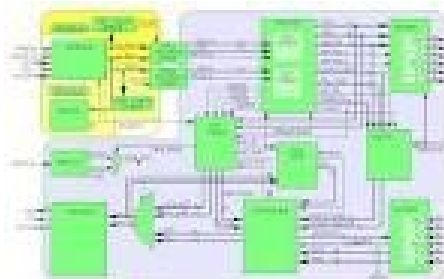
4.OUTCOMES:

- i. Explain the concept of components of computer system
- ii. Know the instructions and addressing modes
- iii.

5.LINK SHEET:

- i. What are the components of Computer system?
- ii. What are the topics covered in overview and instructions
- iii.

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)

Embedded Computer: Performs single function on a microprocessor

- Embedded within a product (e.g. microwave, car, cell phone)
- Objective: Low cost
- Increasingly written in a hardware description language, like Verilog or VHDL
- Processor core allows application-specific hardware to be fabricated on a single chip.

Desktop Computer: Designed for individual use

- Also called personal computer, workstation

Server: Runs large, specialized program(s)

- Shared by many users: more memory, higher speed, better reliability
- Accessed via a network using a request-response (client-server) interface
- Example: File server, Database server, Web server

Supercomputer: Massive computing resources and memory

- Hundreds to thousands of processors within single computer
- Terabytes of memory
- Program uses multiple processors simultaneously
- Rare due to extreme expense
- Applications: Weather forecasting, military simulations, etc.

What types of applications are concerned about:

- Memory?
- Processing speed?
- Usability?
- Maintainability?

How can the following impact performance?

- A selected algorithm?
- A programming language?
- A compiler?
- An operating system?
- A processor?
- I/O system/devices?

Computer Architect must balance speed and cost across the system

- System is measured against specification
- Benchmark programs measure performance of systems/subsystems
- Subsystems are designed to be in balance between each other

Usage:

- Normal: Data communications, time, clock frequencies
- Power of 2: Memory (often)

Memory units:

- **Bit (b):** 1 binary digit
- **Nibble:** 4 binary digits
- **Byte (B):** 8 binary digits
- **Word:** Commonly 32 binary digits (but may be 64).
- **Half Word:** Half the binary digits of a word
- **Double Word:** Double the binary digits of a word

Common Use:

- 10 Mbps = 10 Mb/s = 10 Megabits per second
- 10 MB = 10 Megabytes
- 10 MIPS = 10 Million Instructions Per Second

Moore's Law:

- Component density increase per year: 1.6
- Processor performance increase: 1.5 more recently 1.2 and < 1.2
- Memory capacity improvement: 4/3: 1.33

Tradeoffs in Power versus Clock Rate

- Faster Clock Rate = Faster processing = More power
- More transistors = More complexity = More power

Example Problems:

A disk operates at 7200 Revolutions per minute (RPM). How long does it take to revolve once?

$$\frac{7200 \text{ Revs}}{60 \text{ seconds}} = \frac{1 \text{ Rev}}{x \text{ secs}}$$

$$7200/60 x = 1$$

$$120x = 1$$

$$x = 1/120 = 0.00833 \text{ second} = 8.33 \text{ milliseconds or } 8.33 \text{ ms}$$

A disk holds 600 GB. How many bytes does it hold?

$$600 \text{ GB} = 600 \times 2^{30} = 600 \times 1,073,741,824 = 644,245,094,400$$

A LAN operates at 10 Mbps. How long will it take to transfer a packet of 1000 bytes?

(Optimistically assuming 100% efficiency)

$$\frac{10 \text{ Mb}}{1 \text{ sec}} = \frac{8 \text{ bits}}{x \text{ sec}}$$

$$10,000,000x = 8$$

$$x = 8/10,000,000 = 0.000,000,8 = 800\text{ns}$$

$$1000 \times 800 \text{ ns} = 800\text{us}$$

$$\frac{10 \text{ Mb}}{1 \text{ sec}} = \frac{8000}{x \text{ sec}}$$

$$10,000,000x = 8000$$

$$x = 8000/10,000,000 = 8/10,000$$

$$x = 0.0008 = 800\text{us}$$



8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system

Unit 1 – Lesson No. 2/ 12

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Eight ideas		
Time:	45 Minutes	
Lesson. No		

1.CONTENT LIST:

Eight ideas

2. SKILLS ADDRESSED:

Learning Understanding

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students know the eight ideas of computer system

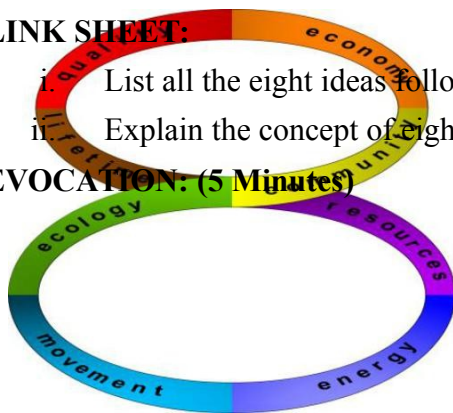
4.OUTCOMES:

- Learn the basics underlined in eight ideas
- Remember the basic concepts of eight ideas

5.LINK SHEET:

- List all the eight ideas followed to design computer system?
- Explain the concept of eight ideas

6.EVOCATION: (5 Minutes)



7. Lecture Notes (40 minutes)

□ **Design for *Moore's Law***

one constant for computer designers is rapid change, which is driven largely by Moore's Law. It states that integrated circuit resources double every 18–24 months. Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel. As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project. Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts. We use an "up and to the right" Moore's Law graph to represent designing for rapid change.



□ **Use *abstraction* to simplify design**

. Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law. A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels. We'll use the abstract painting icon to represent this second great idea



□ **Make the *common case fast***

. Making the common case fast will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is often easier to enhance. This common sense advice implies that you know what the common case is, which is only possible with careful experimentation and measurement. We use a sports car as the icon for making the common case fast, as the most common trip has one or two passengers, and it's surely easier to make a fast sports car than a fast minivan



□ **Performance *via parallelism***

Since the dawn of computing, computer architects have offered designs that get more performance by performing operations in parallel. We'll see many examples of parallelism in this book. We use multiple jet engines of a plane as our icon for parallel performance.



▮ **Performance via *pipelining***

Following the saying that it can be better to ask for forgiveness than to ask for permission, the next great idea is prediction. In some cases it can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a misprediction is not too expensive and your prediction is relatively accurate. We use the fortune-teller's crystal ball as our prediction icon.



Performance via *prediction*

A particular pattern of parallelism is so prevalent in computer architecture that it merits its own name: pipelining. For example, before fire engines, a "bucket brigade" would respond to a fire, which many cowboy movies show in response to a dastardly act by the villain. The townsfolk form a human chain to carry a water source to fire, as they could much more quickly move buckets up the chain instead of individuals running back and forth. Our pipeline icon is a sequence of pipes, with each section representing one stage of the pipeline.



▮ ***Hierarchy of memories***

Programmers want memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is often the majority of computer cost. Architects have found that they can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom. Caches give the programmer the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy. We use a layered triangle icon to represent the memory hierarchy. The shape indicates speed, cost, and size: the closer to the top, the faster and

more expensive per bit the memory; the wider the base of the layer, the bigger the memory.



□ ***Dependability via redundancy***

Computers not only need to be fast; they need to be dependable. Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures. We use the tractor-trailer as our icon, since the dual tires on each side of its rear axels allow the truck to continue driving even when one tire fails. (Presumably, the truck driver heads immediately to a repair facility so the flat tire can be fixed, thereby restoring redundancy!)





8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Components of a computer system		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No. 3/ 12	

1.CONTENT LIST:

Components of a computer system

2. SKILLS ADDRESSED:

Learning Remembering

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students know the main components of computer system

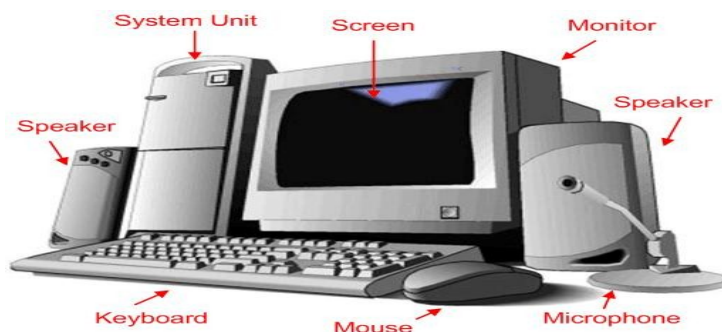
4.OUTCOMES:

- Learn the basics of computer components
- Remember operation of computer components

5.LINK SHEET:

- List the components of computer system?
- Sketch the components of computer system with neat diagram.
- Explain the operation of all the components of computer system.

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)

Computer components include:

- Input: keyboard, mouse, network, disk
- Output: printer, video screen, network, disk
- Memory: DRAM, magnetic disk
- CPU: Intelligence: Includes Datapath and Control

Input/Output

Mouse:

- Electromechanical: Rolling ball indicates change in position as (x,y) coordinates.
- Optical: Camera samples 1500 times per second. Optical processor compares images and determines distance moved.

Displays:

Raster Refresh Buffer: Holds the bitmap or matrix of pixel values.

- Matrix of Pixels: low resolution: 512 x 340 pixels to high resolution: 2560 x 1600 pixels
 - Black & White: 1 bit per pixel
 - Grayscale: 8 bits per pixel
 - Color: (one method): 8 bits each for red, blue, green = 24 bits
- Required: Refresh the screen periodically to avoid flickering

Two types of Displays:

Cathode Ray Tube (CRT): Pixel is source of light

- Scans one line at a time with a refresh rate of 30-75 times per second

Liquid Crystal Display (LCD): LCD pixel control or bends the light for the display.

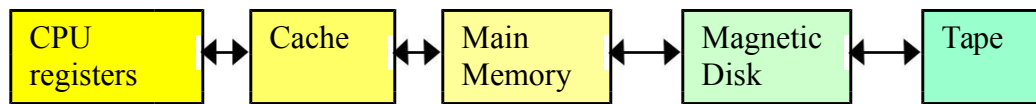
- Color active matrix LCD: Three transistor switches per pixel

Networking: Communications between computers

Local Area Network (LAN): A network which spans a small area: within a building

Wide Area Network (WAN): A network which extends hundreds of miles; typically managed by a communications service provider

Memory Hierarchy:



Fast, expensive, volatile

Slow, cheap, non-volatile

Secondary Memory: Nonvolatile memory used to store programs and data when not running

- Nonvolatile: Does not lose data when powered off
- Includes:
 - Magnetic Disk: Access time: 5-15 ms
 - Tape: Sometimes used for backup
 - Optical Disk: CD or DVD
 - FLASH: Removable memory cards attach via USB
 - Floppy and Zip: Removable form of magnetic disk

Magnetic Disk: Movable arm moves to concentric circle then writes

- Disk diameter: 1 to 3.5 inches
- Latency: Moving head to ‘cylinder’ or concentric track
- Rotation Time: Rotating cylinder to correct location on ‘track’
- Transfer Time: Reading or writing to disk on ‘track’
- Access time: 5-20 ms

Optical Disk: Laser uses spiral pattern to write bits as pits or flats.

- Compact Disc (CD): Stores music
- Digital Versatile Disc (DVD): Multi-gigabyte capacity required for films
- Read-write procedure similar to Magnetic Disk (but optical write, not magnetic)

Flash Memory: Semiconductor memory is nonvolatile

- More expensive than disk, but also more rugged and faster latency.
- Good for 100,000-1M writes.
- Common in cameras, portable music players, memory sticks

Primary or Main Memory: Programs are retained while they are running. Uses:

- Dynamic Random Access Memory (**DRAM**)
 - Built as an integrated circuit, equal speed to any location in memory
 - Access time: 50-70 ns.
- **SIMM** (Single In-line Memory Module): DRAM memory chips lined up in a row, often on a daughter card
- **DIMM** (Dual In-line Memory Module): Two rows of memory chips
- **ROM** (Read Only Memory) or **EPROM** (Erasable Programmable ROM)

Cache: Buffer to the slower, larger main memory. Uses:

- Static Random Access Memory (**SRAM**)
- Faster, less dense and more expensive than DRAM
 - Uses multiple transistors per bit instead of the single transistor for DRAM

Registers: Fastest memory within the CPU.

Central Processing Unit (CPU) or Processor: Intelligence

- **Data Path:** Performs arithmetic operations using registers
- **Control:** Management of flow of information through the data path

Bus: Connects the CPU, Memory, I/O Devices

- Bits are transmitted between the CPU, Memory, I/O Devices in a timeshared way
- Serial buses transmit one bit at a time.
- Parallel buses transmit many bits simultaneously: one bit per line

One bus system: Memory, CPU, I/O Subsystem on same bus

Two bus system:

- One bus: CPU \longleftrightarrow Memory
- One bus: CPU \longleftrightarrow I/O Subsystem

Example: Universal Serial Bus (USB 2.0)



- Hot-pluggable: can be plugged and unplugged without damage to the system
- Operates at 0.2, 1.5 or 60 MB/sec
- Can interface to printer or other slow devices

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system, Notebook Computers, Handheld Computers

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Technology		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No.4/ 12	

1.CONTENT LIST:

Technology

2. SKILLS ADDRESSED:

Listening Understanding

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students know the technology involved in computer architecture

4.OUTCOMES:

- Learn the technology involved in computer system
- Remember the different types of technology

5.LINK SHEET:

- Explain the technology involved in computer architecture
- Discuss the types of technology in detail.

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)

Embedded Computer: Performs single function on a microprocessor

- Embedded within a product (e.g. microwave, car, cell phone)
- Objective: Low cost
- Increasingly written in a hardware description language, like Verilog or VHDL
- Processor core allows application-specific hardware to be fabricated on a single chip.

Desktop Computer: Designed for individual use

- Also called personal computer, workstation

Server: Runs large, specialized program(s)

- Shared by many users: more memory, higher speed, better reliability
- Accessed via a network using a request-response (client-server) interface
- Example: File server, Database server, Web server

Supercomputer: Massive computing resources and memory

- Hundreds to thousands of processors within single computer
- Terabytes of memory
- Program uses multiple processors simultaneously
- Rare due to extreme expense
- Applications: Weather forecasting, military simulations, etc.

CPUs

Device density: 2x every 1.5 years (~60% per year)

Latency: 2x every 5 years (~15% per year)

Memory (DRAM)

Capacity: 4x every 3 years (~60% per year)

(2x every two years lately)

Latency: 1.5x every 10 years

Cost per bit: decreases about 25% per year

Hard drives:

Capacity: 4x every 3 years (~60% per year)

Bandwidth: 2.5x every 4 years

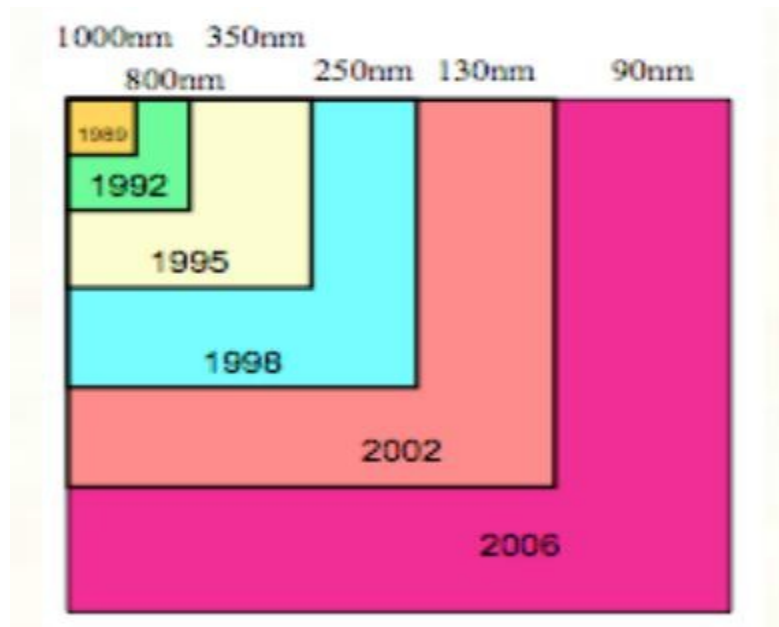
Latency: 2x every 5 years

Boards:

Wire density: 2x every 15 years

Cables:

No change



Physical Hardware

Semiconductor: Conducts electricity poorly

Silicon Ingot: Made of silicon: substance found in sand

Wafer: Ingot is sliced into 0.1-inch blank wafers

Processing: Add materials to wafers: conductor, insulator, or **transistors**: (on/off) switch

Diced: Wafers are cut into smaller components called **dies** or **chips**

Yield: Wafers/dies are tested providing a % success rate. Failures are discarded

Bonding: The chip is connected to the input/output pins of a package

Integrated Circuit = chip: A device containing up to millions of transistors

Very Large Scale Integrated Circuit (VLSI): A device containing hundreds of thousands to millions of transistors

Computer chassis vocabulary:



- **Motherboard:** Holds the processor, system bus, various interfaces and connectors
- **Daughter card:** Small printed circuit board, often contains multiple memory chips.
- **Cage or Chassis:** Holds multiple boards
- **Backplane:** Contains bus interface for boards to communicate
- **3D Packaging:** Transistors interconnect above, beside, below (3D)

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Memory, Processing speed, Usability, Maintainability

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Performance and Powerwall		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No.5/ 12	

1.CONTENT LIST:

Performance and Powerwall

2. SKILLS ADDRESSED:

Learning Understanding

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the performance and Powerwall in computer architecture

4.OUTCOMES:

- Learn the performance of computer architecture
- Understand the power consumption and Powerwall in computer architecture

5.LINK SHEET:

- List the performance factors to design computer system
- Explain the performance involved in computer architecture
- Discuss the types of power wall in detail.

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)



Purchasing perspective

- given a collection of machines, which has the
 - best performance ?
 - least cost ?
 - best cost/performance?



Design perspective

- faced with design options, which has the
 - best performance improvement ?
 - least cost ?
 - best cost/performance?



Both require

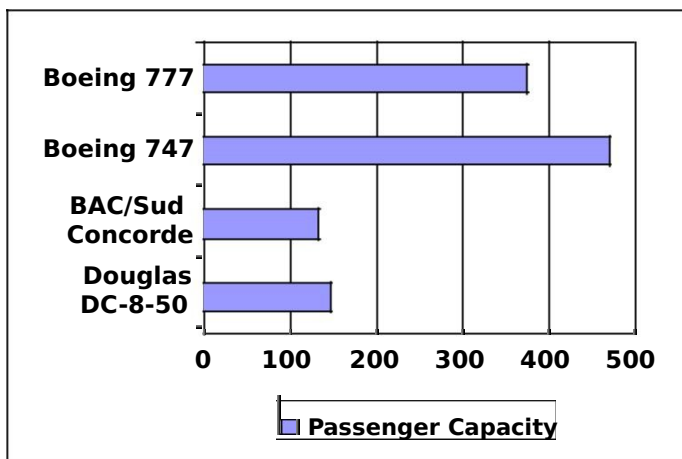
- basis for comparison
- metric for evaluation



Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors



Which airplane has the best performance?



Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time
 - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by

- Replacing the processor with a faster version?
- Adding more processors?

Relative Performance

Define Performance = $1/\text{Execution Time}$

—X is n time faster than Y

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

- Example: time taken to run a program

-

10s on A, 15s on B

-

$\text{Execution Time}_B / \text{Execution Time}_A = 15s / 10s = 1.5$

So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time

- Total response time, including all aspects

- Processing, I/O, OS overhead, idle time

- Determines system performance

- CPU time

- Time spent processing a given job

- Discounts I/O time, other jobs' shares

- Comprises user CPU time and system CPU time

- Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock

- Clock period: duration of a clock cycle

-

e.g., $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$

- Clock frequency (rate): cycles per second

-

e.g., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by

- Reducing number of clock cycles

- Increasing clock rate

- Hardware designer must often trade off clock rate against cycle count

Clock Cycles	Instruction Count	μCycles per Instruction
CPU Time	Instruction Count	μCPI
	Instruction Count	μClock Cycle Time
	μCPI Clock Rate	



- Instruction Count for a program
 - Determined by program, ISA and compiler
- Average cycles per instruction
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Uniprocessors to multiprocessors		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No.6/ 12	

1.CONTENT LIST:

Uniprocessors to multiprocessors

2. SKILLS ADDRESSED:

Learning Remembering

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students know the history of computer system

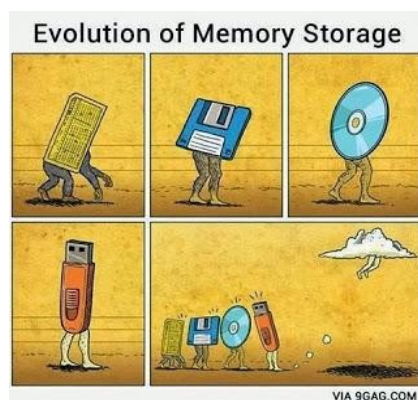
4.OUTCOMES:

- Learn the history of computer architecture
- Understand the advantages of transformation from uniprocessors to multiprocessors

5.LINK SHEET:

- Explain the evolution of computer architecture.
- List the nature of uniprocessors.
- Discuss the transformation of uniprocessors to multiprocessors.

6.EVOCATION: (5 Minutes)



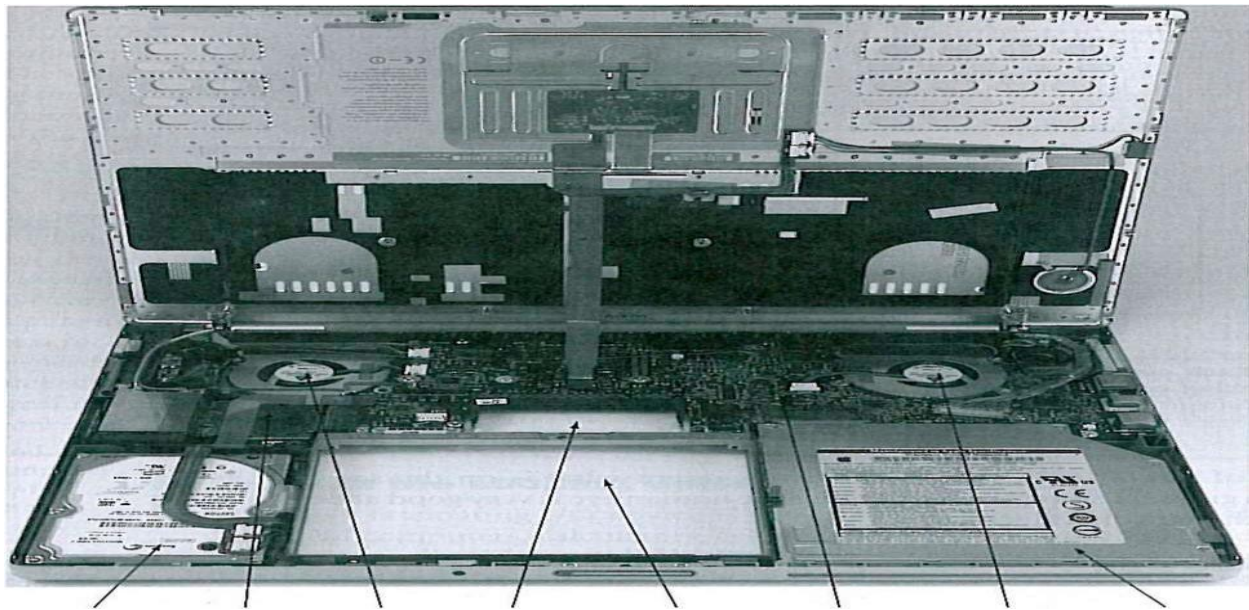
7.Lecture Notes: (40 Minutes)

If we open the box containing the computer, we see a fascinating board of thin plastic, covered with dozens of small gray or black rectangles.

The motherboard is shown in the upper part of the photo. Two disk drives are in front—the hard drive on the left and a DVD drive on the right. The hole in the middle is for the laptop battery. The small rectangles on the motherboard contain the devices that drive our advancing technology, called integrated circuits and nicknamed chips.

The board is composed of three pieces: the piece connecting to the I/O devices mentioned earlier, the memory, and the processor.

The memory is where the programs are kept when they are running; it also contains the data needed by the running programs. Figure 1.8 shows that memory is found on the two small boards, and each small memory board contains eight integrated circuits \



The *processor* is the active part of the board, following the instructions of a program to the letter. It adds numbers, tests numbers, signals I/O devices to activate, and so on.. Occasionally, people call the processor the CPU, for the more bureaucratic-sounding central processor unit.

Descending even lower into the hardware, The processor logically comprises two main components: datapath and control, the respective brawn and brain of the processor.

The datapath performs the arithmetic operations, and control tells the datapath, memory, and I/O devices what to do according to the wishes of the instructions of the program. This explains the datapath and control for a higher-performance design Descending into the depths of any component of the hardware reveals insights into the computer. Inside the processor is another type of memory—cache memory.

Cache memory consists of a small, fast memory that acts as a buffer for the DRAM memory. (The nontechnical definition of *cache* is a safe place for hiding things.)

Cache is built using a different memory technology, **static random access memory (SRAM)**. SRAM is faster but less dense, and hence more expensive, than DRAM You may have noticed a common theme in both the software and the hardware descriptions: delving into the depths of hardware or software reveals more information or, conversely, lower-level details are hidden to



offer a simpler model at higher levels. The use of such layers, or **abstractions**, is a principal technique for designing very sophisticated computer systems.

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system, DVD

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Instructions – operations and operands		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No.7/ 12	

1.CONTENT LIST:

Instructions – operations and operands

2. SKILLS ADDRESSED:

Learning Remembering

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the instruction set and its operations

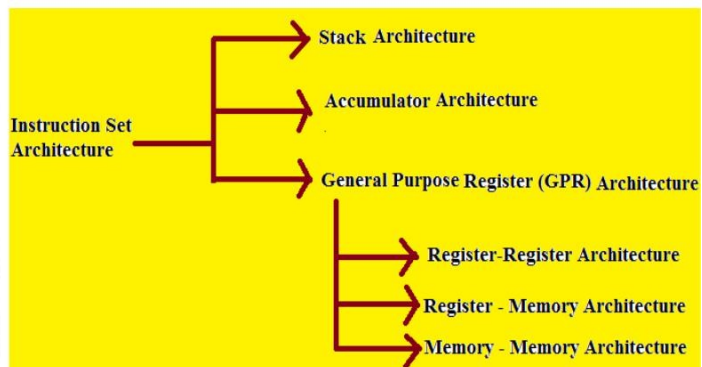
4.OUTCOMES:

- Learn the set of instructions involved in computer architecture
- know the operand and operations of instructions

5.LINK SHEET:

- List the instructions available in computer system
- Explain the operands and operations using instruction set

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)

INSTRUCTION AND INSTRUCTION SEQUENCING

A computer must have instruction capable of performing the following operations. They are,
Data transfer between memory and processor register.

Arithmetic and logical operations on data.

Program sequencing and control.

I/O transfer.

Register Transfer Notation:

The possible locations in which transfer of information occurs are, Memory Location

Processor register

Registers in I/O sub-system.

Location	Hardware Binary Address	Eg	Description
Memory	LOC,PLACE,A,VAR2	$R1 \leftarrow [LOC]$	The contents of memory location are transferred to. the processor register.
Processor	R0,R1,....	$[R3] \leftarrow [R1] + [R2]$	Add the contents of register R1 & R2 and places .their sum into register R3.It is .called Register Transfer Notation.
I/O Registers	DATAIN,DATAOUT		Provides Status information

Assembly Language Notation:

Assembly Language Format	Description
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Instruction Type	Syntax	Eg	Description
Three Address	Operation Source1,Source2,Destination	Add A,B,C	Add values of variable A ,B & place the result into c.
Two Address	Operation Source,Destination	Add A,B	Add the values of A,B & place the result into B.
One Address	Operation Operand	Add B	Content of accumulator add with content of B.

Instruction Execution and Straight–line Sequencing:

Instruction Execution:

There are 2 phases for Instruction Execution. They are, Instruction Fetch
Instruction Execution

Instruction Fetch:

The instruction is fetched from the memory location whose address is in PC.This is placed in IR.

Instruction Execution:

Instruction in IR is examined to determine whose operation is to be performed.

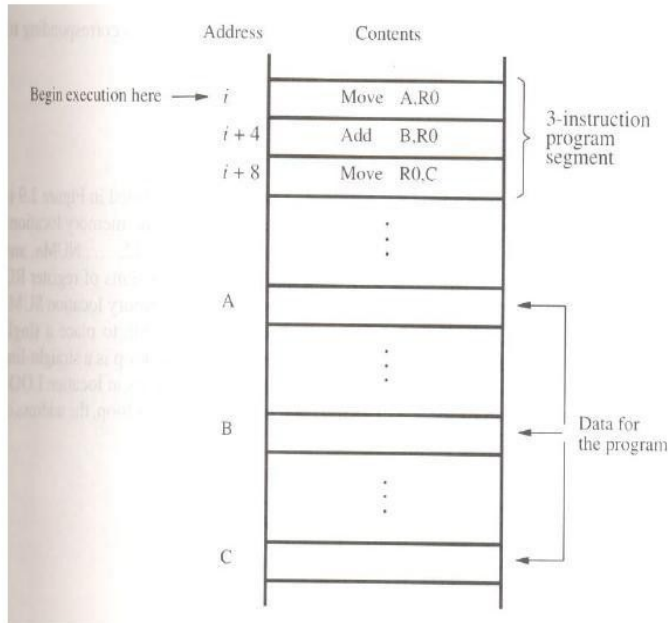
Program execution Steps:

To begin executing a program, the address of first instruction must be placed in PC.

The processor control circuits use the information in the PC to fetch & execute instructions one at a time in the order of increasing order.

This is called Straight line sequencing.During the execution of each instruction,the PC is incremented by 4 to point the address of next instruction.

Fig: Program Execution



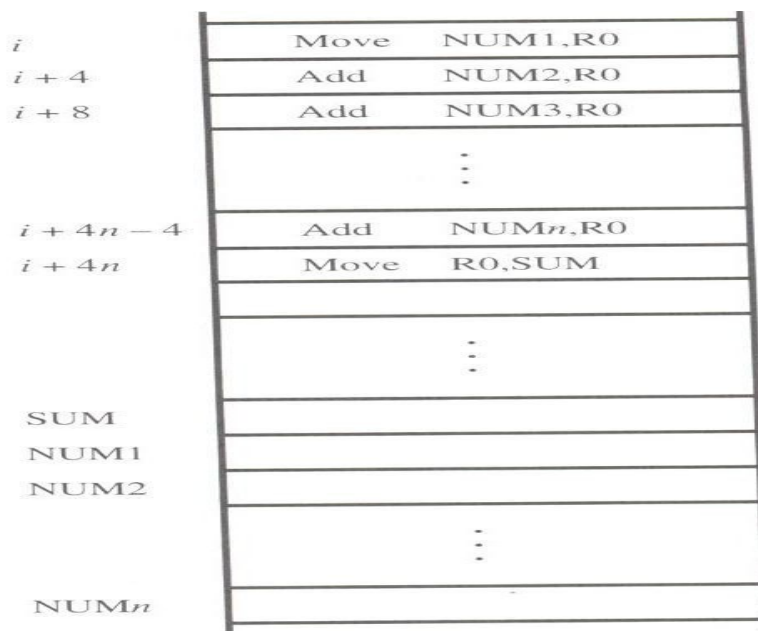
Branching:

The Address of the memory locations containing the n numbers are symbolically given as NUM1,NUM2.....NUM n .

Separate Add instruction is used to add each number to the contents of register R0.

After all the numbers have been added,the result is placed in memory location SUM.

Fig: Straight Line Sequencing Program for adding 'n' numbers



Using loop to add 'n' numbers:

Number of entries in the list „n“ is stored in memory location M. Register R1 is used as a counter to determine the number of times the loop is executed.

Content location M are loaded into register R1 at the beginning of the program.

It starts at location Loop and ends at the instruction Branch>0. During each pass, the address of the next list entry is determined and the entry is fetched and added to R0

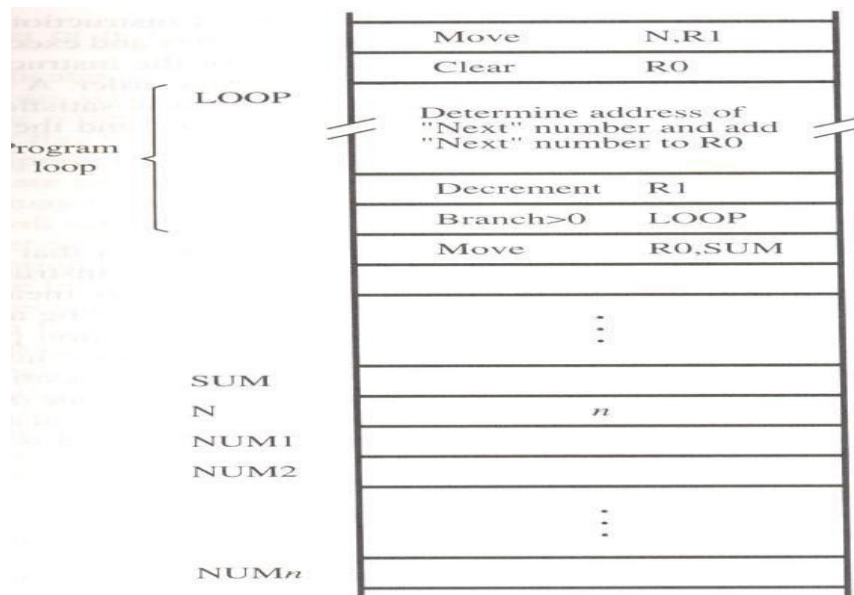
Decrement R1

It reduces the contents of R1 by 1 each time through the loop.

Branch >0 Loop

A conditional branch instruction causes a branch only if a specified condition is satisfied.

Fig:Using loop to add 'n' numbers:



Branch >0 Loop

Conditional Codes:

Result of various operation for user by subsequent conditional branch instruction is accomplished by recording the required information in individual bits often called

Condition code Flags.

Commonly used flags:

N (Negative) set to 1 if the result is -ve ,otherwise cleared to 0.

Z(Zero) set to 1 if the result is 0 ,otherwise cleared to 0.

V(Overflow) set to 1 if arithmetic overflow occurs,otherwise cleared to 0.



C(Carry)set to 1 if carry and results from the operation ,otherwise cleared to 0.

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system, Microchip, Intel cores

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Instructions – Representing instructions		
Time:	50 Minutes	
Lesson. No	Unit 1 – Lesson No.8/ 12	

1.CONTENT LIST:

Representing instructions

2. SKILLS ADDRESSED:

Learning understanding

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students know the representation of instructions

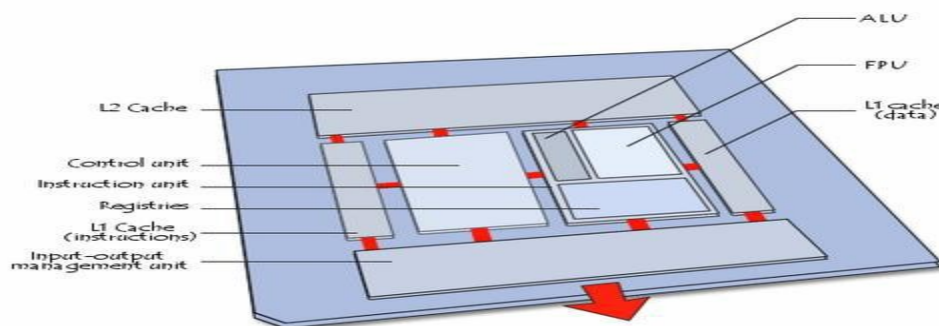
4.OUTCOMES:

- Learn the set of representation involve in computer architecture
- know the way of representing the instructions

5.LINK SHEET:

- Give the different types of representation involved in instruction set
- Discuss in detail the way to represent instruction set.

6.EVOCATION: (5 Minutes)



7.Lecture Notes: (40 Minutes)

INSTRUCTION AND INSTRUCTION SEQUENCING

A computer must have instruction capable of performing the following operations. They are,
Data transfer between memory and processor register.

Arithmetic and logical operations on data.

Program sequencing and control.

I/O transfer.

Register Transfer Notation:

The possible locations in which transfer of information occurs are, Memory Location

Processor register

Registers in I/O sub-system.

Location	Hardware Binary Address	Eg	Description
Memory	LOC,PLACE,A,VAR2	$R1 \leftarrow [LOC]$	The contents of memory location are transferred to. the processor register.
Processor	R0,R1,....	$[R3] \leftarrow [R1] + [R2]$	Add the contents of register R1 & R2 and places .their sum into register R3.It is .called Register Transfer Notation.
I/O Registers	DATAIN,DATAOUT		Provides Status information

Assembly Language Notation:

Assembly Language Format	Description
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Instruction Type	Syntax	Eg	Description
Three Address	Operation Source1,Source2,Destination	Add A,B,C	Add values of variable A ,B & place the result into c.
Two Address	Operation Source,Destination	Add A,B	Add the values of A,B & place the result into B.
One Address	Operation Operand	Add B	Content of accumulator add with content of B.

Instruction Execution and Straight–line Sequencing:

Instruction Execution:

There are 2 phases for Instruction Execution. They are, Instruction Fetch
Instruction Execution

Instruction Fetch:

The instruction is fetched from the memory location whose address is in PC.This is placed in IR.

Instruction Execution:

Instruction in IR is examined to determine whose operation is to be performed.

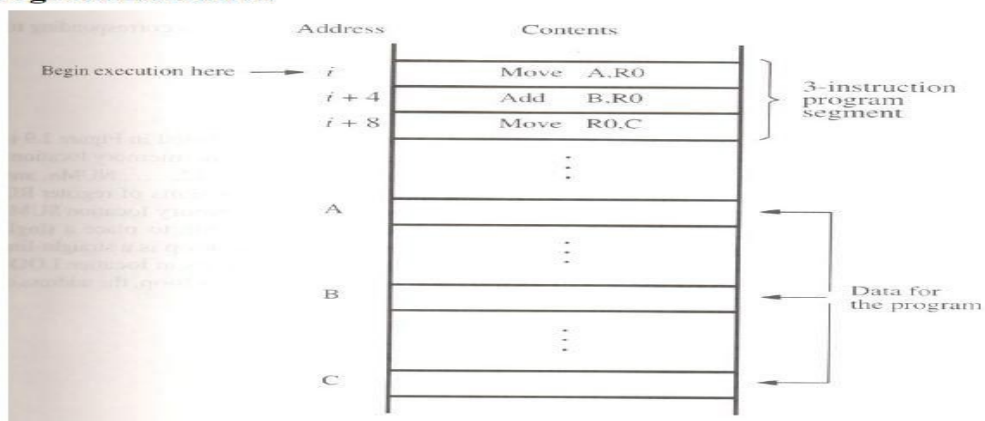
Program execution Steps:

To begin executing a program, the address of first instruction must be placed in PC.

The processor control circuits use the information in the PC to fetch & execute instructions one at a time in the order of increasing order.

This is called Straight line sequencing.During the execution of each instruction,the PC is incremented by 4 to point the address of next instruction.

Fig: Program Execution



Branching:

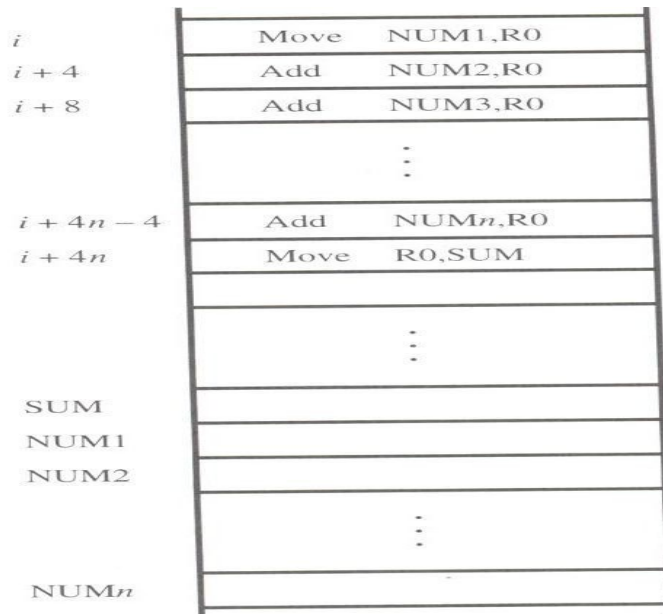
The Address of the memory locations containing the n numbers are symbolically given as NUM1,NUM2.....NUMn.

Separate Add instruction is used to add each number to the contents of register R0.

After all the numbers have been added, the result is placed in memory location SUM.

\

Fig: Straight Line Sequencing Program for adding 'n' numbers



Using loop to add 'n' numbers:

Number of entries in the list „n“ is stored in memory location M. Register R1 is used as a counter to determine the number of times the loop is executed.

Content location M are loaded into register R1 at the beginning of the program.

It starts at location Loop and ends at the instruction. Branch > 0. During each pass, the address of the next list entry is determined and the entry is fetched and added to R0

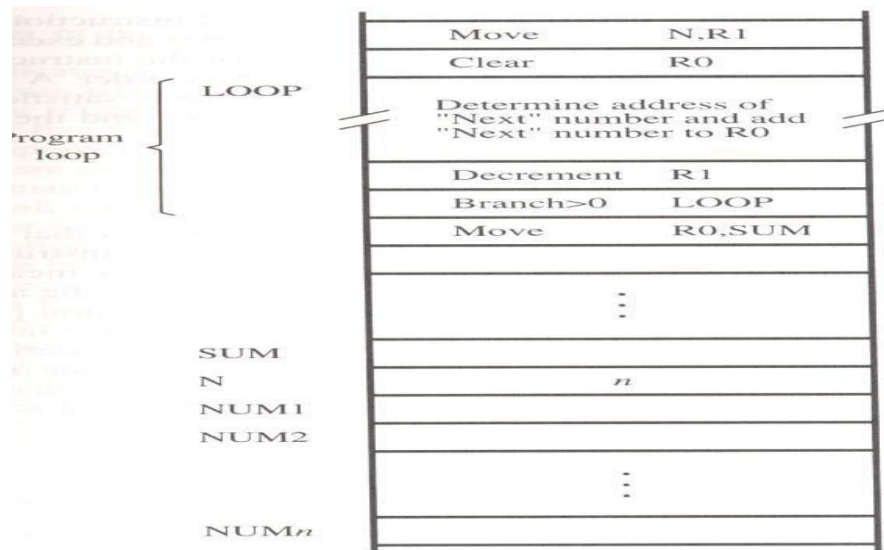
Decrement R1

It reduces the contents of R1 by 1 each time through the loop.

Branch > 0 Loop

A conditional branch instruction causes a branch only if a specified condition is satisfied.

Fig:Using loop to add 'n' numbers:



Branch >0 Loop

Conditional Codes:

Result of various operation for user by subsequent conditional branch instruction is accomplished by recording the required information in individual bits often called

Condition code Flags.

Commonly used flags:

N (Negative) set to 1 if the result is -ve ,otherwise cleared to 0.

Z(Zero) set to 1 if the result is 0 ,otherwise cleared to 0.

V(Overflow) set to 1 if arithmetic overflow occurs,otherwise cleared to 0.



C(Carry)set to 1 if carry and results from the operation ,otherwise cleared to 0.

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9.Application

Processor, Embedded system, intel core and microchip

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Logical operations – control operations		
Time:	45 Minutes	
Lesson. No	Unit 1 – Lesson No.9/ 12	

1.CONTENT LIST:

Logical operations – control operations

2. SKILLS ADDRESSED:

Learning Remembering

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students learn the logical and control operations

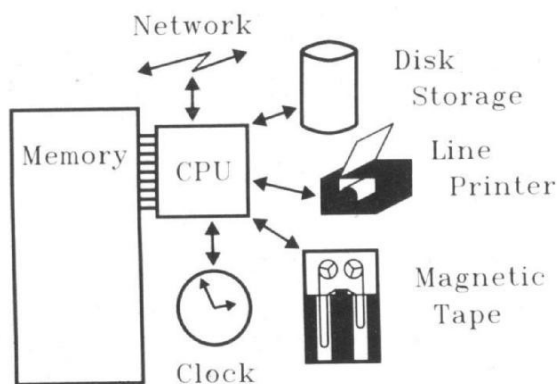
4.OUTCOMES:

- Learn the logical operations of computer system
- Understand the control operations of computer system

5.LINK SHEET:

- Explain the logical operations of computer.
- List the functions of control unit.
- Discuss in detail the operations of control unit.

