



# Computer Organization and Software Systems

## CONTACT SESSION 1

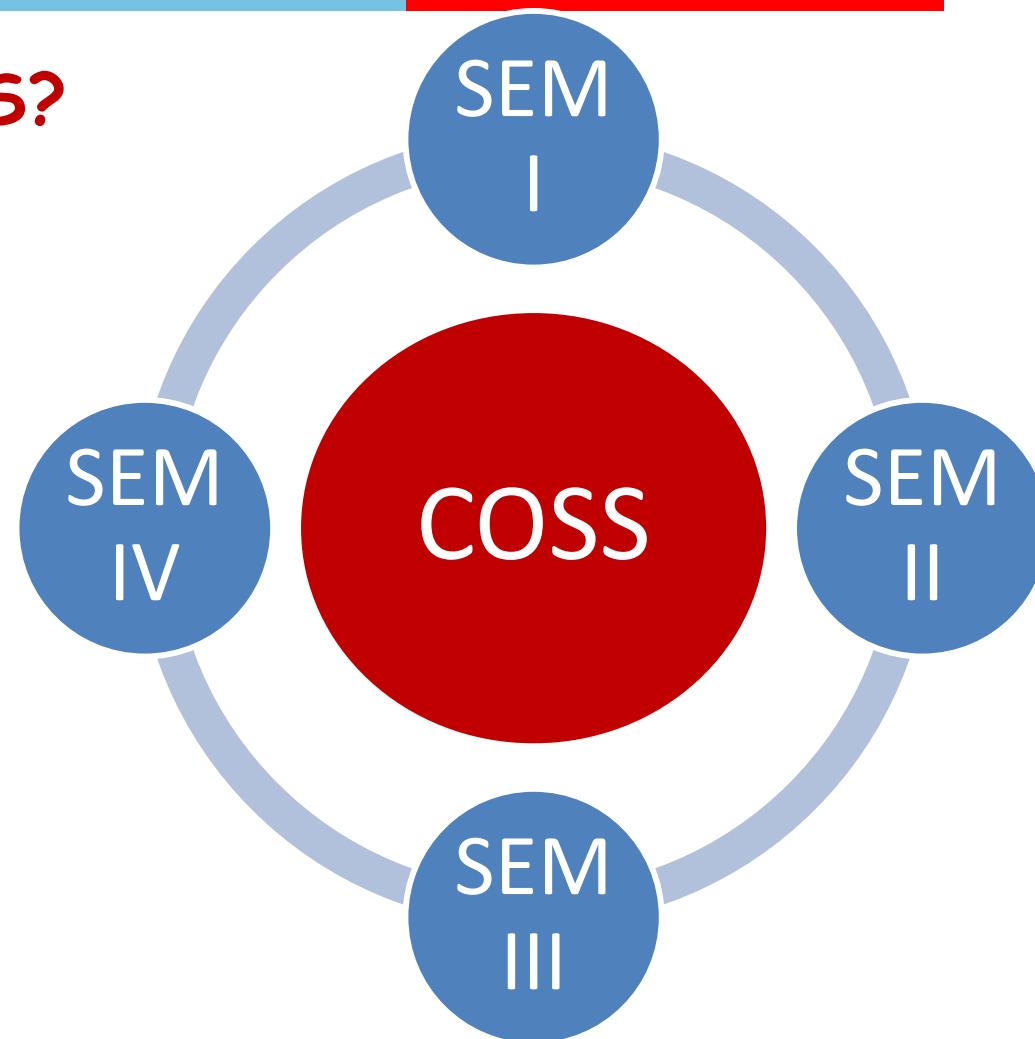
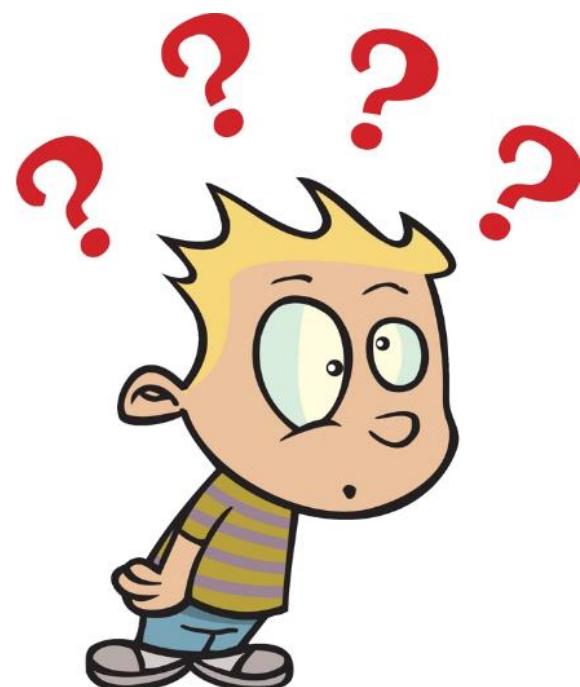
Dr. Lucy J. Gudino  
WILP & Department of CS & IS



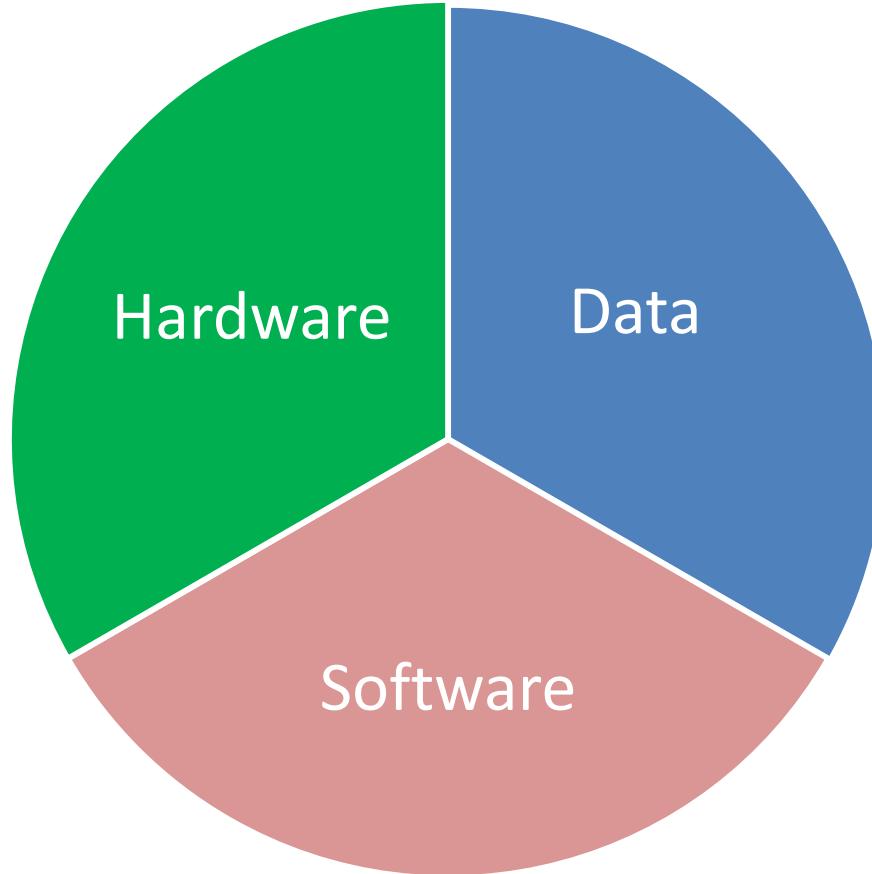
**BITS** Pilani  
Pilani Campus

# Introduction

Why Study COSS?



# Introduction



Data analytics: is the process of examining **data** sets in order to draw conclusions about the information they contain, increasingly with the aid of **specialized systems** and **software**.