6.EVOCATION: (5 Minutes)



Although the first computers operated on full words, it soon became clear that it

7.Lecture Notes: (40 Minutes)

Logical operations were useful to operate on fields of bits within a word or even on individual bits. Examining characters within a word, each of which is stored as 8 bits, is one example of such an operation (see Section 2.9).

It follows that operations were added to programming languages and instruction set architectures to simplify, among other things, the packing and unpacking of bits into words. These instructions are called logical operations. Figure 2.8 shows logical operations in C, Java, and MIPS.

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR			or, ori
Bit-by-bit NOT	~	~	nor

The first class of such operations is called *shifts*. They move all the bits in a word to the left or right, filling the emptied bits with 0s.

For example, if register \$s0 contained 0000 0000 0000 0000 0000 0000 0000 1001_{two} = 9_{ten} and the instruction to shift left by 4 was executed, the new value would be: 0000 0000 0000 0000 0000 1001 0000_{two} = 144_{ten}. The dual of a shift left is a shift right. The actual name of the two MIPS shift instructions are called *shift left logical* (sll) and *shift right logical* (srl).

To place a value into one of these seas of 0s, there is the dual to AND, called OR. It is a bit-by-bit operation that places a 1 in the result if *either* operand bit is a 1.

To elaborate, if the registers \$11 and \$12 are unchanged from the preceding example, the result of the MIPS instruction `or $t0 = reg $t1 | reg $t2` is this value in register \$t0:

NOT A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1. NOR

A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands.

That is, it calculates a 1 only if there is a 0 in *both* operands. 0000 0000 0000 0000 0011 1101 1100 0000_{two}. The final logical operation is a contrarian. NOT takes one operand and places a 1 in the result if one operand bit is a 0, and vice versa.

In keeping with the three-operand format, the designers of MIPS decided to include the instruction NOR (NOT OR) instead of NOT. If one operand is zero, then it is equivalent to NOT: $A \text{ NOR } 0 = \text{NOT}(A \text{ OR } 0) = \text{NOT}(A)$.

Control Unit

Makes all the other parts work together Uses a FSM (like our Traffic FSM but much bigger - many inputs/outputs - and more complicated)

Program Counter (PC)

- Tells control unit which instruction to execute next – Recall program is a sequence of instructions
- Holds address of next instruction (program is in memory)
- Normally, the next PC is the current PC plus one instruction

Instruction Register (IR)



- Holds the instruction currently being executed
- Decoded to feed signals to other units and inputs to FSM

8. Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. Application

Processor, Embedded system, Digital logic gates.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Addressing and addressing modes		
Time:	50 Minutes	
Lesson. No	Unit 1 – Lesson No.10, 11 / 12	

1.CONTENT LIST:

Addressing and addressing modes

2. SKILLS ADDRESSED:

Learning Understanding

3.OBJECTIVE OF THIS LESSON PLAN:

To make the students learn the addressing and addressing modes

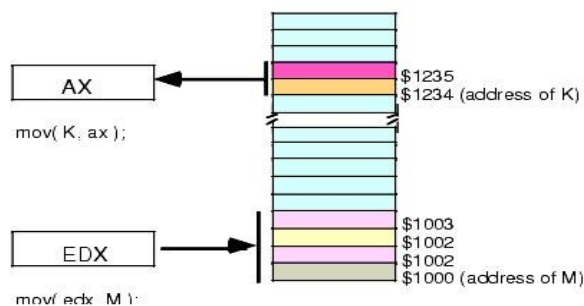
4.OUTCOMES:

- Learn the definition of addressing
- Understand the different modes of addressing

5.LINK SHEET:

- Define Addressing.
- What are the modes of addressing?
- Discuss in detail any type of addressing mode.

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

ADDRESSING MODES

The name(address) of the register is given in the instruction.

Absolute Mode(Direct Mode): The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

Generic Addressing Modes:

- Immediate mode

- Register mode

- Absolute mode

- Indirect mode

- Index mode

Base with index

- Base with index and offset

- Relative mode

- Auto-increment mode

- Auto-decrement mode

Implementation of Variables and Constants:

Variables:

The value can be changed as needed using the appropriate instructions.

There are 2 accessing modes to access the variables. They are

- Register Mode

- Absolute Mode

Register Mode:

The operand

The operand is in new location.

The address of this location is given explicitly in the instruction.

Eg: MOVE LOC,R2

The above instruction uses the register and absolute mode.

The processor register is the temporary storage where the data in the register are accessed using register mode.

The absolute mode can represent global variables in the program.

Mode Assembler Syntax Addressing Function

Register mode Ri EA=Ri

Absolute mode LOC EA=LOC

Where EA-Effective Address

Constants:

Address and data constants can be represented in assembly language using Immediate Mode.

Immediate Mode.

The operand is given explicitly in the instruction.

Eg: Move 200 immediate ,R0

It places the value 200 in the register R0.The immediate mode used to specify the value of source operand.

In assembly language, the immediate subscript is not appropriate so # symbol is used.

It can be re-written as

Move #200,R0

Assembly Syntax: Addressing Function

Immediate #value Operand =value

Indirection and Pointers:

Instruction does not give the operand or its address explicitly.Instead it provides information from which the new address of the operand can be determined.This address is called effective Address (EA) of the operand.

Indirect Mode:

The effective address of the operand is the contents of a register .

We denote the indirection by the name of the register or new address given in the instruction.

instruction.

Fig:Indirect Mode

Add (R1),R0
...
Operand

Add (A),R0
B
Operand

Address of an operand(B) is stored into R1 register.If we want this operand,we can get it through register R1(indirection).

The register or new location that contains the address of an operand is called the **pointer**.

Mode Assembler Syntax Addressing Function Indirect Ri , LOC EA=[Ri] or

EA=[LOC]

Indexing and Arrays:

Index Mode:

The effective address of an operand is generated by adding a constant value to the contents of a register.

The constant value uses either special purpose or general purpose register.

We indicate the index mode symbolically as, **X(Ri)**

Where **X** – denotes the constant value contained in the instruction

Ri – It is the name of the register involved.

The Effective Address of the operand is,

$$EA = X + [Ri]$$

The index register R1 contains the address of a new location and the value of X defines an offset(also called a displacement).

To find operand,

First go to Reg R1 (using address)-read the content from R1-1000

Add the content 1000 with offset 20 get the result.

$$1000 + 20 = 1020$$

Here the constant X refers to the new address and the contents of index register define the offset to the operand.

The sum of two values is given explicitly in the instruction and the other is stored in register.

Eg: Add 20(R1) , R2 (or) EA=>1000+20=1020

Index Mode	Assembler Syntax	Addressing Function
Index	X(Ri)	EA=[Ri]+X
Base with Index	(Ri,Rj)	EA=[Ri]+[Rj]
Base with Index and offset	X(Ri,Rj)	EA=[Ri]+[Rj] +X

Relative Addressing:

It is same as index mode. The difference is, instead of general purpose register, here we can use program counter(PC).

Relative Mode:

The Effective Address is determined by the Index mode using the PC in place of the general purpose register (gpr).

This mode can be used to access the data operand. But its most common use is to specify the target address in branch instruction. Eg. Branch>0 Loop

It causes the program execution to goto the branch target location. It is identified by the name loop if the branch condition is satisfied.

Mode Assembler Syntax Addressing Function

Relative X(PC) $EA=[PC]+X$

Additional Modes:

There are two additional modes. They are

Auto-increment mode

Auto-decrement mode

Auto-increment mode:

The Effective Address of the operand is the contents of a register in the instruction.

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

Mode Assembler syntax Addressing Function

Auto-increment (Ri)+ $EA=[Ri]$;

Increment Ri

Auto-decrement mode:

The Effective Address of the operand is the contents of a register in the instruction.

After accessing the operand, the contents of this register is automatically decremented to point to the next item in the list.

Mode Assembler Syntax Addressing Function

Auto-decrement -(Ri) $EA=[Ri]$;



Decrement Ri

Textbook :

- Carl Hamacher, Zvonko Vranesic and Safwat Zaky, —Computer Organization□, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

Application

Processor, Embedded system.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Introduction to Unit II – Arithmetic Operations		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 1 / 9	

1.CONTENT LIST:

Introduction to Unit II - Arithmetic Operations

2. SKILLS ADDRESSED:

Listening

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basic arithmetic operations.

4.OUTCOMES:

- i. Explain the concept of arithmetic operations
- ii. Listen the major topics covered in unit2

5.LINK SHEET:

- i. What are the arithmetic operations performed by ALU?
- ii. What are the topics covered in Arithmetic operations

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

ALU Design

In computing an **arithmetic logic unit (ALU)** is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under **Arithmetic and logic structures** in the ACM Computing Classification System

FIXED POINT NUMBER AND OPERATION

In computing, a **fixed-point number** representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (*e.g.*, after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating point number representation.

Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

FLOATING POINT NUMBERS & OPERATIONS

Floating point Representation:

To represent the fractional binary numbers, it is necessary to consider binary point. If binary point is assumed to the right of the sign bit, we can represent the fractional binary numbers as given below,

$$B = (b_0 * 2^0 + b_{-1} * 2^{-1} + b_{-2} * 2^{-2} + \dots + b_{-(n-1)} * 2^{-(n-1)})$$

Direct implementation of dedicated units :

always : 1 – 5

in most cases : 6

sometimes : 7, 8

Sequential implementation using simpler units and

several clock cycles (_decomposition) :

sometimes : 6

in most cases : 7, 8, 9

Table lookup

techniques using ROMs :

universal : simple application to all operations efficient

only for singleoperand operations of high complexity (8 – 12) and small word length (note: ROM size)

Approximation techniques using simpler units : 7–12

taylor series expansion

polynomial and rational approximations

convergence of recursive equation systems

Binary adder

This is also called Ripple Carry Adder, because of the construction with full adders are connected in cascade.

Carry Look Ahead Adder

The most widely used technique employs the principle of carry look-ahead to improve the speed of the algorithm.

Binary subtractor

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

For J multiplier bits and K multiplicand bits we need $(J \times K)$ AND gates and $(J - 1)$ K-bit adders to produce a product of $J+K$ bits.



$K=4$ and $J=3$, we need 12 AND gates and two 4-bit adders.

8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for ALU		
Time:	45 Minutes	
Lesson. No	Unit 2– Lesson No. 1 / 9	

1.CONTENT LIST:

ALU

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basic arithmetic operations.

4.OUTCOMES:

- Explain the concept of arithmetic operations
- Listen the major topics covered in unit2

5.LINK SHEET:

- What are the arithmetic operations performed by ALU?
- What are the topics covered in Arithmetic operations

6.EVOCATION: (5 Minutes)



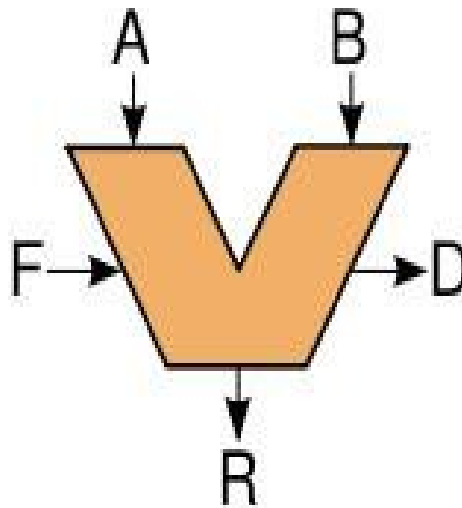
7. Lecture Notes: (40 Minutes)

ALU stands for: Arithmetic Logic Unit

ALU is a digital circuit that performs Arithmetic (Add, Sub, . . .) and Logical (AND, OR, NOT) operations.

John Von Neumann proposed the ALU in 1945 when he was working on EDVAC

Typical Schematic Symbol of an ALU



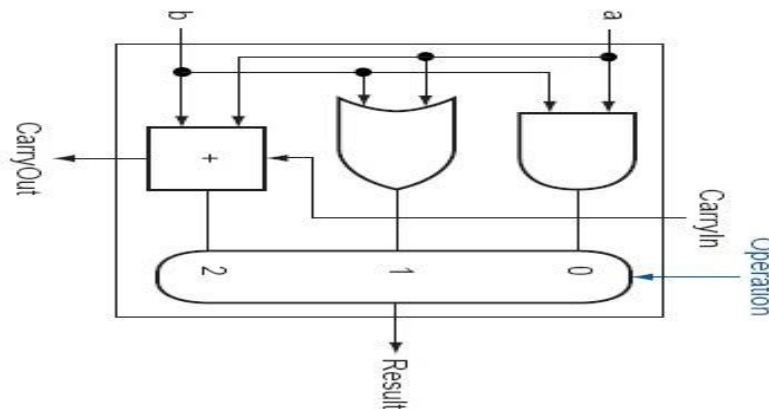
1-Bit ALU

This is an one-bit ALU which can do Logical AND and Logical OR operation.

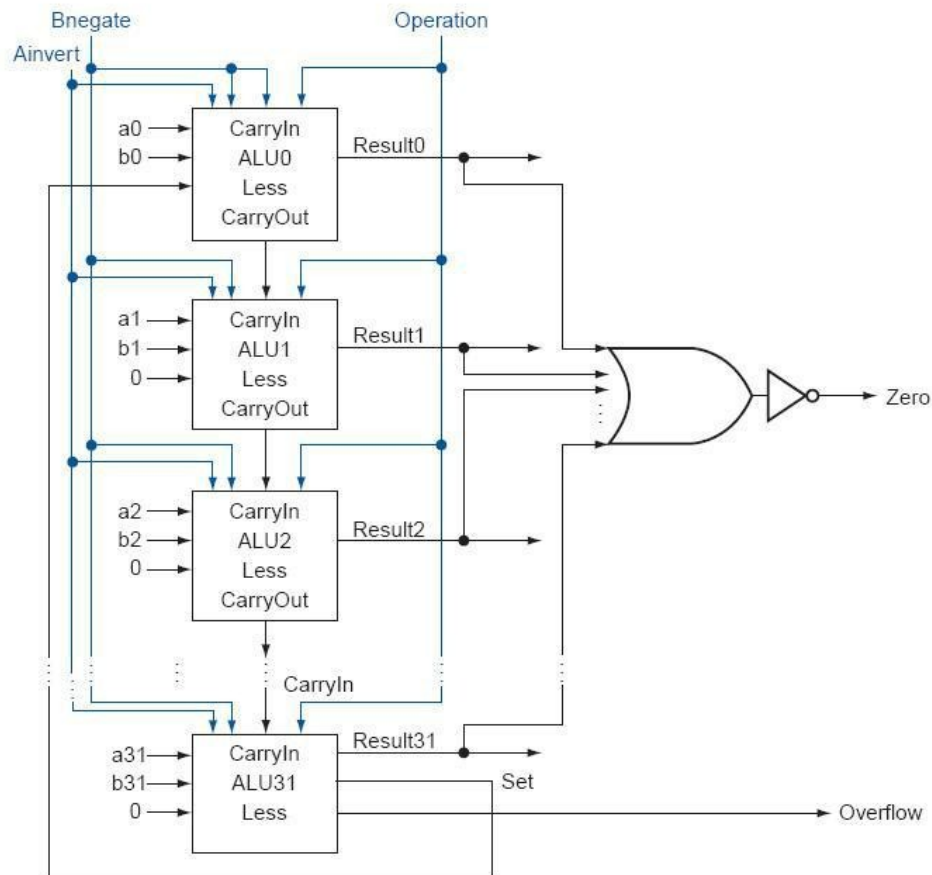
Result = $a \text{ AND } b$ when operation = 0

Result = $a \text{ OR } b$ when operation = 1

The operation line is the input of a MUX



32-Bit ALU





8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques
- Operations performed by computer

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Addition		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 3 / 9	

1.CONTENT LIST:

Addition

2. SKILLS ADDRESSED:

Learning understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the basic operation of Addition.

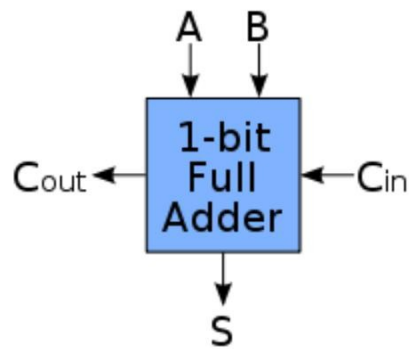
4.OUTCOMES:

- Learn the concept of Addition
- Understand the different types of adder circuit

5.LINK SHEET:

- What is Addition?
- Design Fast adders
- Discuss in detail the operation of all the adders

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

Half Adder:

A combinational circuit that performs the addition of two bits is called a half adder.

Full Adder:

One that performs the addition of three bits (two significant bits and a previous carry) is a full adder.

Binary Adder:

This is also called Ripple Carry Adder, because of the construction with full adders are connected in cascade.

Binary multiplier:

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

Need for using arithmetic circuits in designing combinational circuits:

- reduce cost
 - reduce number of gates (for SSI circuits)
 - reduce IC packages (for complex circuits)
- (ii) increase speed
- (iii) design simplicity (reuse blocks where possible)

Half Adder

The truth table for the half adder is listed below.

Table 4-3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Boolean Expression

$$C = xy$$

$$S = x'y + xy'$$

Implementation of Half Adder Circuit

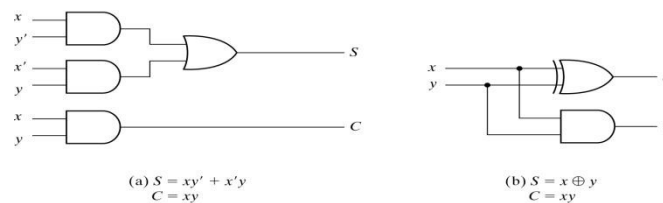


Fig. 4-5 Implementation of Half-Adder

Full Adder

One that performs the addition of three bits (two significant bits and a previous carry) is a full adder.

Truth Table

Table 4-4
Full Adder

<i>x</i>	<i>y</i>	<i>z</i>	<i>C</i>	<i>S</i>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Boolean expression using K map

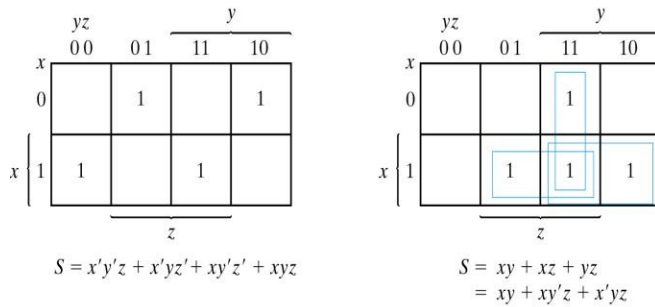


Fig. 4-6 Maps for Full Adder

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$

Implementation of Full adder circuit

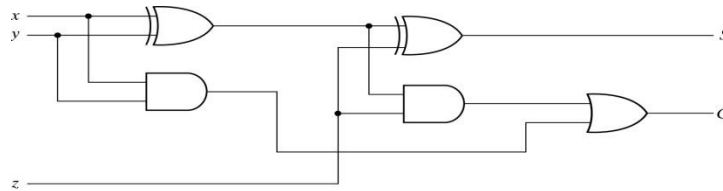


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

Binary adder

This is also called Ripple Carry Adder, because of the construction with full adders are connected in cascade.

Truth Table

Subscript <i>i</i> :	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

Implementation of Binary adder circuit

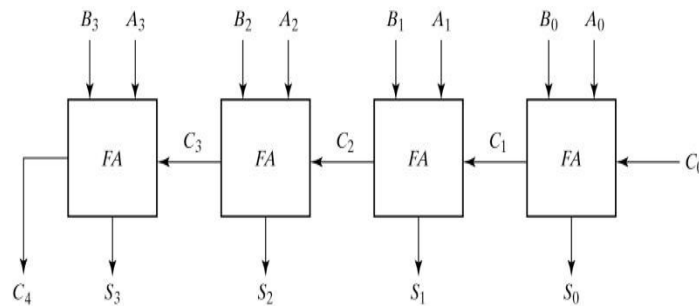


Fig. 4-9 4-Bit Adder

Carry Look Ahead Adder

The most widely used technique employs the principle of carry look-ahead to improve the speed of the algorithm.

Boolean expression

$$P_i = A_i \oplus B_i$$

steady state value

$$G_i = A_i B_i \quad \text{steady state value}$$

Output sum and carry

$$S_i = P_i \oplus C_i$$

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

G_i : carry generate P_i : carry propagate

$$C_0 = \text{input carry}$$

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

Implementation of Carry Look ahead adder circuit

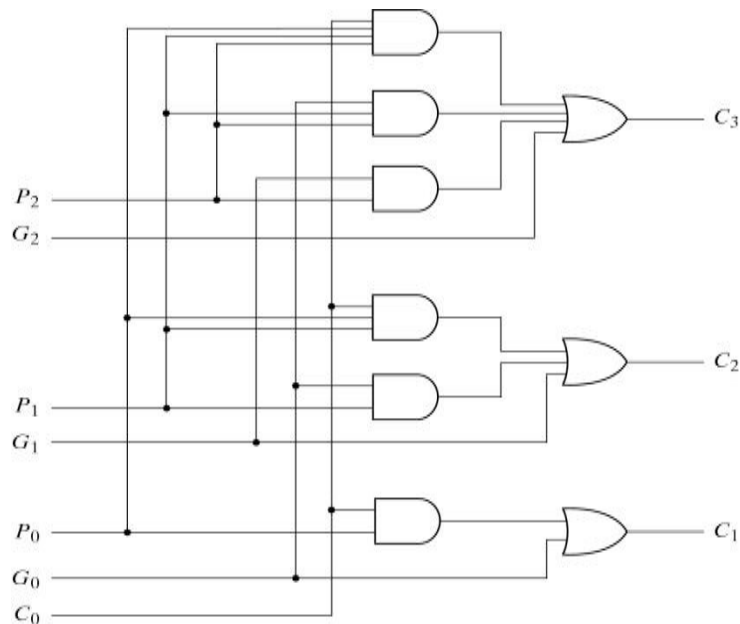




Fig. 4-11 Logic Diagram of Carry Lookahead Generator

8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques
- Operations performed by computer

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Subtraction		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 4 / 9	

1.CONTENT LIST:

Subtraction

2. SKILLS ADDRESSED:

Learning

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students learn the basic operation of Subtraction.

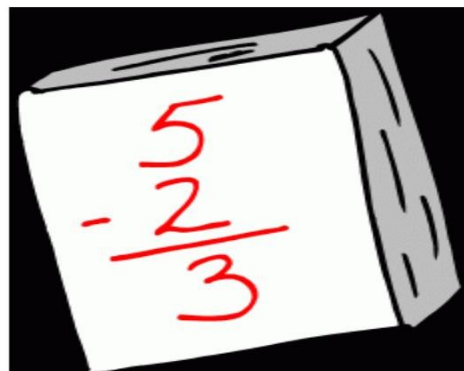
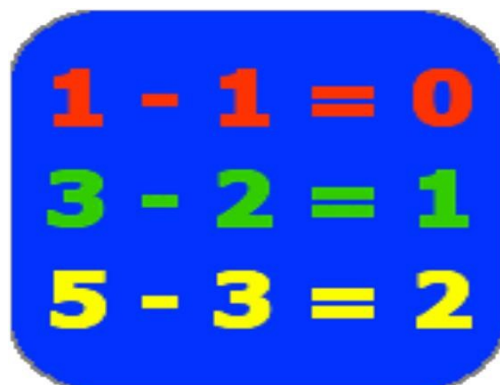
4.OUTCOMES:

- Learn the concept of Subtraction
- Understand the different types of subtractor circuit

5.LINK SHEET:

- What is Subtraction?
- Design any one of the subtractor circuit
- Discuss in detail the operation of all the subtractors

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

Binary subtractor:

$M = 1 \rightarrow$ subtractor ; $M = 0 \rightarrow$ adder

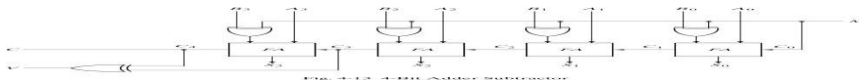
Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains $n+1$ bits cannot be accommodated.

Binary subtractor:

$M = 1 \rightarrow$ subtractor ; $M = 0 \rightarrow$ adder

Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains $n+1$ bits cannot be accommodated.

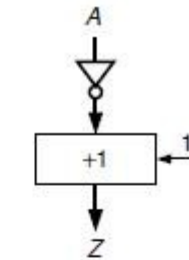
Implementation of Binary Subtractor circuit



5.1 Complement and Subtraction

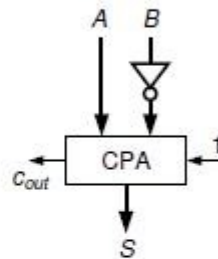
2's complementer (negation)

$$-A = \overline{A} + 1$$



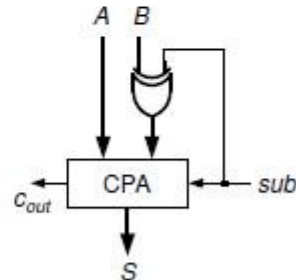
2's complement subtractor

$$\begin{aligned} A - B &= A + (-B) \\ &= A + \overline{B} + 1 \end{aligned}$$



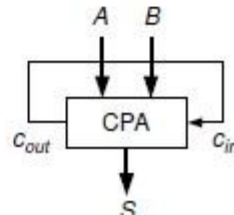
2's complement adder/subtractor

$$\begin{aligned} A \pm B &= A + (-1)^{sub} B \\ &= A + (B \oplus sub) + sub \end{aligned}$$



1's complement adder

$$\begin{aligned} A + B \pmod{2^n - 1} \\ &= A + B + c_{out} \\ &\text{(end-around carry)} \end{aligned}$$





8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques
- Operations performed by computer

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Multiplication		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 5 / 9	

1.CONTENT LIST:

Multiplication

2. SKILLS ADDRESSED:

Remembering

Applying

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students learn the basic operation of Multiplication.

4.OUTCOMES:

- Learn the concept of Multiplication
- Understand the different types of Multiplier circuit

5.LINK SHEET:

- What is Multiplication?
- Design any one of the Multiplier circuit
- Discuss in detail the Booth's algorithm

6.EVOCATION: (5 Minutes)



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12
2	0	2	4	6	8	10	12	14	16	18	20	22	24
3	0	3	6	9	12	15	18	21	24	27	30	33	36
4	0	4	8	12	16	20	24	28	32	36	40	44	48
5	0	5	10	15	20	25	30	35	40	45	50	55	60
6	0	6	12	18	24	30	36	42	48	54	60	66	72
7	0	7	14	21	28	35	42	49	56	63	70	77	84
8	0	8	16	24	32	40	48	56	64	72	80	88	96
9	0	9	18	27	36	45	54	63	72	81	90	99	108
10	0	10	20	30	40	50	60	70	80	90	100	110	120
11	0	11	22	33	44	55	66	77	88	99	110	121	132
12	0	12	24	36	48	60	72	84	96	108	120	132	144

7. Lecture Notes: (40 Minutes)

Binary multiplier:

Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

For J multiplier bits and K multiplicand bits we need (J X K) AND gates and (J – 1) K-bit adders to produce a product of J+K bits.

K=4 and J=3, we need 12 AND gates and two 4-bit adders.

Implementation of 4-bit by 3-bit binary multiplier circuit

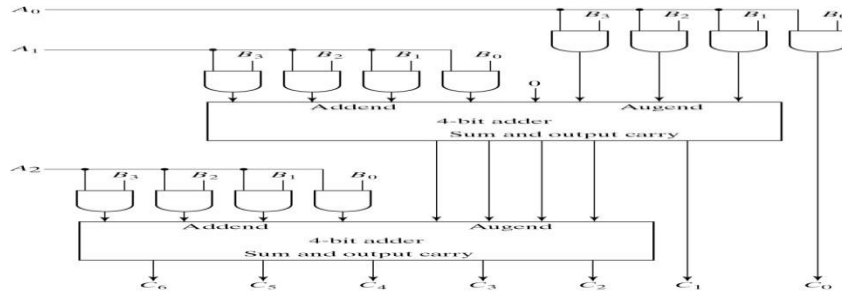


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Multiplication Basics

Multiplies two bit operands and [1, 2]

Product is 2 bit unsigned number or 2 1 bit signed number

Example : unsigned multiplication

$$P = A \cdot B = \sum_{i=0}^{n-1} a_i 2^i \cdot \sum_{j=0}^{n-1} b_j 2^j = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \quad \text{or}$$

$$P_i = a_i \cdot B, \quad P = \sum_{i=0}^{n-1} P_i 2^i; \quad i = 0, \dots, n-1 \quad (\text{r.s.a.})$$

Algorithm

- 1) Generation of partial products
- 2) Adding up partial products :
 - a) sequentially (sequential shiftandadd),
 - b) serially (combinational shiftandadd),
 or
 - c) in parallel

Speedup techniques

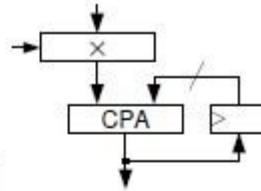
Reduce number of partial products

Accelerate addition of partial products

Sequential multipliers :

partial products generated and added *sequentially* (using *accumulator*)

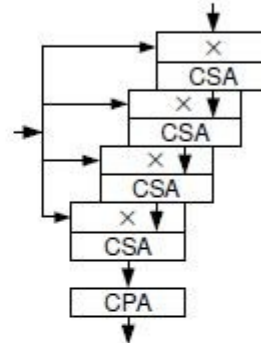
$$A = O(n), T = O(\log n), L = n$$



Array multipliers :

partial products generated and added *simultaneously* in linear array (using *array adder*)

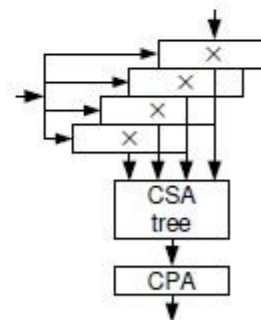
$$A = O(n^2), T = O(n)$$



Parallel multipliers :

partial products generated in *parallel* and added *subsequently* in multi-operand adder (using *tree adder*)

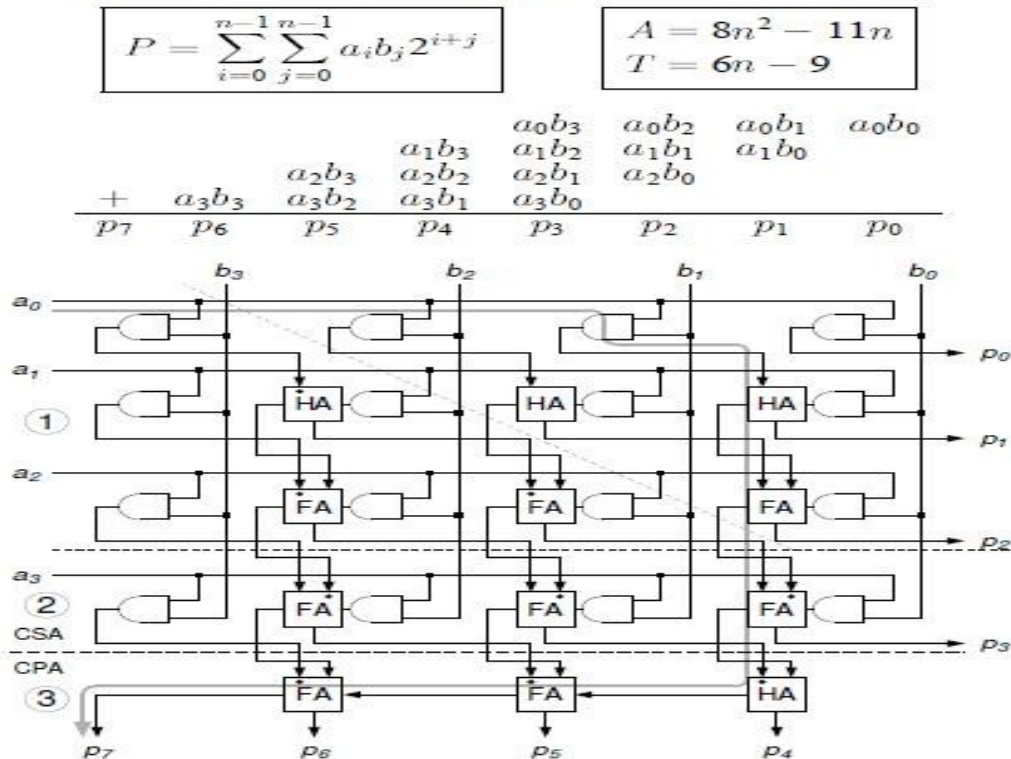
$$A = O(n^2), T = O(\log n)$$



Signed multipliers :

- complement* operands before and result after multiplication \Rightarrow *unsigned* multiplication
- direct* implementation (dedicated multiplier structure)

- **Braun multiplier** : array multiplier for unsigned numbers



Booth Recoding

- **Speed-up technique** : reduction of partial products

Sequential multiplication

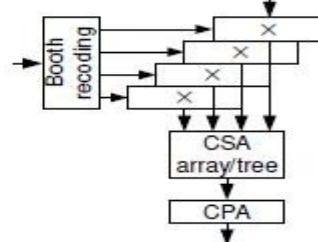
- **Minimal (or canonical) signed-digit (SD) represent.** of A
- + **One cycle per non-zero partial product** (i.e. $\forall a_i \mid a_i \neq 0$)
- **Negative partial products**
- **Data-dependent** reduction of partial products and latency

Combinational multiplication

- Only **fixed** reduction of partial product possible
- **Radix-4 modified Booth recoding** : 2 bits recoded to one multiplier digit $\Rightarrow n/2$ partial products

$$A = \sum_{i=0}^{n/2} \underbrace{(a_{2i-1} + a_{2i} - 2a_{2i+1})}_{\{-2, -1, 0, +1, +2\}} 2^{2i} ; a_{-1} = 0$$

a_{2i+1}	a_{2i}	a_{2i-1}	P_i
0	0	0	+ 0
0	0	1	+ B
0	1	0	+ B
0	1	1	+ 2B
1	0	0	- 2B
1	0	1	- B
1	1	0	- B
1	1	1	- 0





Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

Applicable to sequential, array, and parallel multip

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Division		
Time:	45Minutes	
Lesson. No	Unit 2 – Lesson No. 6 / 9	

1.CONTENT LIST:

Division

2. SKILLS ADDRESSED:

Understanding

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students learn the basic operation of Division.

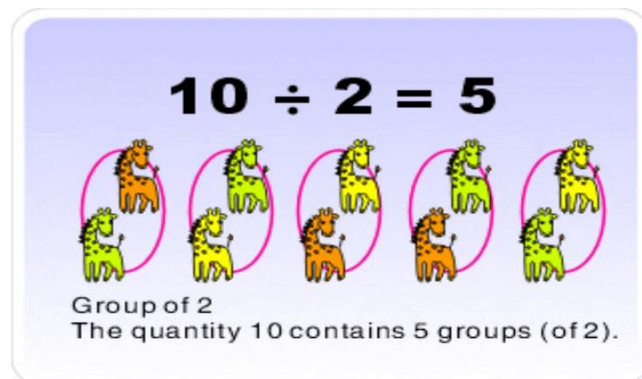
4.OUTCOMES:

- Learn the concept of Division
- Understand the different types of Divider circuit

5.LINK SHEET:

- What is Division?
- Design any one of the Divider circuit
- Discuss in detail the operation of division

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

Division Algorithms and Hardware Implementations

Two types of division operations

- Integer division: with integer operands and result
- Fractional division: operands and results are fractions

Any division algorithm can be carried out independent of

- Position of the decimal point
- Sign of operands

Restoring Division Algorithm

Put x in register A, d in register B, 0 in register P, and perform n divide steps (n is the quotient wordlength)

Each step consists of

- Shift the register pair (P,A) one bit left
- Subtract the contents of B from P, put the result back in P
- If the result is -ve, set the low-order bit of A to 0 otherwise to 1
- If the result is -ve, restore the old value of P by adding the contents of B back in P

P	A	Operation
00000	1110	Divide 14 = 1110 by 3 = 11. B register always contains 0011
00001	110	step 1(i): shift
-00011		step 1(ii): subtract

-00010	1100	step 1(iii): quotient is -ve, set quotient bit to 0
00001	1100	step 1(iv): restore
00011	100	step 2(i): shift
-00011		step 2(ii): subtract

00000	1001	step 2(iii): quotient is +ve, set quotient bit to 1
00001	001	step 3(i): shift
-00011		step 3(ii): subtract

-00010	0010	step 3(iii): quotient is -ve, set quotient bit to 0
00001	0010	step 3(iv): restore
00010	010	step 4(i): shift
-00011		step 4(ii): subtract

-00001	0100	step 4(iii): quotient is -ve, set quotient bit to 0
00010	0100	step 4(iv): restore

- The quotient is 0100 and the remainder is 00010
- The name *restoring* because if subtraction by b yields a negative result, the P register is restored by adding b back

Non-Restoring Division Algorithm

A variant that skips the restoring step and instead works with negative residuals

If P is negative

- (i-a) Shift the register pair (P,A) one bit left
- (ii-a) Add the contents of register B to P

If P is positive

- (i-b) Shift the register pair (P,A) one bit left
- (ii-b) Subtract the contents of register B from P
- (iii) If P is negative, set the low-order bit of A to 0, otherwise set it to 1
- After n cycles
- The quotient is in A
- If P is positive, it is the remainder, otherwise it has to be restored (add B to it) to get the remainder

P	A	Operation
00000	1110	Divide 14 = 1110 by 3 = 11. B register always contains 0011
00001	110	step 1(i-b): shift
+00011		step 1(ii-b): subtract b (add two's complement)

11110	1100	step 1(iii): P is negative, so set quotient bit to 0
11101	100	step 2(i-a): shift
+00011		step 2(ii-a): add b

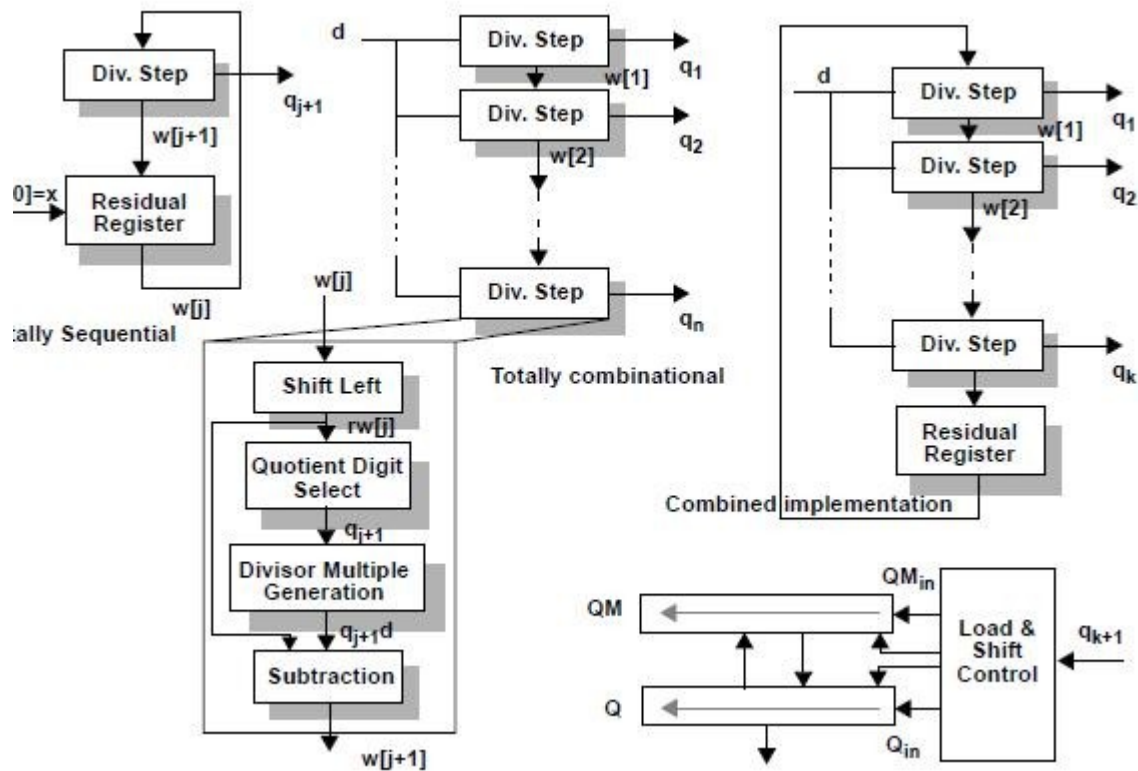
00000	1001	step 2(iii): P is +ve, so set quotient bit to 1
00001	001	step 3(i-b): shift
+11101		step 3(ii-b): subtract b

11110	0010	step 3(iii): P is -ve, so set quotient bit to 0
11100	010	step 4(i-a): shift
+00011		step 4(ii-a): add b

11111	0100	step 4(iii): P is -ve, set quotient bit to 0
+00011		Remainder is negative, so do final restore step

00010		

- The quotient is 0100 and the remainder is 00010
- *restoring* division seems to be more complicated since it involves extra addition in step (iv)
- This is not true since the sign resulting from the subtraction is tested at adder o/p and only if the sum is +ve, it is loaded back to the P register





8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques
- Operations performed by computer

	<div>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</div> <div>Department of Information Technology</div>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Floating Point operations		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 7/ 9	

1.CONTENT LIST:

Floating Point operations

2. SKILLS ADDRESSED:

Understanding

Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the concept of floating point operations.

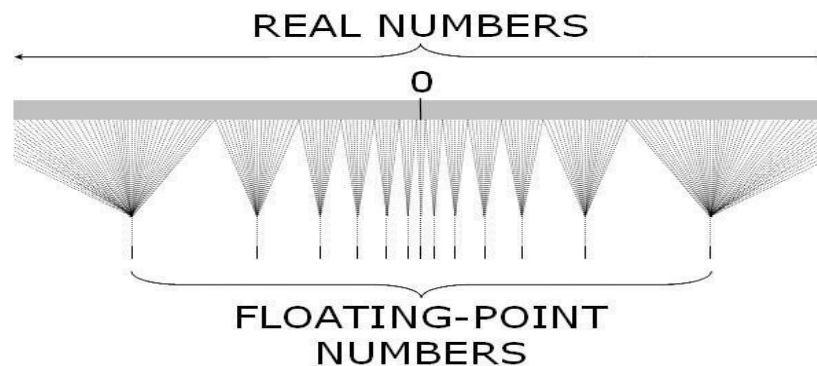
4.OUTCOMES:

- i. Learn the concept of floating numbers
- ii. Understand the different floating point operationst

5.LINK SHEET:

- i. What is floating point?
- ii. Discuss in detail the operation of floating numbers

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

Arithmetic operations on floating point numbers consist of addition, subtraction, multiplication and division the operations are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation) -- example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction.

ADDITION

example on decimal value given in scientific notation:

$$\begin{array}{r} 3.25 \times 10^{**3} \\ + 2.63 \times 10^{** -1} \\ \hline \end{array}$$

first step: align decimal points

second step: add

$$\begin{array}{r} 3.25 \quad \times 10^{**3} \\ + 0.000263 \times 10^{**3} \\ \hline 3.250263 \times 10^{**3} \end{array}$$

(presumes use of infinite precision, without regard for accuracy)

third step: normalize the result (already normalized!)

SUBTRACTION

like addition as far as alignment of radix points then the algorithm for subtraction of sign mag. numbers takes over.

before subtracting,

compare magnitudes (don't forget the hidden bit!)

change sign bit if order of operands is changed.

don't forget to normalize number afterward.

MULTIPLICATION

example on decimal values given in scientific notation:

$$\begin{array}{r} 3.0 \times 10^{**1} \\ + 0.5 \times 10^{**2} \end{array}$$

algorithm: multiply mantissas
add exponents

$$\begin{array}{r} 3.0 \times 10^{**1} \\ + 0.5 \times 10^{**2} \\ \hline 1.50 \times 10^{**3} \end{array}$$

example in binary: use a mantissa that is only 4 bits so that
I don't spend all day just doing the
multiplication part.

DIVISION

similar to multiplication.

true division:

do unsigned division on the mantissas (don't forget the hidden bit)
subtract TRUE exponents

The IEEE standard is very specific about how all this is done.
Unfortunately, the hardware to do all this is pretty slow.

Some comparisons of approximate times:

2's complement integer add	1 time unit
fl. pt add	4 time units
fl. pt multiply	6 time units
fl. pt. divide	13 time units

There is a faster way to do division. Its called
division by reciprocal approximation. It takes about the same
time as a fl. pt. multiply. Unfortunately, the results are not
always the same as with true division.