# Text Books and Reference Books

## Text Books:

- (T1) W. Stallings, *Computer Organization & Architecture*, PHI, 10<sup>th</sup> ed., 2010.
- (T2) A Silberschatz, Abraham and others, *Operating Systems Concepts*, Wiley Student Edition, 8<sup>th</sup> Edition

## Reference Books:

- (R1) Patterson, David A & J L Hennenssy, *Computer Organization and Design – The Hardware/Software Interface*, Elsevier, 5th Ed., 2014.
- (R2) Randal E. Bryant, David R. O'Hallaron, *Computer Systems – A Programmer's Perspective*, Pearson, 3<sup>rd</sup> Ed, 2016.
- (R3) Tanenbaum, *Modern Operating Systems*: Pearson New International Edition, Pearson Education, 2013 (Pearson Online)
- (R4) Stallings, *Operating Systems: Internals and Design Principles* : International Edition, Pearson Education, 2013 (Pearson Online)

# Evaluation Scheme

5 unit course.

Sl No.	Evaluation Component	Duration	Weightage %	Nature of Component
1	Mid Sem Exam } f[2]	90 min	30% -	CB / OB
2	Comprehensive Examination } f[3]	180 min	40% -	OB
3	Quiz → BO 2 5 Q, A1, A2 } EC 1	-----	5% ----- 30M	OB
4	Assignments	---	25%	OB

B1  
A1  
ILM  
A2

# Assignments

- Two assignments:
  - One pre-midsem exam : 12%
  - One post-midsem : 13%
- Lab based
- Simulator to be used : CPU-OS simulator
  - Open source tool  
[https://drive.google.com/open?id=12YUK52RQ-JhPOddj6CD\\_oifW4sTMbsBI](https://drive.google.com/open?id=12YUK52RQ-JhPOddj6CD_oifW4sTMbsBI)
  - Virtual lab (Platify)

# Assignment should not be

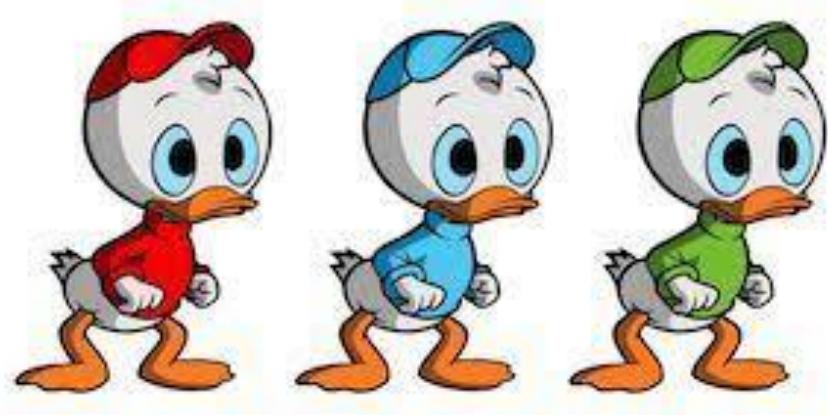
FILL IT.



SHUT IT.



FORGET IT.



CRACK!!

[lottoland.com.au](http://lottoland.com.au)



# General Instructions

- 
1. Always use note book for writing important points and for solving problems
  2. Use chat box for writing subject related questions
  3. Do not repeat the questions on chat box. Questions will be answered during last 10 minutes of the session
  4. Unanswered questions will be put up on the canvas forum

Contact details: [krpruthvi@wilp.bits-pilani.ac.in](mailto:krpruthvi@wilp.bits-pilani.ac.in)

# Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
1-2	<p><b>Introduction to Computer Systems</b></p> <ul style="list-style-type: none"> <li>• Hardware Organization of a computer</li> <li>• Running a Hello Program</li> <li>• Instruction Cycle State Diagram</li> <li>• Operating System role in Managing Hardware</li> </ul>	T1

# Definition of a Computer

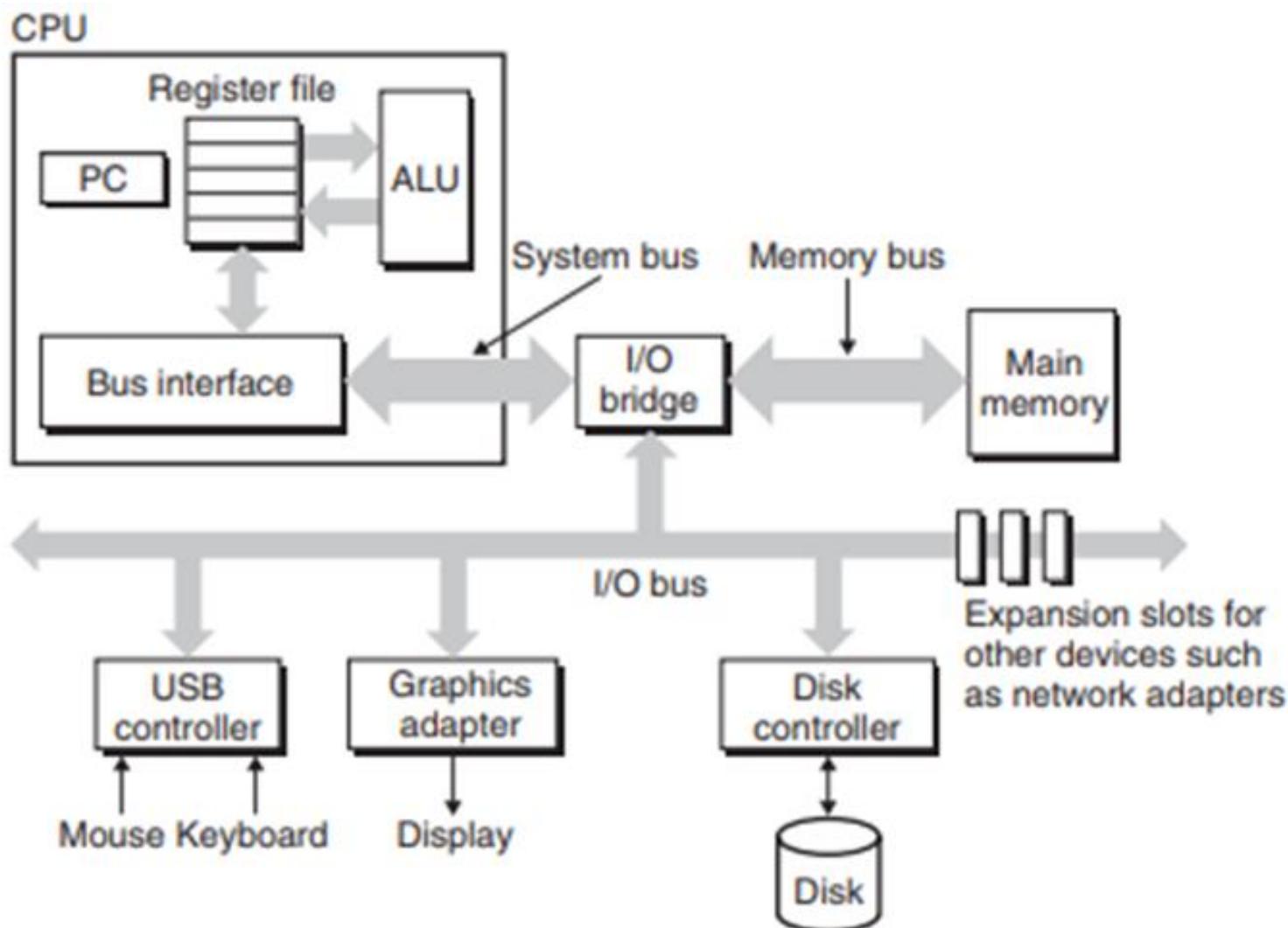
- Is a complex system
- Is a programmable device
- Must be able to **process** data
- Must be able to **store** data
- Must be able to **move** data
- Must be able to **control** above three functions