Division by reciprocal approximation:

instead of doing a / b

they do $a \times 1/b$.

figure out a reciprocal for b, and then use the fl. pt.
multiplication hardware.

example of a result that isn't the same as with true division.

true division: $3/3 = 1$ (exactly)

reciprocal approx: $1/3 = .33333333$

$3 \times .33333333 = .99999999$, not 1



It is not always possible to get a perfectly accurate reciprocal.

8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

- They present special design challenges, because there are simply too many inputs to list all possible combinations in a truth table.
- In applying this method, bus-wide operations are broken into simpler bit-by-bit operations that are more easily defined by truth-tables, and more tractable to familiar design techniques
- Operations performed by computer

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Information Technology	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	P.Kaviya	
Lesson Plan for Subword parallelism		
Time:	45 Minutes	
Lesson. No	Unit 2 – Lesson No. 8/ 9	

1.CONTENT LIST:

Subword parallelism

2. SKILLS ADDRESSED:

Understanding

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the concept of subword parallelism

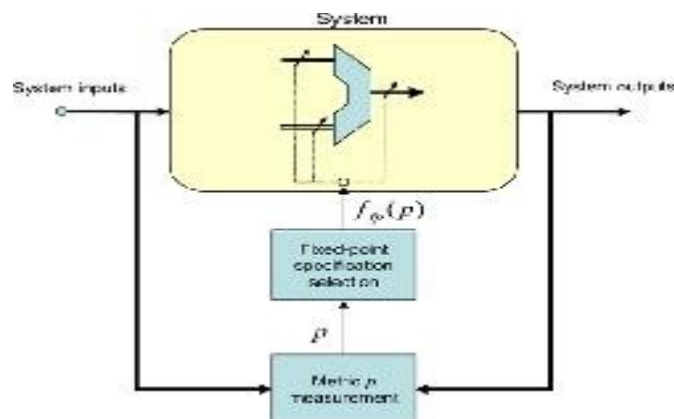
4.OUTCOMES:

- i. Learn the concept of parallelism
- ii. Understand the manipulation of subword parallelism

5.LINK SHEET:

- i. What is parallelism?
- ii. Discuss in detail the operation of subword parallelism

6.EVOCATION: (5 Minutes)



7. Lecture Notes: (40 Minutes)

Subword Parallelism

The term SIMD was originally defined in 1960s as category of multiprocessor with one control unit and multiple processing elements - each instruction is executed by all processing elements on different data streams, e.g., Illiac IV. Today the term is used to describe partitionable ALUs in which multiple operands can fit in a fixed-width register and are acted upon in parallel.

(other terms include subword parallelism, microSIMD, short vector extensions, split-ALU, SLP / superword-level parallelism, and SIGD / single-instruction-group[ed]-data)

The structure of the arithmetic element can be altered under program control. Each instruction specifies a particular form of machine in which to operate, ranging from a full 36-bit computer to four 9-bit computers with many variations.

Not only is such a scheme able to make more efficient use of the memory in storing data of various word lengths, but it also can be expected to result in greater over-all machine speed because of the increased parallelism of operation.

Peak operating rates must then be referred to particular configurations. For addition and multiplication, these peak rates are given in the following table:

PEAK OPERATING SPEEDS OF TX-2

Word Lengths (in bits)	Additions per second	Multiplications per second
36	150,000	80,000
18	300,000	240,000
9	600,000	600,000

Univac 1107 (ca. 1962) - a 36-bit machine that included add/subtract halves instructions (two 18-bit operations in parallel) and add/subtract thirds instructions (three 12-bit operations in parallel)

Intel i860 (ca. 1989) added three packed graphics data types (e.g., eight 1-byte pixel values per 64-bit word) and a special graphics function unit (e.g., z-buffer interpolation)



Motorola 88110 (ca. 1991) included six graphics data types and performed saturating arithmetic

8. Textbook :

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth Edition, Tata McGraw Hill, 2002, PP. 3-9

9. APPLICATIONS

MicroSIMD, superword-level parallelism, and SIGD / single-instruction-group[ed]-data)

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Introduction to Unit III – Processor and Control Unit		
Time:	45Minutes	
Lesson. No	Unit 3 – Lesson No. 1 / 13	

1. CONTENT LIST:

Introduction to Unit III - Processor and Control Unit

2. SKILLS ADDRESSED:

Listening

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basics of processor and control unit.

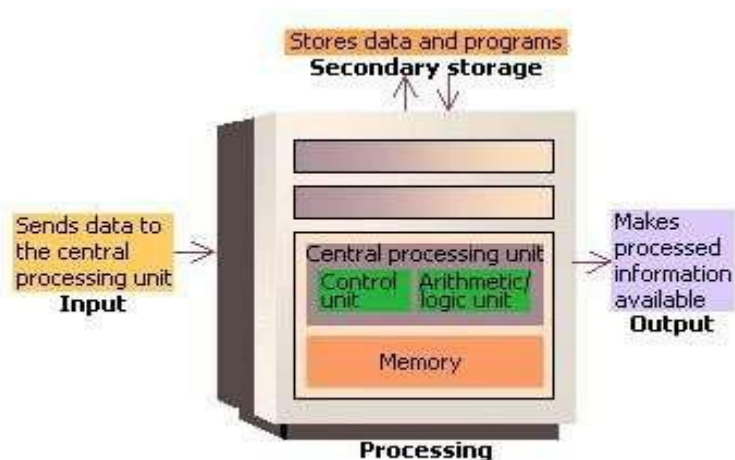
4. OUTCOMES:



- Explain the concept of processor and control unit
- Listen the major topics covered in unit3

5. LINK SHEET:

- Give the detailed concept of processor and control unit
- What are the topics covered in processor and control unit?

6. EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Basic MIPS implementation		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 2 ,3 / 13	

1. CONTENT LIST:

Basic MIPS implementation

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the implementation of MIPS.

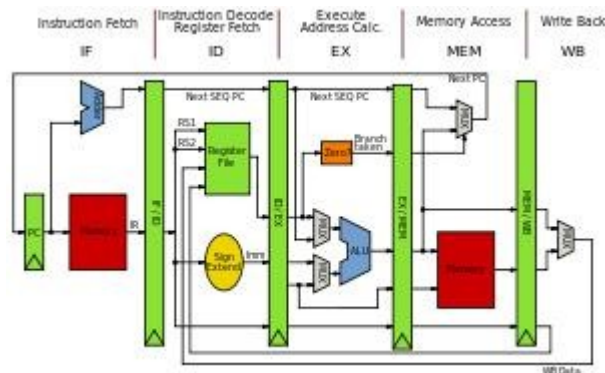
4.OUTCOMES:



- Learn the concept of MIPS implementation
- Know the different types of formats used in MIPS

5.LINK SHEET:

- List the formats in MIPS Implementation
- Explain the concept of MIPS Implementation
- Give the Features of MIPS Implementation

6.EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Building datapath		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 4 / 13	

1.CONTENT LIST:

Building datapath

2. SKILLS ADDRESSED:

Learning

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students learn the building of datapath

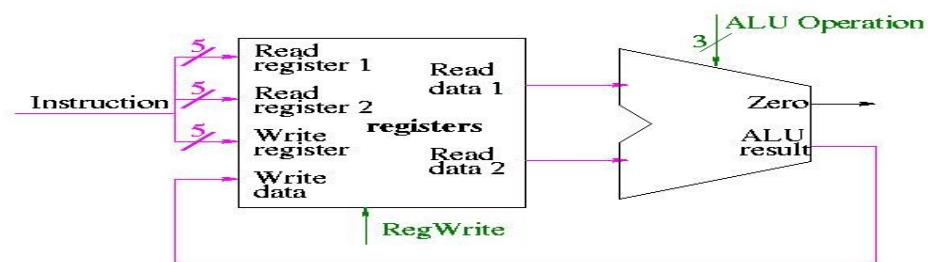
4.OUTCOMES:

- Understand the concept of datapath building
- Know the register format and other types of format in detail



5.LINK SHEET:

- List the formats in datapath building
- Design datapath with various formats
- Explain any one of the formats in detail

6.EVOCATION: (5 Minutes)



Control lines in green
Buses in magenta, 32-bit if not labelled

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Control Implementation scheme		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 5, 6 / 13	

1. CONTENT LIST:

Control Implementation scheme

2. SKILLS ADDRESSED:

Learning

Applying

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the scheme of control implementation

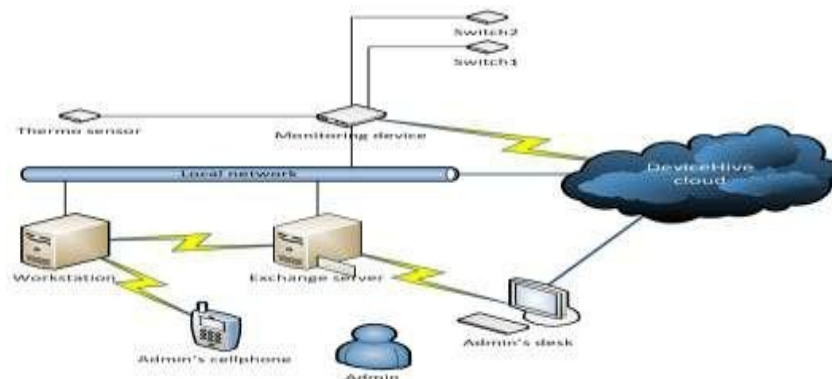
4. OUTCOMES:



- Learn the scheme to design datapath
- Understand the different cycles in implementing control scheme

5. LINK SHEET:

- Give the formats in control implementation scheme
- Design the scheme to characterize the datapath
- Explain control implementation scheme in detail.

6. EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Pipelining		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 5, 6 / 13	

1.CONTENT LIST:

Pipelining

2. SKILLS ADDRESSED:

Remembering Applying

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students remember the pipelining procedures

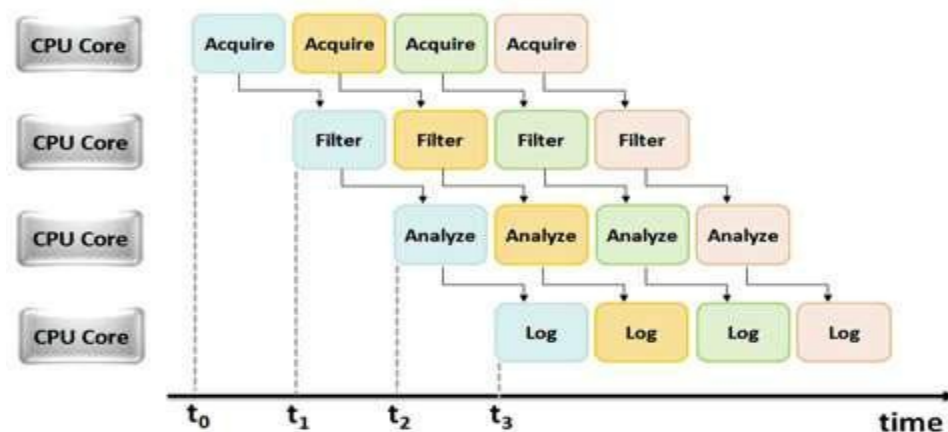
4.OUTCOMES:



- Learn the sequence steps for pipelining
- Understand the features of pipelining

5.LINK SHEET:

- Give the need and role of cache memory in pipelining
- Explain in detail the basic concepts of pipelining

6.EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Pipelined datapath and control		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 8, 9 / 13	

1. CONTENT LIST:

Pipelined datapath and control

2. SKILLS ADDRESSED:

Understanding

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students understand the datapath and control considerations for pipelining

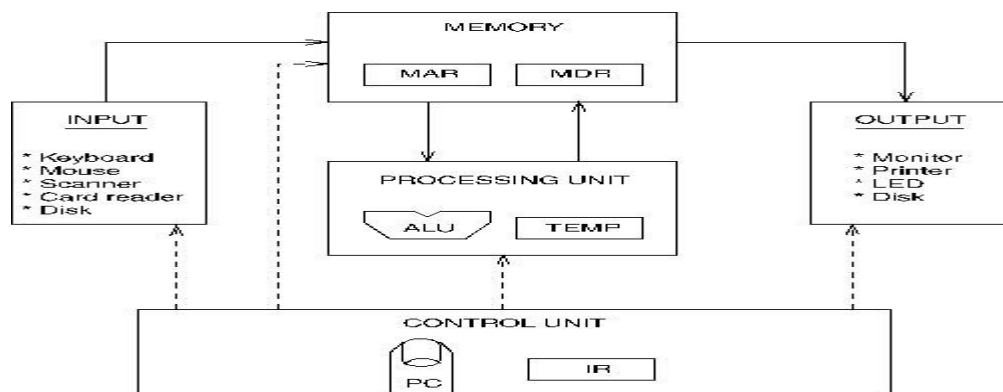
4. OUTCOMES:



- Remember the important terms used in pipelining datapath and control concept
- Understand the operations performed in control and datapath pipelining

5. LINK SHEET:

- Construct pipelined datapath and control architecture
- Discuss in detail the operation of data and control pipelining
- Draw the control and data pipelining architecture.

6. EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Handling Data hazards & Control hazards		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 10, 11 / 13	

1. CONTENT LIST:

Handling Data hazards & Control hazards

2. SKILLS ADDRESSED:

Understanding Applying

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students understand the Von Neumann architecture and data hazards exclusively

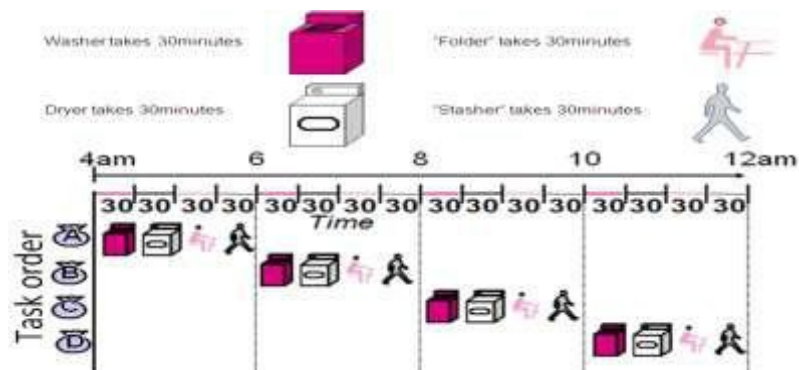
4. OUTCOMES:



- Learn the operation performed in data and control hazards.
- Understand the various structure to handle data and control hazards

5. LINK SHEET:

- What is hazard
- Discuss in detail the operation of data hazard.
- Explain in detail the features of control hazard.

6. EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Exceptions.		
Time:	45 Minutes	
Lesson. No	Unit 3 – Lesson No. 12 / 13	

1.CONTENT LIST:

Exceptions.

2. SKILLS ADDRESSED:

Learning

Analyzing

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students understand the way to handle the exceptions

4.OUTCOMES:

- Know the manipulation performed in exceptions.
- Understand the procedure to handle exceptions

5.LINK SHEET:

- Construct exceptions with neat clock cycle
- Discuss in detail the manipulation of exception handling
- Write the features of exception handling?

6.EVOCATION: (5 Minutes)



UNIT-3

Processor and Control Unit

3 Introduction

→ clock cycle time and number of CPs are determined by processor implementation
→ Effect of different implementation choices on clock rate and CPs. This chapter holds the basic MIPS implementation and its building nature.

3.1 Basic MIPS Implementation :-

Examine the implementation that includes a subset of the core MIPS instruction set

→ The memory reference instructions load word (lw) and store word (sw)

→ The arithmetic logical instructions add, sub, AND, OR and sli

→ The instruction branch equal (beq) and jump(j) which added last.

(2)

To simplify the instruction, ALU and memory reference is used. After using the ALU, the action required to complete the different instruction classes differ. A memory reference instruction will need to access the memory either to write data for a store or read data for a load. An arithmetic logical instruction must write the data from the ALU back into a register or memory.

Fig 3.1 shows the high level view of MIPS implementation.

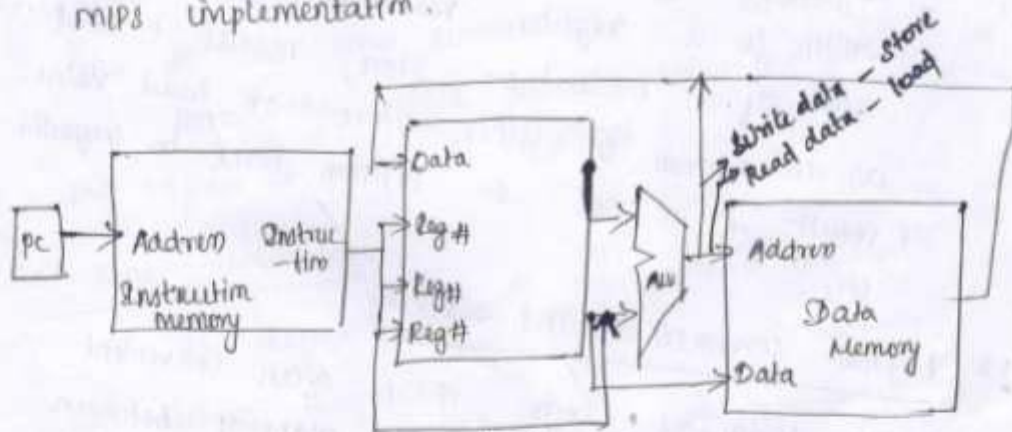


Fig 3.1 MIPS Implementation - processor

Add R_i, R_i
Add R_0, C

Fig 3.1 shows the MIPS subset showing the major functional units and the major connections between them. PC 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 that instruction. After fetching the register operand, they can be operated on to compute memory address, to compute an arithmetic result or a compare. If the instruction is an arithmetic logical instruction, the result from ALU must be written to a register. ~~Result written back to~~ If it is load or store, result is used as an address to either store or load value. result from ALU is written back to register file.

3.1.2 Logical conventions and clocking

→ By designing logic, it is often convenient for designer to change the mapping between logically true or false signal and high or low voltage level.

(8)

→ The functional units in MIPS implementation consists of 2 different logic elements.

a. elements on data values → Combinational (depend only on current ip)

b. elements on State.

↳ contain internal storage — state elements or sequential element (depend on input and internal state)

→ A state element has two inputs and one output. The required inputs are the data value to be written into the element and the clock, which determines when the data value is written.

→ output from a state element provides the value that was written in an earlier clock cycle. Here the state elements are memories and registers as given in Fig 8-1.

3.1.3 Clocking Methodology

It defines when signals can be read and when they can be written. It is important to specify the timing of reads and writes, because if a signal is written at the same time it is read, the value of the read could correspond to old value, the newly written value or even mix of two.

⑤

→ The functional units in MIPS Implementation consists of 2 different logic elements.

- a. elements on data values → Combinational (depend only on current (ip))
- b. elements on ~~on~~ State.

↳ Contain Internal Storage — State elements or Sequential element (depend on Input and internal state)

→ A state element has two inputs and one output. The required inputs are the data value to be written into the element and the clock, which determines when the data value is written.

→ output from a state element provides the value that was written in an earlier clock cycle. Here the state elements are memories and registers as given in Fig 5.1.

3.1.3 Clocking methodology

It defines when signals can be read and when they can be written. It is important to specify the timing of reads and writes, because if a signal is written at the same time it is read, the value of the read could correspond to old value, the newly written value or even

Fig 3.2 shows the two state elements surround a block of combinational logic which operates in a single clock cycle. All signals propagate from state element 1 through combinational logic to state element 2. In the of one clock cycle.

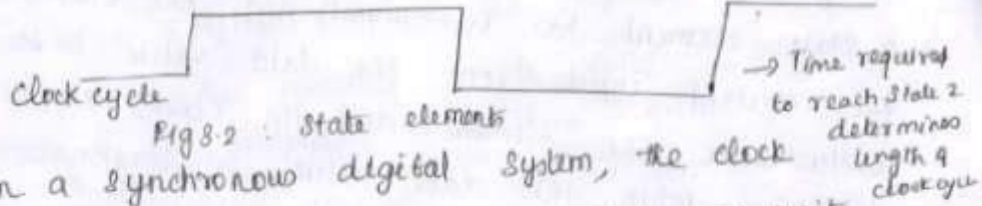
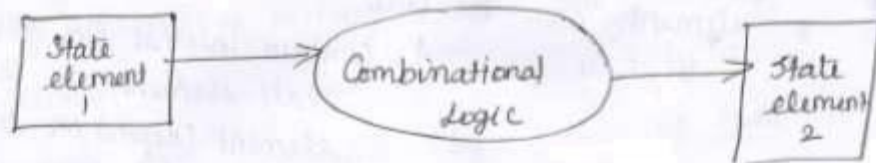


Fig 3.2 : State elements

In a synchronous digital system, the clock determines when elements with state will write values into internal storage. write control signal and clock edge determines the ^{change in} state element.

2.1.4 Edge triggered Methodology

An edge triggered methodology allows us to read the contents of a register, send the value through combinational logic, and write that register in the same clock cycle. With an edge triggered timing methodology, there is no feedback within a single clock cycle and it is given in Fig 3.3

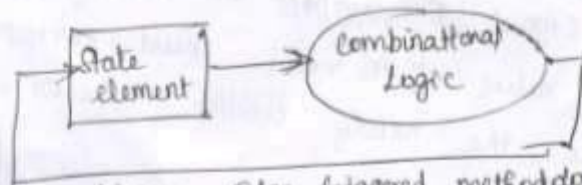


Fig 3.3 : Edge triggered methodology

The clock cycle CPU must be long enough so that the input values are stable when the active clock edge occurs.

MIPS subset implementation

21-10

3.2 Building a datapath = registers, ALU and interconnecting bus

→ The datapath is the brain of a processor since it implements the fetch-decode-execute cycle. The general discipline for datapath design is to

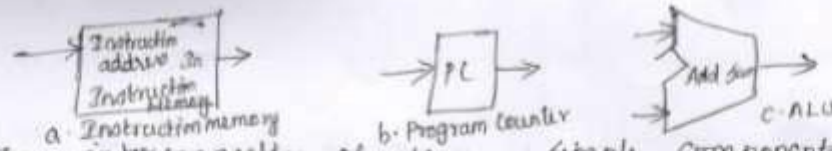
1. Determine the instruction classes and formats in the ISA.

2. Design datapath components and interconnections for each instruction class or format

3. Compose the datapath segments designed in (step 2) to yield a composite datapath.

8.

→ Simple datapath components include memory (stores the current instruction), PC or program counter (stores the address of current instruction) and ALU (executes current instruction).



The interconnection of these simple components to form a basic datapath is given in Fig 3.4.

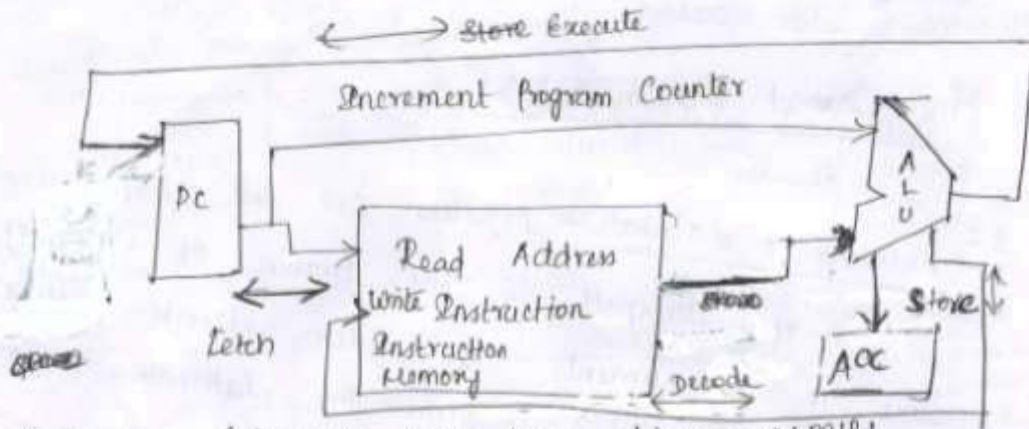


Fig 3.4: Schematic high level diagram of MIPS datapath.

Implementation of datapath for I and J format instructions requires two or more components

1. data memory
2. sign Extender

→ given in Fig 3.5

I → op, (rt), rs, imm

J Format → opcode | Address

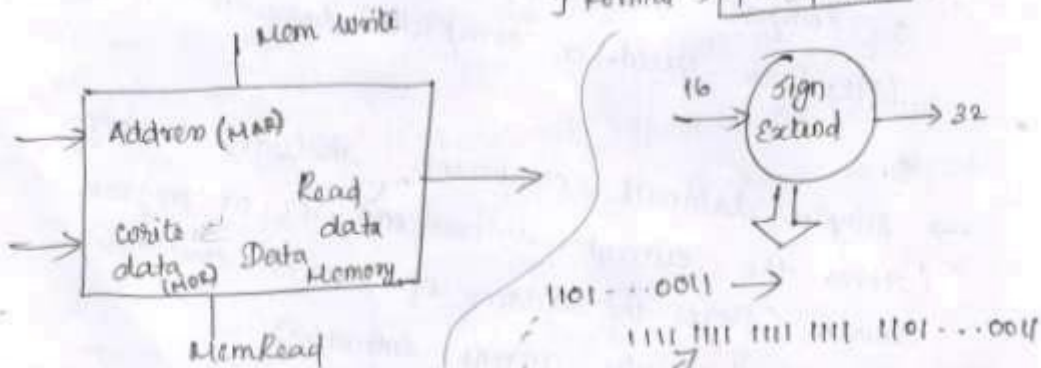


Fig 3.5 Schematic diagram of data memory & sign Extender

From Fig 3.5

1. data memory stores ALU results and operands including instructions, and has two enabling inputs (mem write and mem read) that cannot be both be active (have a logical high value at the same time).

2. The data memory accepts an address and either accepts data (write data port if mem write is enabled) or outputs data (read data port mem read is enabled), at the indicated address. The sign extender adds 16 leading digits to a 16 bit word with most significant bit b, to produce a 32 bit word. In particular, the additional 16 digits have the same value as b, thus implementing sign extension in two's complement representation.

6

3.2.2 R-Format Datapath → used when all the data values are located in registers $[OP, r_d, r_s, r_t]$

→ Implementation of datapath for R-format instructions is fairly straight forward - the register file and the ALU are all that is required. The R format is given in Fig 3.6

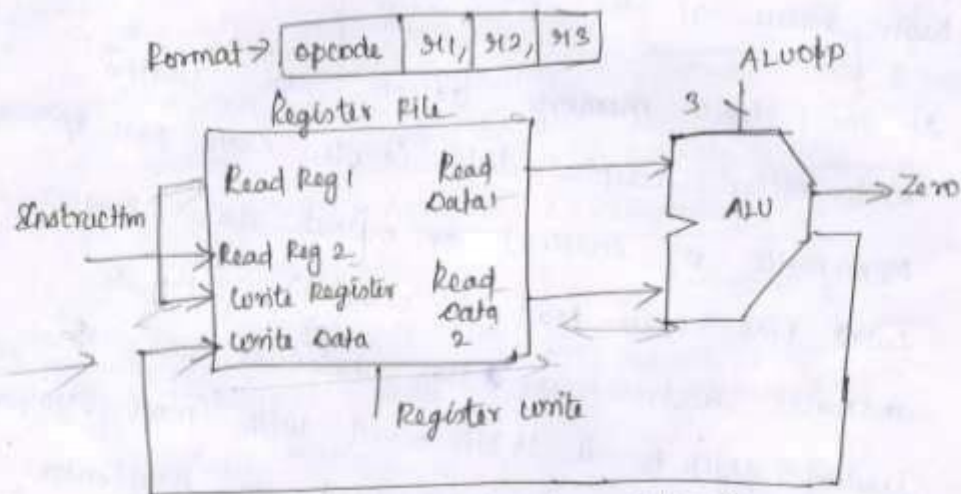
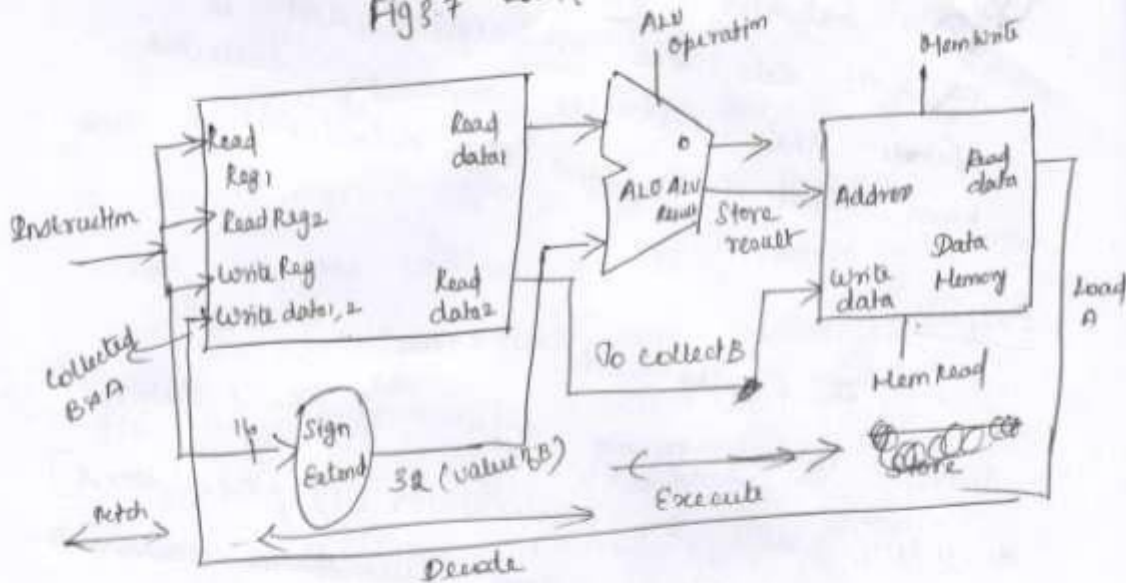


Fig 3.6: R Format instruction datapath.

3.2.3 Load/Store Datapath

The load/store datapath instruction such as lw \$t1, offset(\$t2) where offset denotes a memory address offset applied to the base address in register \$t2. The lw instruction reads from memory and writes into register \$t1. The sw instruction reads from register \$t1 and writes into memory.

Fig 3.7 Load/Store datapath



Eg :- Load A
Add B
Store C

Fetch:- Instruction is fetched from memory by program counter

Decode :- The operands are collected from memory and (register file if needed).

First In order to perform above Eg: Read data from memory is enable and it will copy contents of A into register. It will be stored in read data. Again to read B one signal will be sent and write the data B in register file.

→ The value of B is given by Sign Extender. Both values will be stored as write data 1 & 2 in registers and to operate the data it will be collected from Read data 1 and sign extender.

Execute:-

Finally the Execution Steps starts with giving input as Read data 1 as which will be accumulator (ie) $[AC \leftarrow A]$ and sign extender where ~~accumulator contains B~~ is $AC \leftarrow B$. It contains the value B. At the end the result will be coming as Add result which contains $AC \leftarrow [AC] + B$

Store:-

Finally the accumulator result will be stored in memory

(7)

3.2.4 Branch / Jump Datapath

The branch datapath ~~fits~~ uses instructions such as beq \$t1, \$t2, offset where offset is a 16 bit address offset for computing the branch Target address via PC-relative addressing.

The beq instruction reads from registers \$t1 and \$t2 then compares the data obtained from these registers to see if they are equal. If equal the branch is taken. Otherwise, the branch is not taken.

By taking the branch, ISA specification means that ALU adds a sign extended offset to the program counter (PC).

The Branch instruction datapath is illustrated in Fig 3.8

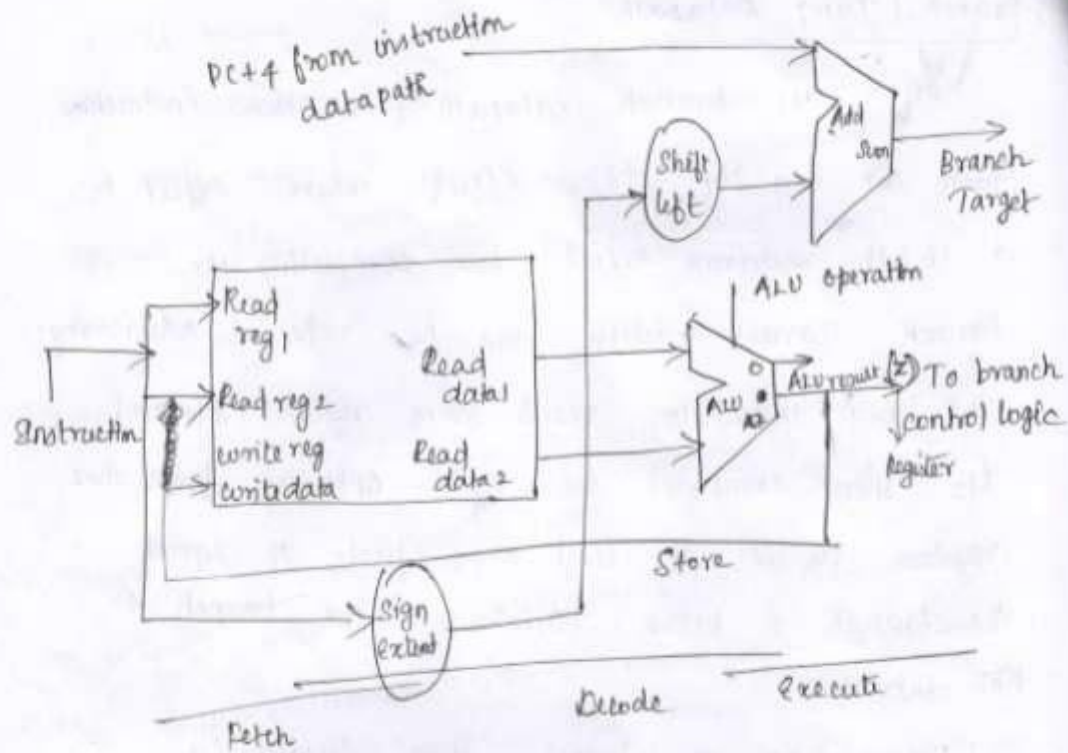


Fig 3.8 Branch Instruction Datapath.

→ The branch datapath takes operand from the instruction input to the register file, then sign extends the offset. The sign extended offset and the program Counter are combined by ALU to yield the branch target address. operands

obtained from register file via Read Data ports.

→ MIPS has the special feature of delayed branch, i.e. which follows the branch is always fetched, decoded and prepared for execution.

(8)

→ If the branch condition is false, a normal branch occurs. If the branch condition is true then I_b is executed.

3.3 Pipelining - ops of one element is ip to next element

Pipelining is used to achieve high performance.

3.3.1 Basics of pipelining

→ The speed of execution of programs is influenced by many factors. One way to improve the performance is to use faster circuit technology to build the processor and the main memory.

→ pipelining is a particularly effective way of organizing concurrent activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly line operation.

3.3.2 Operation of pipelining in computer

The processor executes a program by fetching and executing instructions, one after the other. Let P_i and E_i refer to the fetch and execute steps for instruction I_i . Execution of a program consists of a sequence of fetch and execute steps as in Fig 3.9.a

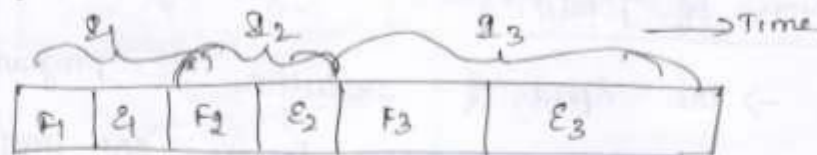
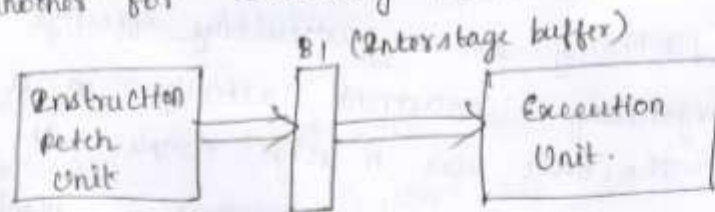


Fig 3.9a Sequential Execution

→ Consider a computer that has 2 separate hardware units, one for fetching instructions and another for executing them as in Fig 3.9b



3.9b Hardware Organization

9

- The instruction fetched by the fetch unit is deposited in intermediate storage buffer B₁.
- Buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction.
- The result stored in the destination location specified by the instruction.
- The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can be completed in one clock cycle. operation can be given in 3.9c

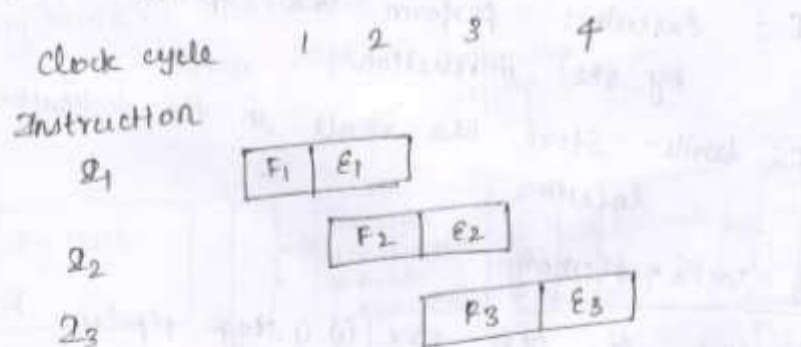


Fig 3.9c Basic idea of instruction pipelining

→ In the first clock cycle, the fetch unit fetches an instruction I_1 (step 1) and stores it in Buffer B_1 at the end of the clock cycle and this continues so on.

→ The processing of an instruction need not be divided into ~~two~~ only two steps. For example, a pipelined processor may process each instruction in four steps as follows

P : Fetch: read the instruction from memory

D : Decode: decode the instruction and fetch the source operand(s)

E : Execute: perform the operation specified by the instruction

W : Write: Store the result in the destination location.

Role of Cache Memory

The sequence of this case (i) 4 stage pipeline is shown in Fig 3.10a

(10)

→ Four Instructions are in program at any given time. This means that four distinct hardware units are needed, as shown in Fig 3.10b.

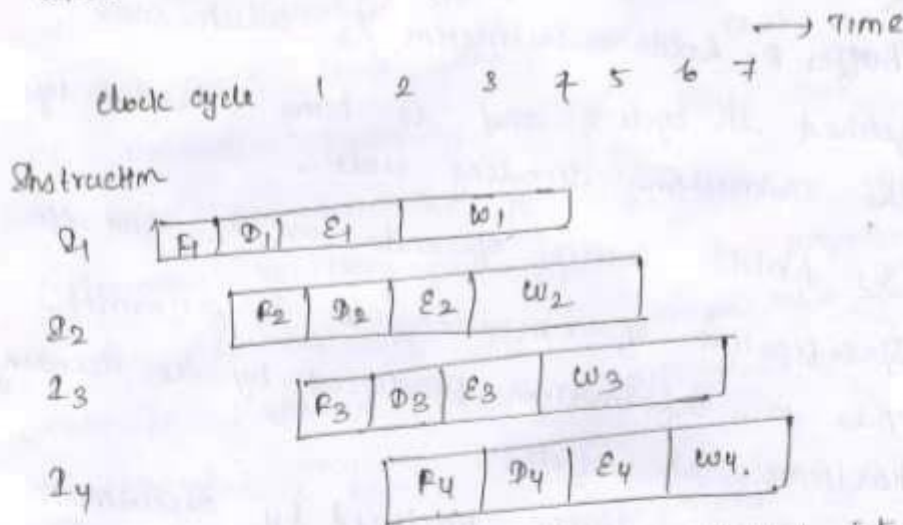


Fig 3.10a. Instruction execution divided into four steps

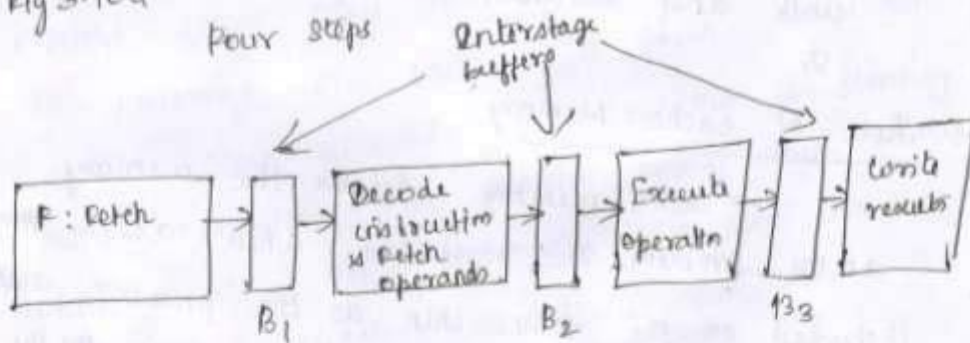


Fig 3.10b: Hardware Organization

~~For the information needed by the stages~~

During clock cycle 4, the information in the buffer is as follows.

- Buffer B₁ holds instruction I₃ which was fetched in cycle 3 and is being decoded by the instruction decoding unit
- B₂ holds source operands for I₂ and the specification of operation to be performed. This is information produced by the decoding hardware in cycles.
- B₃ holds results produced by execution unit and destination information for instruction I₁.

3.5.3 Role of cache memory

→ cache memory solves the memory access problem. In particular when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor.

(11)

3.3.4 Pipeline Performance and Hazard

→ Fig 3.10 completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation. The potential increase in performance resulting from pipeline is proportional to the number of pipeline stages. This increase would be sustained only without interruption throughout program execution.

→ For different stages reasons, one of the pipeline stages may not be able to complete its processing step task for a given instruction in the time allotted. eg. Execution of floating point operation may involve many clock cycles, which prevent other stages to proceed for some cycles, resulting in pipeline stalls.

Eg →

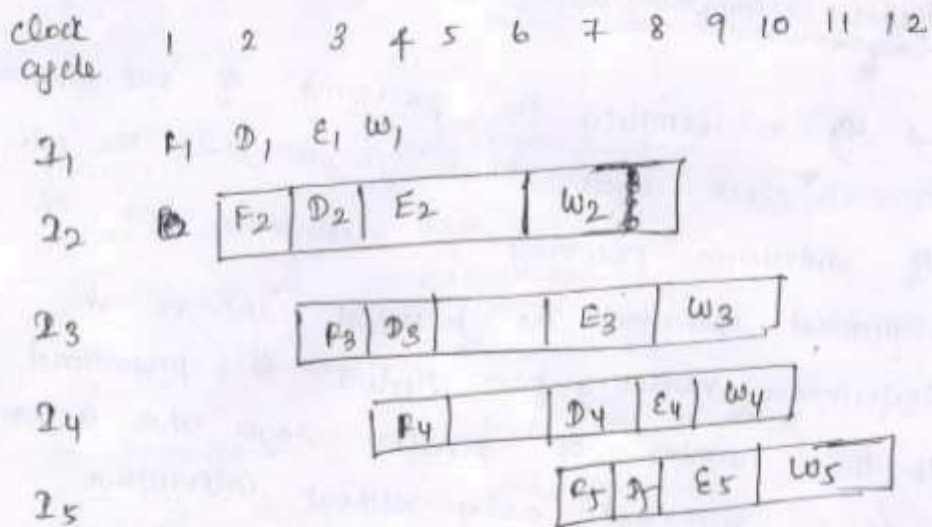


Fig 3-11 Execution of floating point operations

In the above example, pipelined operation is stalled for 3 clock cycles. Any condition that causes pipeline to stall is called a hazard. A hazard is created whenever there is a dependence between instructions and they are close enough that the overlap caused by pipelining would change the order of access to an operand.

- 2) Hazard - when next instruction cannot execute in following clock cycle and can potentially lead to incorrect computation results.

Q. There are three types of hazards

Structural hazards

These hazards are because of conflicts due to insufficient resources when even with all possible combinations, it may be possible to overlap the operation.

This mainly occurs when two instructions require the use of a given hardware.

Eg: memory access !

Data hazards or Data dependent hazards

These result when instruction in the pipeline depends on the result of previous instructions which are still in pipeline and not completed.

Eg $I_1 : A \leftarrow A + 5$

$I_2 : A \leftarrow A \times 2$

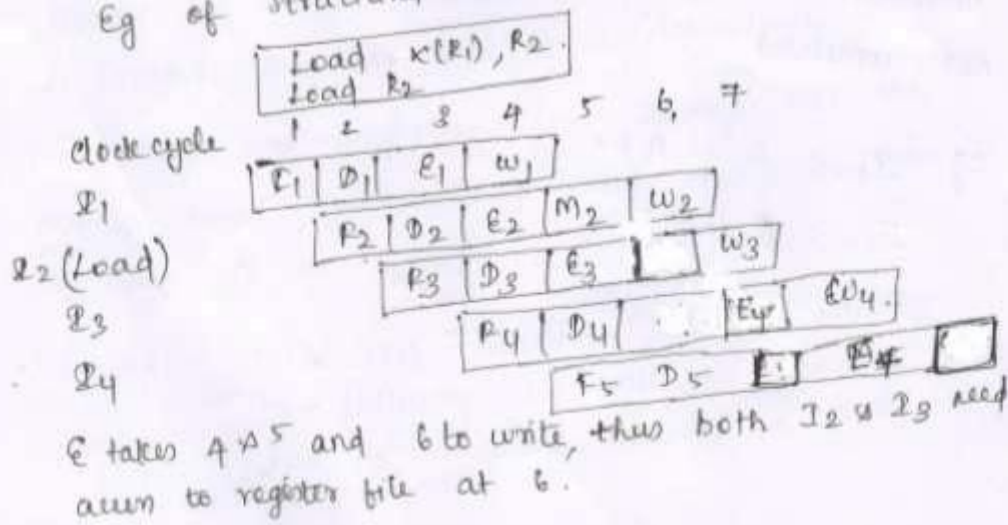
Instruction or Control hazard

They arise while pipelining branch and other instructions that change the contents of program counter.

The simplest way to handle these hazards is to stall the pipeline. Stalling the pipeline allows few instructions to proceed to completion while stopping the execution of those which results in hazards.

Removal of Structural Hazard ← Separate instruction caches for data and instruction

Eg of Structural Hazard



8.3.4.5 Throughput :-

The number of tasks that can be completed by a pipeline per unit time is called its throughput.

8.4 Data Hazards

→ A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason. The potential for obtaining incorrect results when operations performed concurrently can be demonstrated by a simple example.

→ Assume $A = 5$ and consider the following 2 operations

$$\text{Eg 1} \quad A \leftarrow 3 + A$$

$$B \leftarrow 4 \times A$$

When these operations are performed in the order given result is $B = 32$. But if they are performed concurrently, the value of A used in computing B would be original value 5 leading to an incorrect result.

(15)

→ If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in second instruction depends on result of first instruction.

Eg 2

$$A \leftarrow 5 \times C$$

$$B \leftarrow 20 + C$$

These can be performed concurrently, because they are independent.

Eg 3

Consider the following sequence of instructions

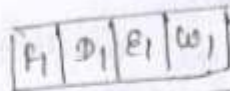
Add ~~R₂~~, R₀, R₁

$$R_2 \leftarrow R_0 + R_1$$

$$R_3 \leftarrow R_4 \times R_2$$

clock cycle 1 2 3 4 5 6 7 8 9 10

I₁ (Add)



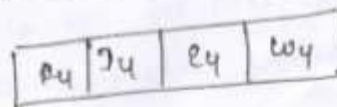
I₂ (Mul)



I₃



I₄



RAW Dependency

→ In the above example, the RAW dependency arises when the destination of one instruction is used as a source in the next instruction.

Since the register R_2 is used by both the instructions, the MUL instruction has to wait until the writeback step of the ADD instruction has been completed. So instruction I_2 is fetched in cycle 2, but its decoding must be delayed until the W step of I_1 is completed. Hence pipelined execution is stalled for 2 cycles for instruction I_2 & I_3 .

eg 4. Consider the pipeline
The two instructions are

$MUL\ R_4, R_5, R_2$

$ADD\ R_5, R_4, R_6$ give rise to a data

dependency. The result of the multiply instruction is placed into register R_4 , which in turn is one of the two source operands of the ADD instruction.

16

10 step of that instruction cannot be completed until the W step of the multiply instruction has been completed. Completion of step O_2 must be delayed to clock cycle 5 and is shown as O_{2A} in the figure 3-15. Instruction I_3 is fetched in cycle 3, but its decoding must be delayed because step O_3 cannot precede O_2 . Hence pipelined execution is

stalled for two cycles.

3.4.1 Handling Data Hazards

3.4.1.1 Operand Forwarding → Simple hardware technique to handle data hazard.

The data hazard just described arises because one instruction I_2 in figure 3-15 is waiting for data to be written in the register file. ALU stores output once the ~~execute~~ state completes step 4. ~~possibility eliminated~~

→ ~~is waiting for the result of instruction~~

Fig 3-16 shows part of processor datapath involving the ALU and the register file. ALU results are fed back as ALU ops. When the forwarding logic detects data dependency, it forwards ALU

output available in result register using data forwarding path to ALU for next operation.

Hence execution of next instruction proceeds without

Operand forwarding or register by passing

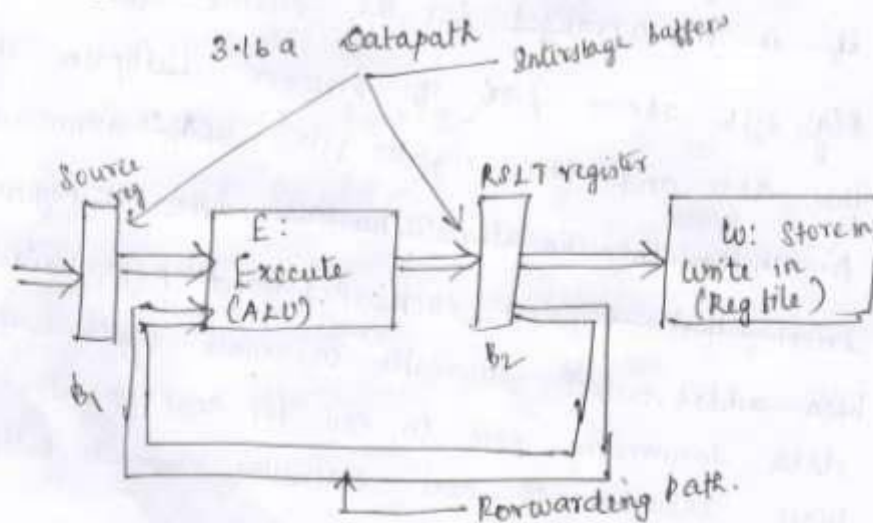
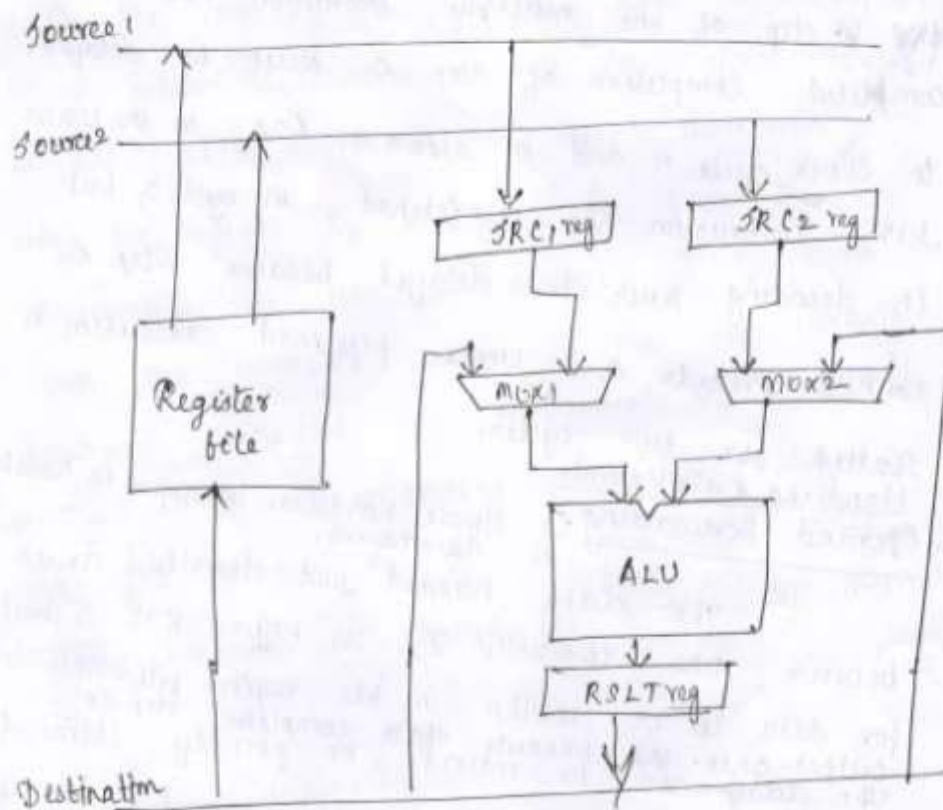
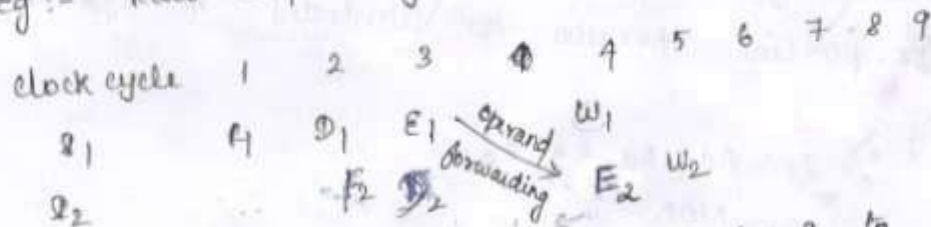


Fig 3.6b position of source

(17)

Eg:- Raw Dependency



→ In this example, instead of I₂ waiting for I₁ to complete W step, the operands from ALU are transferred directly to the E step of I₂, process called operand forwarding. Hence the delay can be reduced, or possible eliminated sometimes, if it is arranged for the result of instruction I₁ to be forwarded directly for use in step E₂.

→ After decoding I₂ and detecting the data dependency, a decision is made to use data forwarding and hence the execution of I₂ proceeds without interruption.

3.4.2 Handling Data Hazards in Software

A software approach to deal with data dependencies is that the compiler can introduce the required delays between instruction I₁ and I₂ by inserting NOP (No operation).

These registers constitute interstage buffers needed for pipeline operation as illustrated in Fig. 3.16b.

Eg:-

I₁: Add R₄, R₂, R₁₃

NOP

NOP

I₂: Add R₅, R₄, R₆

Another possibility is the compiler can rearrange the instructions to perform useful tasks in the NOP slots and thus can achieve better performance.

3.4.2.1 Drawbacks

- NOP increases the size of the code
- May lead to reduce performance on different implementations that have hardware supports

3.4.3 Side Effects

1. Consider instructions used for stack operations PUSH and POP, these instructions, in addition to storing new data in its destination location, the instruction changes the contents of a source register used to access one of its operands (stack pointer SP). All the techniques needed to handle data dependencies involving destination registers / location must also be applied to registers (source) affected by auto-increment / auto-decrement operations (PUSH & POP).

1- Implicit Data dependencies may be created by the instruction that involves condition codes

Eg:

Add R₁, R₃ ← sets the carry flag
 ADC R₄, R₄ ← uses the carry flag
 $R_4 \leftarrow R_4 + [R_4] + \text{carry}$

Instruction Hazards or Control Hazards [Branch: execute cause computer to begin execution of diff instruction sequence?]

→ Instruction hazards can cause a greater performance loss. The purpose of Instruction fetch unit is to supply the execution unit with a steady stream of instructions. whenever there is distraction in this stream, the pipeline stalls. Eg. Branch Instructions change the contents of PC

Handling control or Instruction Hazards

Instruction Queue and Prefetching

To reduce the effect of cache miss or branch penalty, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue

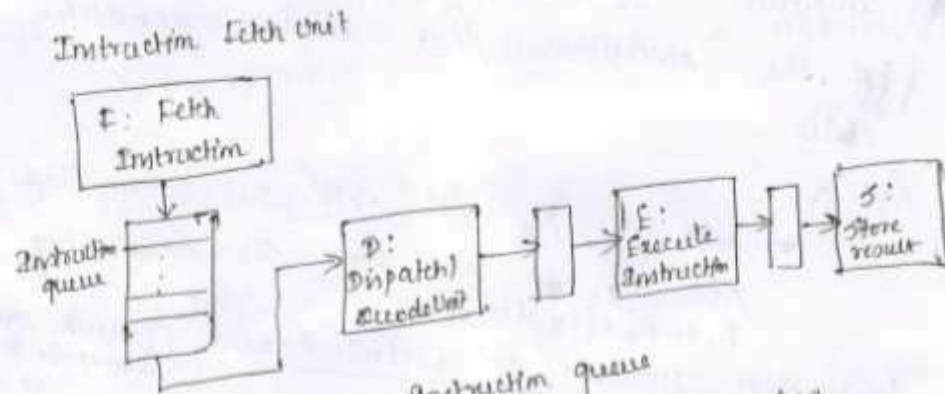
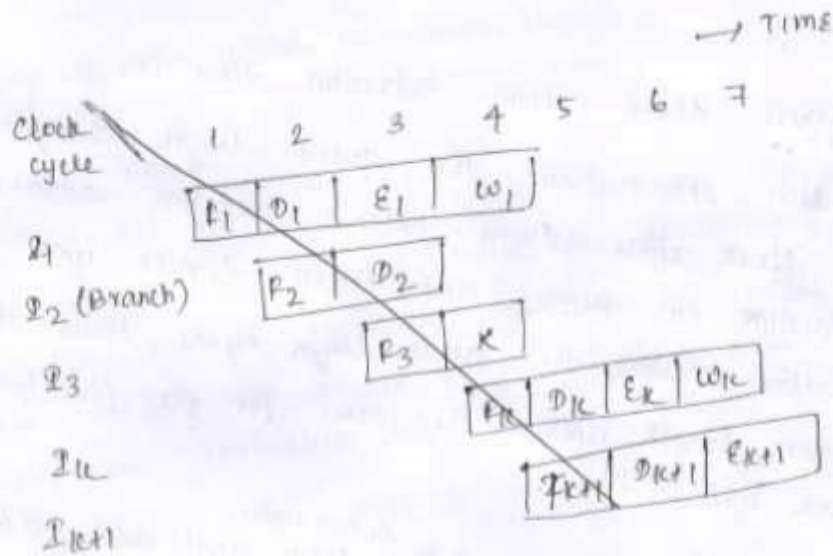


Fig. Use of Instruction queue

- A separate unit called dispatch unit takes instruction from front of the queue and sends them to execution unit. It also performs the decoding function.
- The fetch unit attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions during cache miss.
- In case of data hazard, the dispatch unit is not able to issue instructions from instruction queue. However fetch unit continues to fetch instruction and add them to queue.

29



Use of Instruction Queue during branch

Fig: Use of Instruction Queue during branch

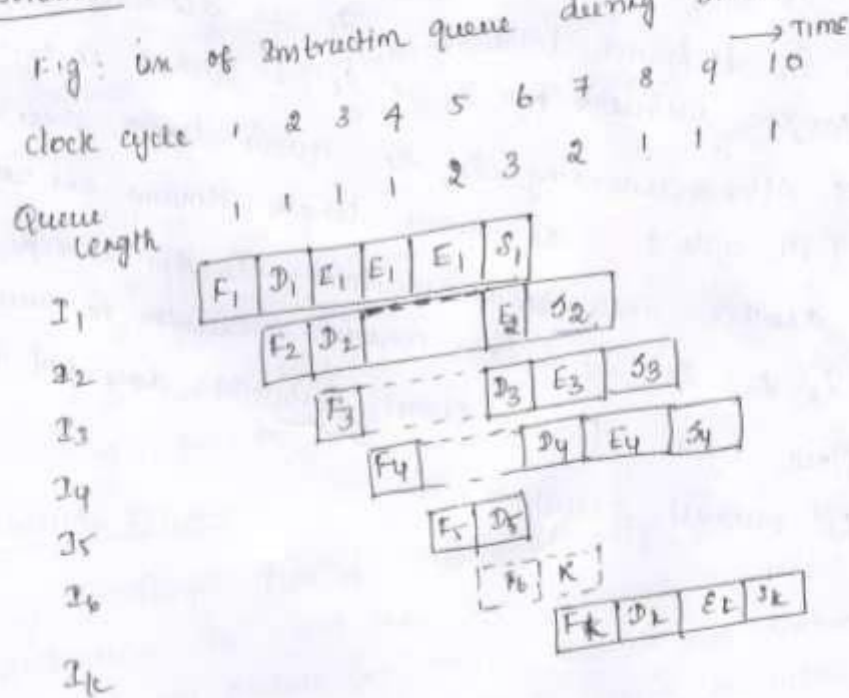


Figure above shows instruction time line.

→ It also shows how the queue length changes over the clock cycle. Every fetch operation adds one instruction to queue and every dispatch unit operation reduces the queue length by one. Hence the queue length remains the same for the first four clock cycles.

→ I_1 has 2 cycle stall ^{In two cycles}, Fetch unit adds 2 instructions but dispatch unit does not issue any instruction. Due to this queue length rises to 3 in clock cycles.

→ I_5 is branch instruction, I_6 is discarded and target instruction of I_5 , I_2 is fetched in cycle 7.

→ After discarding I_6 , the queue length drops to 1 in cycle 8. The queue length remains one until

another stall is encountered. In this example, I_1, I_2, I_3, I_4 & I_2 complete execution in successive clock cycles. Hence Branch instruction does not increase the overall execution time.

Handling of Instruction or Control Hazards

2. Branch Folding

The technique in which instruction fetch unit executes the branch instruction concurrently with the execution of other instruction is called branch folding. It occurs only if there exists at least one instruction in the queue other than branch. Instruction queue prevents the delay that may occur due to cache miss.

3. Conditional branching

It is a major factor that affects the performance of instruction pipelining. There are several approaches to deal with conditional branching.

They are

1. Multiple streams.
2. Prefetch Branch Target.
3. Loop buffer.
4. Branch prediction.

Multiple streams

Simple pipeline suffers penalty for a branch instruction. To avoid this the approach uses two streams; one stream to store the fetched instruction and fetched instruction after conditional branch instruction. It is not valid when branch is not valid.

The other stream stores the instruction from branch address. It is used when branch is valid.

Drawback

- 1) Need additional stream
- 2) Due to multiple streams, there are conditional delays.

Prefetch branch target

In this approach, when a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. It is used when branch is valid.

Loop Buffer

Loop buffer is ^{small} a very high speed memory. It is used to store recently prefetched instructions in sequence. The instructions are fetched from buffer, instead of ~~accessing~~ memory, thus avoiding memory access.

Advantages

In case of conditional branch, the instructions are fetched from buffer saving memory access time.

Branch Prediction

Prediction techniques can be used to

Check whether a branch will be valid or not valid.

These techniques reduce the branch penalty

The common prediction techniques are

1. Predict never taken
2. Predict always taken
3. Predict by opcode
4. Taken / Not taken switch
5. Branch history table.

→ In the first two approach, if prediction is wrong a page fault or protection violation error occurs, the processor then halts prefetching and fetches the instruction from desired address.

→ In third prediction, it depends on branch opcode

→ The fourth and fifth prediction techniques are dynamic, they depend on execution history of previously executed conditional branch instructions

Branch Prediction strategies

There are two types of branch prediction strategies.

1. Static branch strategy
2. Dynamic branch strategy

Static branch strategy

In this strategy, branch can be predicted based on branch code types statically (probability of branch). This won't produce accurate results every time.

Dynamic Branch strategy

This strategy uses recent branch history during program execution to predict whether or not the branch will be taken next time when it occurs.

Branch prediction statistics includes

- T: Branch Taken
- N: Not Taken
- NN: Last two branches not taken
- NT: Not branch taken & previous taken
- TT: Both last two branch taken
- TN: Last branch taken & previous not taken

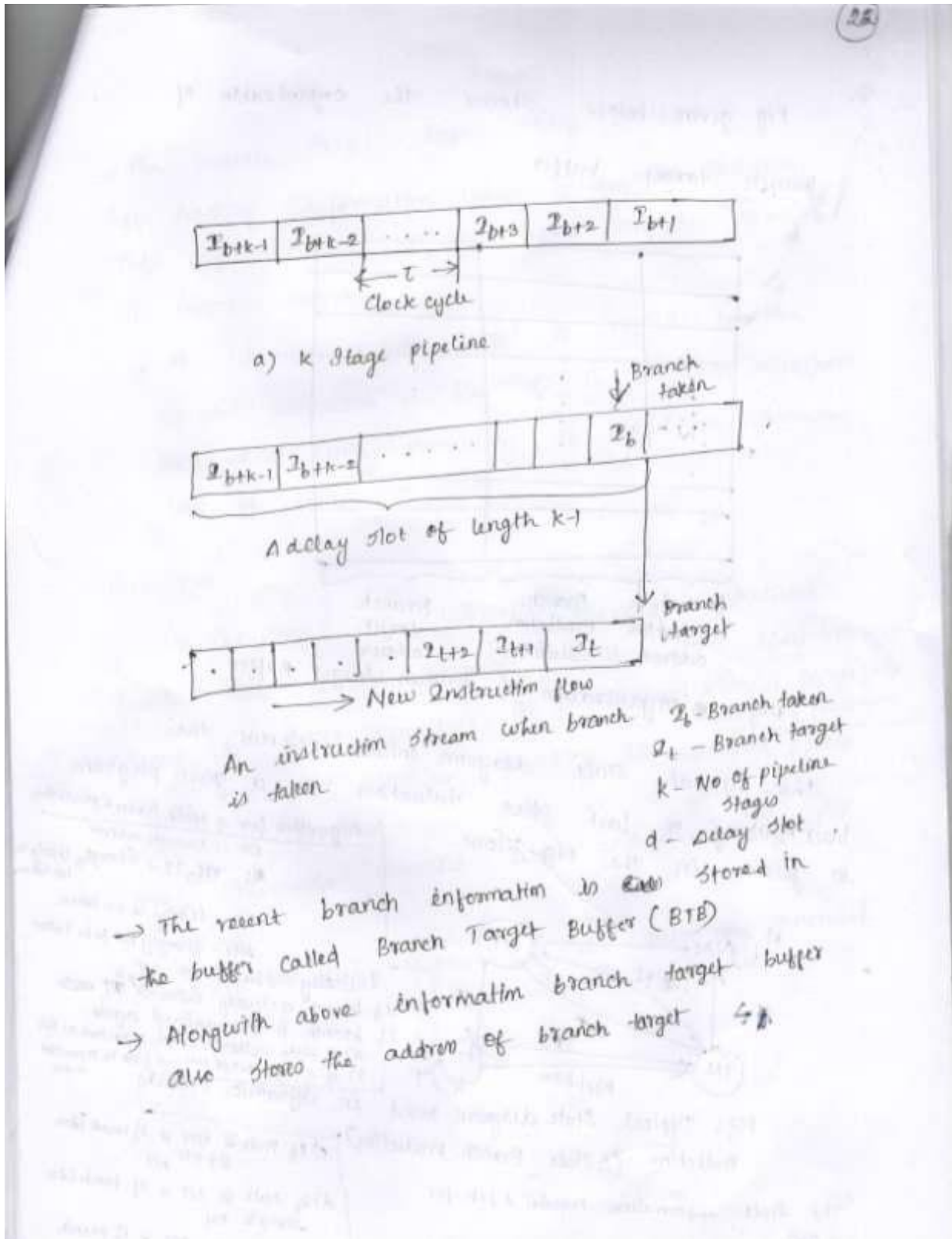


Fig given below shows the organization of branch target buffer

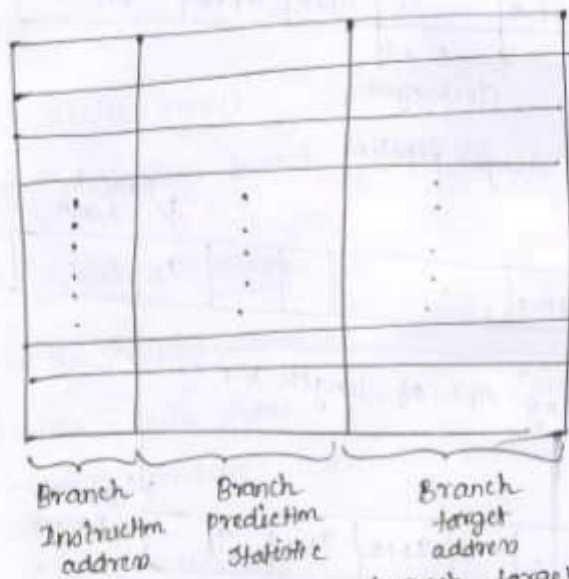


Fig: organization of branch target Buffer

The typical state diagram which shows the backtracking of last two instructions in a given program is given in the fig below.

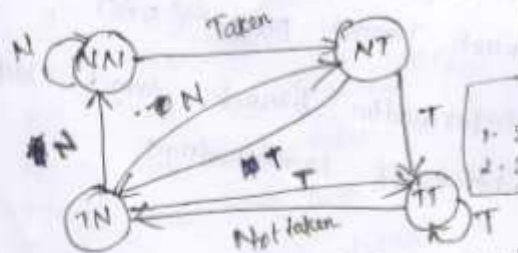


Fig: Typical State diagram used in dynamic branch Prediction (4 State Branch prediction)

The state information requires 2 bits for 4 states

Algorithm for 4 State Branch Prediction

BA - Branch address
 NT, TN, TT - Strongly likely to be taken
 TN - Likely to be taken
 NN - Strongly not to be taken

Initially state is NT
 1. If branch actually taken → change to TT
 2. If branch is encountered again
 a) If state is either TT or TN, fetch inst at BA
 b) If " is NN or NN → fetch is sequential order

c) If state is NN & if branch taken → change to NT
 d) If state is NT & if branch taken → change to TN
 e) If state is TN & if branch taken → change to TT

25

→ The branch target buffer entry contains base tracking information which guides the prediction. This information is updated upon completion of the current branch.

→ The branch target buffer is extended to store target instruction & few of its successor instructions itself and hence processing of conditional branches will be with zero delay.

Exceptions

→ Exceptions are internally generated unscheduled events that disrupt program execution and they are used to detect overflow. On the otherhand interrupt comes from outside of processor

→ Arithmetic overflow, invoking the operating system from user program and using an undefined instruction are internally generated events and hence called exceptions.

→ I/O device request is an externally generated event and hence called interrupt

Handling exceptions in MIPS architecture

Types of exceptions

There are two types of exceptions can occur in basic MIPS architecture implementation.

1. ~~Execution~~ 0

1. Execution of an undefined instruction
2. Arithmetic overflow in the instruction $\text{add } R_1, R_2, R_3$

Response to an exception

→ When an exception occurs the processor saves the address of the offending instruction in exception program counter (EPC) and then transfers control to the operating system at some specified address.

→ The operating system then takes the appropriate action, which may involve providing some service to the user program, taking some predefined action in response to an exception, or stopping the ~~execution~~ execution of the program & reporting an error.

→ Then by itself restart the program

methods used to communicate the reason for an exception

To handle the exception, it is must for the operating system to know the reason for exception and instruction that caused it. There are 2 main methods

1. Status register method

Uses a status register (called the cause register) which holds a field that indicates the reason for the exception

2. Vectored Interrupts Method

The address to which control is transferred is determined by cause of the exception

Exceptions in a pipelined Implementation

Multiple exceptions can occur simultaneously in a single clock cycle. In such cases, exception having highest priority is serviced first

In pipelined computers, the interrupts and exceptions are further classified as

- 1) Imprecise Interrupts or Imprecise exceptions
- 2) precise interrupts or precise exception

Imprecise interrupts or imprecise exceptions

Interrupts or exceptions in pipelined computers that are not associated with the exact instruction that was the cause of the interrupt or exception are called imprecise interrupts or imprecise exception

Precise interrupts or precise exception

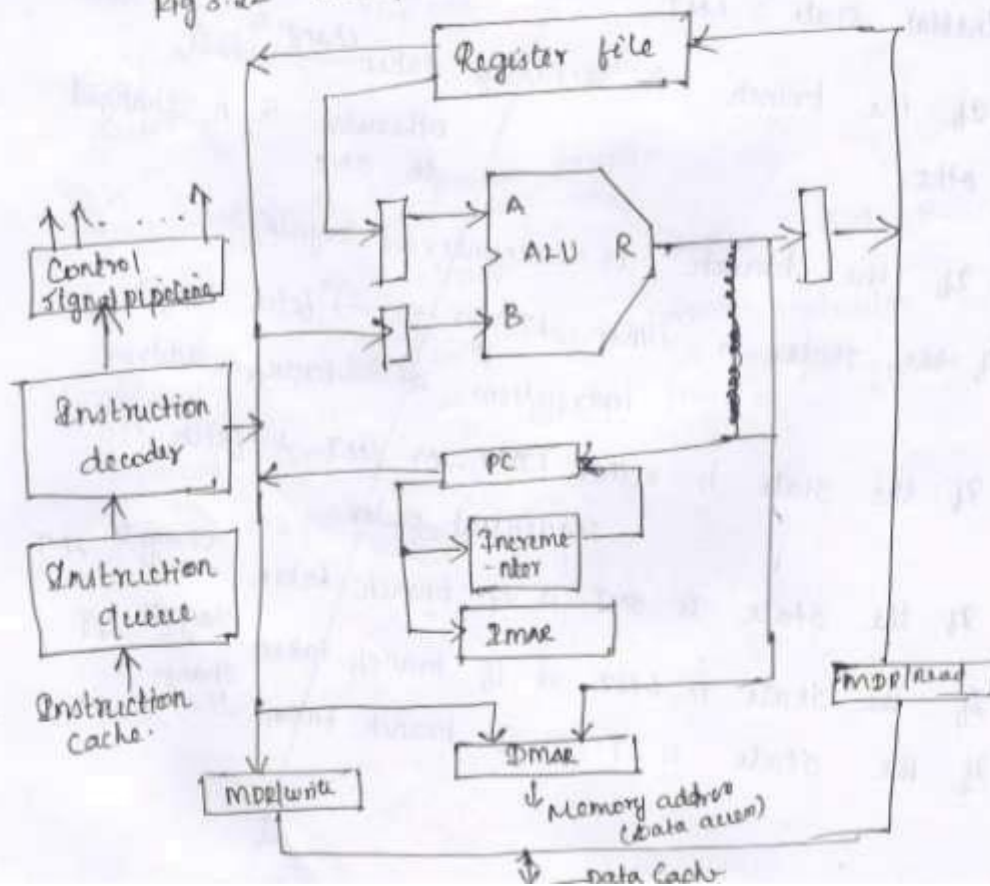
An interrupt or exception that is always associated with the correct instruction in pipelined computers is called ^{interrupt} precise or precise exception

→ The state information used in dynamic branch prediction algorithms requires 2 bits for 4 states and may be kept by the processor in a variety of ways

3.6 Datapath and Control Considerations

The datapath given below supports a 4 stage pipeline

Fig 3.22 Datapath modified for pipeline execution



Datapath has following features

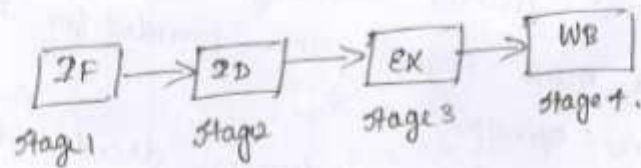
1. There are separate Instruction and data caches
2. Use two registers IMAR & DMAR for addressing instruction caches and data caches
3. The PC is connected directly to IMAR for parallel operation with ALU.
4. The instruction queue is present to load instruction from instruction cache
5. DMAR gets the address either from register file or from the ALU to support registers indirect and indexed addressing modes
6. Separate MDR register are provided for read and write operations
7. The output of the instruction decoder is connected to the control signal pipeline.

→ The following operation can be performed independently in the processor of Fig 5.22

- * Reaching an instruction from the instruction cache
- * Incrementing the PC

- * Decoding an instruction
- * Reading from or writing into the data cache
- * Reading the contents of up to two registers from the register file
- * Writing into one register in the register file
- * Performing an ALU operation.

There is no any shared resources, ~~they~~ these operations can be performed simultaneously in any combination. mostly in 4 stage pipeline



3.2 Exceptions Handling

Exceptional situations occur where the instruction execution order is changed in unexpected ways. Exceptional situations are harder to handle in a pipelined machine because the overlapping of instructions makes it more difficult to know whether an instruction can safely change the state of machine.

3.7 Control Implementation Scheme

Control unit

To perform an operation in computer system, the control unit generates control signals needed in proper sequence. The execution of an instruction may consist of

1. Fetch cycle
2. Decode cycle
3. Execute cycle
4. Store cycle.

The execution time done in sequence it is called by cycles.

The two main schemes to design control unit and to perform these operations in a controlled sequence namely

- (i) Hardwired control
- (ii) Microprogrammed control

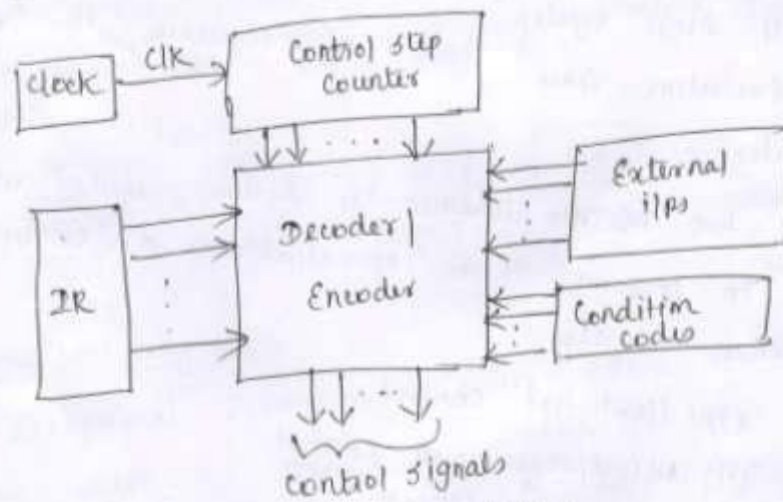
Hardwired control

- 1) Uses gates to generate signals (combinational circuit)
- 2) Its input logic signals governed by current machine instruction are transferred into a set of output control signals.

→ It gets set of inputs (from IR, flags, clock, system bus) and transforms them into a set of control signals. The sequence of operations carried out by machine is determined by wiring logic elements, hence "hardwired".

→ The required control signals are determined by the following information [Control signals: MAR, Read, Select, Add, Zin, Pout, Zout, Yin, End]

1. Contents of control step counter
2. Contents of IR
3. Contents of condition code flags
4. External input signals such as interrupt requests

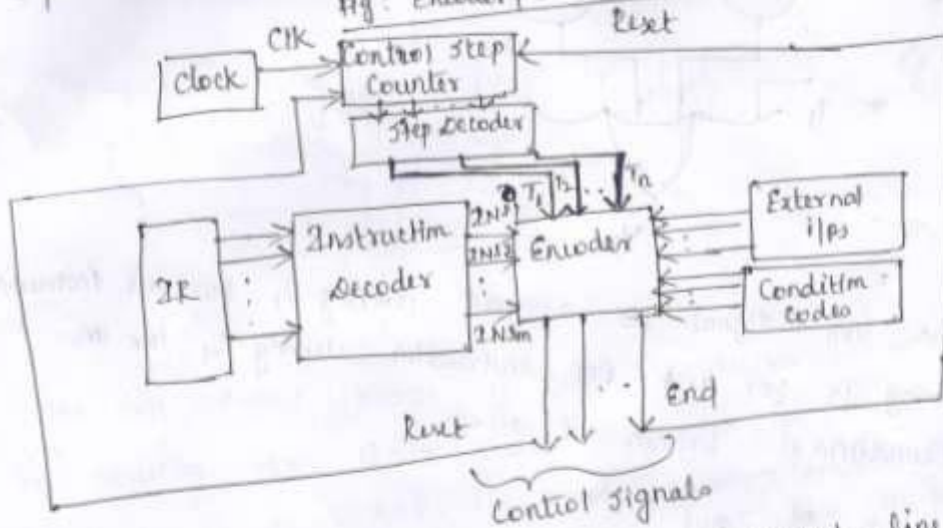


Encoder/Decoder to generate control signal Control Unit Organization

→ The decoder and encoder block is a combinational circuit that generates the required control o/p depending on the state of all its inputs.

→ The block can be separated to give detailed explanation of control signal

Fig: Encoder/Decoder Block for Control Signals

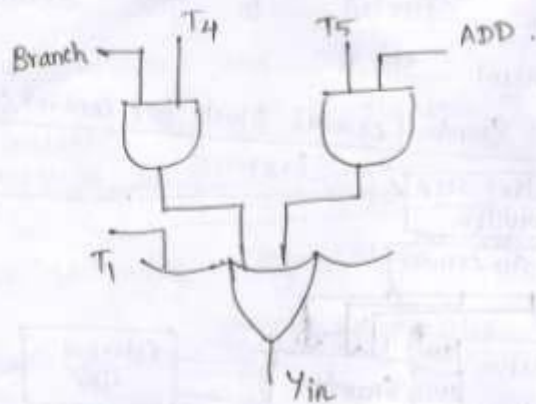


→ The step decoder provides a separate signal line for each step, or time slot in the control sequence.
 → It contains separate line for each machine instruction. One of $2N-1$ through $2N-m$ is set 1 and all other lines are set to 0.

Implem. Generation of control signal Y_{in} (implemented by logic function)

$$Y_{in} = T_1 + T_5 \cdot ADD + T_4 \cdot BRANCH + \dots$$

The Encoder circuit implements the above logic function to generate Y_{in} .

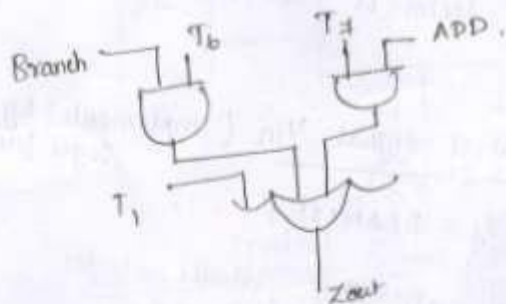


→ The Yin signal is asserted during T_1 for all instructions, during T_5 for an ADD instruction, during T_4 for an unconditional branch & so on...

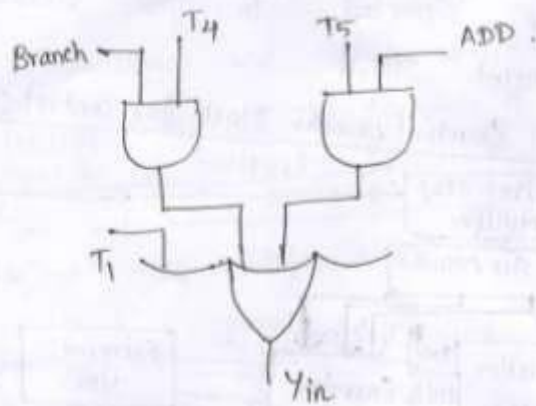
Generation of Zout

$$Z_{out} = T_2 + T_4 \cdot ADD + T_6 \cdot BRANCH + \dots$$

This can be generated at one clock period



Z_{out} is asserted during T_2 , during T_4 for ADD and T_6 for Branch.

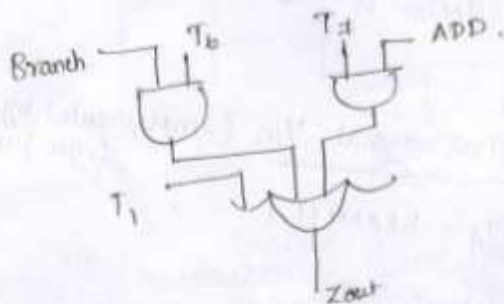


→ The Y_{in} signal is asserted during T_1 for all instructions, during T_5 for an ADD instruction, during T_4 for an unconditional branch & so on...

Generation of Z_{out}

$$Z_{out} = T_2 + T_4 \cdot ADD + T_6 \cdot BRANCH + \dots$$

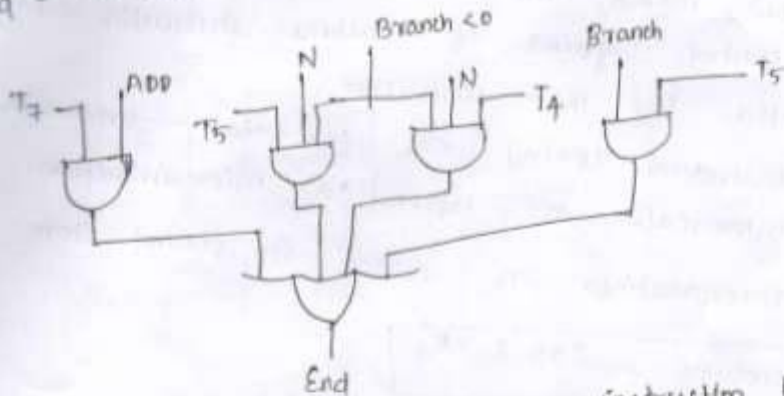
This can be generated at one clock period



Z_{out} is asserted during T_2 , during T_4 for ADD and T_6 for Branch.

Generation of
END:-

$$\text{End} = T_7 \cdot \text{ADD} + T_5 \cdot \text{BR} + (T_5 \cdot \text{NB} + T_4 \cdot \text{N}) \cdot \text{BRN} + \dots$$



The End signal starts a new instruction fetch cycle by resetting the control step counter to its starting value. Adv: High speed Disadv: Complex and less flexible.

Microprogrammed Control

→ Stores the control signals as microcode in the sequence
→ This microcode is read and executed in every clock cycle.

→ Here the control signals are generated by a program similar to machine language programs.

[For each instruction, microprogram is needed to generate control signal. They are stored in control store.

They are available in the form of microinstructions - combination of 0's and 1's called control word. microinstruction arranged in sequence called microprogram.)

→ A sequence of control word (where individual bits represent the various control signals as PCin, PCout, MARin, Read, MDout etc) corresponding to the control sequence of machine instruction constitutes microroutine for that instruction.

→ The individual control word for ~~that instruction~~ this microroutine are referred as microinstruction.

→ The microroutines are stored in control store.

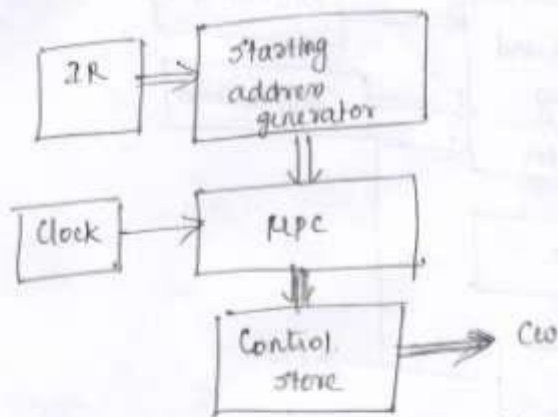
Microroutine: ADD R₁, R₂.

step 1 PCout, MARin, Read, Sel₄, Add, Xin
step 2 Zout, PCin, Yin
step 3 MDout, IRin
step 4 R₂out, MARin, Read
step 5 R₁out, Yin
step 6 MDout

• The instruction is executed based on the control signal.

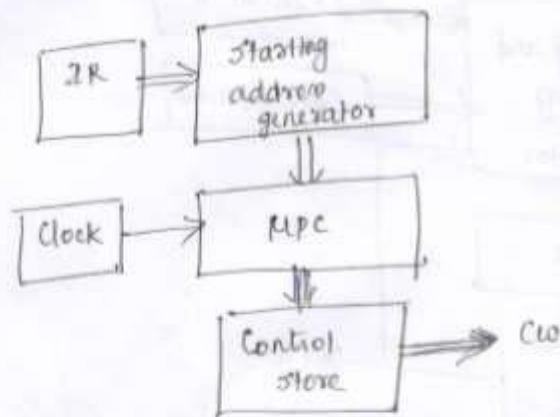
The control signal can be enabled by assuming 1 to the control word and can be disabled by assuming 0 to the control word.

Fig: Basic organization of microprogrammed control unit





→ MPC is used to read sequentially the control word from control store. Every time a new instruction is loaded into ZR, the output of block labelled starting address generator is loaded in MPC. The MPC is then automatically incremented by clock, causing successive microinstruction to be read from control store. Hence control signals are delivered to various parts of processor in correct sequence.

Fig. Basic organization of Microprogrammed Control unit



→ MPC is used to read sequentially the control word from control store. Every time a new instruction is loaded into IR, the output of block labelled starting address generator is loaded in MPC. The MPC is then automatically incremented by clock, causing successive microinstructions to be read from control store. Hence control signals are delivered to various parts of processor in correct sequence.