# Computer System

- Hardware
  - Central Processing Unit (CPU)
  - Memory
  - I/O devices
- Software
  - System Software
    - System Management Software
    - Tools and Utilities for Developing the software
  - Application Software
    - General Purpose Software
    - Specific Purposed Software

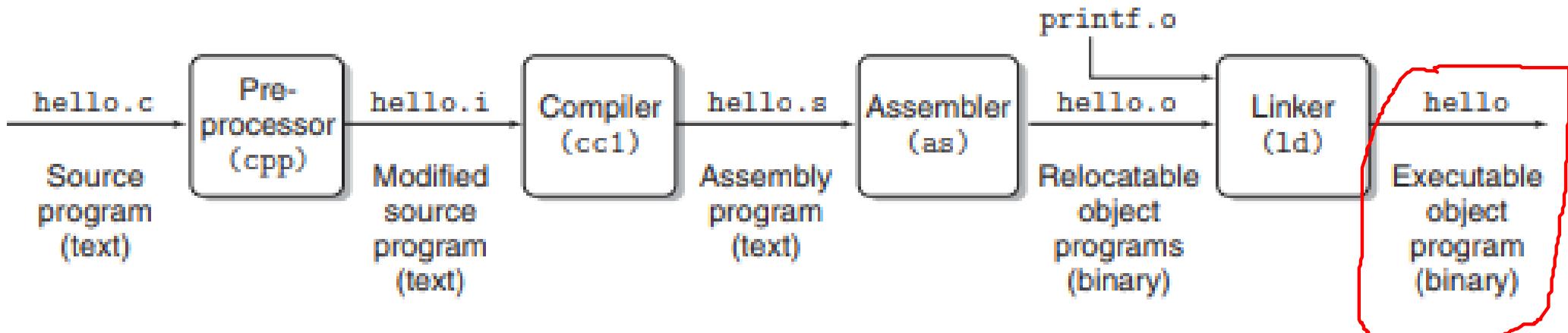
# Hardware Organization of a computer



# Running a Hello.c Program

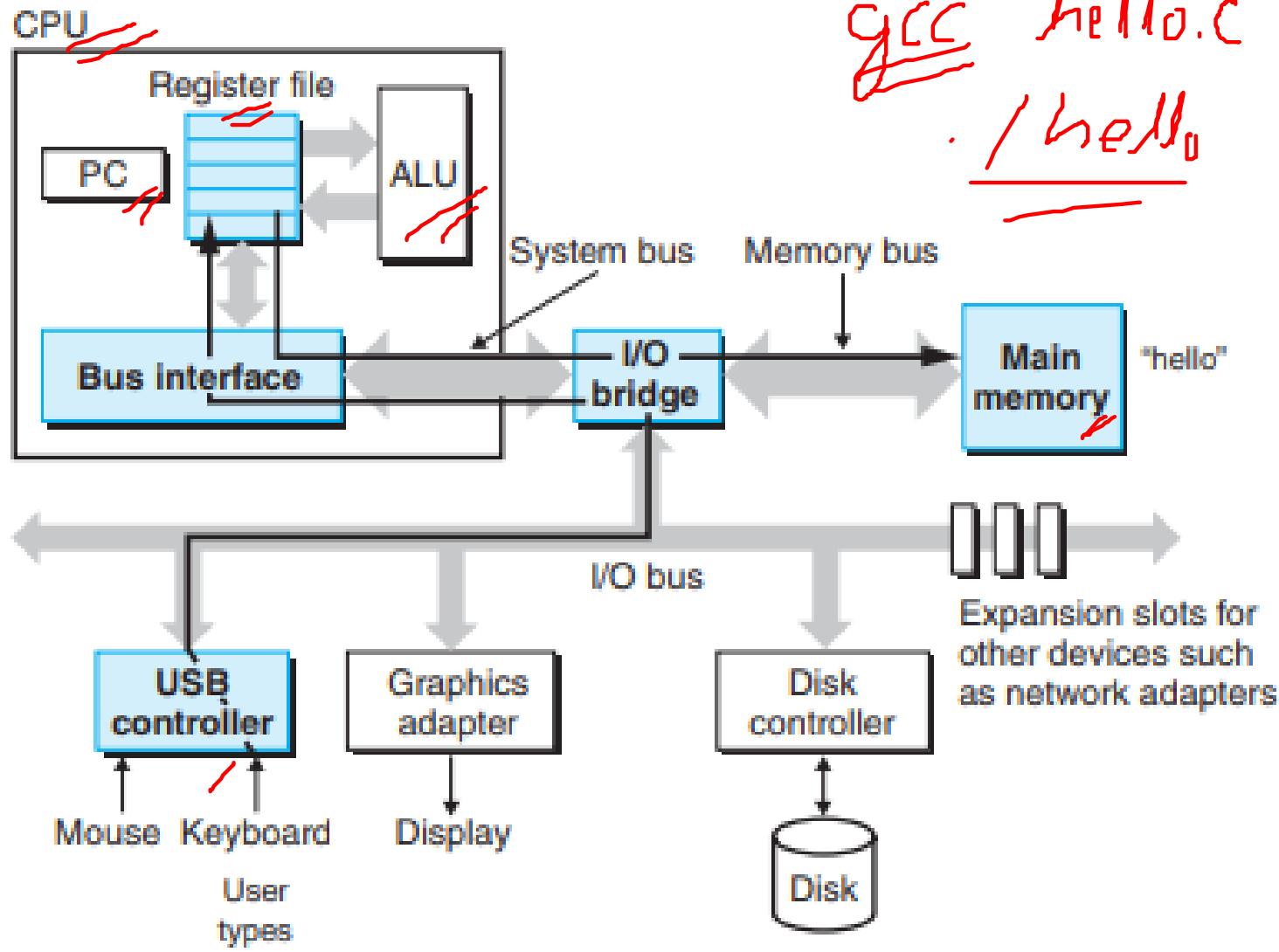
```
#include <stdio.h>

int main()
{
    printf("hello, world\n");
}
```