	<div>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</div> <div>Department of Computer Science & Engineering</div>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Introduction to Unit IV – PARALLELISM		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 1 / 10	

1. CONTENT LIST:

Introduction to Unit IV – Parallelism

2. SKILLS ADDRESSED:

Listening

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basics of parallelism

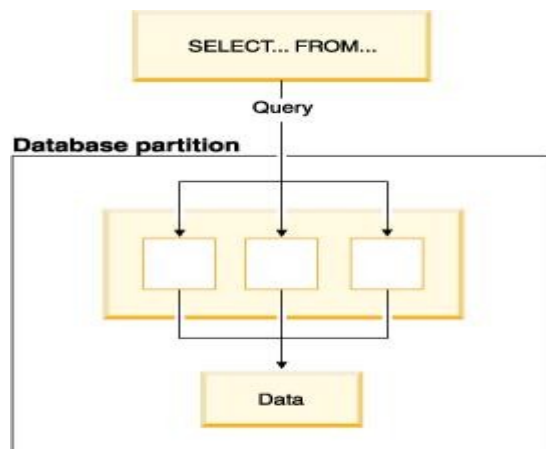
4. OUTCOMES:



- i. Explain the concept of parallelism in computer architecture
- ii. Demonstrate the major topics covered in unit4

5.LINK SHEET:

- i. Define parallelism
- ii. What are the topics covered in parallelism?

6.EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Instruction-level-parallelism		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 2 / 10	

1. CONTENT LIST:

Instruction-level-parallelism

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the instruction level parallelism and its features

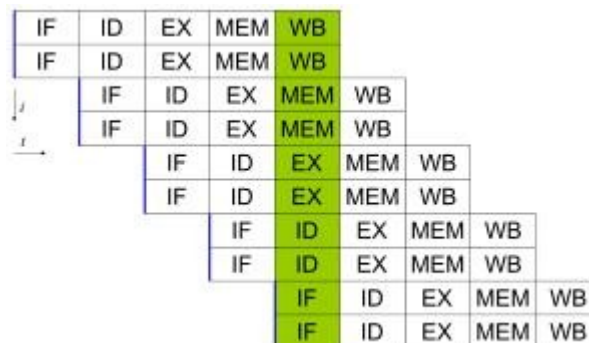
4. OUTCOMES:

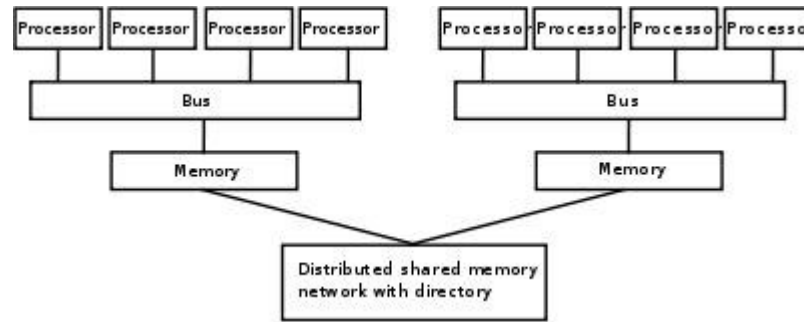
- iii. Enumerate the features of instruction level parallelism
- iv. Explain the detailed concept of instruction level parallelism



5.LINK SHEET:

- iii. Define instruction level parallelism
- iv. What is speculation?
- v. List the major factors which influence instruction set to undergo parallelism

6.EVOCATION: (5 Minutes)





	<p>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</p> <p>Department of Computer Science & Engineering</p>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Parallel processing challenges		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 3,4 / 10	

1. CONTENT LIST:

Parallel processing challenges

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate the students understand the challenges over parallel processing

4. OUTCOMES:



- v. Enumerate the characteristics of parallel processing
- vi. Explain the detailed concept of challenges in parallel processing

5.LINK SHEET:

- vi. Define data dependencies over parallel processing
- vii. What is static multiple issue processor?
- viii. List the factors to implement dynamic multiple issue processor

6.EVOCATION: (5 Minutes)



	<div>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</div> <div>Department of Computer Science & Engineering</div>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Flynn's classification		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 5/ 10	

1. CONTENT LIST:

Flynn's classification

2. SKILLS ADDRESSED:

Remembering

Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the classifications given by flynn

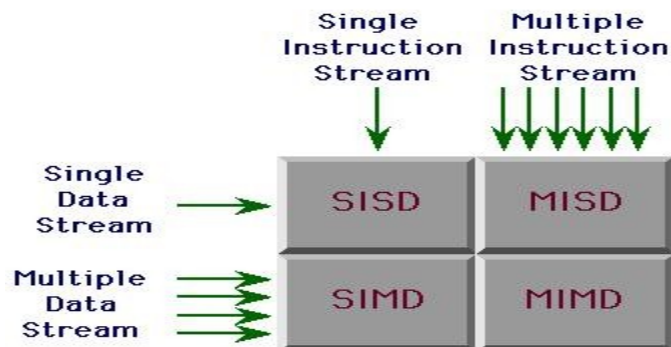
4. OUTCOMES:



- vii. Enumerate the basic classification of flynn
- viii. Explain the detailed concept of SIMD,MIMD and other classification of flynn

5.LINK SHEET:

- ix. Define SIMD
- x. What is Flynn classification?
- xi. Explain each classification in detail

6.EVOCATION: (5 Minutes)



	<div>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</div> <div>Department of Computer Science & Engineering</div>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Flynn's classification		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 6/ 10	

1. CONTENT LIST:

Flynn's classification

2. SKILLS ADDRESSED:

Remembering

Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the classifications given by flynn

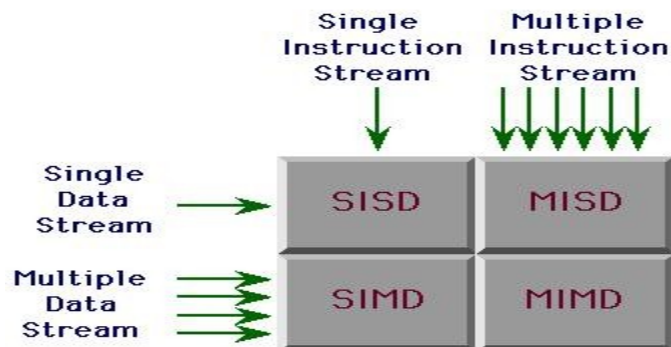
4. OUTCOMES:



- ix. Enumerate the basic classification of flynn
- x. Explain the detailed concept of SIMD,MIMD and other classification of flynn

5.LINK SHEET:

- xii. Define SIMD
- xiii. What is Flynn classification?
- xiv. Explain each classification in detail

6.EVOCATION: (5 Minutes)



	<div>SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR</div> <div>Department of Computer Science & Engineering</div>	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Hardware multithreading		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 7/ 10	

1. CONTENT LIST:

Hardware multithreading

2. SKILLS ADDRESSED:

Remembering Learning

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the detail concept of hardware multithreading

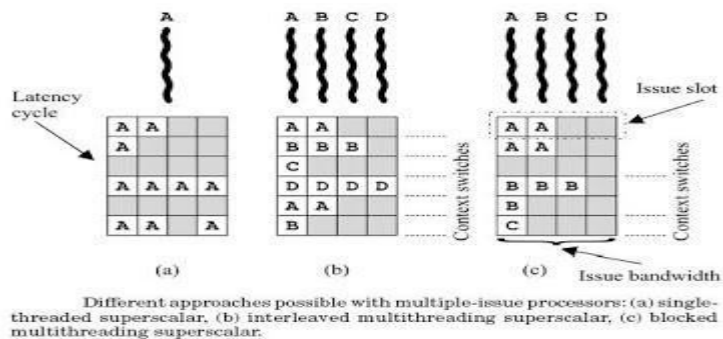
4. OUTCOMES:

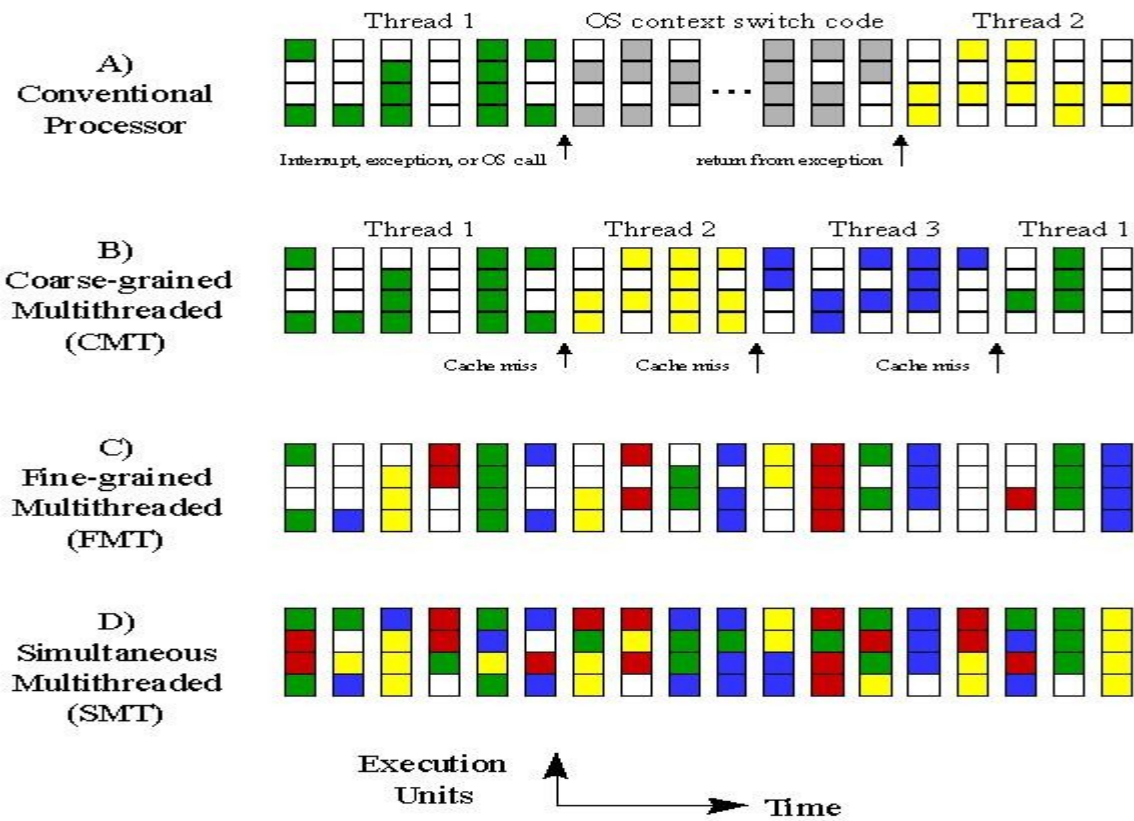
- xi. Demonstrate hardware multithreading in detail
- xii. Explain the detailed concept of hardware multithreading and its features



5. LINK SHEET:

- xv. Define multithreading
- xvi. List the types of multithreading?
- xvii. Discuss the factors influencing hardware multithreading in detail.

6. EVOCATION: (5 Minutes)





	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Multicore processors		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 8/ 10	

1. CONTENT LIST:

Multicore processors

2. SKILLS ADDRESSED:

Remembering

Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the detail concept of multicore processors

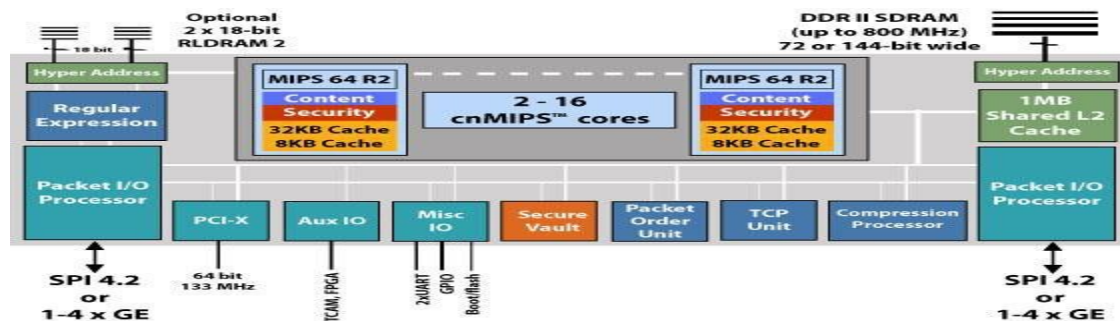
4. OUTCOMES:



- xiii. Demonstrate multicore processors in detail
- xiv. Sort the different types of multicore processors

5. LINK SHEET:

- xviii. Define processor
- xix. List the types of multicore processors?
- xx. Discuss the various architecture underlying with multicore processors.

6. EVOCATION: (5 Minutes)



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Multicore processors		
Time:	45 Minutes	
Lesson. No	Unit 4 – Lesson No. 9/ 10	

1. CONTENT LIST:

Multicore processors

2. SKILLS ADDRESSED:

Remembering

Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the detail concept of multicore processors

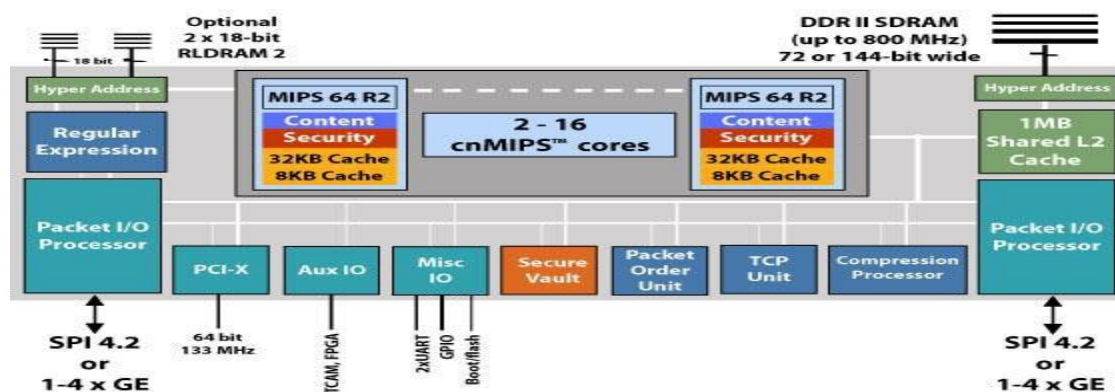
4. OUTCOMES:

- xv. Demonstrate multicore processors in detail
- xvi. Sort the different types of multicore processors

5. LINK SHEET:

- xxi. Define processor
- xxii. List the types of multicore processors?
- xxiii. Discuss the various architecture underlying with multicore processors.

6. EVOCATION: (5 Minutes)



UNIT- 4

Parallelism

→ In order to execute the program sequentially there is need to follow parallelism.

A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. parallelism has sometimes been viewed as rare and exotic subarea of computing, interesting but of little relevance to the average programmer.

→ In parallelism, multiple copies of hardware unit is used. All copies can operate simultaneously

Characteristics of parallelism

1. Microscopic vs macroscopic
 2. Symmetric vs Asymmetric
 3. Fine grain vs Coarse grain
 4. Explicit vs Implicit
1. Microscopic - characterize parallel facilities, but not especially visible.

Examples of Microscopic parallelism: parallel operation in an ALU, parallel access to general purpose registers, parallel transfer across I/O bus.

Macroscopic parallelism: Multiple, identical processors

2. Symmetric parallelism

Refers to Multiple identical processors.

Example to Macroscopic parallelism: eg: dual processor PC

Asymmetric parallelism

Refers to multiple dissimilar processors.
It also comes under Macroscopic parallelism. Example
PC with graphics processor

3. Fine grain: parallelism among individual instructions
or data elements

Coarse grain: parallelism among programs or large
blocks of data

4. Explicit and Implicit parallelism

Explicit: visible to programmer
Requires programmer to initiate and
control parallel activities.

Implicit: Invisible to programmer
Hardware runs multiple copies of program
automatically

Instruction Level Parallelism

Pipelining exploits the potential parallelism among instructions. This parallelism is called Instruction level parallelism (ILP). ~~That~~ This means that there are many instructions in code that don't depend on each other. Thus the instructions can be executed in parallel.

Instruction level parallelism can be done by

Building compilers to analyze the code

Building hardware to be even smarter than that code.

Exploiting the parallelism in Pipeline

Two Methods of exploiting the parallelism

→ Increase pipeline depth

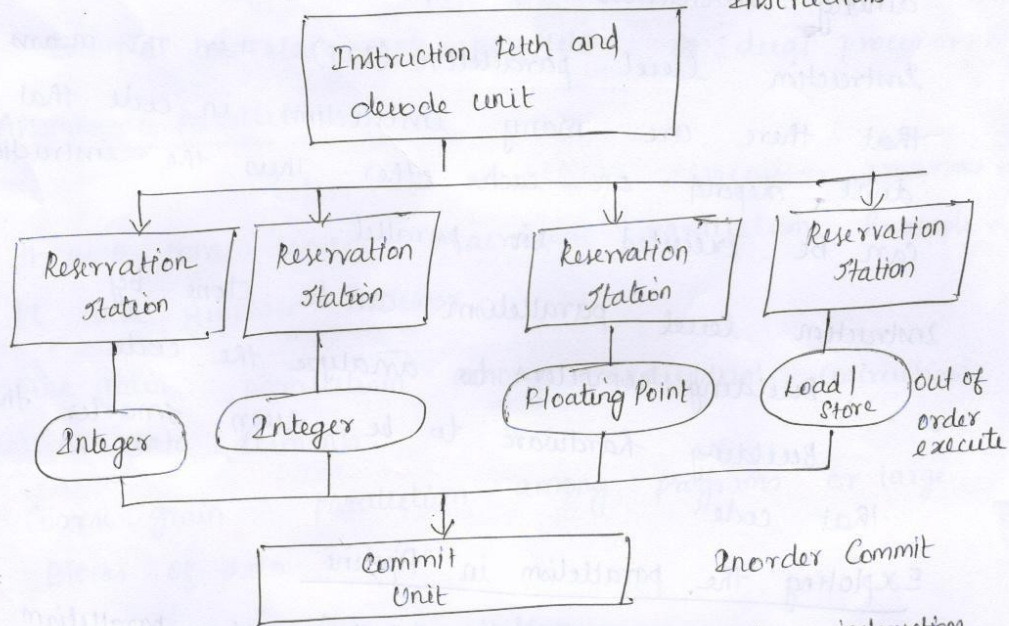
→ Multiple Issue: * Replicate internal components

* Launch multiple instructions in every pipeline stage.

It is sometimes useful to flip the metric and use IPC or Instructions per clock cycle. 4 GHz four way multiple issue MP can execute a peak rate of 16 billion instructions per second or an IPC of 4.

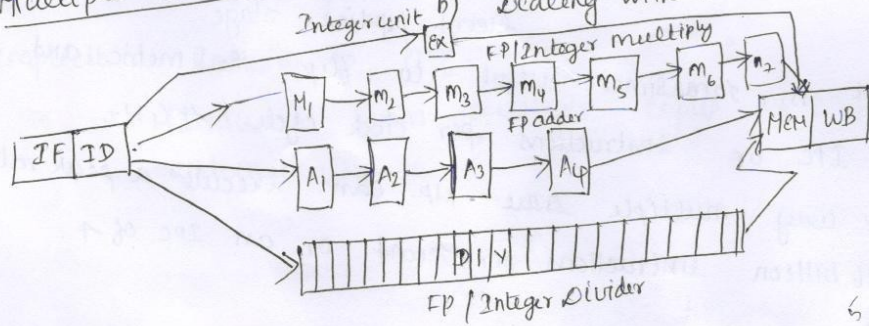
Assuming a five stage pipeline, such a processor would have 20 instructions in execution at any given time.

Fig: Exploiting parallelism in pipeline using Instructions



Today's high end microprocessor issues 3 to 8 instructions every clock cycle

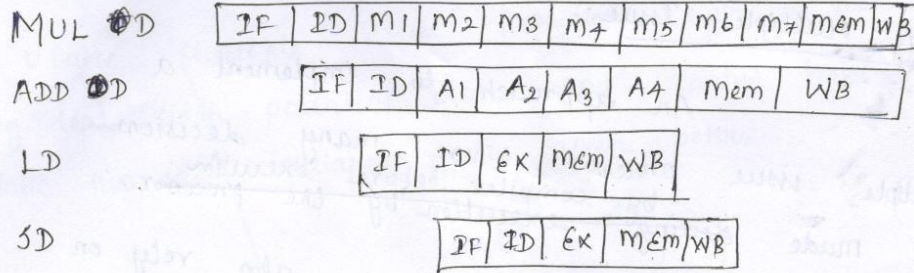
Multiple Issue Pipeline: a) packaging instructions into issue slots



1 2 3 4 5 6 7 8

(8)

Double - floating point
Integer - whole no



Concept of Speculation

→ One of the most important methods for finding and exploiting IP is speculation. It is an approach that allows the compiler or processor to guess about the properties of an instruction, so as to enable execution to begin for other instructions that may depend on the speculated instruction.

→ Speculation may be done in compiler or by the hardware. For example, the compiler can use speculation to reorder instructions, moving the instruction across a branch or a load across a store. The processor hardware can perform the same transformation at runtime using techniques.

Since speculation can improve performance when done properly and decrease performance when done carelessly, significant effort goes into deciding when it is appropriate to speculate. The static and dynamic multiple

Static Multiple Issue:-

An approach to implement a multiple issue processor where many decisions are made ~~during~~ ^{by compiler before execution} ~~execution by the processor~~.

Most static issue processors also rely on the compiler to take on some responsibility for handling data and control hazards.

Static Multiple Issue with MIPS ISA

To give a flavour of static multiple issue, let us use simple two issue MIPS processor where one of the instructions can be an integer ALU operation or branch and the other can be a load or store. The figure given below shows how the instructions look as they go into the pipeline in pairs.

Instruction type	pipe stages						
	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			IF	ID	EX	MEM	WB
Load or store				IF	ID	EX	MEM
ALU or branch					IF	ID	EX
Load or store						IF	ID

static two issue pipeline in operation

Static Multiple Issue vary in how they deal with potential data and control hazards.

Dynamic Multiple Issue

An approach to implement a multiple issue processor where many decisions are made during execution by the processor. This however shows the difference between the compiler and hardware.

Parallel Processing Challenges or 2LP challenges - hardware/software

In order to attain parallelism, there should not be dependencies among instructions which are executing in parallel. [parallel programming - scheduling, partitioning task into parallel pieces, balancing load evenly]

Two such report to face challenges on parallelism are

H/W Terminology Data Hazards (RAW, WAR, WAW)

s/w Terminology Data Dependencies (RAW, WAR, WAW)
Amdahl's Law: calculate performance gain can be obtained by improving some portion of computer: speedup:
Dependencies and Hazards per for entire task using ~~deep~~

→ Dependencies are a property of programs
→ If two instructions are data dependent they cannot execute simultaneously.

Speed up: $\frac{\text{Performance of entire task using Improved machine}}{\text{Performance for entire task using Old machine}}$

→ A dependence results in a hazard and the hazard causes a stall.

→ Data dependencies may occur through registers or memory.

Types of Dependencies

* Name dependencies

Output dependence

Anti dependence

* Data True Dependence

* Control Dependence

* Resource Dependence.

Name Dependencies

Output Dependence

Instruction j writes operand before Instr i writes it

→ I : Sub r_1, r_4, r_3
→ J : add r_1, r_2, r_3

K : Mul r_6, r_1, r_7

called an output dependence by compiler writers.

This also results from reuse of name r_1 . If it causes hazard, write after write

Anti dependence

when two instructions use the same register or memory location, called a name, but no flow of data between the instructions is associated with that name;

2 versions of name dependence.

Inst_j writes operand before inst_i reads it

→ I : sub r₄, r₁, r₃
J : add r₁, r₂, r₃

K : mul r₆, r₁, r₇

called an 'antidependence' by compiler writers.

This results from reuse of name r₁.

If antidependence caused a hazard in the pipeline,

Called write After Read (WAR) hazard

Data Dependencies Through register and memory

* Dependencies through registers are easy:

lw r₁₀, 10(r₁₁)

add r₁₂, r₁₀, r₈

Just compare register names.

* Dependencies through memory are harder.

sw r₁₀, 4(r₂)

lw r₆, 0(r₄)

is r₂ + 4 = r₄ + 0? If so they are dependent, if not

they are not.

6

Data True Dependencies

An Instruction j is data dependent on instruction i if either of the following hold:

Instruction i produces a result that may be used by instruction j , or instruction j is data dependent on instruction k , and instruction k is data dependent on instruction i .

```

Loop LD  F0, 0(R1)
      ADD F4, F0, F2
      SD  F4, 0(R1)
      SUB R1, R1, #8
      BNE R1, R2, Loop
    
```

Control Dependencies

Control dependence determines the order of an instruction i , with respect to branch instruction so that the instruction i is executed in correct program order. Every instruction is control dependent on some set of branches and in general, these control dependencies must be preserved to preserve program order.

if P_1 {

S_1 ;

}

if P_2 {

S_2 ;

}

S_1 is control dependent on P_1 and
 S_2 is control dependent on P_2 but not on P_1 .

Two constraints imposed by control dependencies

1. An instruction that is control dependent on a branch cannot be moved before the branch
2. An instruction that is not control dependent on a branch cannot be moved after the branch.

Resource Dependencies

An instruction is resource dependent on a previously issued instruction if it requires a hardware resource which is still being ~~used~~ by a previously issued instruction

eg: $\text{div } r_1, r_2, r_3$
 $\text{div } r_4, r_2, r_5$

7

Instruction Dependence example

For the following code identify all data and name dependencies between instructions and give dependency graph.

1. LD $F_0, 0(R_1)$
2. ADD.D F_4, F_0, F_2
3. S.D $F_4, 0(R_1)$
4. LD $F_0, -8(R_1)$
5. ADD.D F_4, F_0, F_2
6. S.D $F_4, -8(R_1)$

True Data Dependence

→ Instruction 2 depends on Instruction 1 (instruction 1 result in F_0 used by instruction 2), similarly instr(4,5)
 → Instruction 3 depends on instruction 2 (instruction 2 result in F_4 used by instruction 3) ||| Instruction (5,6)

Name Dependencies

Output name (WAW)

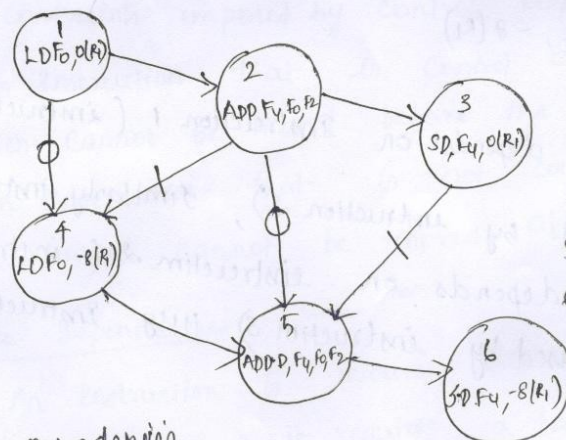
Instr 1 has an output name dependencies over result register (name) F_0 with instr 4.
 Instr 2 has an output name dependence over result reg name F_4 with Instr 5.

Anti Dependence (WAT)

Instr2 has an antidependence with instr4 over reg (name) F0 which is an operand of instr1 & the result instr4.

Instr3 has an antidependence with instr5 over register (name) F4 which is an operand of instr3 and the result of instr5.

Instruction Dependency Graph



- 1 LD F0, 0(R1)
- 2 ADD.D F4, F0, F2
- 3 SD F4, 0(R1)
- 4 LD F0, -8(R1)
- 5 ADD.D F4, F0, F2
- 6 SD F4, -8(R1)

Data dependencies

(1, 2) (2, 3) (4, 5) (5, 6)

O/P dependencies (0)

(1, 4) (2, 5)

Anti dependence (-)
(2, 4) (3, 5)

(8)

Flynn's classification or Taxonomy of Computer

Architecture

The most popular taxonomy of computer Architecture was defined by Flynn in 1966. Flynn's classification scheme is based on the notion of stream of information. The most common type of two information is instruction and data. Instruction is performed by processing unit. The data stream is performed between memory & processing unit. Mostly it is single or multiple.

He distinguished the taxonomy into four types

1. single Instruction single data streams (SISD)
2. single Instruction multiple data streams (SIMD)
3. Multiple Instruction single data streams (MISD)
4. Multiple Instruction Multiple data streams (MIMD)

SISD

A single processor computer system is called single Instruction stream, single data stream system (SISD). A program executed by the processor constitutes the single Instruction stream, and the sequence of data items that it operates on constitutes the single data stream. Fig 4.1 shows SISD Architecture

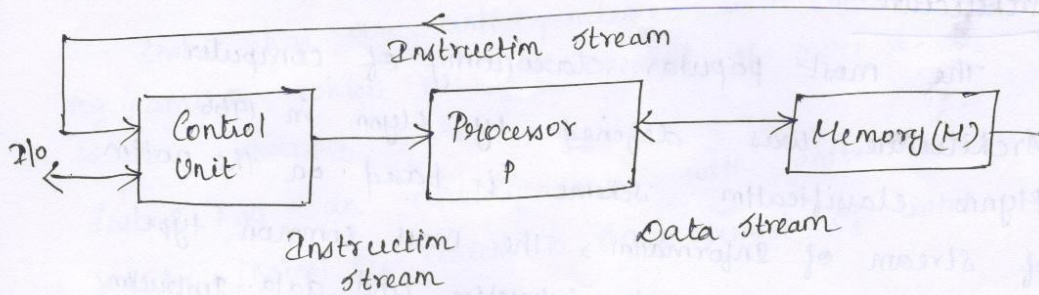
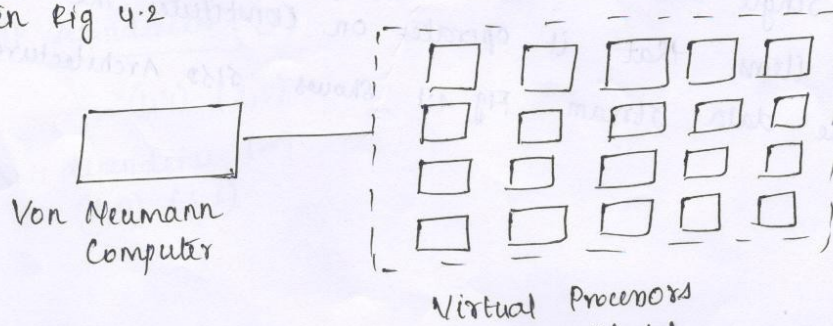


Fig 4.1 SIMD Architecture

SIMD Architecture - A single Stream of Instruction is broadcast to no of processors.

The SIMD form of parallel processing also called array processing. The SIMD model of parallel computing consists of 2 parts : a front end computer of the usual von Neumann style, and a processor array. The processor array is a set of identical synchronised processing elements capable of simultaneously performing the same operation on different data. SIMD architecture model is given in Fig 4.2



9

Fig 4.3 illustrates the structure of an array processor related with SIMD architecture model

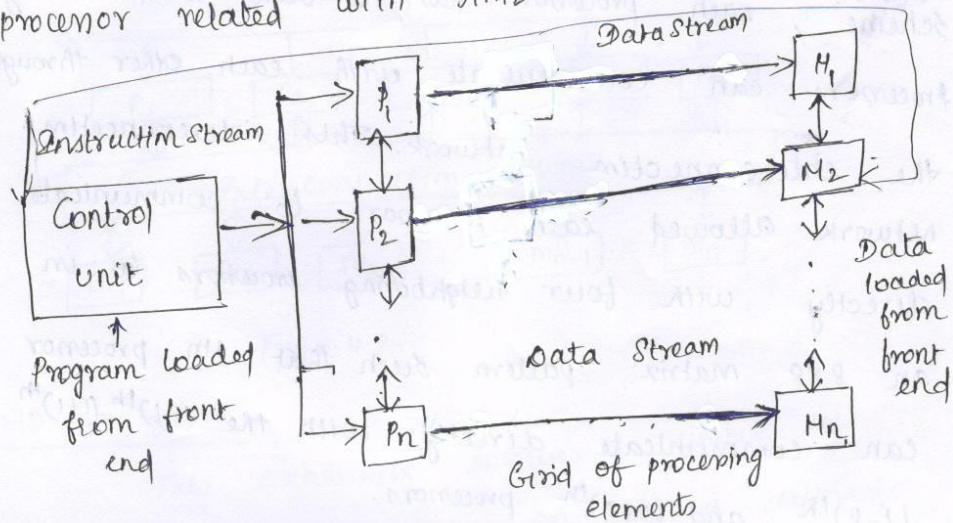


Fig 4.3 SIMD Architecture

A two dimensional grid of processing elements executes an instruction stream that is broadcast from a central control processor. As each instruction is broadcast, all elements execute it simultaneously. Each processing element is connected to its four nearest neighbors for purpose of exchanging data and around connections to memory. End around connections may be provided in both rows and columns. The grid of processing elements can be used to solve two dimensional problems. If each of the element of the grid represents a point in space the array can be used to compute temperature at points in the interior of conducting plane.

Fig 4.4.1 shows Two SIMO scheme. In the first scheme, each processor has its own local memory. Processors can communicate with each other through the interconnection network. This interconnection network allowed each processor to communicate directly with four neighboring processors in an 8x8 matrix pattern such that i th processor can communicate directly with the $(i-1)$ th, $(i+1)$ th, $(i-8)$ th and $(i+8)$ th processors.

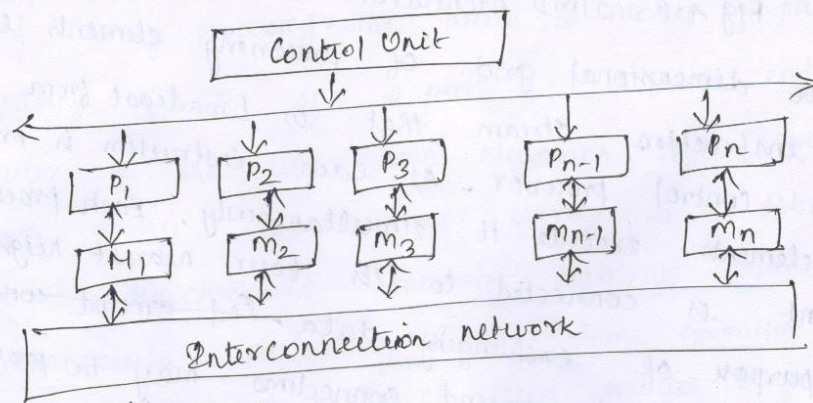


Fig 4.4.1

Fig 4.4.2 shows processors and memory modules communicate with each other via the interconnection network. Two processors can transfer data between each other via intermediate memory module(s) or possibly through intermediate processor(s).

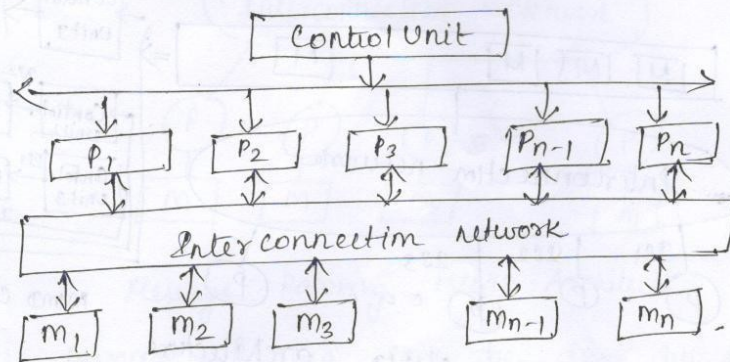


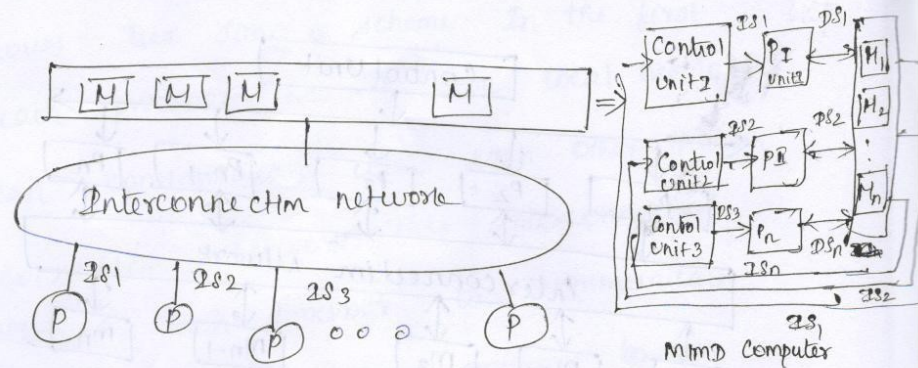
Fig 4.4.2

MIMD Architecture

This architecture involves a number of independent processors each executing a different program and accessing its own sequence of data items. Multiple Instruction Multiple Data Stream (MIMD) parallel architectures are made of multiple processors and multiple memory modules connected together via some interconnection network. They fall into two broad categories : shared memory or message passing

Shared memory

Processors exchange information through their central shared memory in shared memory systems. Fig 4.5 related to shared memory is shown below

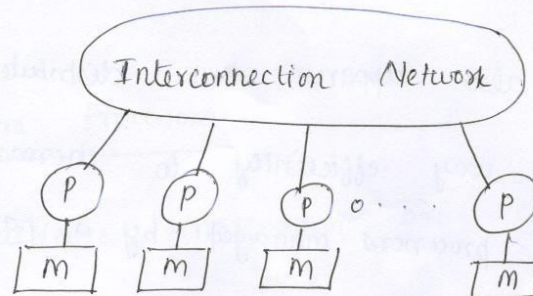


Shared Memory MIMD architecture

A shared memory system typically accomplishes interprocessor coordination through a global memory shared by all processors. There are typically a sender system that communicate through a bus and cache memory controller. A shared memory organization is the one in which processors communicate by reading and writing locations in a shared memory that is equally accessible to all processors.

Message Passing MIMD Architecture

A message passing (also referred to as distributed memory) typically combines the local memory and processor at each node of the interconnection network. There is no global memory, so it is necessary to move data from local one local memory to another by means of message passing.



Message Passing MIMD Architecture

The Message Passing can be done by ~~the~~ send)

Receive pair of commands, which must be written into the application software by a programmer.

Message passing systems are a class of Multiprocessors in which each processor has access to its own local memory.

Unlike Shared Memory systems, Communications in message passing systems are performed through send and receive operations.

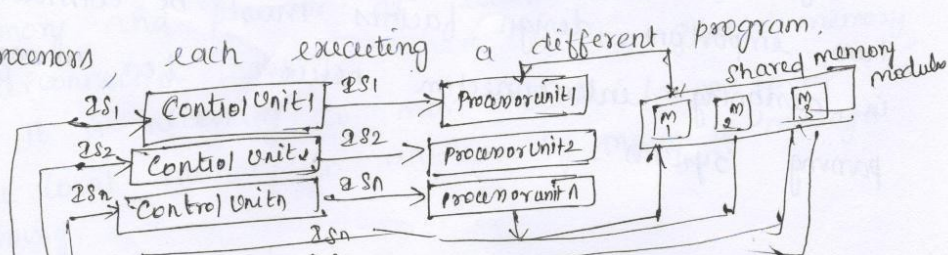
Message Passing Multiprocessors employ a variety of static networks in local communication. Two important design factors must be considered in designing interconnection network for message passing systems.

→ It was also apparent that distributed memory is one only way efficiently to increase the number of processors managed by a parallel and distributed system.

The Distributed shared memory (DSM) architecture began to appear in systems like the SGI Origin 2000 and others. In such systems, memory is physically distributed; for example the hardware architecture follows the message passing school of design, but the programming model follows the shared memory school of thought.

MISD (Multiple Instruction Stream, single data stream)

The fourth possibility is a Multiple Instruction Stream, single data stream. In such a system, a common data structure is manipulated by separate processors, each executing a different program.



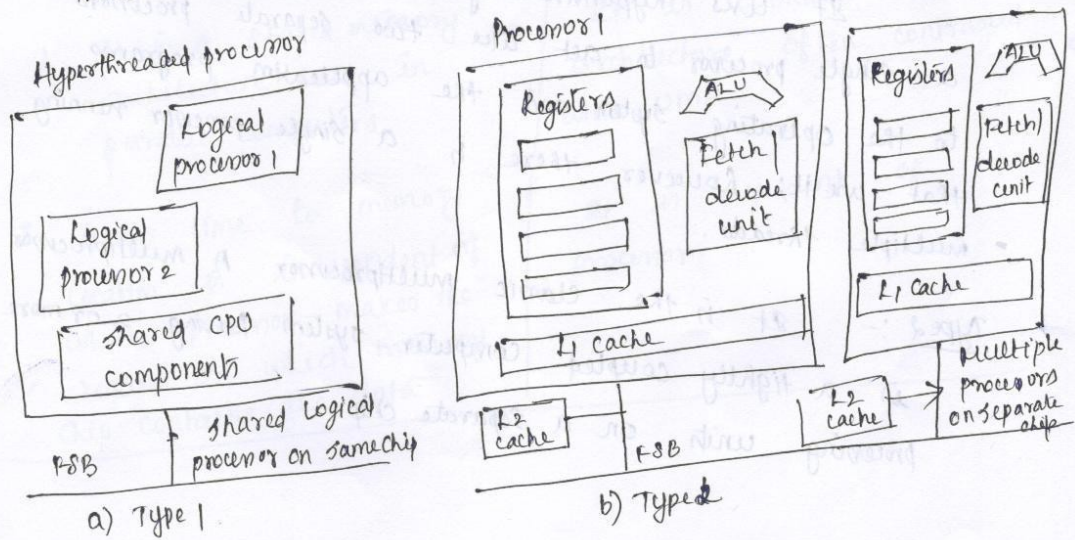
Multicore Processors

Multicore architectures are classified according to

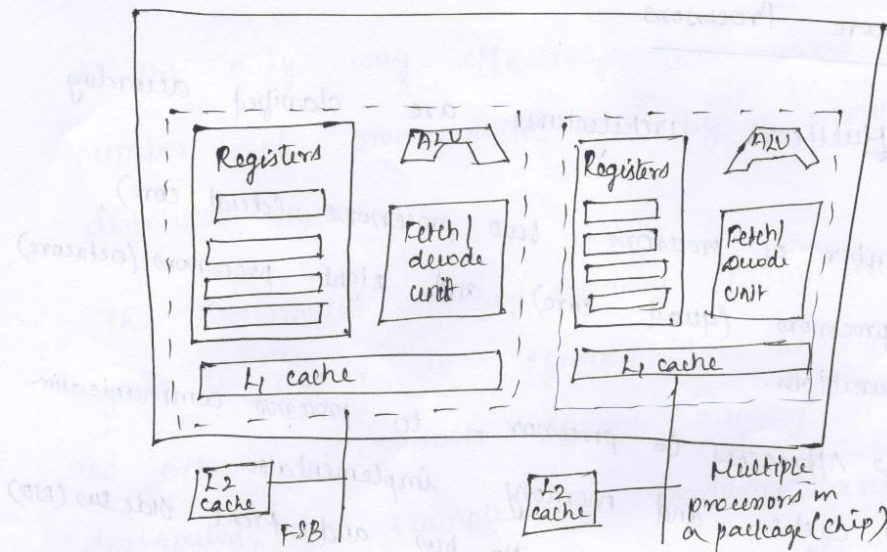
1. Number of processors: two processors (dual core), four processors (quad core), and eight processors (octacore) configurations.

- Approaches to processor - to processor communication
- Cache and memory implementation
- Implementation of system bus and front side bus (FSB)

Fig shown below shows three common configurations that support multiprocessing



Multicore (CMP)



c) Type 3

Fig: Common configurations that support multiprocessing

Type 1

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.

Type 2 :-

It is the classic multiprocessor. A multiprocessor is a tightly coupled computer system having 2 or more processing units on a separate chip.

Types

It represents multicore system. It provides two or more complete processors on single chip.

→ A shared memory multiprocessor (SMP) is one that offers the programmer a single physical address across all processors. They are also known as shared address multiprocessor.

Uniform Memory Access (UMA) multiprocessors	Non Uniform Memory access (NUMA) multiprocessors
<ol style="list-style-type: none"> 1. A multiprocessor in which latency to any word in main memory is about the same 2. It is a shared memory architecture used in parallel computers 3. Access time to memory location is independent of which processor makes the request or which memory chip contains the data 	<p>A type of single address space multiprocessor in which some memory accesses are much faster than others</p> <p>It is unshared memory architecture, often contrasted with UMA</p> <p>It is dependent of processor</p>

- In UMA, each processor uses private cache.
- The UMA is suitable for general purpose and time sharing applications by multiple users.
- It can be used to speed up the execution of single large program in time critical application.