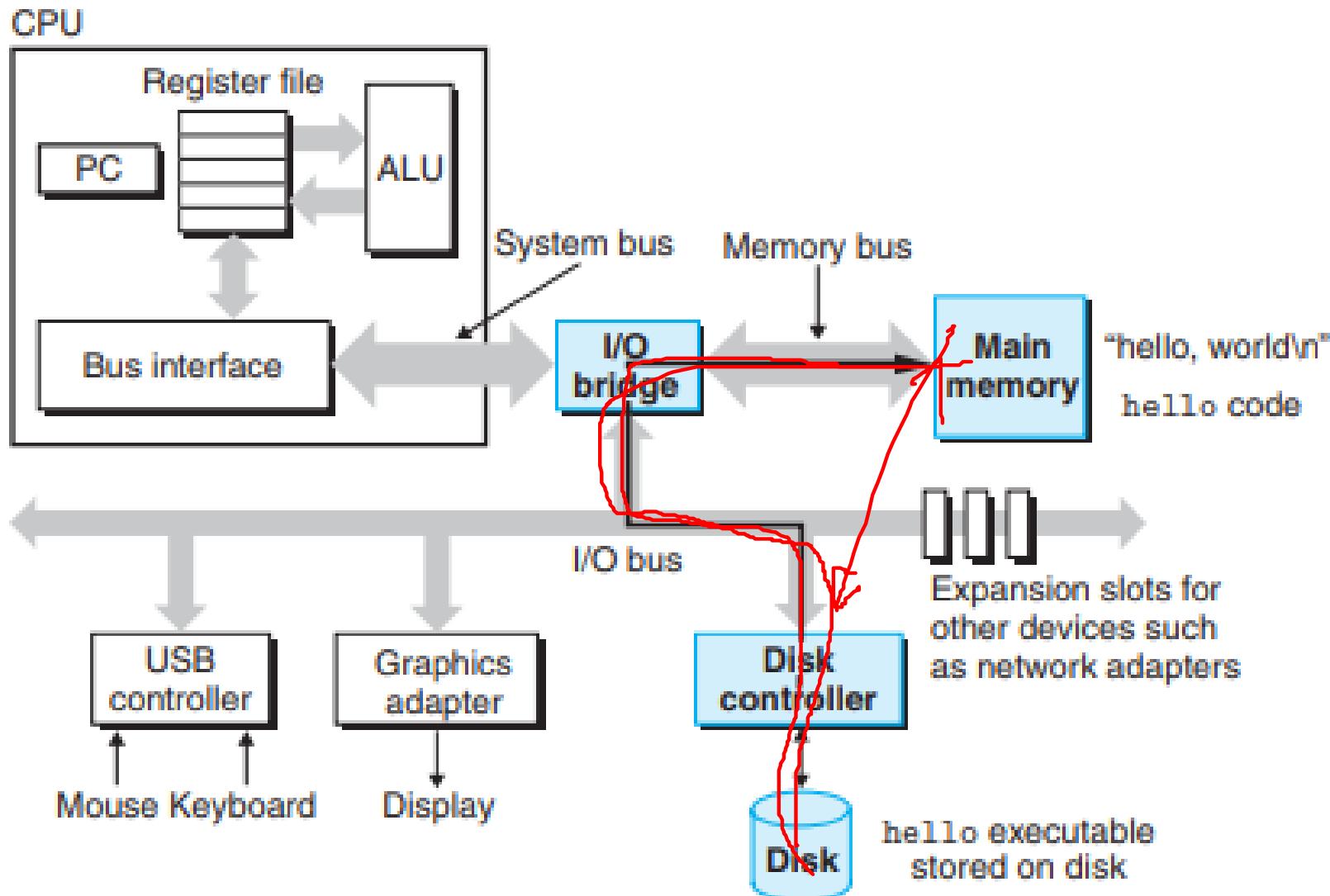
The compilation system.

# Reading ./hello command from Keyboard

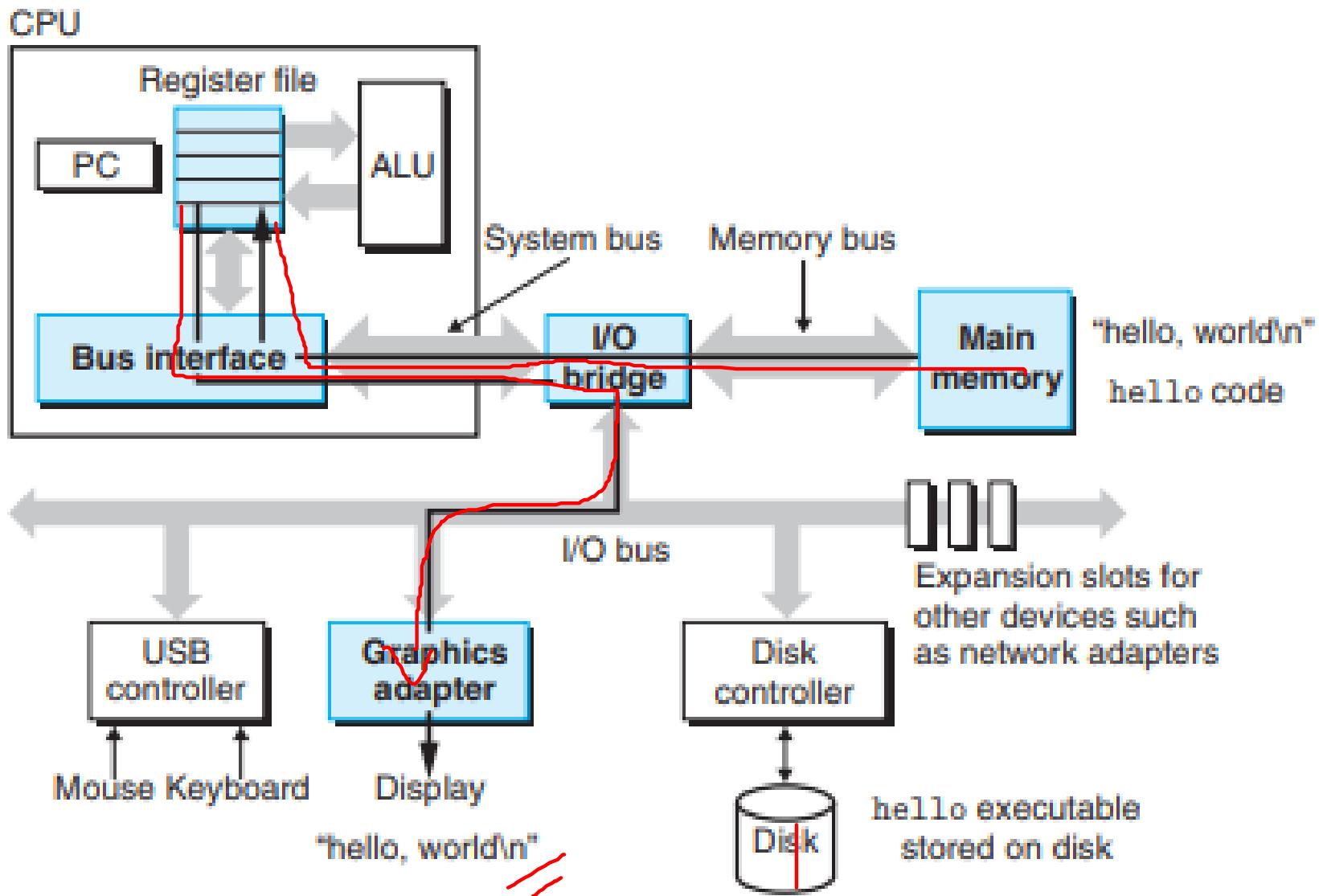


~~gcc hello.c -o hello~~  
./hello

# Loading the executable from disk into main memory



# Writing the output string from memory to the display



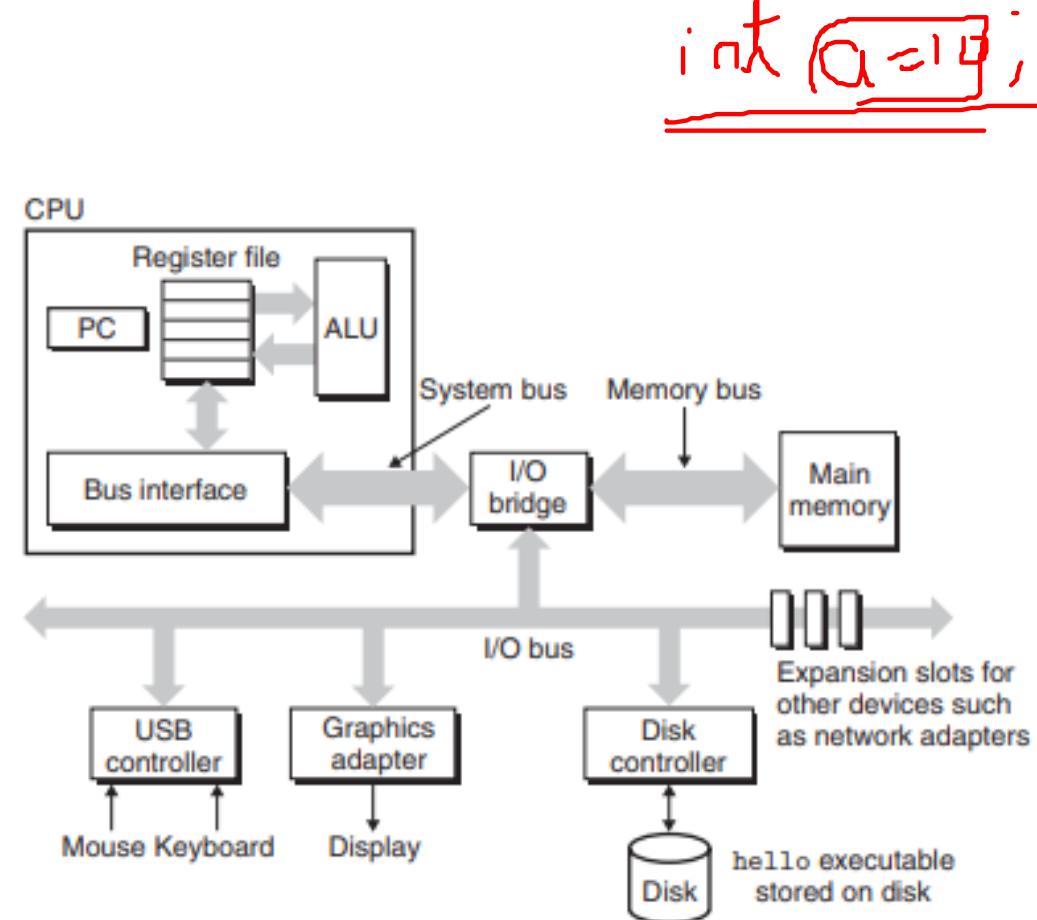
# Why do we need to know how compilation works?

---

- Optimizing program performance.
- Understanding link-time errors.
- Avoiding security holes.

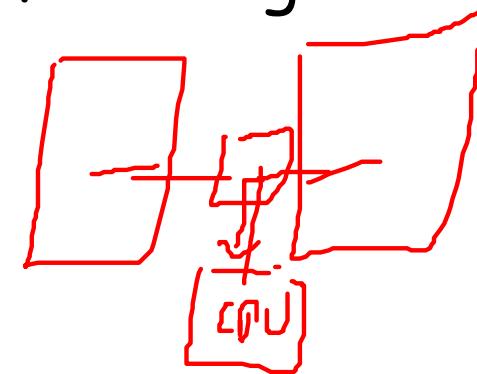
# Von Neumann Architecture

- Three key concepts:
  - Data and instructions are stored in a single read - write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next



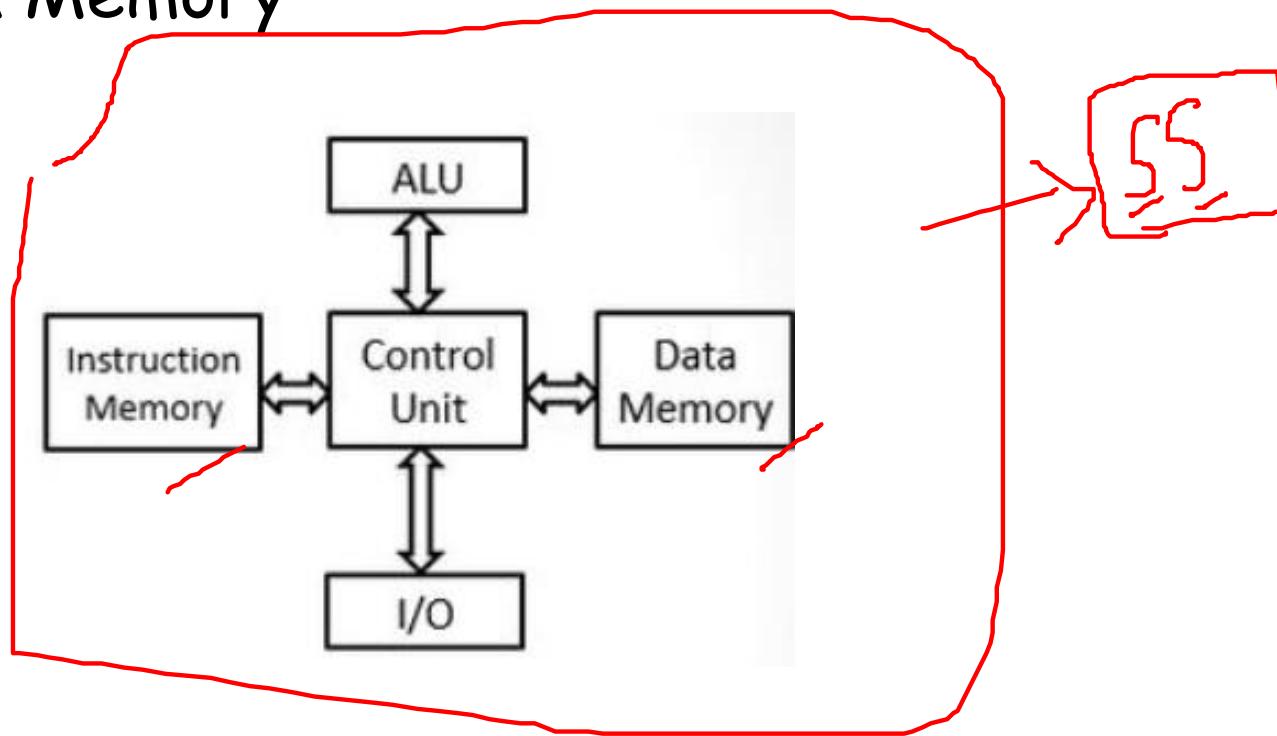
# Von Neumann Architecture...

- Stored-program computers have the following characteristics:
  - Three hardware systems:
    - A central processing unit (CPU)
    - A main memory system
    - An I/O system
  - The capacity to carry out sequential instruction processing.
  - A single path between the CPU and main memory.
    - This single path is known as the von Neumann bottleneck.
    - Side effect : reduced throughput (Data Rate)