Hardware Multithreading

→ The performance of a processor can be measured by the rate of instruction execution.

It is given by $\text{MIPS rate} = f_{\text{clk}} \times I_{\text{pc}}$.

→ Thus by increasing I_{pc} , performance can be increased

→ I_{pc} can be increased by instruction pipeline

→ When pipelining is used, it is essential to maximize the utilization of each pipeline stage to improve throughput. To do this, instructions must be executed in different order. However this approach needs complex mechanisms in the design. Therefore an another approach called Multithreading is used.

In Multithreading, the instruction stream is divided into several smaller streams, called threads, such that the threads can be executed in parallel. Thus high degree of instruction level parallelism can be achieved without increasing circuit complexity or power consumption.

Implicit and Explicit Multithreading

Some important terms used in multithreaded processors.

Process :-

A process is an instance of a program running on a computer. The process image is the collection of program data, stack and attributes that define the process. ~~Three~~ Important characteristics of process are

a) Resource Ownership

A process may get control of resources such as main memory, I/O channels, I/O devices and files from time to time.

b) Scheduling / execution :-

A process execution takes place through one or more programs.

c) Process switch :-

A process switch is an operation that switches the process or control from one process to another.

Thread

A thread includes the program counter, stack pointer and its own area for a stack. It executes sequentially and can be interrupted to transfer control to another thread.

Thread Switch

A thread switch is an operation that switches the processor control from one thread to another within the same process.

Explicit Threads

User level threads which are visible to application program and kernel level threads which are visible only to operating system, both are referred to as explicit threads.

Implicit and explicit Multithreading

Implicit :-

Refers to the concurrent execution of multiple threads extracted from a single sequential program.

Explicit :-

Refers to the concurrent execution of instructions from different explicit threads, either by interleaving instructions from different threads or by parallel execution in parallel pipelines.

Approaches to Explicit Multithreading

Interleaved or fine grained Multithreading

The processor executes two or more threads at a time. It switches from one thread to another at each clock cycle.

Blocked or coarse grained Multithreading

The processor executes instructions of a thread sequentially and if an event (eg. cache miss) that causes any delay occurs, it switches to another thread.

Simultaneous Multithreading (SMT)

The wide super scalar instruction is executed by executing multiple threads simultaneously using multiple execution units of a superscalar processor.

Chip Multiprocessing:-

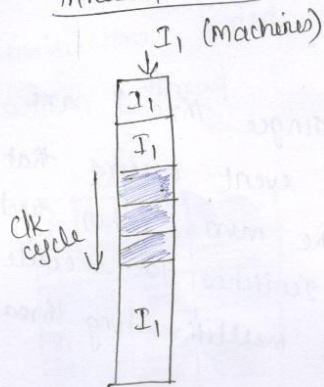
The processor is replicated on a single chip and each processor executes separate threads. This is referred as multicore.

Scalar approaches for threads

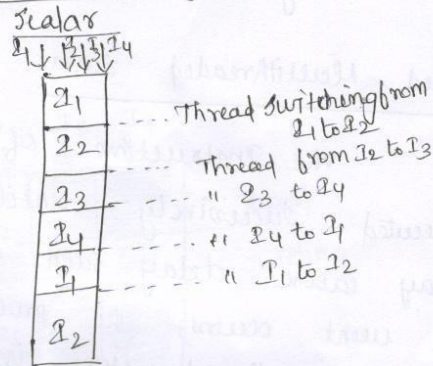
Single threaded scalar

Fig shown below gives single threaded scalar.

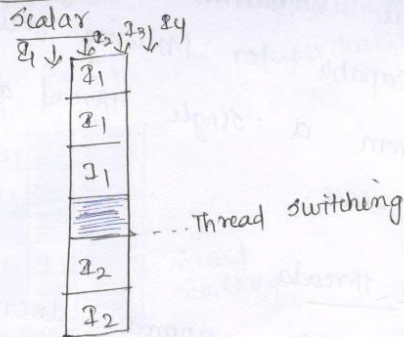
Fig a single threaded scalar



Figb: Interleaved multithreading



Figc: Block Multithreading



I_1, I_2, I_3, \dots so m indicates instruction on different threads. Each row indicates single clock cycle. Shaded block shows context execution slot.

Interlaved Multithreaded scalar

The pipeline stages are kept fully occupied (or close to it) by switching from one thread to another at each clock cycle. The hardware should support this switching operation.

Blocked Multithreaded scalar

The instructions of a single thread are executed successively until an event occurs that may cause delay such as cache miss. When such an event occurs, the processor switches to execute another thread. Here in Block multithreading thread switching needs one clock cycle.

Fig (a), (b), (c) show variations among processors that have hardware capable for issuing four instructions per cycle. The instructions from a single thread are executed in single cycle in each case.

Superscalar approaches for threads

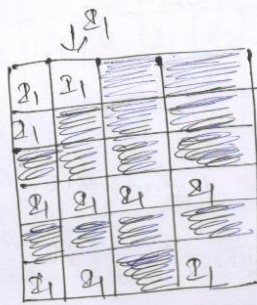
The basic superscalar approach does not support multithreading. It provides parallelism within a processor.

Not all of the available issue slots are used: referred as horizontal loss

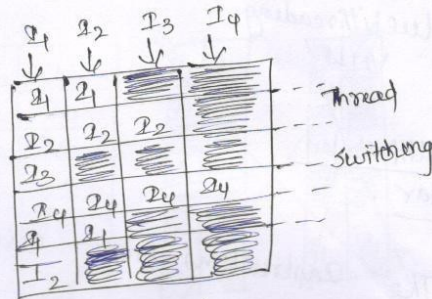
no issue slots are used, called vertical loss.

Interleaved Multithreading Super Scalar

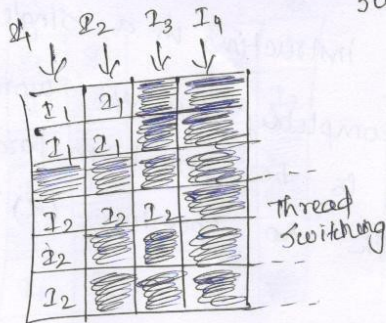
In this approach, as many instructions as possible are issued from a single thread and delay due to thread switching are eliminated. The no of instructions issued in a cycle is limited by dependencies within thread.



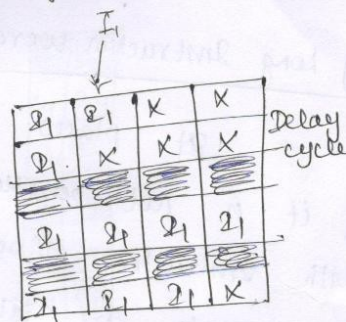
a) Super scalar



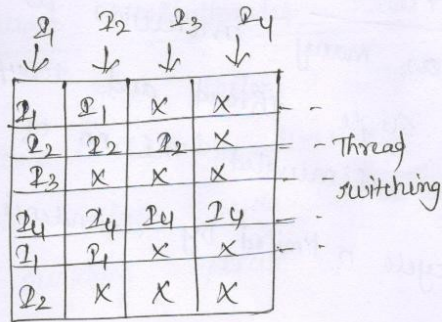
(b) Interleaved Multithreading Super scalar



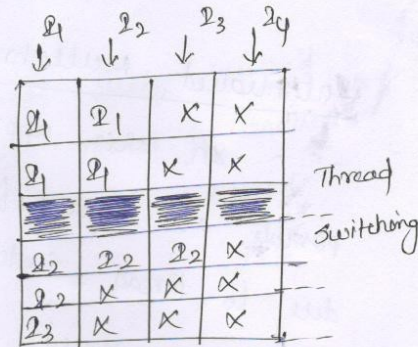
c) Blocked Multithreading Super scalar



d) VLIW



(a) Interleaved Multithreading VLIW



(b) Blocked Multithreading VLIW

Blocked Multithreading Superscalar :-

The instructions from only one thread are issued during a clock cycle and block multithreading is used

Very long Instruction word (VLIW)

It places multiple instructions in a single word. If it is not possible to completely fill the word with instructions those are to be issued in parallel, those slots are filled with no operation (X)

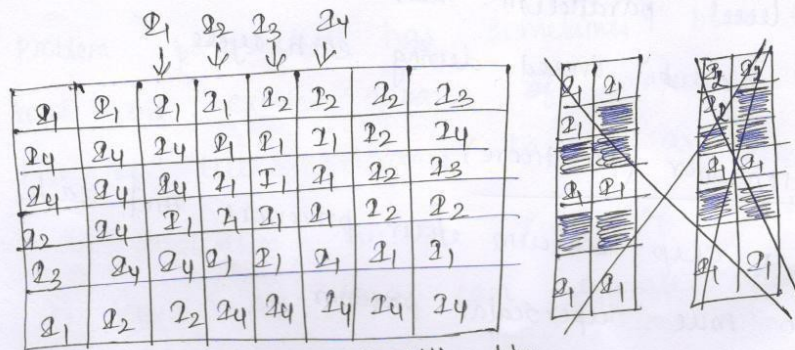
Interleaved Multithreading VLIW

provides similar efficiencies as that of interleaved multithreading on superscalar architecture

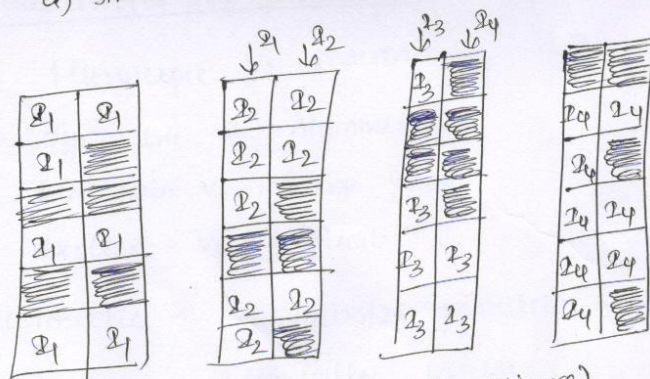
Blocked Multithreading VLIW

provides similar efficiencies as that of blocked multithreading on a superscalar architecture.

Other approaches to execute multiple threads



a) Simultaneous Multithreading



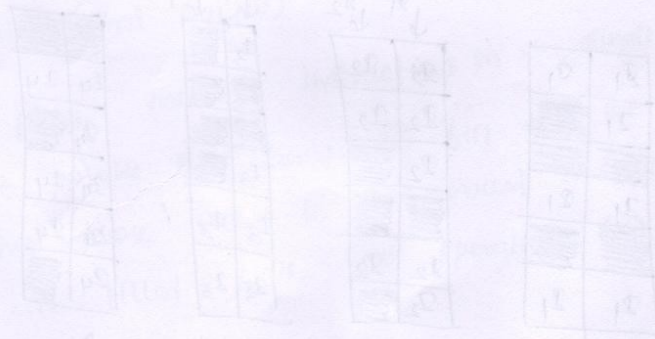
b) Chip Multiprocessor (Multicore)



SIMULTANEOUS MULTITHREADING

A high degree of efficiency can be achieved using system capable of issuing no of instructions at a time. All horizontal slots can be filled by a single instruction if the thread has a high degree of instruction level parallelism. Then the maximum no of instructions can be issued during each cycle.

Chip Multiprocessor (Multicore)

a chip contains four processors and each has a two mule superscalar processor.



	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Introduction to Unit V – MEMORY AND I/O SYSTEMS		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No. 1 / 13	

1.CONTENT LIST:

Introduction to Unit V - Memory and I/O Systems

2. SKILLS ADDRESSED:

Listening

3. OBJECTIVE OF THIS LESSON PLAN:

To facilitate students understand the basics of memory, input output systems

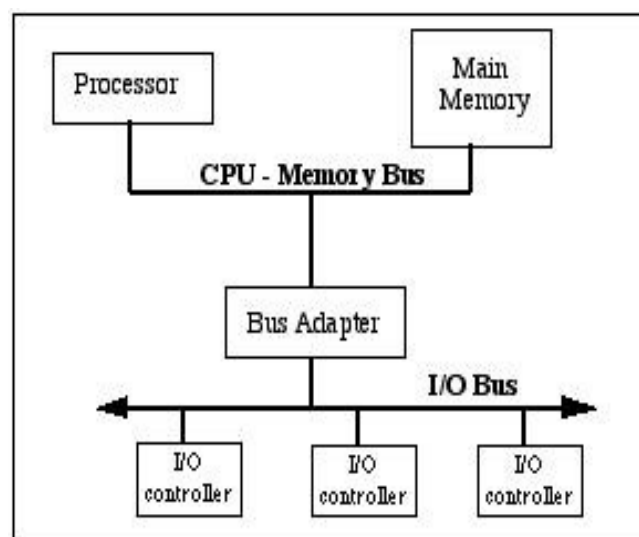
4.OUTCOMES:

- Explain the concept of memory units and its features
- Demonstrate the major topics covered in unit5

5.LINK SHEET:

- Define memory
- What are the topics covered in memory and input / output devices?

6.EVOCATION: (5 Minutes)



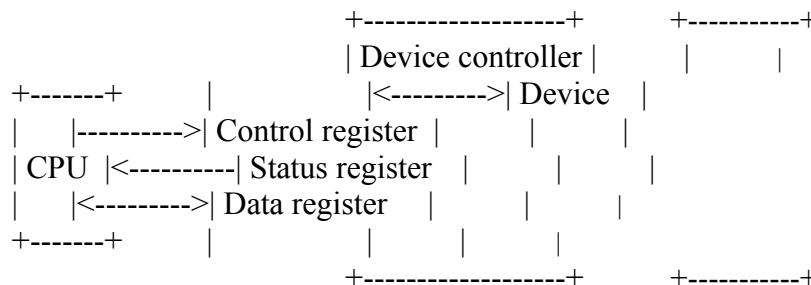
7. Lecture Notes

From the CPU's perspective, an I/O device appears as a set of special-purpose registers, of three general types:

- Status registers provide status information to the CPU about the I/O device. These registers are often read-only, i.e. the CPU can only read their bits, and cannot change them.
- Configuration/control registers are used by the CPU to configure and control the device. Bits in these configuration registers may be write-only, so the CPU can alter them, but not read them back. Most bits in control registers can be both read and written.
- Data registers are used to read data from or send data to the I/O device.

In some instances, a given register may fit more than one of the above categories, e.g. some bits are used for configuration while other bits in the same register provide status information.

The logic circuit that contains these registers is called the *device controller*, and the software that communicates with the controller is called a *device driver*.

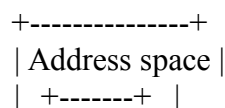


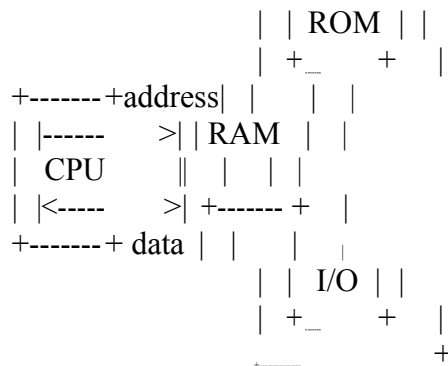
Simple devices such as keyboards and mice may be represented by only a few registers, while more complex ones such as disk drives and graphics adapters may have dozens.

Each of the I/O registers, like memory, must have an address so that the CPU can read or write specific registers.

Some CPUs have a separate address space for I/O devices. This requires separate instructions to perform I/O operations.

Other architectures, like the MIPS, use *memory-mapped I/O*. When using memory-mapped I/O, the same address space is shared by memory and I/O devices. Some addresses represent memory cells, while others represent registers in I/O devices. No separate I/O instructions are needed in a CPU that uses memory-mapped I/O. Instead, we can perform I/O operations using any instruction that can reference memory.





On the MIPS, we would access ROM, RAM, and I/O devices using load and store instructions. Which type of device we access depends only on the address used!

```
lw    $t0, 0x00000004    # Read ROM
sw    $t0, 0x00000004    # Write ROM (bus error!)

lbu   $t0, 0x0000ffc1     # Read RAM
sb    $t0, 0x0000ffc1     # Write RAM

lbu   $t0, 0xffff0000     # Read an I/O device
sb    $t0, 0xffff0004     # Write to an I/O device
```

The 32-bit MIPS architecture has a 32-bit address, and hence an address space of 4 gigabytes. Addresses 0x00000000 through 0xffffefff are used for memory, and addresses 0xffff0000 - 0xffffffff (the last 64 kilobytes) are reserved for I/O device registers. This is a very small fraction of the total address space, and yet far more space than is needed for I/O devices on any one computer.

Each register within an I/O controller must be assigned a unique address within the address space. This address may be fixed for certain devices, and auto-assigned for others. (PC plug-and-play devices have auto-assigned I/O addresses, which are determined during boot-up.)



8. TEXT BOOKS:

1. V.Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc,. 2012
2. David A. Patterson and John L. Hennessey, —Computer organization and design— Morgan auffman Isevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

Unit 5– Lesson No.2 / 13

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Memory hierarchy		
Time:	45 Minutes	
Lesson. No		

1.CONTENT LIST:

Memory hierarchy

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students understand the hierarchy of memories

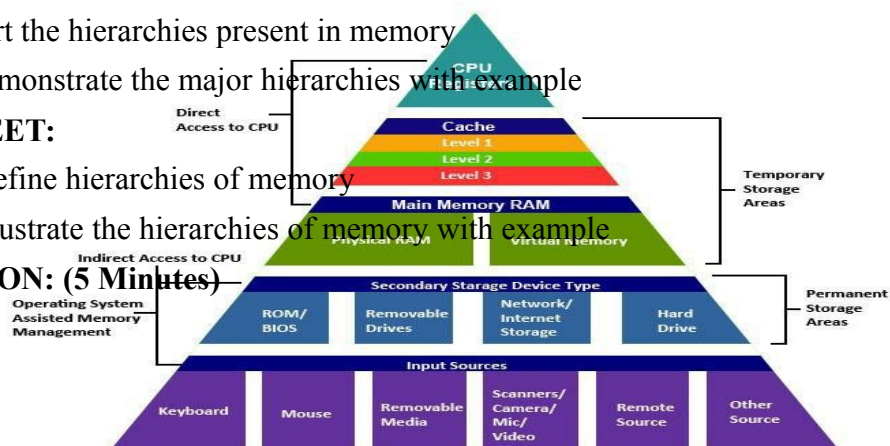
4.OUTCOMES:

- Sort the hierarchies present in memory
- Demonstrate the major hierarchies with example

5.LINK SHEET:

- Define hierarchies of memory
- Illustrate the hierarchies of memory with example

6.EVOCATION: (5 Minutes)



②

Primary memory - costly & have limited size. mainly used for storing the currently processing data

ROM

RAM.

Secondary Memory

used to store data & instructions (programs) when they are not being processed

Eg: hard disks, floppies, CD-ROMs, magnetic tapes etc.

Memory Hierarchy:-

→ Ideally, computer memory should be fast, large and inexpensive. Unfortunately it is impossible to meet all the three of these requirements using one type of memory.

→ Increased speed and size are achieved at increased cost.

→ Very fast memory system can be achieved if SRAM chips are used. These chips are expensive and for the cost reason it is impracticable to build a large main memory using SRAM chips. The only alternative is to use DRAM chips for large main memories.

- Based on the ~~at~~
- DRAM should insert wait states in memory read/write cycles. This reduces the speed of execution.
 - In the memory system, small section of DRAM is added along with main memory, referred to as cache memory.
 - The program which is to be executed is loaded in the main memory, but the part of program (code) and data that work at a particular time is usually accessed from the cache memory.
 - The size of memory is still small compared to the demands of large programs with voluminous data. Very large disks are available at a reasonable price, sacrificing the speed.
 - Thus it is realized that to make efficient computer system ~~it~~ is not possible to rely on single memory component, but to employ a memory hierarchy.
 - Using Memory hierarchy, all of different types of memory units are employed to give efficient computer system.

The typical Memory hierarchy is given in Fig below.

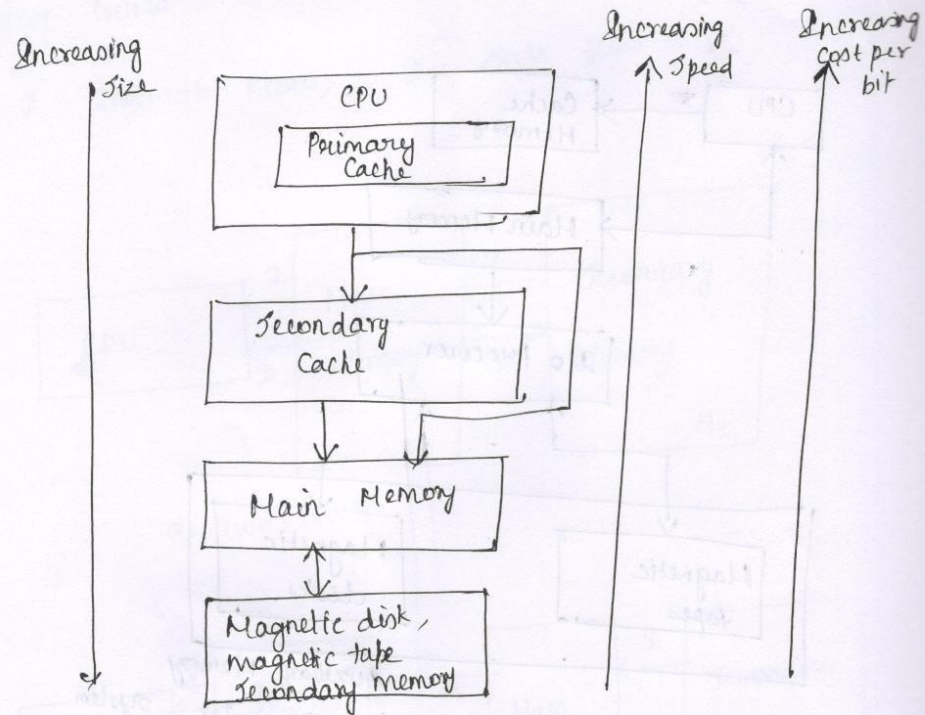


Fig : Typical memory hierarchy.

∴ Huge amount of cost effective storage can be provided by magnetic disks. Memory with DRAM technology can be built along with cache memory to achieve better performance.

Memory hierarchy can be employed in a computer system is shown below

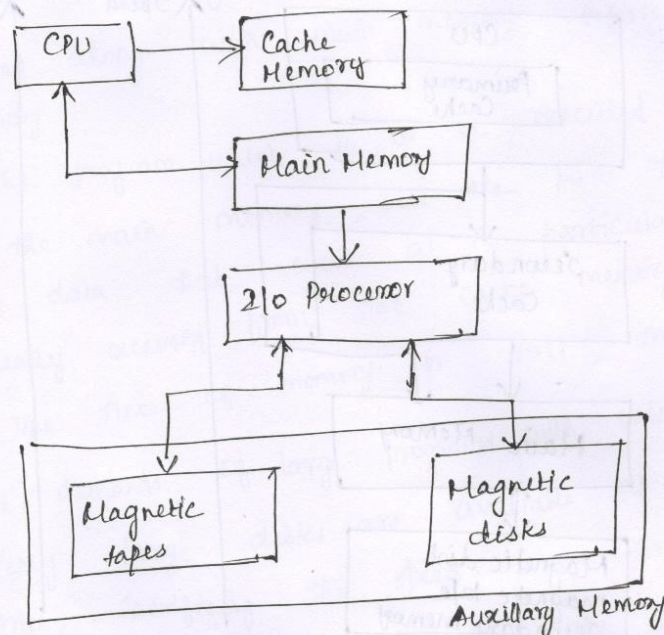


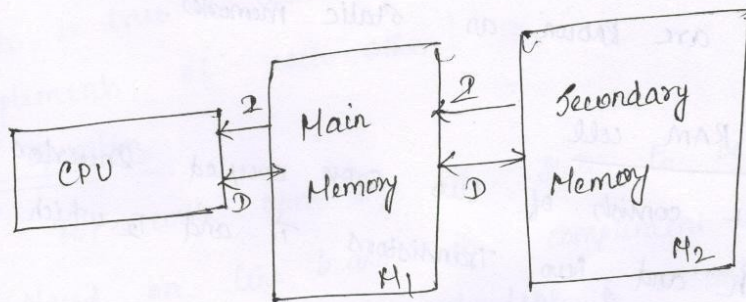
Fig: Memory hierarchy in computer system

Magnetic tapes and Magnetic disks are used as secondary memory. This memory is also known as auxiliary memory.

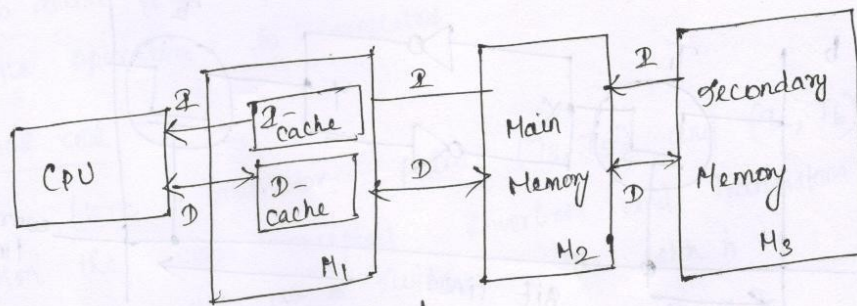
(4)

Common Memory hierarchy with two, three and four levels

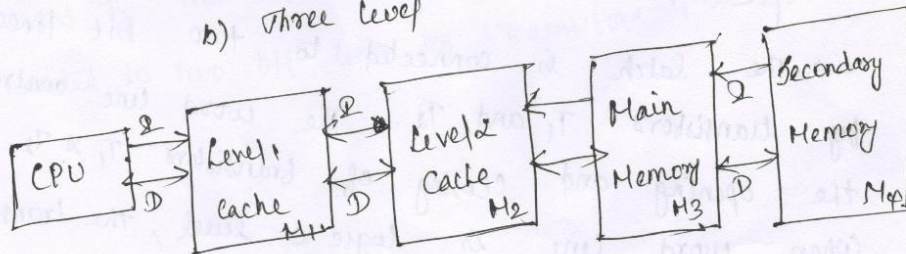
I - Instruction flow, D - Data flow



a) Two level



b) Three level



c) Four level



7. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design— Morgan auffman Isevier, Fifth edition, 2014

8. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Memory technologies		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.3,4 / 13	

1.CONTENT LIST:

Memory technologies

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students remember the technologies present in memory device

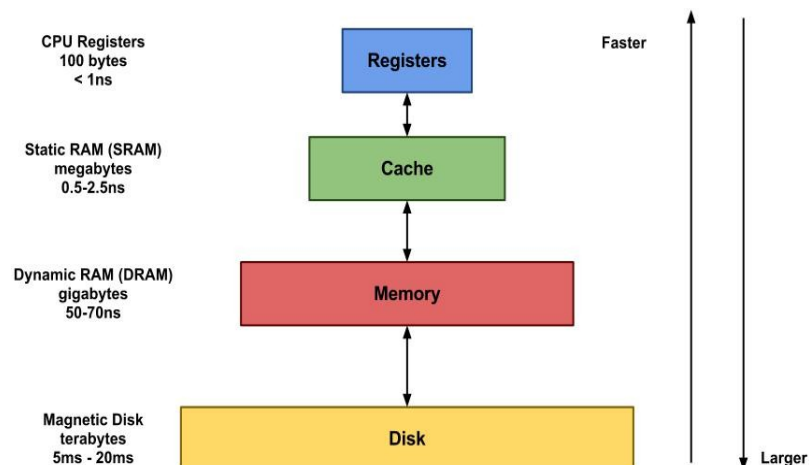
4.OUTCOMES:

- Sort the technologies present in memory
- Demonstrate the major technologies with example

5.LINK SHEET:

- Define technologies of memory
- Illustrate the technologies of memory with example

6.EVOCATION: (5 Minutes)



7. Lecture Notes

MEMORY TECHNOLOGIES

Much of the success of computer technology stems from the tremendous progress in storage technology.

Early computers had a few kilobytes of random-access memory. The earliest IBM PCs didn't even have a hard disk.

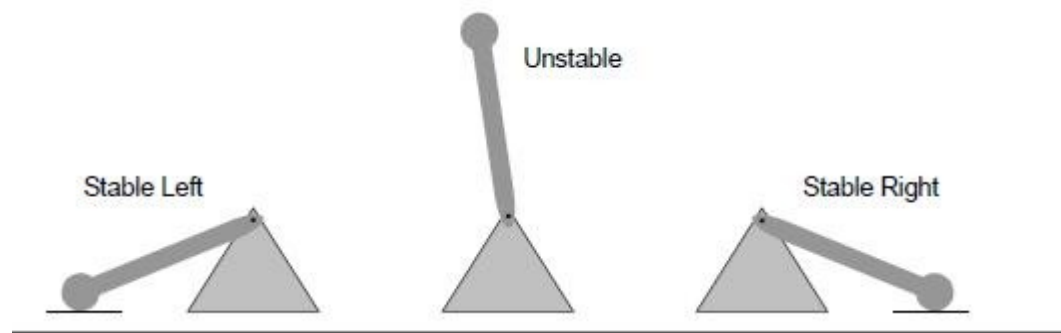
That changed with the introduction of the IBM PC-XT in 1982, with its 10-megabyte disk. By the year 2010, typical machines had 150,000 times as much disk storage, and the amount of storage was increasing by a factor of 2 every couple of years.

Random-Access Memory

Random-access memory (RAM) comes in two varieties—*static* and *dynamic*. *Static RAM* (SRAM) is faster and significantly more expensive than *Dynamic RAM* (DRAM). SRAM is used for cache memories, both on and off the CPU chip. DRAM is used for the main memory plus the frame buffer of a graphics system. Typically, a desktop system will have no more than a few megabytes of SRAM, but hundreds or thousands of megabytes of DRAM.

Static RAM

SRAM stores each bit in a *bistable* memory cell. Each cell is implemented with a six-transistor circuit. This circuit has the property that it can stay indefinitely in either of two different voltage configurations, or *states*. Any other state will be unstable—starting from there, the circuit will quickly move toward one of the stable



Dynamic RAM

DRAM stores each bit as charge on a capacitor. This capacitor is very small—typically around 30 femtofarads, that is, 30×10^{-15} farads. Recall, however, that a farad is a very large unit of measure. DRAM storage can be made very dense—each cell consists of a capacitor and a single access-transistor. Unlike SRAM, however, a DRAM memory cell is very sensitive to any disturbance. When the capacitor voltage is disturbed, it will never recover. Exposure to light rays will cause the capacitor voltages to change. In fact, the sensors in digital cameras and camcorders are essentially arrays of DRAM cells.

	Transistors per bit	Relative access time	Persistent?	Sensitive?	Relative cost	Applications
SRAM	6	1X	Yes	No	100X	Cache memory
DRAM	1	10X	No	Yes	1X	Main mem, frame buffers

Conventional DRAMs

The cells (bits) in a DRAM chip are partitioned into d *supercells*, each consisting of w DRAM cells. A $d \times w$ DRAM stores a total of dw bits of information. The supercells are organized as a rectangular array with r rows and c columns, where $rc = d$. Each supercell has an address of the form (i, j) , where i denotes the row, and j denotes the column.

For example, Figure 6.3 shows the organization of a 16×8 DRAM chip with $d = 16$ supercells, $w = 8$ 534 bits per supercell, $r = 4$ rows, and $c = 4$ columns. The shaded box denotes the supercell at address $(2, 1)$.

Information flows in and out of the chip via external connectors called *pins*. Each pin carries a 1-bit signal.

Figure shows two of these sets of pins: eight data pins that can transfer 1 byte in or out of the chip, and two addr pins that carry two-bit row and column supercell addresses. Other pins that carry control information are not shown.

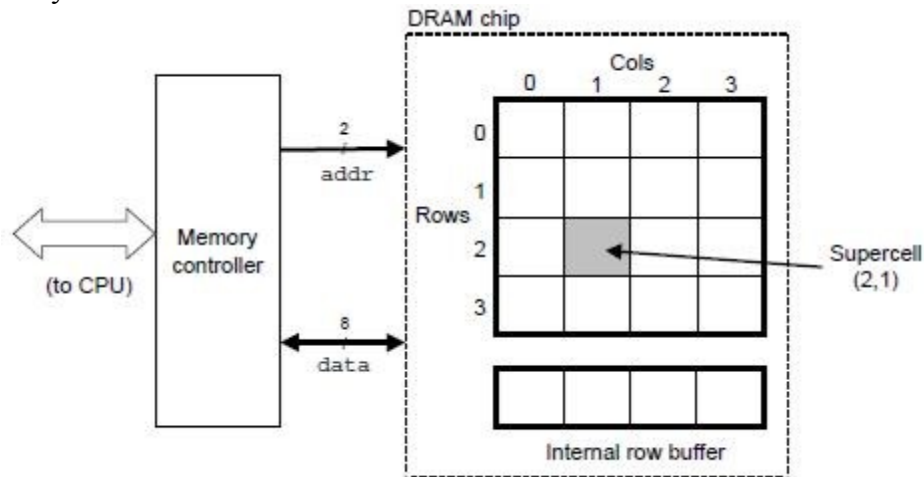
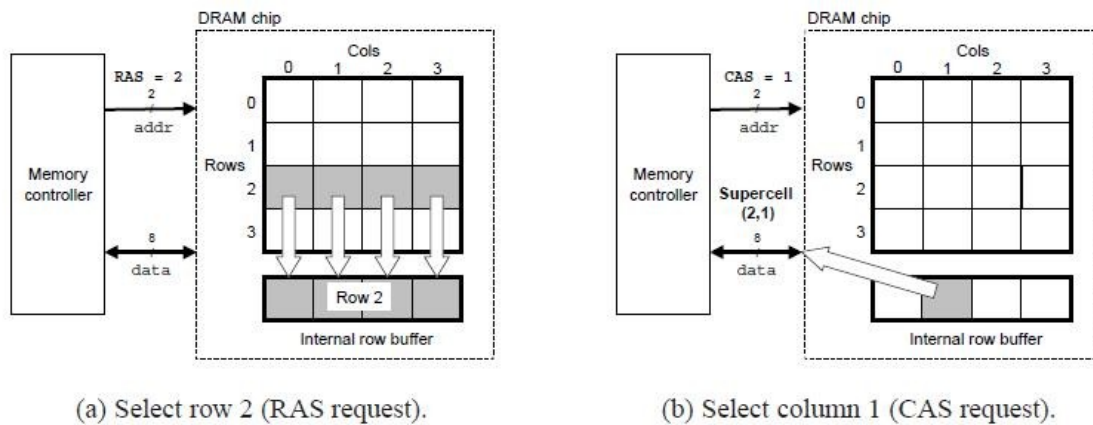


Fig: Conventional DRAM

One reason circuit designers organize DRAMs as two-dimensional arrays instead of linear arrays is to reduce the number of address pins on the chip. For example, if our example 128-bit DRAM were organized as a linear array of 16 supercells with addresses 0 to 15, then the chip would need four address pins instead of two. The disadvantage of the two-dimensional array organization is that addresses must be sent in two distinct steps, which increases the access time.



Enhanced DRAMs

There are many kinds of DRAM memories, and new kinds appear on the market with regularity as manufacturers attempt to keep up with rapidly increasing processor speeds. Each is based on the conventional DRAM cell, with optimizations that improve the speed with which the basic DRAM cells can be accessed.

Accessing Main Memory

Data flows back and forth between the processor and the DRAM main memory over shared electrical conduits called *buses*. Each transfer of data between the CPU and memory is accomplished with a series of steps called a *bus transaction*. A *read transaction* transfers data from the main memory to the CPU. A *write transaction* transfers data from the CPU to the main memory.

A *bus* is a collection of parallel wires that carry address, data, and control signals. Depending on the particular bus design, data and address signals can share the same set of wires, or they can use different sets. Also, more than two devices can share the same bus. The control wires carry signals that synchronize the transaction and identify what kind of transaction is currently being performed.

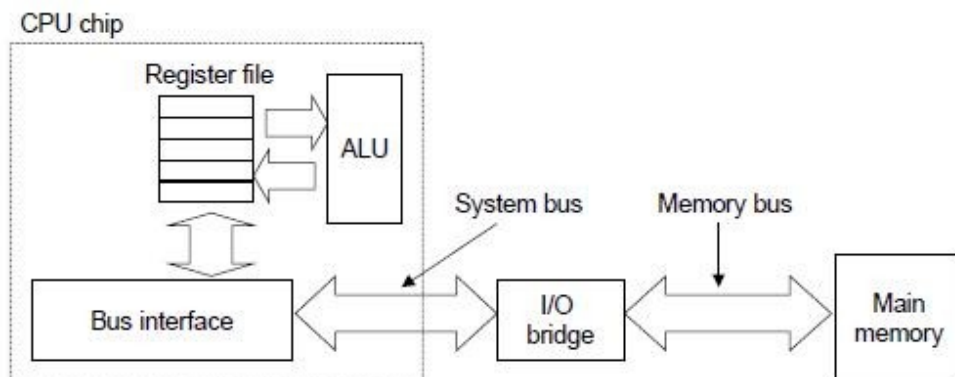


Figure : Example bus structure that connects the CPU and main memory.

Disk Storage

Disks are workhorse storage devices that hold enormous amounts of data, on the order of hundreds to thousands of gigabytes, as opposed to the hundreds or thousands of megabytes in a RAM-based memory. However, it takes on the order of milliseconds to read information from a disk, a hundred thousand times longer than from DRAM and a million times longer than from SRAM.

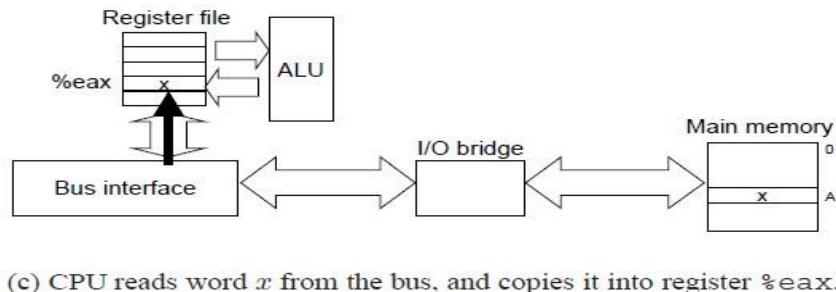
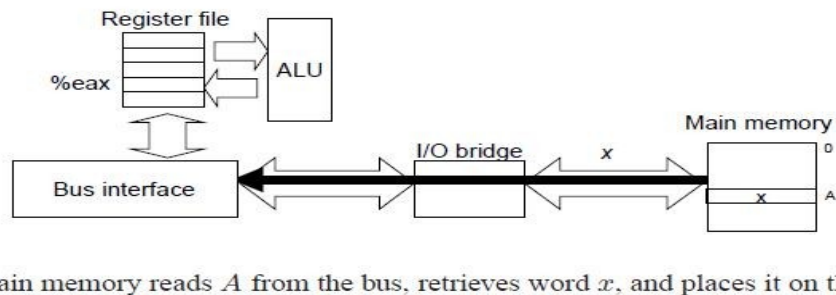
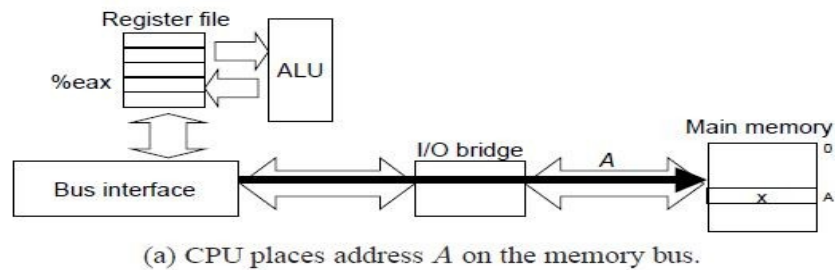




Figure: **Memory read transaction for a load operation**

8. TEXT BOOKS:

1. V.Carl Hamacher, Zvonko G. Varanescic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012
2. David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan auffman Isevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Cache basics – Measuring and improving cache performance		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.5 / 13	

1.CONTENT LIST:

Cache basics – Measuring and improving cache performance

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of cache and its performance measurement

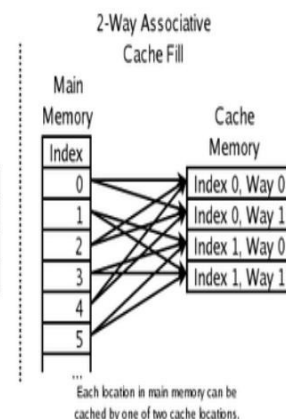
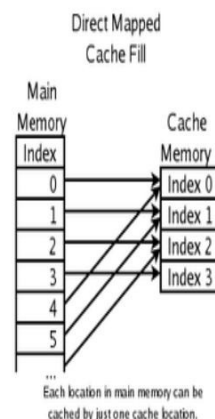
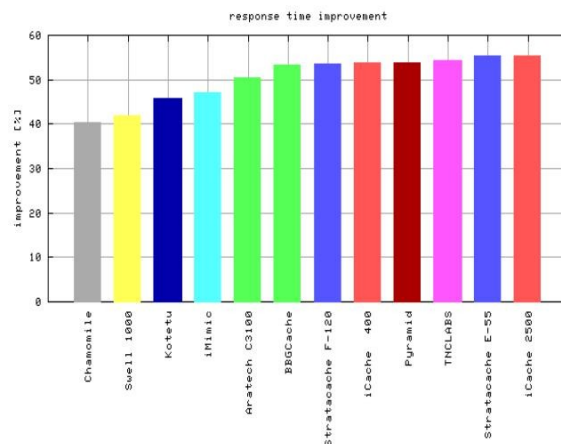
4.OUTCOMES:

- Describe cache memory
- Analyze the performance of cache and measure it

5.LINK SHEET:

- Define cache memory
- Explain the criteria to improve the performance of cache memory

6.EVOCATION: (5 Minutes)



7. Lecture Notes

Cache basics – Measuring and improving cache performance

One focuses on reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location. The second technique reduces the miss penalty by adding an additional level to the hierarchy. This technique, called *multilevel caching*, first appeared in high-end computers selling for more than \$100,000 in 1990; since then it has become common on desktop computers selling for less than \$500! CPU time can be divided into the clock cycles that the CPU spends executing the program and the clock cycles that the CPU spends waiting for the memory system. Normally, we assume that the costs of cache accesses that are hits are part of the normal CPU execution cycles. Thus,

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles})$$

The memory-stall clock cycles come primarily from cache misses, and we make that assumption here. We also restrict the discussion to a simplified model of the memory system. In real processors, the stalls generated by reads and writes can be quite complex, and accurate performance prediction usually requires very detailed simulations of the processor and memory system.

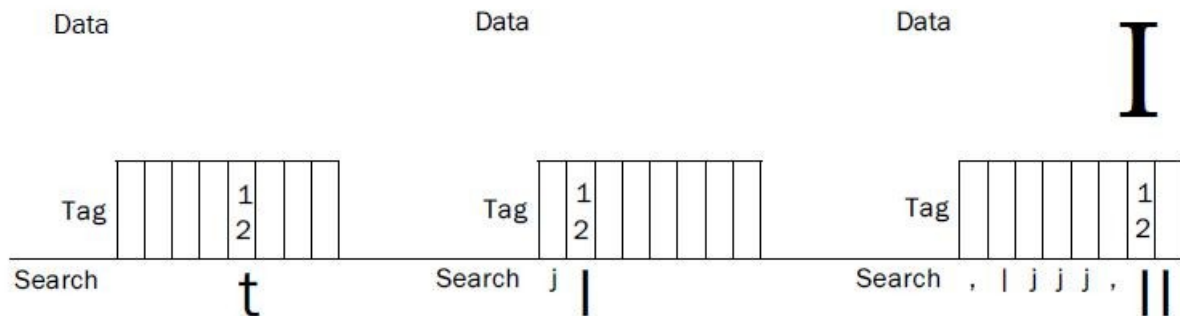
Reads Read-stall cycles = Program x Read miss rate x Read miss penalty
Writes are more complicated. For a write-through scheme, we have two sources of stalls: write misses, which usually require that we fetch the block before continuing the write (see the *Elaboration* on page 467 for more details on dealing with writes), and write buffer stalls, which occur when the write buffer is full when a write occurs.

Calculating Cache Performance:

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%.