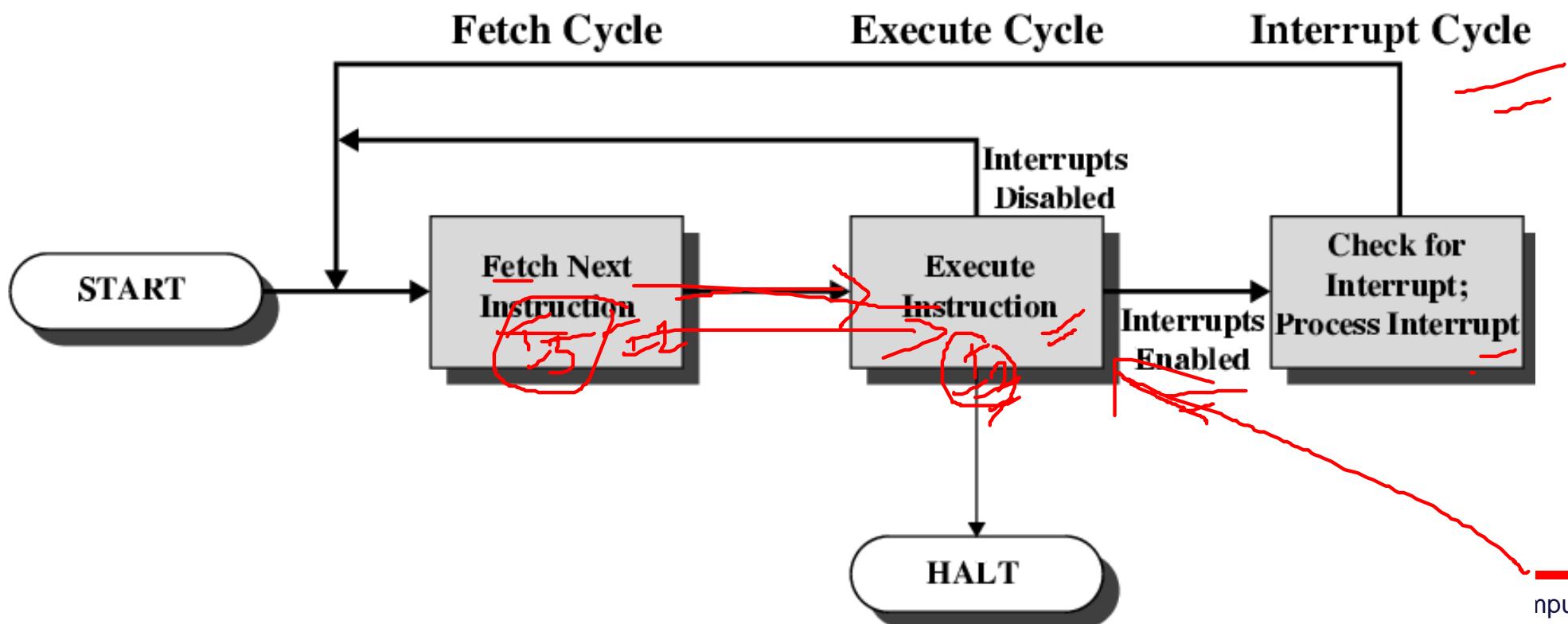
# Harvard Architecture

- Uses two memory systems and two separate busses
  - Instruction Memory
  - Data Memory



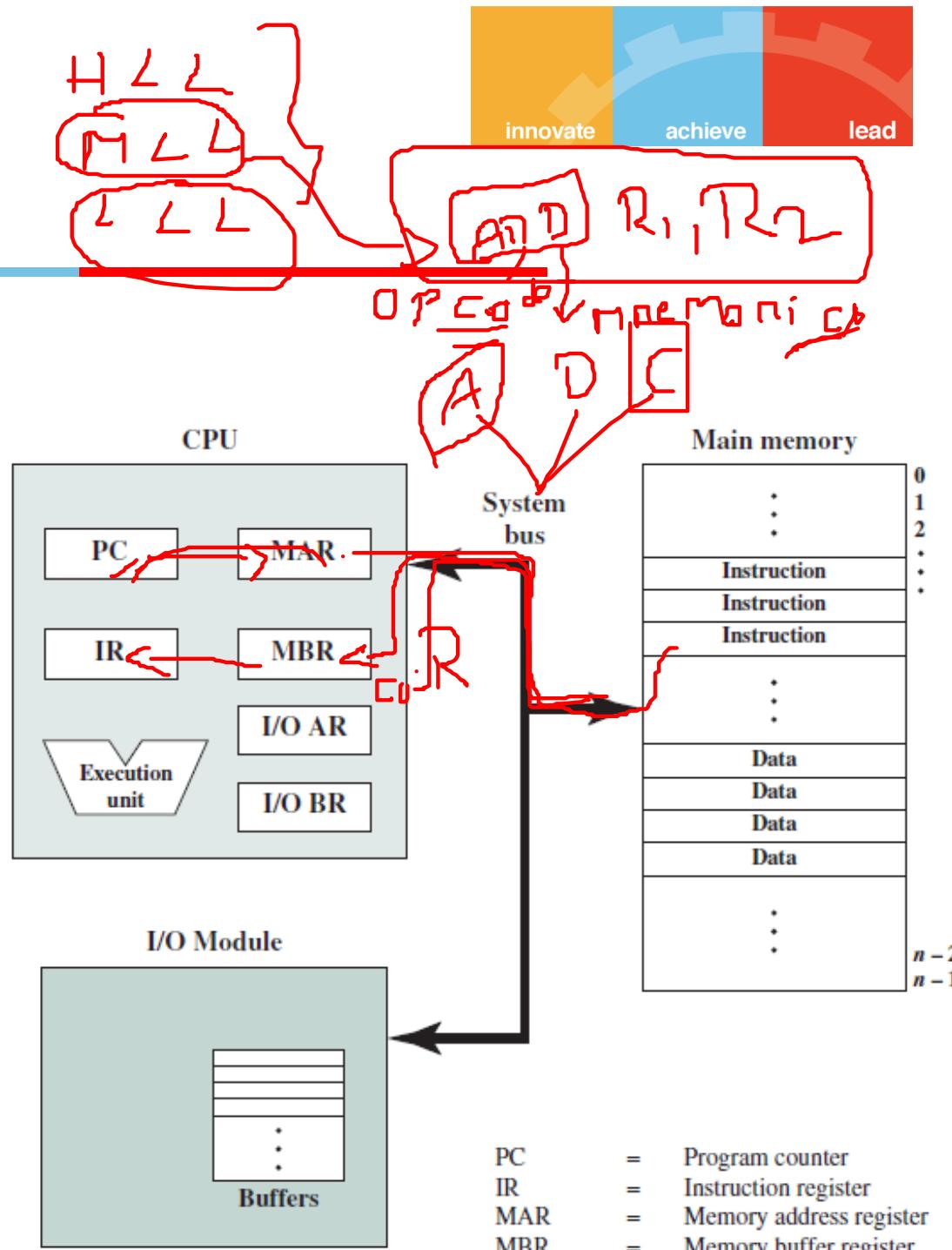
# Instruction Cycle Diagram

- Instruction execution : Two steps:
  - Fetch
  - Execute
- Interrupt: Interrupt is checked at the end of Instruction cycle



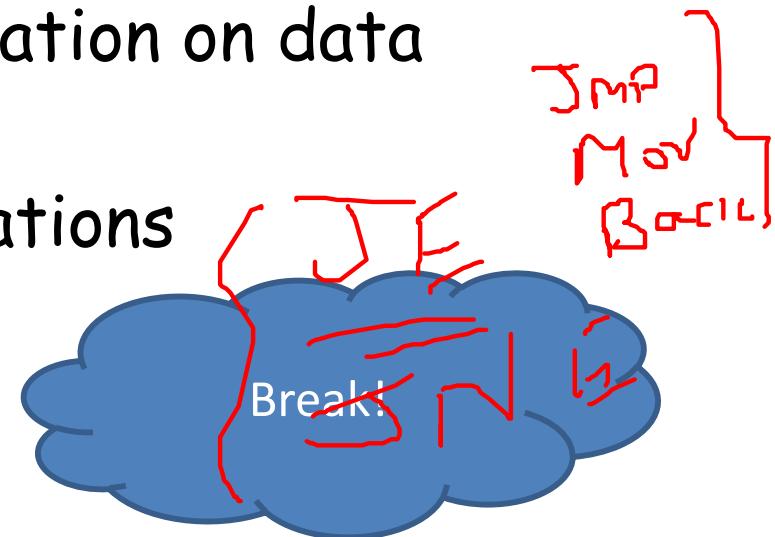
# Fetch Cycle

- Program Counter (PC) holds address of next instruction to be fetched
- Processor fetches instruction from memory location pointed to by PC
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions during execution cycle
- Increment PC
  - Unless told otherwise



# Execute Cycle

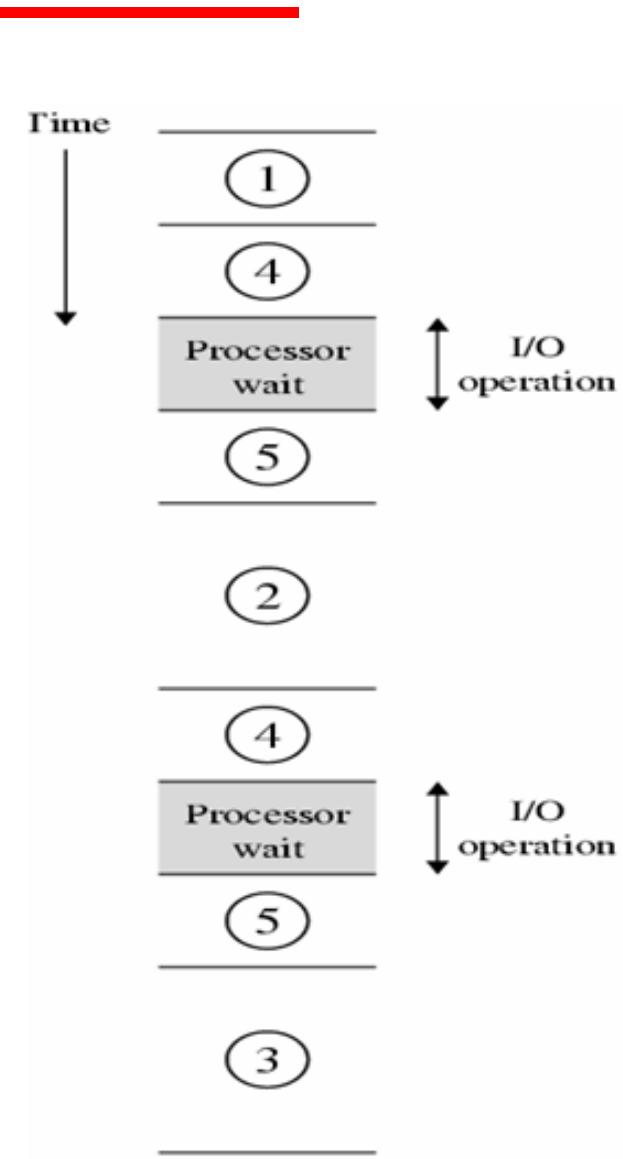
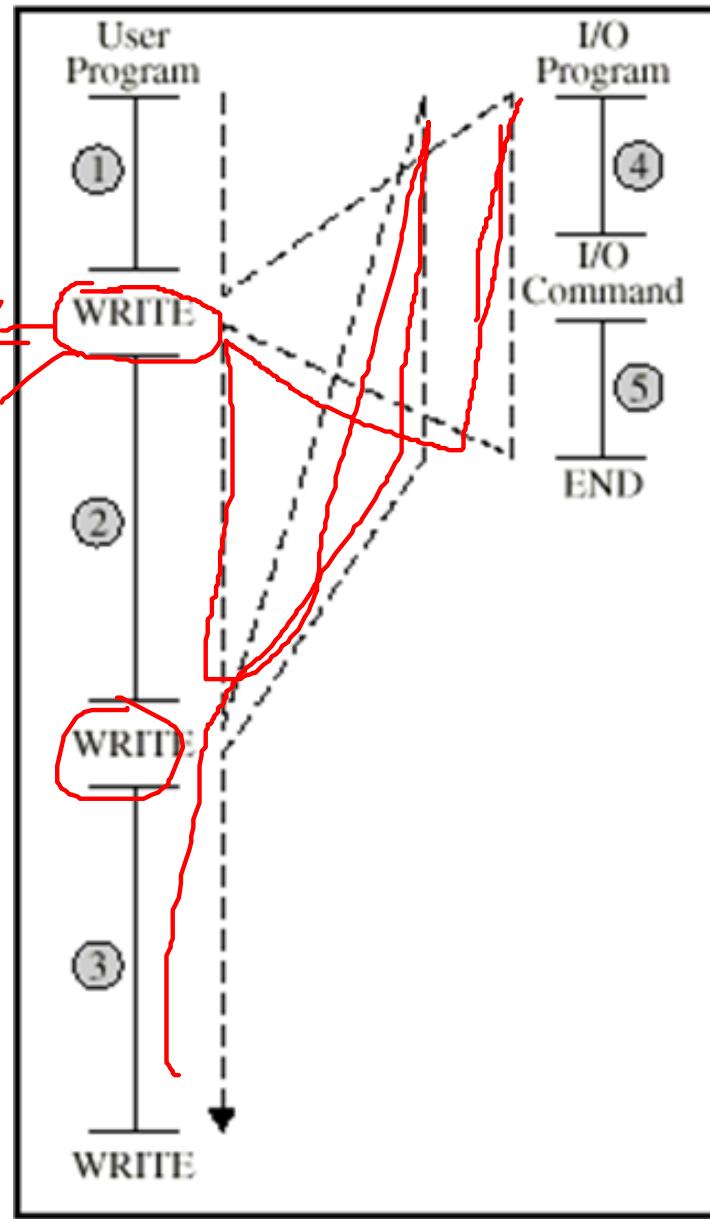
- Processor - memory
  - Data transfer between CPU and main memory
- Processor - I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above



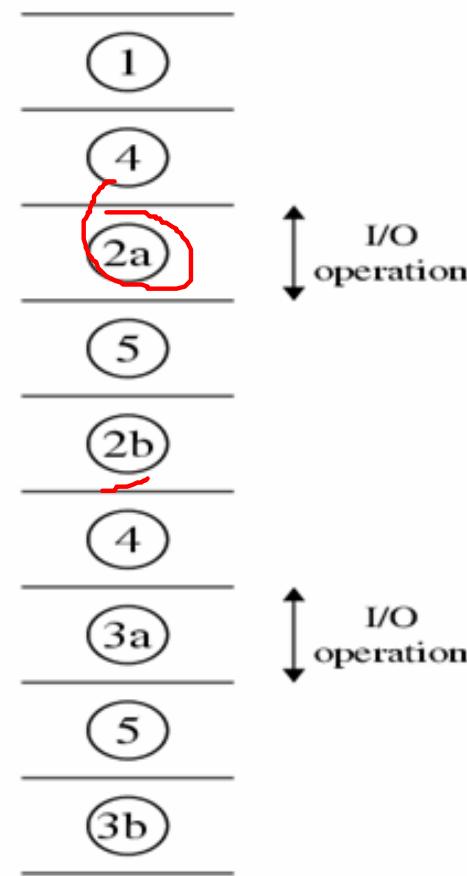
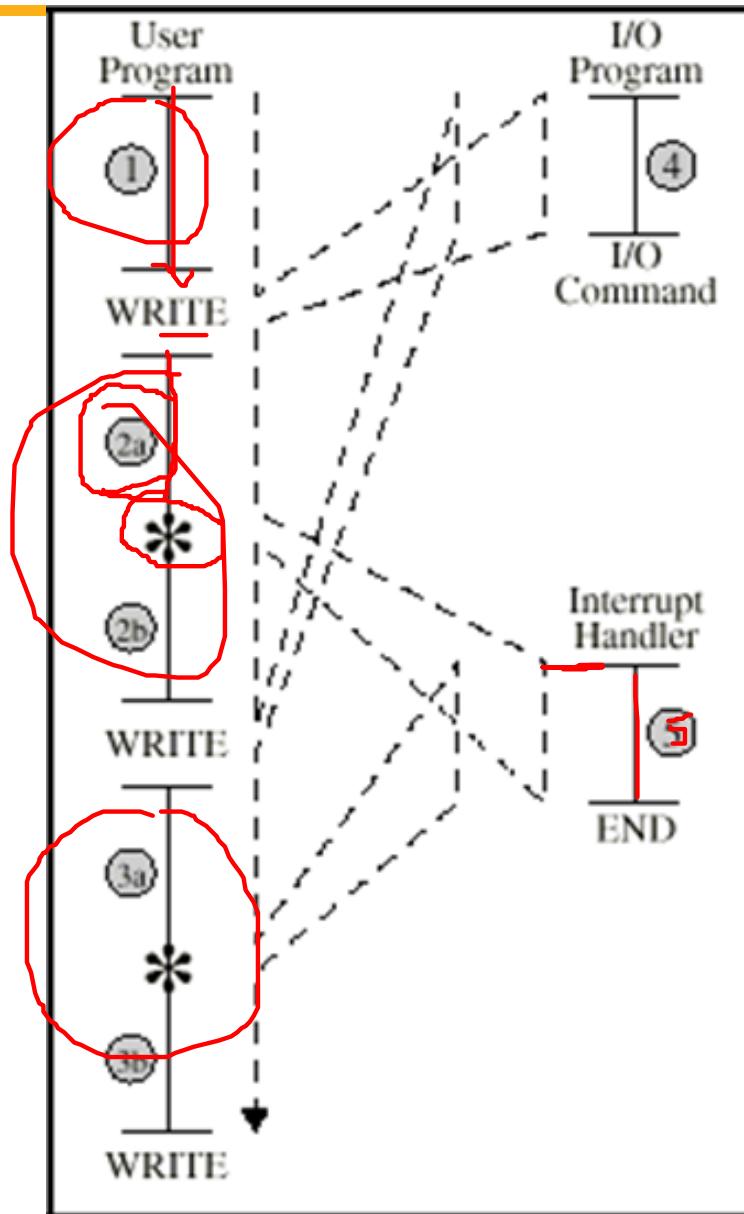
# Interrupt Cycle