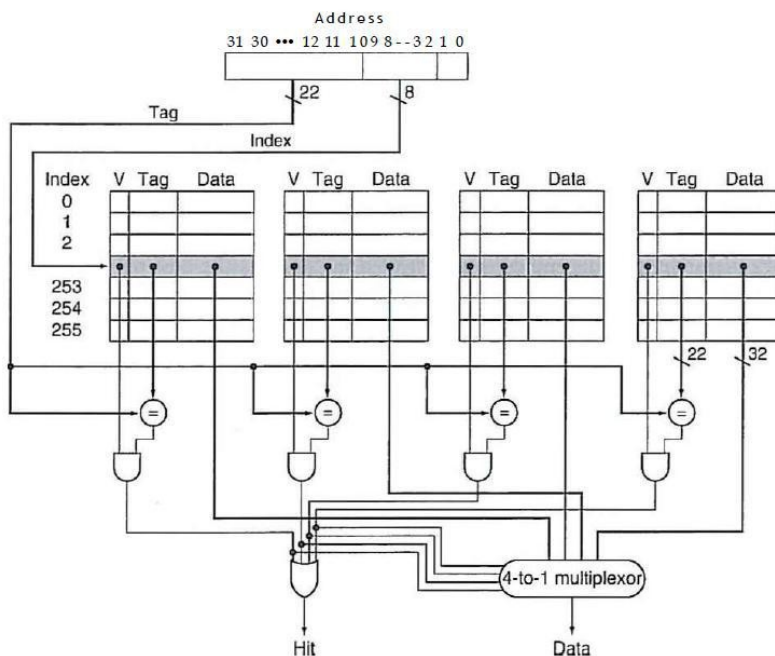
Reducing Cache Misses by *Move* Flexible Placement of Blocks

So far, when we place a block in the cache, we have used a simple placement scheme: A block can go in exactly one place in the cache. As mentioned earlier, it is called *direct mapped* because there is a direct mapping from any block address in memory to a single location in the upper level of the hierarchy. However, there is actually a whole range of schemes for placing blocks. Direct mapped, where a block can be placed in exactly one location, is at one extreme. At the other extreme is a scheme where a block can be placed in *any* location in the cache. Such a scheme is called **fully associative**, because a block in memory may be associated with any entry in the cache. To find a given block in a fully associative cache, all the entries in the cache must be searched because a block can be placed in any one. To make the search practical, it is done in parallel with a comparator associated with each cache entry. These comparators significantly increase the hardware cost, effectively making fully associative placement practical only for caches with small numbers of blocks.



Choosing Which Block to Replace :

When a miss occurs in a direct-mapped cache, the requested block can go in exactly one position, and the block occupying that position must be replaced. In an associative cache, we have a choice of where to place the requested block, and hence a choice of which block to replace. In a fully associative cache, all blocks are candidates for replacement. In a set-associative cache, we must choose among the blocks in the selected set. The most commonly used scheme is **least recently used (LRU)**, which we used in the previous example. In an LRU scheme, the block replaced is the one that has been unused for the longest time. The set associative example on page 482 uses LRU, which is why we replaced Memory(0) instead of Memory(6). LRU replacement is implemented by keeping track of when each element in a set was used relative to the other elements in the set. For a two-way set-associative cache, tracking when the two elements were used can be implemented by keeping a single bit in each set and setting the bit to indicate an element whenever that element is referenced. As associativity increases, implementing LRU gets harder; in Section 5.5, we will see an alternative scheme for replacement.





8. TEXT BOOKS:

1. V.Carl Hamacher, Zvonko G. Varanesic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012
2. David A. Patterson and John L. Hennessey, —Computer organization and design— Morgan auffman lsevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Cache basics – Measuring and improving cache performance		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.6 / 13	

1.CONTENT LIST:

Cache basics – Measuring and improving cache performance

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of cache and its performance measurement

4.OUTCOMES:

- i. Describe cache memory
- ii. Analyze the performance of cache and measure it

5.LINK SHEET:

- i. Define cache memory
- ii. Explain the criteria to improve the performance of cache memory

6.EVOCATION: (5 Minutes)



(15)

Two techniques can be used to improve cache performance are.

- Reducing the miss rate by reducing the probability that two different memory blocks will contend for same cache location
- Reducing the miss penalty by adding an additional level to hierarchy. This technique is called multilevel caching

$$\text{CPU time} = \left(\frac{\text{CPU execution clock}}{\text{cycles}} + \frac{\text{Memory stall}}{\text{clock cycles}} \right) \times \text{clock cycle time.}$$

- cache hit indicates normal CPU execution cycles
- cache misses indicates memory stall clock cycles.
- Memory stall clock cycles are sum of stall cycles coming from read plus those coming from writes.

$$\text{Memory stall clock cycle} = \left(\frac{\text{Read stall}}{\text{cycles}} + \frac{\text{Write stall}}{\text{cycles}} \right)$$

$$\text{Read stall cycles} = \frac{\text{Reads}}{\text{program}} \times \text{Read miss rate} \times \text{Read miss penalty.}$$

- For a write through scheme, @ two sources of stalls → write misses and write buffer stalls.

$$\text{Write Stall cycles} = \left(\frac{\text{Writes}}{\text{program}} \times \text{write miss rate} \times \text{write miss penalty} \right) + \text{write buffer stalls}$$

One can combine read and write penalties together if they are same as

$$\text{Memory Stall clock cycles} = \frac{\text{Memory accesses}}{\text{program}} \times \text{Miss rate} \times \text{Miss penalty}$$

Also represented as

$$\text{Memory Stall clock cycles} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Average Memory access time (AMAT)

It is the average time to access memory considering both hits and misses and the frequency of different accesses.

$$\text{AMAT} = \text{Time for a hit} + \text{miss rate} \times \text{miss penalty}$$

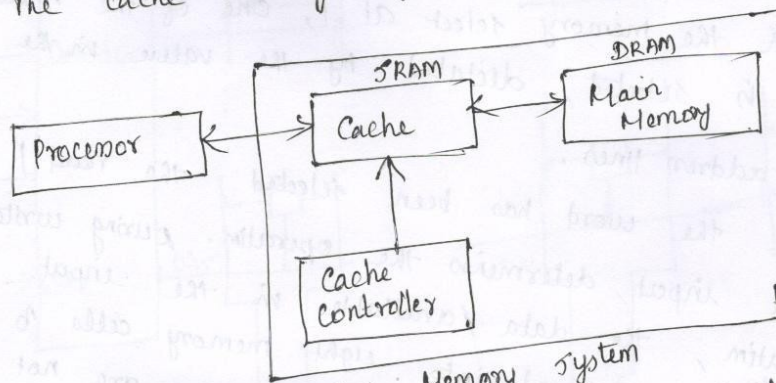
Reducing Cache misses by more flexible placement of blocks

In direct mapping, the position of memory block is given by

Cache Basics

The small section of SRAM memory added between processor and main memory to speed up execution process is known as cache memory.

The cache memory system is shown below



Cache Memory system

- The cache memory system includes a small amount of fast memory (SRAM) and a large amount of slow memory (DRAM).
- Cache Controller implements the cache logic. If processor finds that the addressed code or data is not available in the cache - the condition referred as cache miss, the desired memory block is copied from main memory to cache using cache controller.
- The cache block also known as cache slot or line

(11)

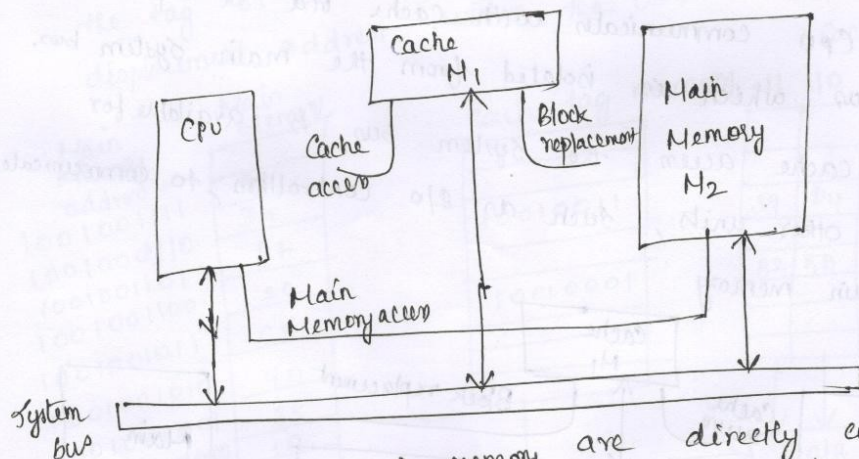
→ the percentage of accesses where the processor finds the code or data word it needs in the cache memory is called hit rate / hit ratio.

$$\text{Hit rate} = \frac{\text{Number of hits}}{\text{Total no of bus cycles}} \times 100\%$$

Most commonly Used Cache Organization

1. Look aside
2. Look Through.

Look aside system organization



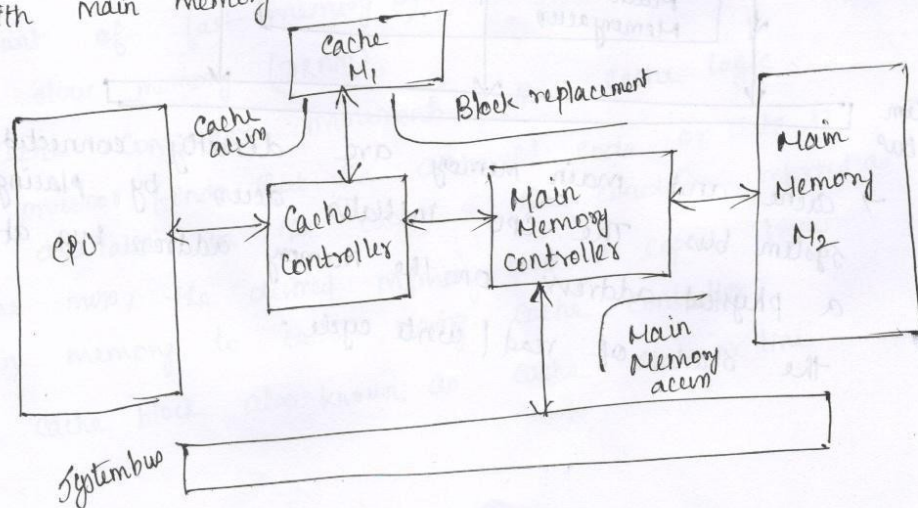
→ cache and main memory are directly connected to system bus. The CPU initiates access by placing a physical address on the memory address bus at the start of read / write cycle.

→ M_1 immediately compares physical address to tag address currently residing in tag memory. If match is found, ~~the access~~ in case of cache hit, the access is completed by read/write operation executed in the cache.

→ If ~~match~~ match is not found, in case of cache miss, the desired access is completed by a read/write operation directed to M_2 . The system bus is used to transfer ~~the~~ target address from M_2 to M_1 .

Look Through system organization

CPU communicates with cache via a separate (local) bus which is isolated from the main system bus. During cache access, the system bus is available for use by other units, such as I/O controllers, to communicate with main memory.



(12)

→ Look Through cache system does not automatically send all memory requests to main memory; it does so only after a cache miss

→ Look Through cache systems use wider local bus to link M_1 & M_2 , thus speeding up cache main memory transfers. It is faster.

Cache Read operation

→ Here each cache block is 4 bytes and each memory is 10 bit long. 8 higher order bits form the tag or block address & 2 lower order bits define displacement address within the block.

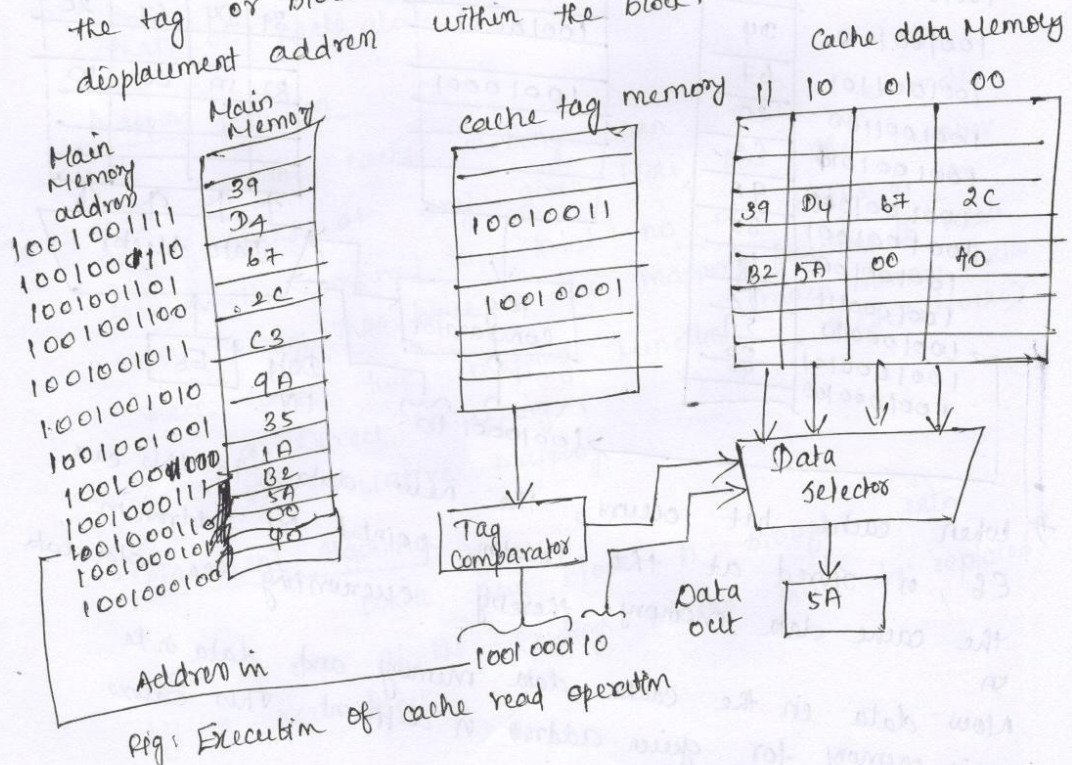
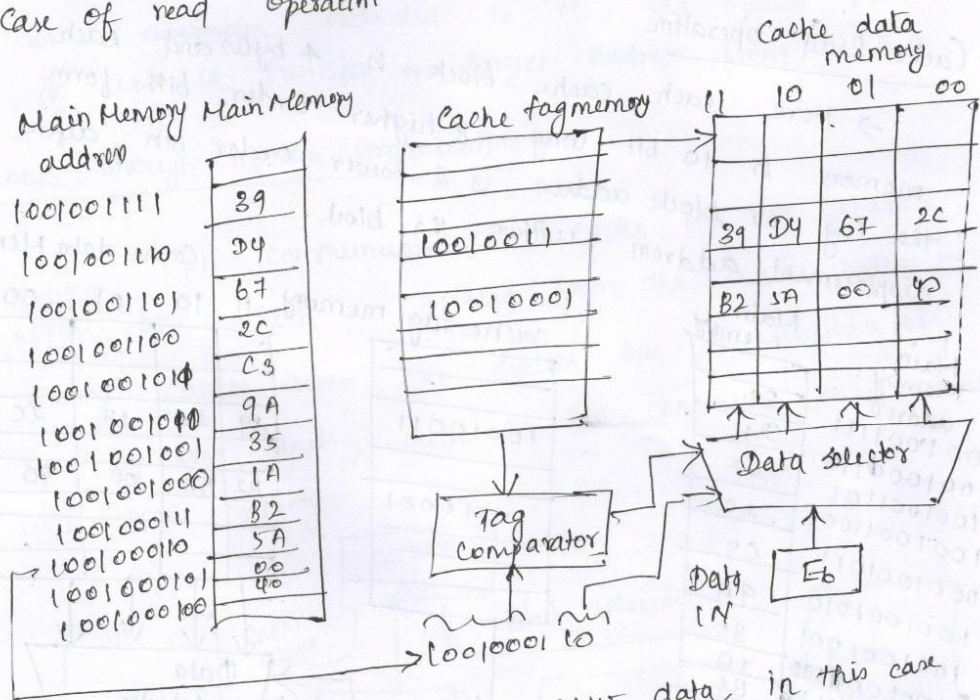


Fig: Execution of cache read operation

→ The stored tag pinpoints the corresponding block in cache data memory and 2-bit displacement is used to read the target word.

Cache write operation

It uses same addressing technique as in case of read operation



→ When cache hit occurs, the new data, in this case **E6**, is stored at the location pointed by address in the cache data memory, thereby overwriting the old data. Now data in the cache data memory and data in the main memory for given address is different. This causes Cache consistency problem.

Elements of cache design

The cache design elements include cache size, mapping function, replacement algorithm write policy, block size and number of caches.

Cache size:-

The size of the cache should be small enough so that the overall average cost per bit is close to that of main memory and large enough so that the overall average access time is close to that of cache alone.

Mapping function

The cache memory can store reasonable no of blocks at any given time, but this number is small compared to total no of blocks in the main memory. Thus have to use mapping functions to relate

There are two mapping functions (main memory blocks & cache blocks)

- Direct mapping
- Associative mapping

Replacement Algorithm

When new block is brought into cache, one of the existing blocks must be replaced by a new block.

There are four most common replacement algorithms.

1. Least Recently Used (LRU)
2. First IN First OUT (FIFO)
3. Least Frequently Used (LFU)
4. Random

Cache Coherency

In a single CPU system, two copies of same data, one in cache memory and another in main memory may become different. This data inconsistency is called as cache coherence problem.

To protect cache coherence, there are four different approaches

1. Bus watching (snooping) - main memory resides in cache memory
2. Hardware transparency - access to all devices to main memory are routed through same cache
3. Non-cacheable memory
4. Cache flushing

By designing shared memory as noncacheable memory cache coherency can be maintained, since shared memory never copied to cache

→ To avoid data inconsistency, a cache flush writes any altered data to main memory and caches in the system are flushed before a device writes to shared memory

Cache Updating policies

Two different sets of data become associated with same address. To prevent this, the cache system has updating systems such as write through system, buffered write through system and write back system.

Write through system

The cache controller copies data to main memory immediately after it is written to cache. Due to this, main memory always contains a valid data and thus any block in the cache can be overwritten immediately without data loss.

Buffered write through system

In buffered write through system, the processor can start a new cycle before the write cycle to the main memory is completed. This means that the write accesses to main memory are buffered.

Write back system

All altered blocks must be written to main memory before another device can access these blocks in main memory.

Measuring and Improving Cache Performance

The performance of a memory system depends mainly on the following

Address reference statistics

It means the order and frequency of the logic address generated by programs that use the memory hierarchy.

Access time :- The access time (t_A) of each memory level relative to the CPU

Storage Capacity

Storage capacity of each memory level.

Block size

The size of blocks (pages) transferred between adjacent levels.

Allocation Algorithm

The algorithm used to determine the regions of memory to which blocks are transferred by the block replacement process.

(16)

$(\text{Block number}) \bmod (\text{Number of blocks in the cache})$

~~then~~

→ In set associative mapping, there is a set that contains the memory block and it is given as $(\text{Block number}) \bmod (\text{Number of sets in the cache})$

Thus to access a particular data all the tags of all elements within the particular set must be searched.

→ In a fully associative mapping, as the block goes anywhere, all the tags of all the blocks must be searched.

Thus increase in associativity decreases the miss rate and hence increases the performance of cache.



7. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan Kaufmann, Fifth edition, 2014

8. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Virtual memory		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.7/ 13	

1.CONTENT LIST:

Virtual memory

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of virtual memory

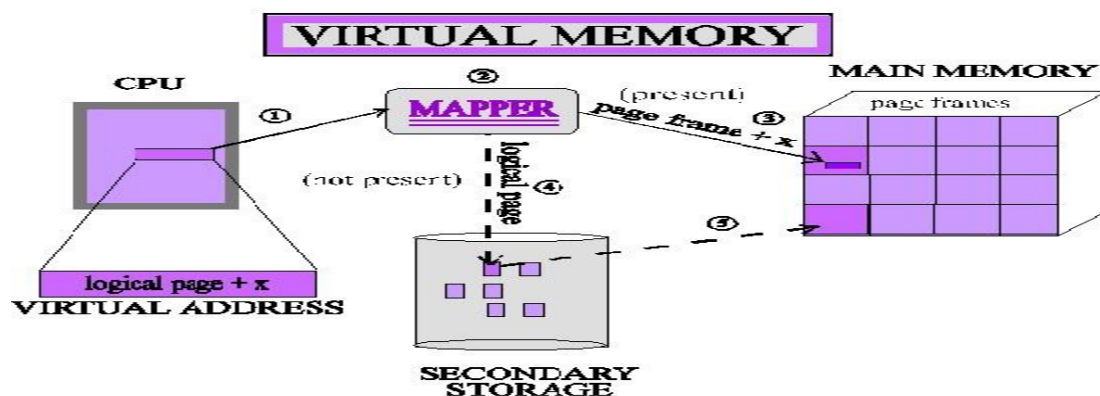
4. OUTCOMES:

- Describe virtual memory
- Analyze the features of virtual memory

5.LINK SHEET:

- Define virtual memory
- Explain the criteria to improve the performance of virtual memory

6.EVOCATION: (5 Minutes)



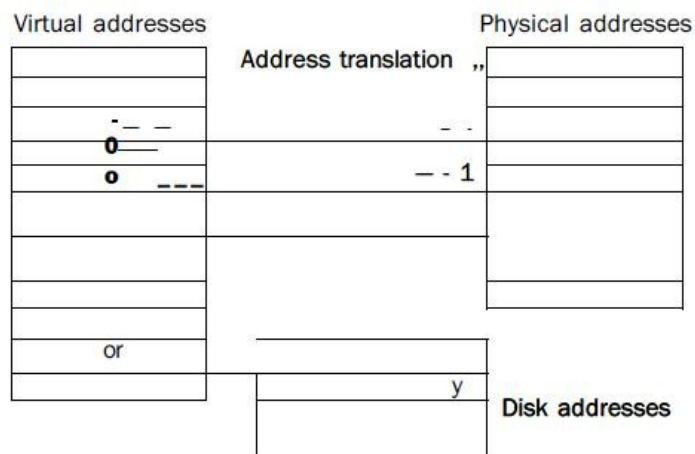
7. Lecture Notes

Virtual Memory

Similarly, the main memory can act as a "cache" for the secondary storage, usually implemented with magnetic disks. This technique is called virtual memory. Historically, there were two major motivations for virtual memory: to allow efficient and safe sharing of memory among multiple programs, and to remove the programming burdens of a small, limited amount of main memory. Four decades after its invention, it's the former reason that reigns today.

Consider a collection of programs running all at once on a computer. Of course, to allow multiple programs to share the same memory, we must be able to protect the programs from each other, ensuring that a program can only read and write the portions of main memory that have been assigned to it. Main memory need contain only the active portions of the many programs, just as a cache contains only the active portion of one program. Thus, the principle of locality enables virtual memory as well as caches, and virtual memory allows us to efficiently share the processor as well as the main memory.

The second motivation for virtual memory is to allow a single user program to exceed the size of primary memory. Formerly, if a program became too large for memory, it was up to the programmer to make it fit. Programmers divided programs into pieces and then identified the pieces that were mutually exclusive. These *overlays* were loaded or unloaded under user program control during execution, with the programmer ensuring that the program never tried to access an overlay that was not loaded and that the overlays loaded never exceeded the total size of the memory. Overlays were traditionally organized as modules, each containing both code and data.



In virtual memory, the address is broken into a *virtual page number* and a *page offset*. Figure 5.20 shows the translation of the virtual page number to a *physical page number*. The physical page number constitutes the upper portion of the physical address, while the page offset, which is not changed, constitutes the lower portion. The number of bits in the page offset field determines the page size. The number of pages addressable with the virtual address need not match the number of pages addressable with the physical address. Having a larger number of virtual pages

than physical pages is the basis for the illusion of an essentially unbounded amount of virtual memory.



8. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan Kaufmann Elsevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for TLBs - Input/output system		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.8/ 13	

1.CONTENT LIST:

TLBs - Input/output system

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN: To

make the students know the basics of TLB

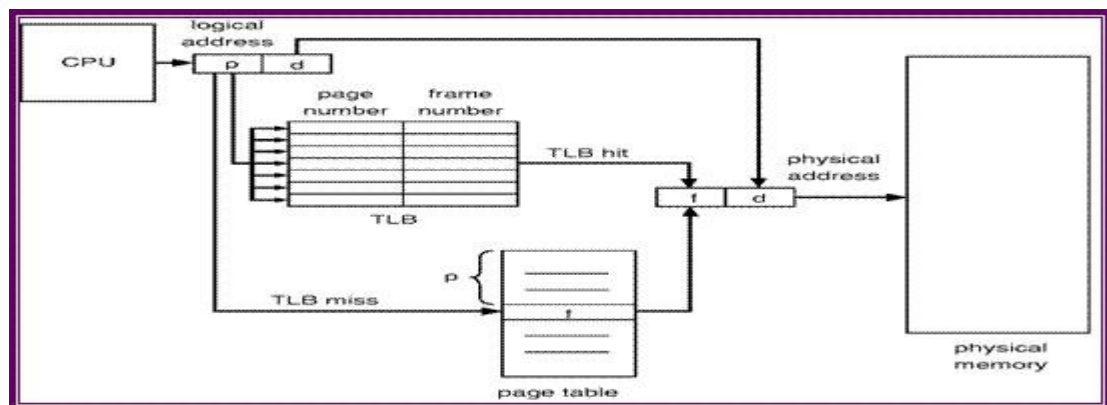
4.OUTCOMES:

- i. Describe TLB
- ii. Analyze the features of TLB

5.LINK SHEET:

- i. Define TLB
- ii. Describe TLB system in detail

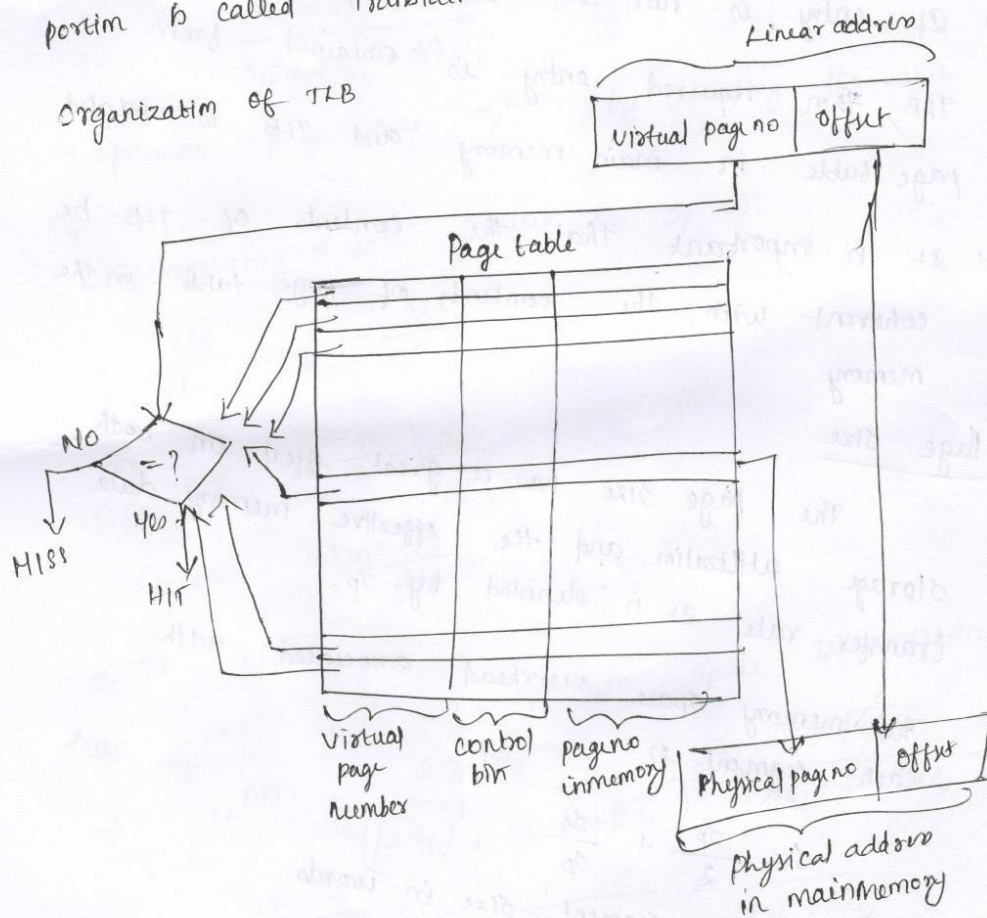
6.EVOCATION: (5 Minutes)



Translation Lookaside Buffer (TLB)

To reduce the access time and degradation of performance, a small portion of the page table is accommodated in the memory management unit. This portion is called Translation Lookaside Buffer (TLB).

Organization of TLB



- A given virtual address is compared with TLB entries for the reference page by mmu. If page table entry for this page is found in TLB, physical address is obtained immediately.
- If entry is not found, there is a miss in TLB, then required entry is obtained from page table in main memory and TLB is updated.
- It is important that the contents of TLB be coherent with the contents of page table in the memory

Page size

The page size has a great effect on both storage utilization and the effective memory data transfer rate. It is denoted by \mathcal{P} .

The memory space overhead associated with each segment is

$$\mathcal{S} = \frac{\mathcal{P}}{2} + \frac{\mathcal{S}_s}{\mathcal{P}}$$

\mathcal{S}_s - average segment size in words

$\frac{\mathcal{S}_s}{\mathcal{P}}$ - size of the page table

Space Utilization factor is

$$u = \frac{s_b}{s_b + s} = \frac{s_b}{s_b + \frac{s_p}{2} + \frac{s_b}{s_p}} = \frac{2s_b s_p}{2s_b s_p + s_p^2 + 2s_b}$$

$$= \frac{2s_b s_p}{s_p^2 + 2s_b(1 + s_p)} \rightarrow \textcircled{1}$$

→ optimum page size s_p^{OPT} by denoting the value

of s_p ,

Differentiating s with respect to $s_p \Rightarrow$

$$\frac{ds}{ds_p} = \frac{1}{2} - \frac{s_b}{s_p^2}$$

$$\frac{ds}{ds_p} = 0 \Rightarrow \frac{1}{2} - \frac{s_b}{s_p^2}$$

$\therefore s$ is min when $\frac{ds}{ds_p} = 0$

$$s_p^{\text{OPT}} = \sqrt{2s_b}$$

Sub s_p^{OPT} in $\textcircled{1} \Rightarrow$ optimum space Utilization

$$\Rightarrow u^{\text{OPT}} = \frac{2s_b \sqrt{2s_b}}{(\sqrt{2s_b})^2 + 2s_b(1 + \sqrt{2s_b})} = \frac{2s_b \sqrt{2s_b}}{4s_b + 2s_b \sqrt{2s_b}}$$

$$= \frac{1}{\frac{4s_b}{2s_b \sqrt{2s_b}} + 1} = \frac{1}{\frac{2}{\sqrt{2s_b}} + 1} = \frac{1}{1 + \sqrt{2/s_b}}$$

$$u^{\text{OPT}} = \frac{1}{1 + \sqrt{2/s_b}}$$

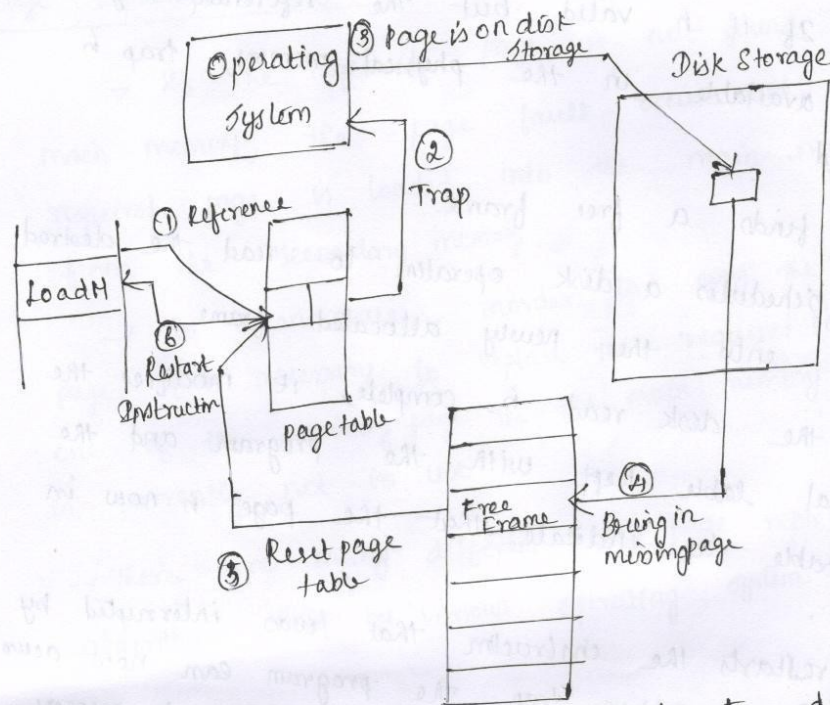
Effect of page size and segment size on
Space Utilisation. Space utilisation factor increases with
increase in segment size and it is optimum when $S_p = \sqrt{2} S_s$.

Page fault and Demand paging

→ If the page required by the processor is not in
the main memory, the page fault occurs and the
required page is loaded into main memory from
secondary storage memory by special routine called
Page fault routine.

→ A demand paging system is similar to paging
system with swapping

→ When a program is to be swapped in the
pager guesses which pages will be used before the
program is swapped out again. ~~the~~ Instead of swapping
in a whole process, the pager brings only those
necessary pages into memory.



- The valid-invalid bits are used to distinguish between those pages that are in memory and those pages that are on the disk.
- when bit is set to valid, it indicates that associated page is both legal and in memory.
- If it is set to invalid, it indicates that the page either is not valid, or is valid but is currently on the disk.

This procedure goes through following steps.

1. It checks an internal table for this program, to determine whether the reference was a valid or invalid memory access.

→ If the reference is invalid, it terminates the process. If it is valid, but the referenced page is not available in the physical memory, trap is activated.

→ It finds a free frame

→ It schedules a disk operation to read the desired page into the newly allocated frame.

→ When the disk read is complete, it modifies the internal table kept with the program and the page table to indicate that the page is now in memory.

→ It restarts the instruction that was interrupted by the illegal address trap. The program can now access the page as though it had always been in memory.

→ The hardware to implement demand paging is the same as the hardware for paging and swapping.

Page table: This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits

(21)

Page Replacement Algorithm

→ If the required page is not found in the main memory, the page fault occurs and the required page is loaded into the main memory from the secondary memory.

→ no vacant space, in order to copy the required page, it is necessary to replace the required page with one of the existing page in the main memory which is currently not in use.

→ There are many different ~~rep~~ page replacement algorithms used by various operating systems. They are

FIFO Algorithm

→ It is the simplest page replacement algorithm.

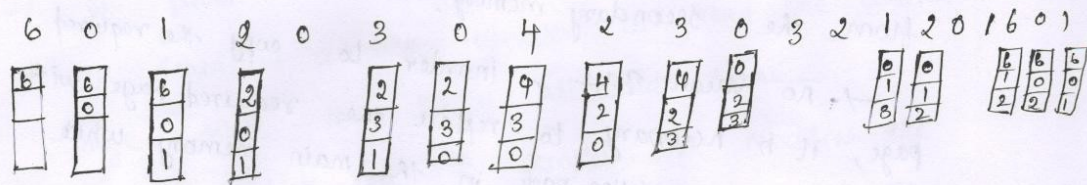
A First In First out replacement algorithm replaces the new page with oldest page in the main memory

Eg: Reference string, our three frames are initially empty.

6, 0, 1 cause page faults and required pages are brought into these empty frames.

next reference (2) replaces page 6, because page 6 was brought in first. Since 0 is next reference and 0 is already in memory, no fault for this reference

Reference string



page frames

Fig: FIFO page replacement Algorithm

- First reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1, 2) to be brought in. This process continues as in Fig above ~~shown~~ shown above.
- FIFO page replacement algorithm is easy to understand and program. It may replace the most needed page as its oldest page.

Optimal Algorithm

An optimal page replacement algorithm has the lowest page fault rate of all algorithms. It has been called OPT or MIN. It states that replace the page that will not be used for the longest period of time.

(22)

Eg:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	6	0	1
<div>6 0 1</div>	<div>6 0 1</div>	<div>6 0 1</div>	<div>2 0 1</div>		<div>3 0 1</div>		<div>4 0 1</div>		<div>3 0 1</div>		<div>3 0 1</div>		<div>2 0 1</div>				<div>6 0 1</div>		

page frames

The reference to page 2 replaces page 6, 6 will not be used until reference 18, whereas page 0 will be used at 5 and page 1 at 14. page 3 replaces page 1 as page 1 will be the last of three pages in memory to be referenced again. No replacement algorithm can process the reference string in three frames with less than nine faults.

Disadvantage

It is difficult to implement, because it requires future knowledge of reference string.

LRU algorithm

The algorithm which replaces the page that has not been used for the longest period of time is referred as least recently used algorithm (LRU).

Reference String

6	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	6	0	1																																																								
<table><tr><td>6</td></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	6	0	1	<table><tr><td>6</td></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	6	0	1	<table><tr><td>6</td></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	6	0	1	<table><tr><td>2</td></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	2	0	1	<table><tr><td>2</td></tr><tr><td>0</td></tr><tr><td>3</td></tr></table>	2	0	3	<table><tr><td>2</td></tr><tr><td>0</td></tr><tr><td>3</td></tr></table>	2	0	3	<table><tr><td>4</td></tr><tr><td>0</td></tr><tr><td>3</td></tr></table>	4	0	3	<table><tr><td>4</td></tr><tr><td>0</td></tr><tr><td>2</td></tr></table>	4	0	2	<table><tr><td>4</td></tr><tr><td>3</td></tr><tr><td>2</td></tr></table>	4	3	2	<table><tr><td>0</td></tr><tr><td>3</td></tr><tr><td>2</td></tr></table>	0	3	2	<table><tr><td>0</td></tr><tr><td>3</td></tr><tr><td>2</td></tr></table>	0	3	2	<table><tr><td>3</td></tr><tr><td>2</td></tr><tr><td>1</td></tr></table>	3	2	1	<table><tr><td>3</td></tr><tr><td>2</td></tr><tr><td>1</td></tr></table>	3	2	1	<table><tr><td>1</td></tr><tr><td>3</td></tr><tr><td>2</td></tr></table>	1	3	2	<table><tr><td>2</td></tr><tr><td>0</td></tr><tr><td>1</td></tr></table>	2	0	1	<table><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>6</td></tr></table>	0	1	6	<table><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>6</td></tr></table>	0	1	6	<table><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>6</td></tr></table>	0	1	6	<table><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>6</td></tr></table>	0	1	6
6																																																																											
0																																																																											
1																																																																											
6																																																																											
0																																																																											
1																																																																											
6																																																																											
0																																																																											
1																																																																											
2																																																																											
0																																																																											
1																																																																											
2																																																																											
0																																																																											
3																																																																											
2																																																																											
0																																																																											
3																																																																											
4																																																																											
0																																																																											
3																																																																											
4																																																																											
0																																																																											
2																																																																											
4																																																																											
3																																																																											
2																																																																											
0																																																																											
3																																																																											
2																																																																											
0																																																																											
3																																																																											
2																																																																											
3																																																																											
2																																																																											
1																																																																											
3																																																																											
2																																																																											
1																																																																											
1																																																																											
3																																																																											
2																																																																											
2																																																																											
0																																																																											
1																																																																											
0																																																																											
1																																																																											
6																																																																											
0																																																																											
1																																																																											
6																																																																											
0																																																																											
1																																																																											
6																																																																											
0																																																																											
1																																																																											
6																																																																											

→ when reference to page 4 occurs, LRU replacement sees that, of the ~~from~~ three frames in memory, page 2 was used least recently. recently used page is 0 and just before that page 3 was used. LRU algorithm replaces page 2 not knowing that page 2

→ when page 2 fault, LRU replaces page 3, among (0, 3, 4) page 3 is least recently used

→ LRU with 12 faults is still much better than FIFO replacement with 15.

Counting algorithm

In the counting algorithm, a counter of the number of references that have been made to each page are kept, and based on these counts, the following two schemes work.

LRU algorithm

The Least Frequently Used page replacement algorithm requires page with smallest count be replaced

MFU algorithm :- Most Frequently Used page replacement algorithm is based on argument that page with smallest count was probably just brought in and yet to be used



7. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan Kaufmann, Fifth edition, 2014

8. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for TLBs - Input/output system		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.9/ 13	

1.CONTENT LIST:

TLBs - Input/output system

2. SKILLS ADDRESSED:

Learning Understanding

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of input/ output system

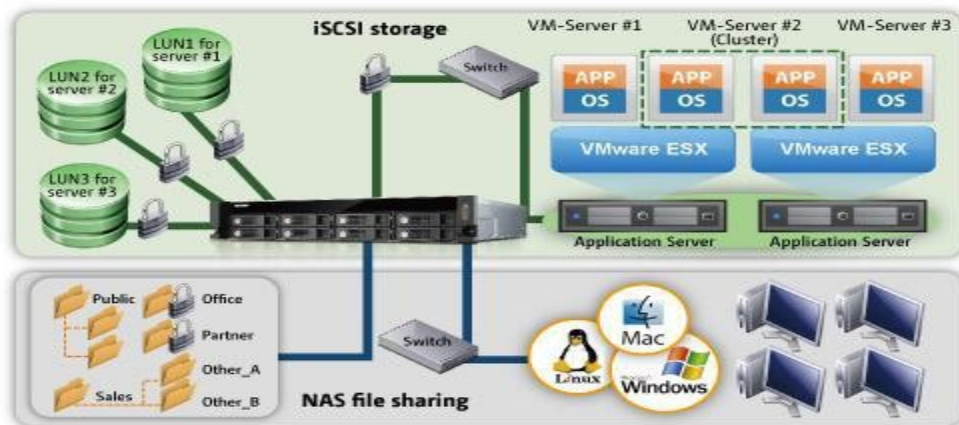
4.OUTCOMES:

- Describe input and output system
- Analyze the features of input/ output system

5.LINK SHEET:

- Define input and output system
- (ii) Design the hardware and software using advanced processors

6.EVOCATION: (5 Minutes)



7. Lecture Notes

Input/Output

The computer system's I/O architecture is its interface to the outside world. This architecture is designed to provide a systematic means of controlling interaction with the outside world and to provide the operating system with the information it needs to manage I/O activity effectively.

There are three principal I/O techniques: programmed I/O, in which I/O occurs under the direct and continuous control of the program requesting the I/O operation; interrupt-driven I/O, in which a program issues an I/O command and then continues to execute, until it is interrupted by the I/O hardware to signal the end of the I/O operations; and direct memory access (DMA), in which a specialized I/O processor takes over control of an I/O operation to move a large block of data.

Two important examples of external I/O interfaces are FireWire and Infiniband.

Peripherals and the System Bus

There are a wide variety of peripherals each with varying methods of operation

Impractical for the processor to accommodate all

Data transfer rates are often slower than the processor and/or memory

Impractical to use the high-speed system bus to communicate directly

Data transfer rates may be faster than that of the processor and/or memory This mismatch may lead to inefficiencies if improperly managed

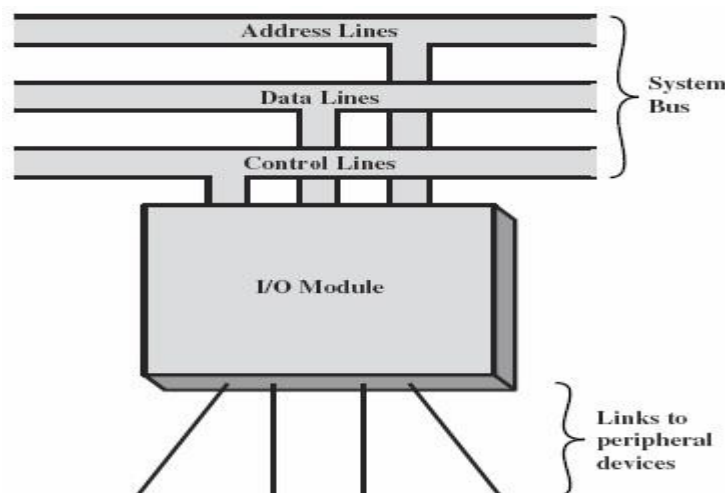
Peripheral often use different data formats and word

lengths Purpose of I/O Modules

Interface to the processor and memory via the system bus or control switch
Interface to one or more peripheral devices

Purpose of I/O Modules

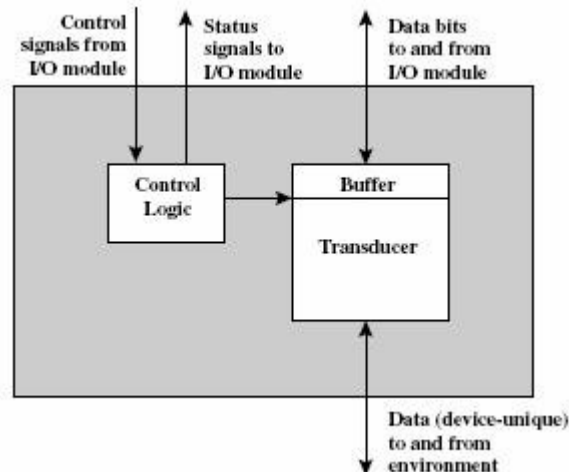
- Interface to the processor and memory via the system bus or control switch
- Interface to one or more peripheral devices



External Devices:

External device categories

- **Human readable:** communicate with the computer user – CRT
- **Machine readable:** communicate with equipment – disk drive or tape drive
- **Communication:** communicate with remote devices – may be human readable or machine readable



The External Device – I/O Module

- **Control signals:** determine the function that will be performed
- **Data:** set of bits to be sent or received
- **Status signals:** indicate the state of the device
- **Control logic:** controls the device's operations
- **Transducer:** converts data from electrical to other forms of energy
- **Buffer:** temporarily holds data being transferred

Keyboard/Monitor

- Most common means of computer/user interaction
- Keyboard provides input that is transmitted to the computer
- Monitor displays data provided by the computer
- The character is the basic unit of exchange
- Each character is associated with a 7 or 8 bit code

Disk Drive

- Contains electronics for exchanging data, control, and status signals with an I/O module
- Contains electronics for controlling the disk read/write mechanism
- Fixed-head disk – transducer converts between magnetic patterns on the disk surface and bits in the buffer
- Moving-head disk – must move the disk arm rapidly across the surface

I/O Modules

Module Function

- Control and timing
- Processor communication

- Device communication
- Data buffering
- Error detection

I/O control steps

- Processor checks I/O module for external device status
- I/O module returns status
- If device ready, processor gives I/O module command to request data transfer
- I/O module gets a unit of data from device
- Data transferred from the I/O module to the processor

Processor communication

Command decoding: I/O module accepts commands from the processor sent as signals on the control bus

Data: data exchanged between the processor and I/O module over the data bus Status reporting: common status signals BUSY and READY are used because peripherals are slow

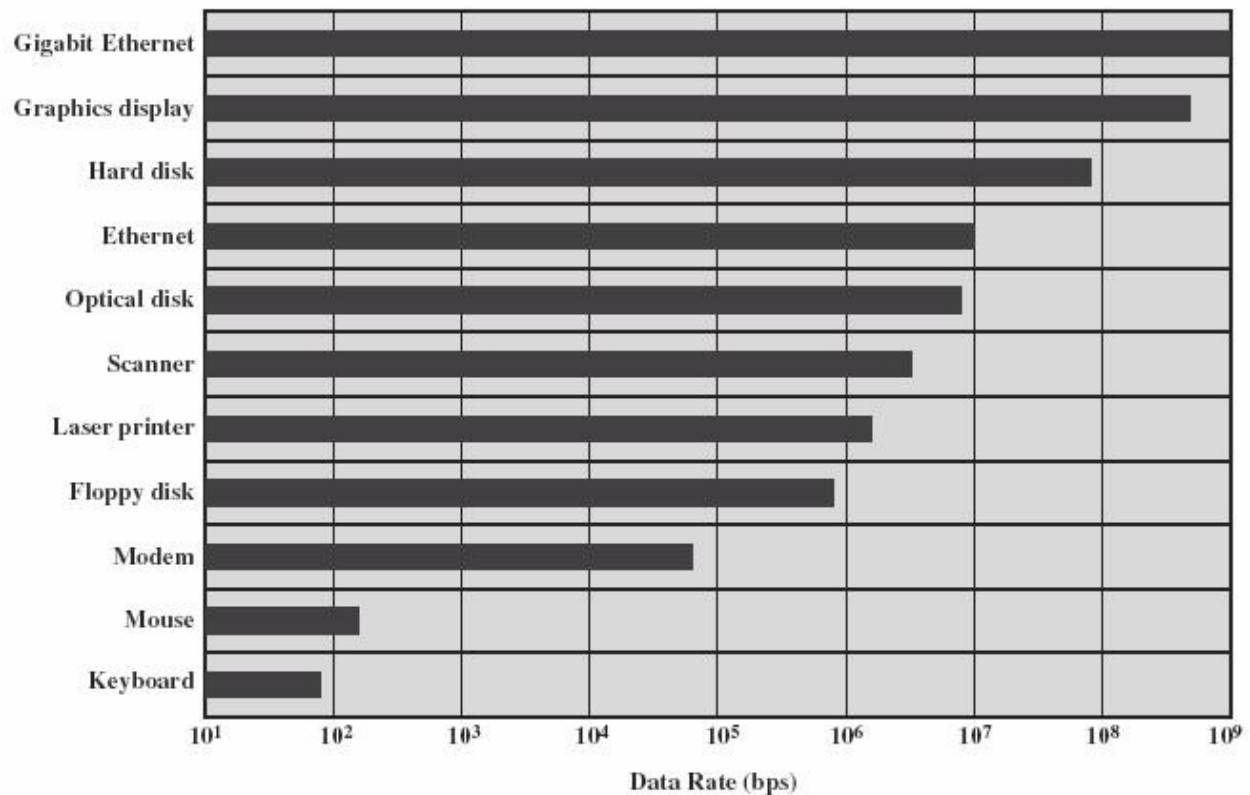
Address recognition: I/O module must recognize a unique address for each peripheral that it controls I/O module communication

Device communication: commands, status information, and data

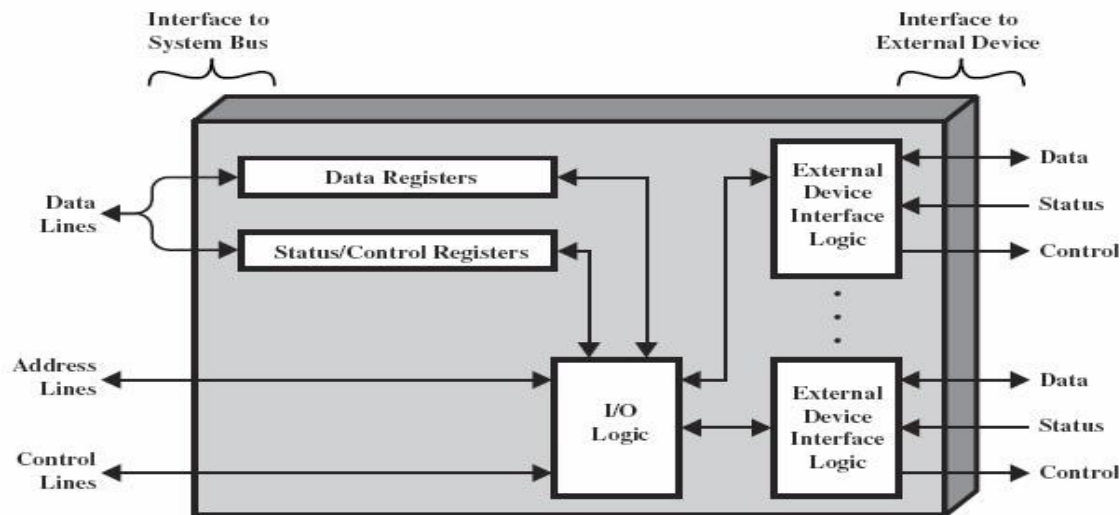
Data buffering: data comes from main memory in rapid burst and must be buffered by the I/O module and then sent to the device at the device's rate

Error detection: responsible for reporting errors to the processor

Typical I/O Device Data Rates



I/O Module Structure: Block Diagram of an I/O Module



Module connects to the computer through a set of signal lines – system bus

- Data transferred to and from the module are buffered with data registers
- Status provided through status registers – may also act as control registers
- Module logic interacts with processor via a set of control signal lines
- Processor uses control signal lines to issue commands to the I/O module
- Module must recognize and generate addresses for devices it controls
- Module contains logic for device interfaces to the devices it controls
- I/O module functions allow the processor to view devices in a simple-minded way
- I/O module may hide device details from the processor so the processor only functions in terms of simple read and write operations – timing, formats, etc...
- I/O module may leave much of the work of controlling a device visible to the processor – rewind a tape, etc...

I/O channel or I/O processor

- I/O module that takes on most of the detailed processing burden
- Used on mainframe computers

I/O controller of device controller

- Primitive I/O module that requires detailed control
- Used on microcomputers



8. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanescic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan Kaufmann/Pearson, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for Programmed I/O		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.10/ 13	

1.CONTENT LIST:

Programmed I/O

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of programmed output and input controller

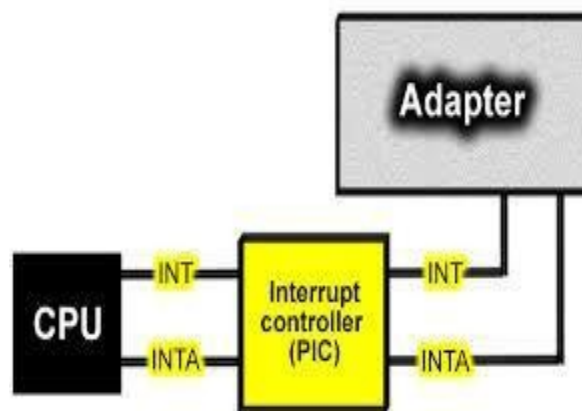
4.OUTCOMES:

- Sort the applications of programmed input and output controller
- Analyze the features of programmed input and output controller

5.LINK SHEET:

- Define programmed input and output controller
- (ii) Design the programmed input and output controller

6.EVOCATION: (5 Minutes)



7. Lecture Notes

Programmed I/O

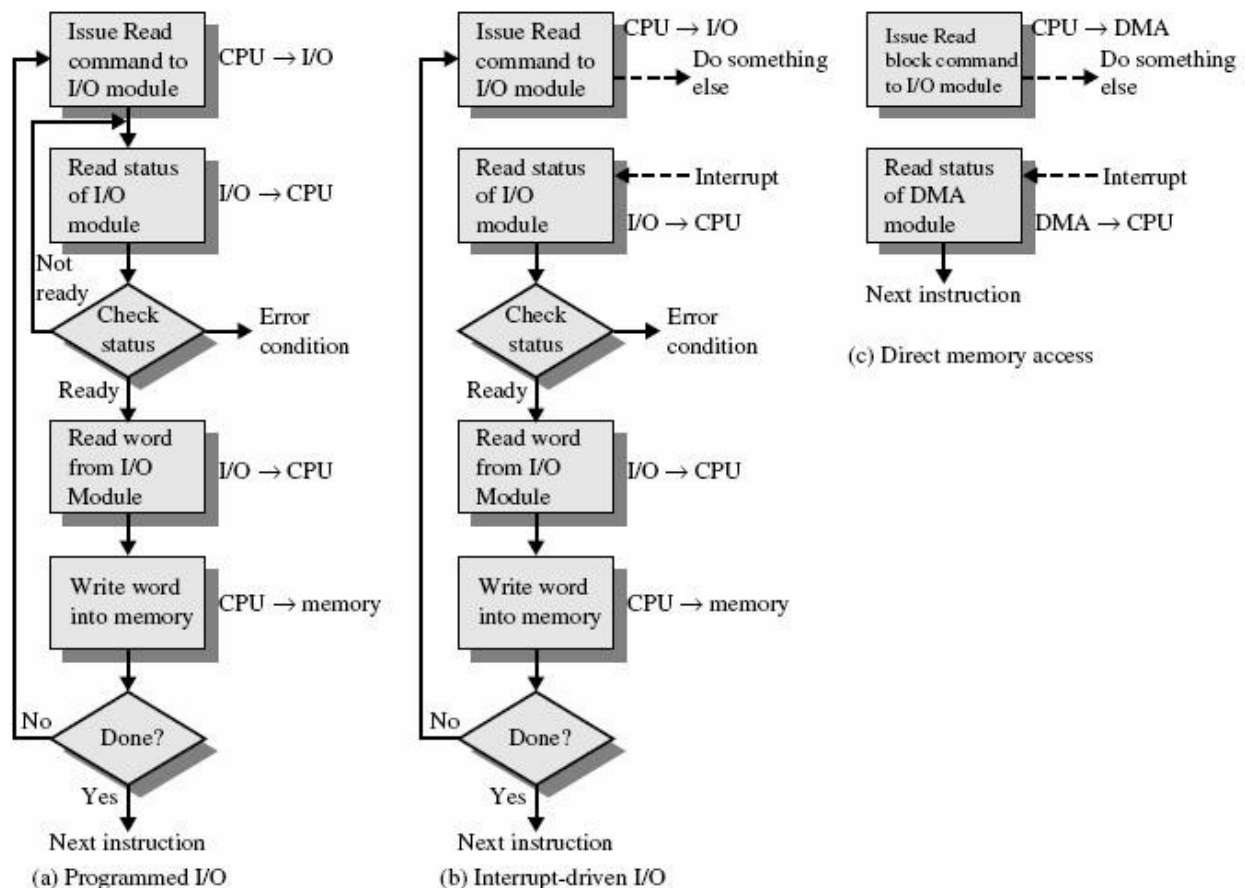
Overview of Programmed I/O

- Processor executes an I/O instruction by issuing command to appropriate I/O module
- I/O module performs the requested action and then sets the appropriate bits in the I/O status register – I/O module takes no further action to alert the processor – it does not interrupt the processor
- The processor periodically checks the status of the I/O module until it determines that the operation is complete

I/O Commands

The processor issues an address, specifying I/O module and device, and an I/O command. The commands are:

- **Control:** activate a peripheral and tell it what to do
- **Test:** test various status conditions associated with an I/O module and its peripherals
- **Read:** causes the I/O module to obtain an item of data from the peripheral and place it into an internal register
- **Write:** causes the I/O module to take a unit of data from the data bus and



Three Techniques for Input of a Block of Data

I/O Instructions

Processor views I/O operations in a similar manner as memory operations Each device is given a unique identifier or address

Processor issues commands containing device address – I/O module must check address lines to see if the command is for itself.

I/O mapping

Memory-mapped I/O

- ▢ Single address space for both memory and I/O devices
 - Disadvantage – uses up valuable memory address space
- ▢ I/O module registers treated as memory addresses
- ▢ Same machine instructions used to access both memory and I/O devices
 - Advantage – allows for more efficient programming
- ▢ Single read line and single write lines needed
- ▢ Commonly used

• Isolated I/O

- ▢ Separate address space for both memory and I/O devices
- ▢ Separate memory and I/O select lines needed
- ▢ Small number of I/O instructions
- ▢ Commonly used

8. TEXT BOOKS:



V.Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design“, Morgan auffman lsevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

Unit 5– Lesson No.11/ 13

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for DMA and interrupts		
Time:	45 Minutes	
Lesson. No		

1.CONTENT LIST:

DMA and interrupts

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of DMA and interrupts

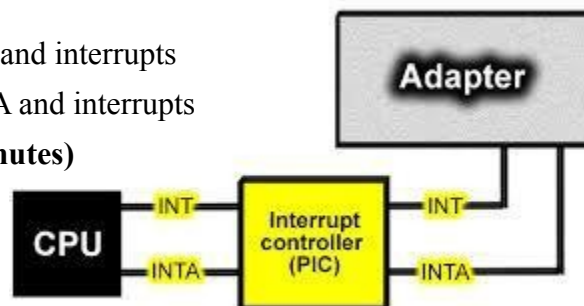
4.OUTCOMES:

- Sort the applications of DMA and interrupts
- Analyze the features of DMA and interrupts

5.LINK SHEET:

- Define DMA and interrupts
- Design the DMA and interrupts

6.EVOCATION: (5 Minutes)



7. Lecture Notes

Interrupt-Driven I/O

- Overcomes the processor having to wait long periods of time for I/O modules
- The processor does not have to repeatedly check the I/O module status

I/O module view point

- I/O module receives a READ command from the processor
- I/O module reads data from desired peripheral into data register
- I/O module interrupts the processor
- I/O module waits until data is requested by the processor
- I/O module places data on the data bus when requested

Processor view point

- The processor issues a READ command
- The processor performs some other useful work
- The processor checks for interrupts at the end of the instruction cycle
- The processor saves the current context when interrupted by the I/O module
- The processor reads the data from the I/O module and stores it in memory
- The processor restores the saved context and resumes execution

Design Issues

How does the processor determine which device issued the interrupt
How are multiple interrupts dealt with

Device identification

Multiple interrupt lines – each line may have multiple I/O modules
Software poll – poll each I/O module

Separate command line – TESTI/O

Processor reads status register of I/O module
Time consuming

Daisy chain

Hardware poll

Common interrupt request line

Processor sends interrupt acknowledge

Requesting I/O module places a word of data on the data lines – vector that uniquely identifies the I/O module – vectored interrupt

• Bus arbitration

- I/O module first gains control of the bus
- I/O module sends interrupt request
- The processor acknowledges the interrupt request
- I/O module places its vector on the data lines

Multiple interrupts

- The techniques above not only identify the requesting I/O module but provide methods of assigning priorities

- Multiple lines – processor picks line with highest priority
- Software polling – polling order determines priority
- Daisy chain – daisy chain order of the modules determines priority
- Bus arbitration – arbitration scheme determines priority

Intel 82C59A Interrupt Controller

Intel 80386 provides

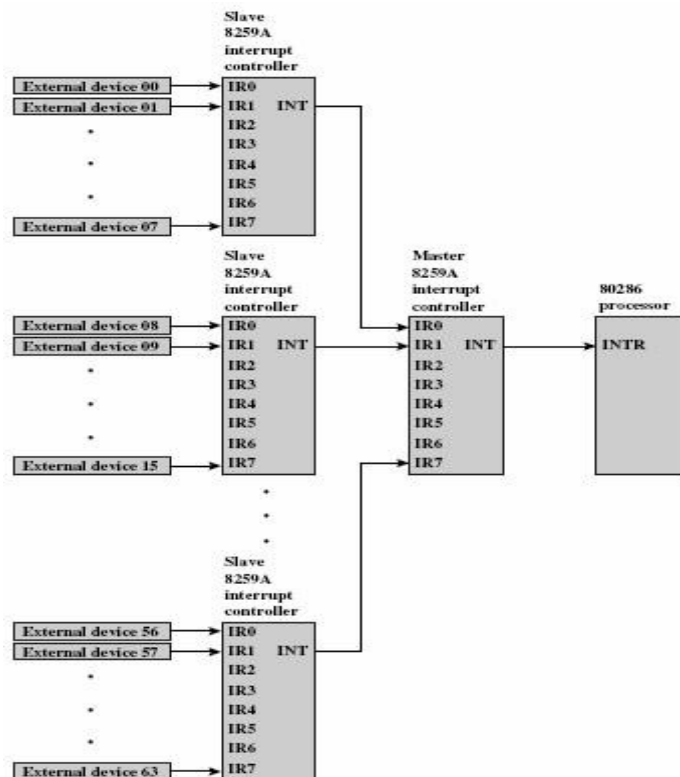
- Single Interrupt Request line – INTR
- Single Interrupt Acknowledge line – INTA
- Connects to an external interrupt arbiter, 82C59A, to handle multiple devices and priority structures
- 8 external devices can be connected to the 82C59A – can be cascaded to 64 82C59A operation – only manages interrupts
- Accepts interrupt requests
- Determines interrupt priority
- Signals the processor using INTR
- Processor acknowledges using INTA
- Places vector information of data bus
- Processor process interrupts and communicates directly with I/O module

82C59A interrupt modes

Fully nested – priority form 0 (IR0) to 7 (IR7)

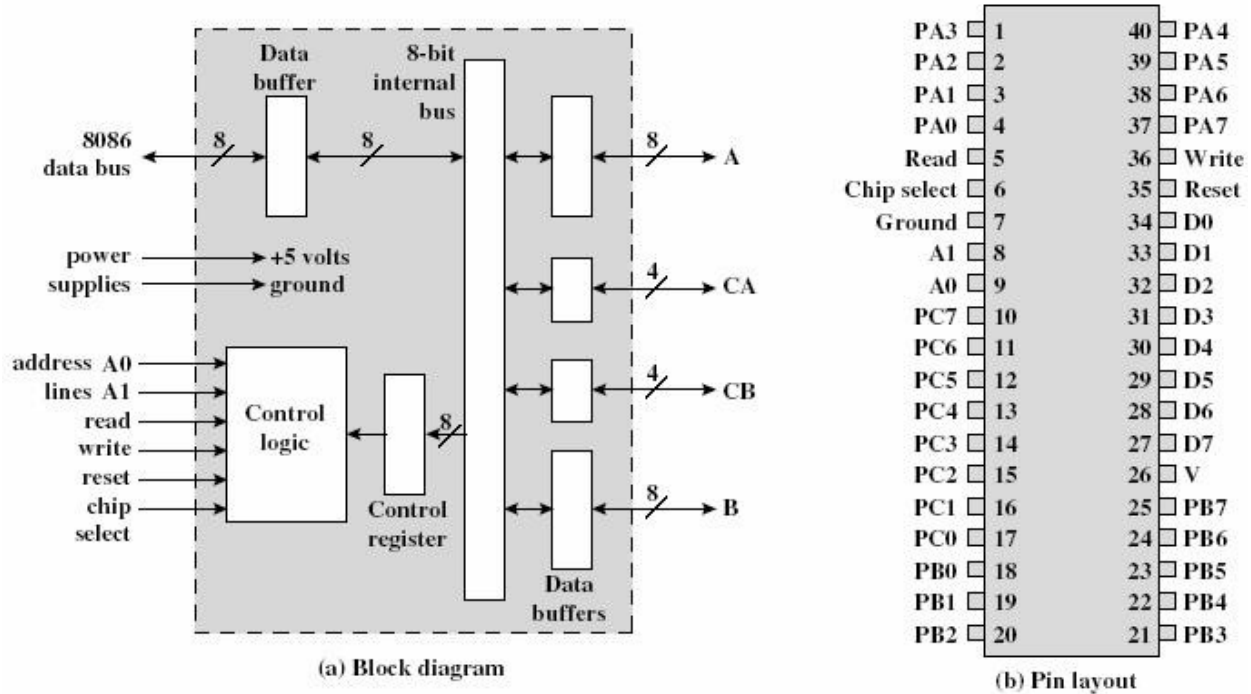
Rotating – several devices same priority - most recently device lowest priority

Special mask – processor can inhibit interrupts from selected devices.

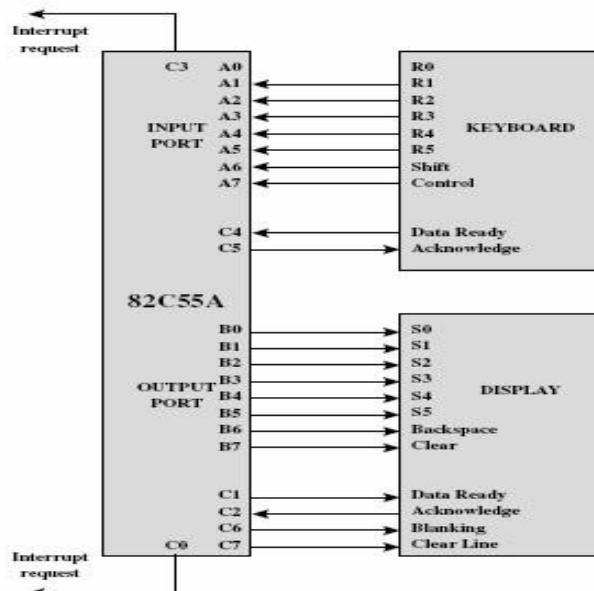


Intel 82C55A Programmable Peripheral Interface

- Single chip, general purpose I/O module
- Designed for use with the Intel 80386
- Can control a variety of simple peripheral devices



A, B, C function as 8 bit I/O ports (C can be divided into two 4 bit I/O ports) Left side of diagram show the interface to the 80386 bus.



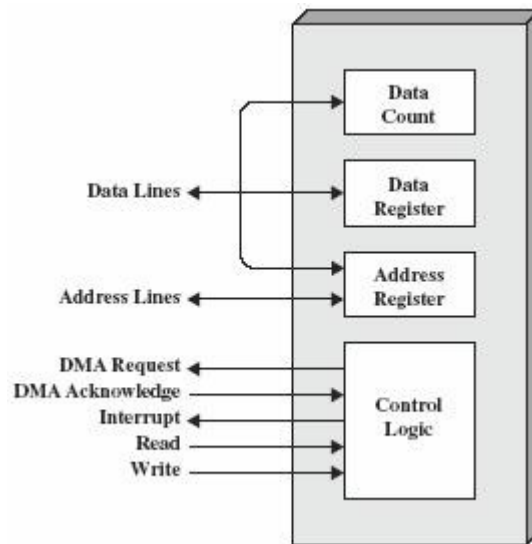
Direct Memory Access

Drawback of Programmed and Interrupt-Driven I/O

- I/O transfer rate limited to speed that processor can test and service devices
- Processor tied up managing I/O transfers

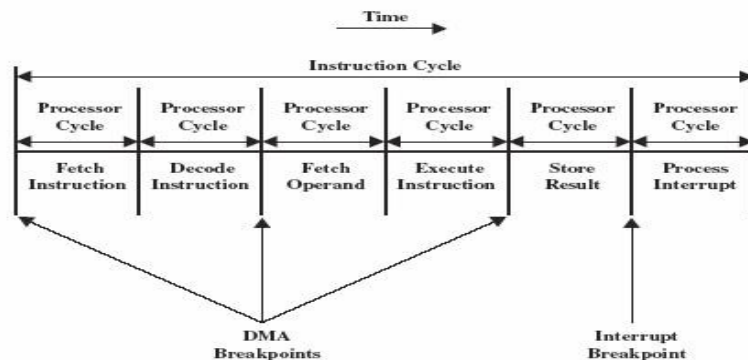
DMA Function

- DMA module on system bus used to mimic the processor.
- DMA module only uses system bus when processor does not need it.
- DMA module may temporarily force processor to suspend operations – cycle stealing.



DMA Operation

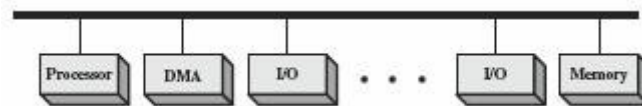
- The processor issues a command to DMA module
- Read or write
- I/O device address using data lines
- Starting memory address using data lines – stored in address register
- Number of words to be transferred using data lines – stored in data register
- The processor then continues with other work
- DMA module transfers the entire block of data – one word at a time – directly to or from memory without going through the processor DMA module sends an interrupt to the processor when complete



DMA and Interrupt Breakpoints during Instruction Cycle

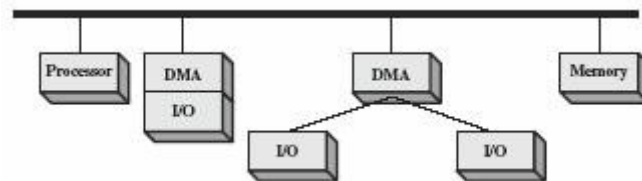
- The processor is suspended just before it needs to use the bus.
- The DMA module transfers one word and returns control to the processor.
- Since this is not an interrupt the processor does not have to save context.
- The processor executes more slowly, but this is still far more efficient than either programmed or interrupt-driven I/O.

DMA Configurations



Single bus – detached DMA module

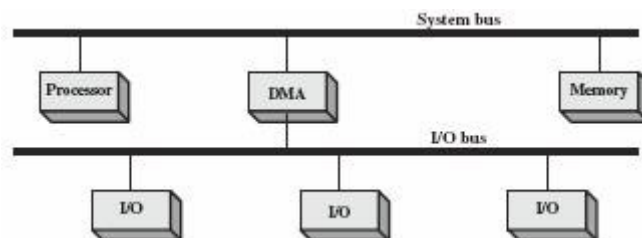
Each transfer uses bus twice – I/O to DMA, DMA to memory
Processor suspended twice.



Single bus – integrated DMA module

Module may support more than one device

Each transfer uses bus once – DMA to memory
Processor suspended once.



Separate I/O bus

Bus supports all DMA enabled devices

Each transfer uses bus once – DMA to memory
Processor suspended once.



8. TEXT BOOKS:

V.Carl Hamacher, Zvonko G. Varanasic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design“ , Morgan auffman lsevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

	SRI VIDYA COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRUDHUNAGAR Department of Computer Science & Engineering	
Class	II Year (03Semester)	
Subject Code	CS6303	
Subject	Computer Architecture	
Prepared By	Kaviya.P	
Lesson Plan for I/O processors		
Time:	45 Minutes	
Lesson. No	Unit 5– Lesson No.12/ 13	

1.CONTENT LIST:

I/O processors

2. SKILLS ADDRESSED:

Learning Remembering

3. OBJECTIVE OF THIS LESSON PLAN:

To make the students know the basics of I/O processors

4.OUTCOMES:

- i. Sort the applications of I/O processors
- ii. Analyze the features of I/O processors

5.LINK SHEET:

- i. Define I/O processors
- (ii) Design the I/O processors

6.EVOCATION: (5 Minutes)



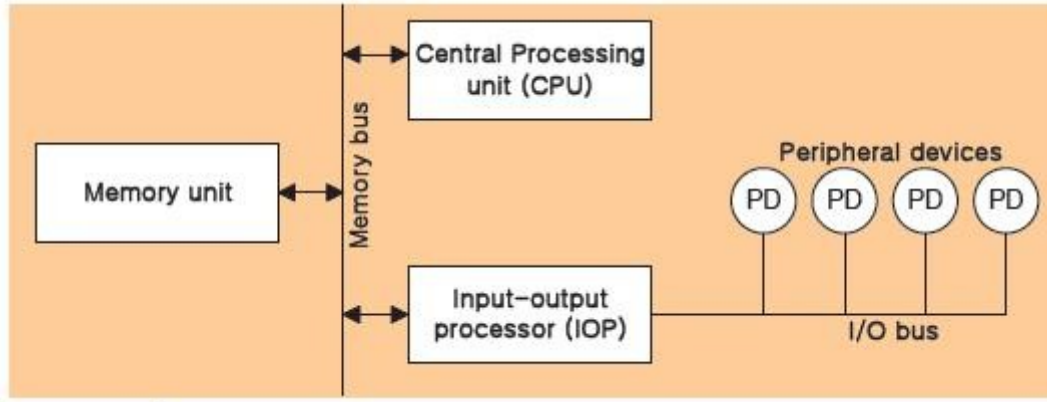
7. Lecture Notes

Input-Output Processor (IOP)

Communicate directly with all I/O devices

Fetch and execute its own instruction

IOP instructions are specifically designed to facilitate I/O transfer



Command

Instruction *that are read form memory by an IOP*

Distinguish from instructions that are read by the CPU

Commands are prepared by experienced programmers and are stored in memory

Command word = IOP program Memory

I/O Channels and Processors

The Evolution of the I/O Function

1. Processor directly controls peripheral device
2. Addition of a controller or I/O module – programmed I/O
3. Same as 2 – interrupts added
4. I/O module direct access to memory using DMA
5. I/O module enhanced to become processor like – I/O channel
6. I/O module has local memory of its own – computer like – I/O processor
 - More and more the I/O function is performed without processor involvement.
 - The processor is increasingly relieved of I/O related tasks – improved performance.

Characteristics of I/O Channels

- Extension of the DMA concept
- Ability to execute I/O instructions – special-purpose processor on I/O channel – complete control over I/O operations
- Processor does not execute I/O instructions itself – processor initiates I/O transfer by instructing the I/O channel to execute a program in memory
- Program specifies

Device or devices
 Area or areas of memory
 Priority
 Error condition actions

Two type of I/O channels

• Selector channel

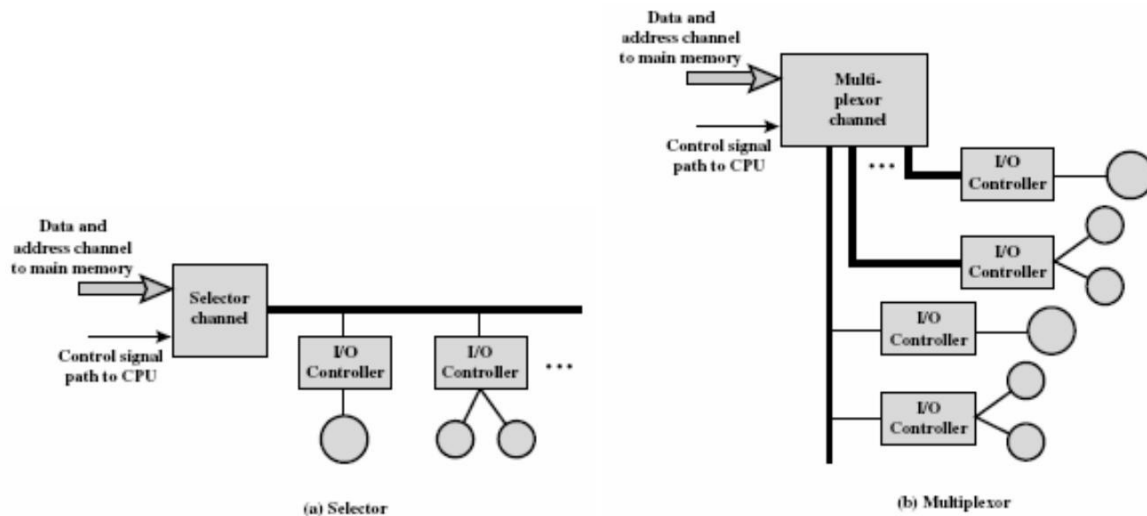
- Controls multiple high-speed devices
- Dedicated to the transfer of data with one of the devices
- Each device handled by a controller, or I/O module
- I/O channel controls these I/O controllers

• Multiplexor channel

Can handle multiple devices at the same time

Byte multiplexor – used for low-speed devices

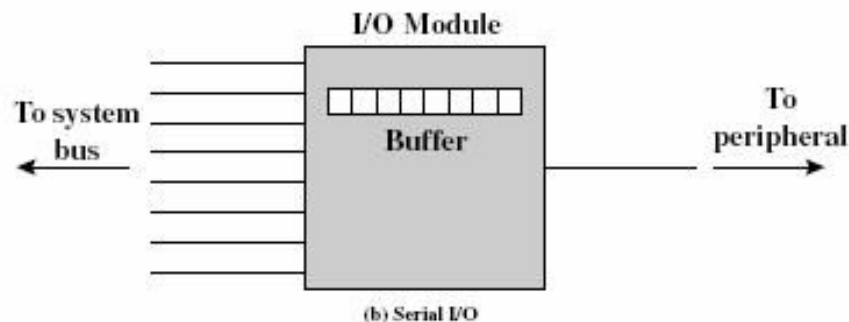
Block multiplexor – interleaves blocks of data from several devices.

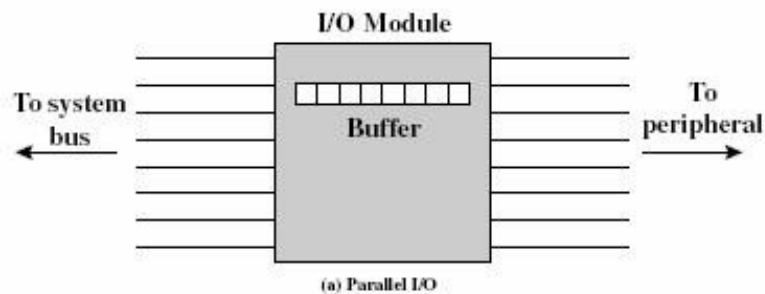


The External Interface: FireWire and Infiniband

Type of Interfaces

- Parallel interface – multiple bits transferred simultaneously
- Serial interface – bits transferred one at a time





I/O module dialog for a write operation

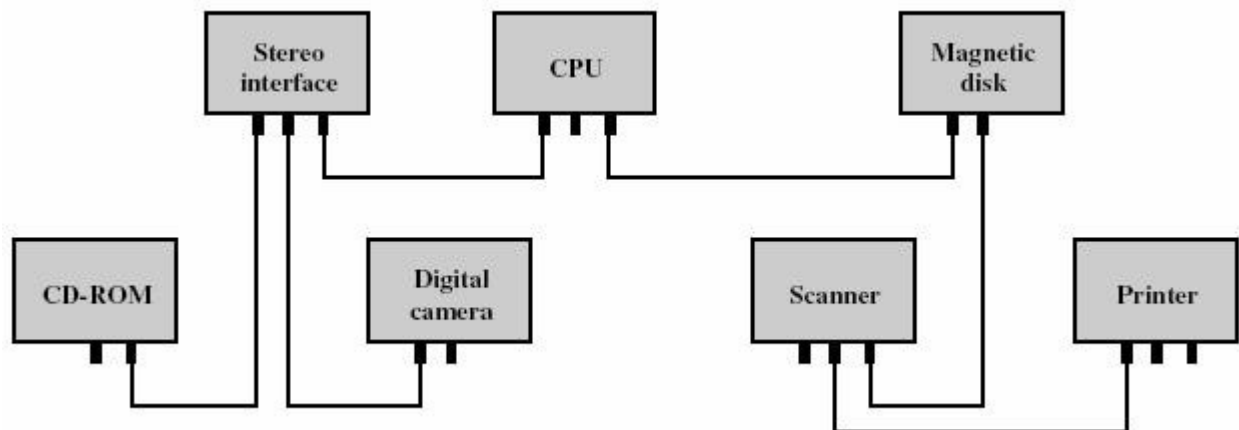
1. I/O module sends control signal – requesting permission to send data
2. Peripheral acknowledges the request
3. I/O module transfer data
4. Peripheral acknowledges receipt of data

FireWire Serial Bus – IEEE 1394

- Very high speed serial bus
- Low cost
- Easy to implement
- Used with digital cameras, VCRs, and televisions

FireWire Configurations

- Daisy chain
- 63 devices on a single port – 64 if you count the interface itself
- 1022 FireWire busses can be interconnected using bridges
- Hot plugging
- Automatic configuration
- No terminations
- Can be tree structured rather than strictly daisy chained



FireWire three layer stack:Physical layer

Defines the transmission media that are permissible and the electrical and signaling characteristics of each 25 to 400 Mbps

Converts binary data to electrical signals

Provides arbitration services

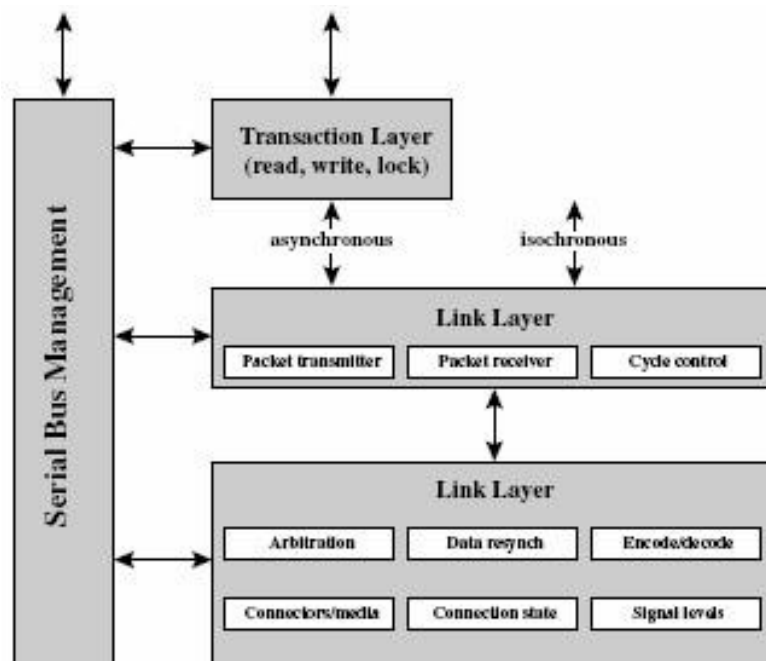
- Based on tree structure
- Root acts as arbiter
- First come first served
- Natural priority controls simultaneous requests – nearest root
- Fair arbitration
- Urgent arbitration

Link layer

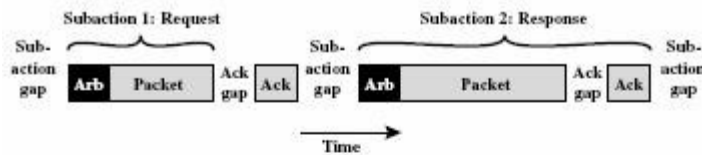
- Describes the transmission of data in the packets
- Asynchronous
 - Variable amount of data and several bytes of transaction data transferred as a packet
 - Uses an explicit address
 - Acknowledgement returned
- Isochronous
 - Variable amount of data in sequence of fixed sized packets at regular intervals
 - Uses simplified addressing
 - No acknowledgement

Transaction layer

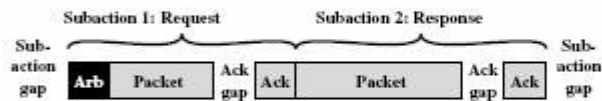
- Defines a request-response protocol that hides the lower-layer detail of FireWire from applications.



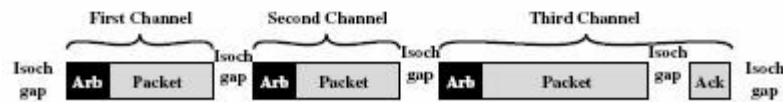
FireWire Protocol Stack



(a) Example asynchronous subaction



(b) Concatenated asynchronous subactions

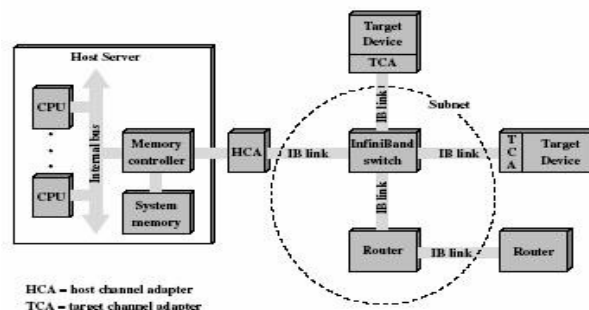


(c) Example isochronous subactions

FireWire Subactions

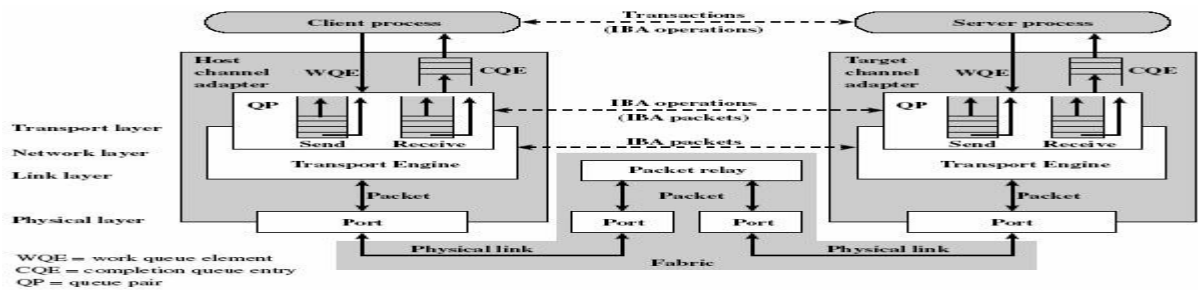
InfiniBand

- Recent I/O specification aimed at high-end server market
- First version released early 2001
- Standard for data flow between processors and intelligent I/O devices
- Intended to replace PCI bus in servers
- Greater capacity, increased expandability, enhanced flexibility
- Connect servers, remote storage, network devices to central fabric of switches and links
- Greater server density
- Independent nodes added as required
- I/O distance from server up to
 - o 17 meters using copper
 - o 300 meters using multimode optical fiber
 - o 10 kilometers using single-mode optical fiber
- Transmission rates up to 30 Gbps



InfiniBand Operations

- 16 logical channels (virtual lanes) per physical link
- One lane for fabric management – all other lanes for data transport
- Data sent as a stream of packets
- Virtual lane temporarily dedicated to the transfer from one end node to another
- Switch maps traffic from incoming lane to outgoing lane



8. TEXT BOOKS:

V. Carl Hamacher, Zvonko G. Varanescic and Safat G. Zaky, —Computer Organisation—, VI edition, McGraw-Hill Inc., 2012

David A. Patterson and John L. Hennessey, —Computer organization and design—, Morgan auffman lsevier, Fifth edition, 2014

9. APPLICATIONS

Real Life Application Of memory concepts Memory is the ability to encode, store, and retrieve a stimulus.