- Interrupts: Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Interrupts enhances processing efficiency

# Program Flow Control (No Interrupts)



# Program Flow Control (With Interrupts)



# Contd...

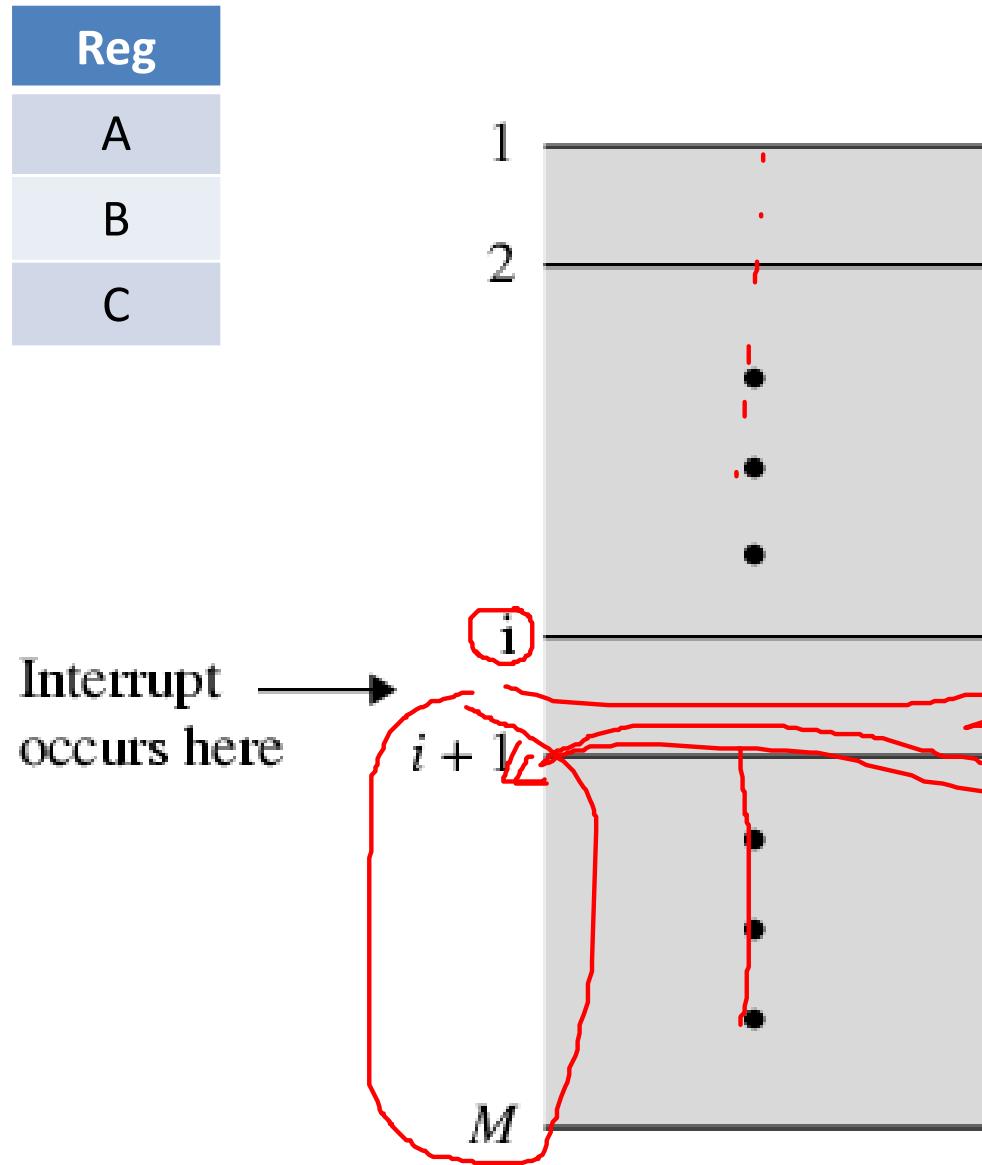
- Classes of interrupts:
  - Program
    - e.g. overflow, division by zero
  - Timer
    - Generated by internal processor timer
    - Used in pre-emptive multi-tasking
  - I/O
    - from I/O controller
  - Hardware failure
    - e.g. memory parity error

# Interrupt Cycle

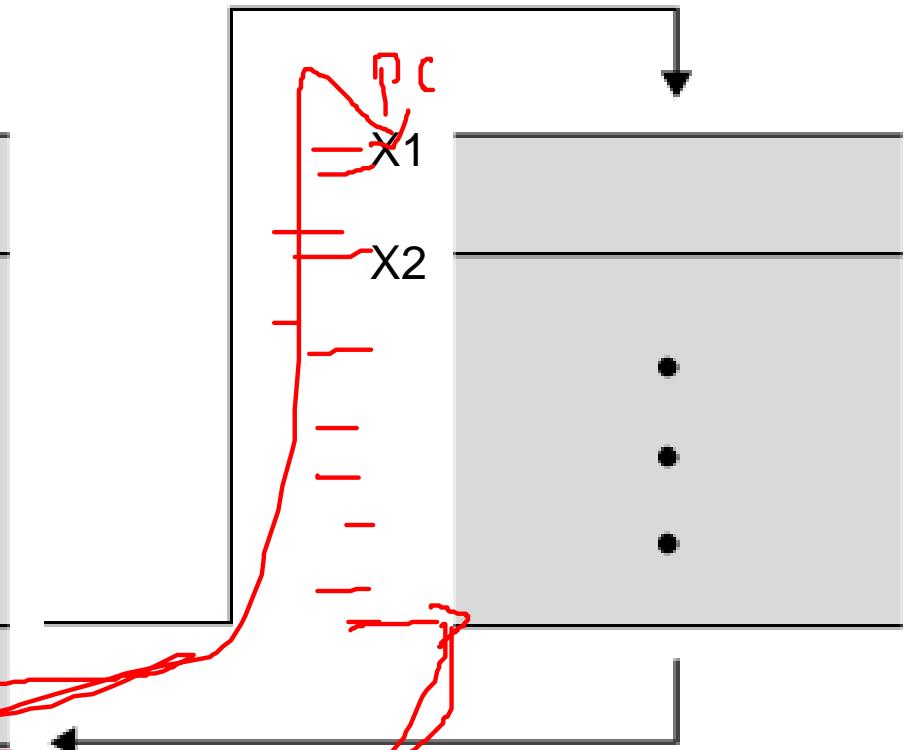
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

# Transfer of Control via Interrupts

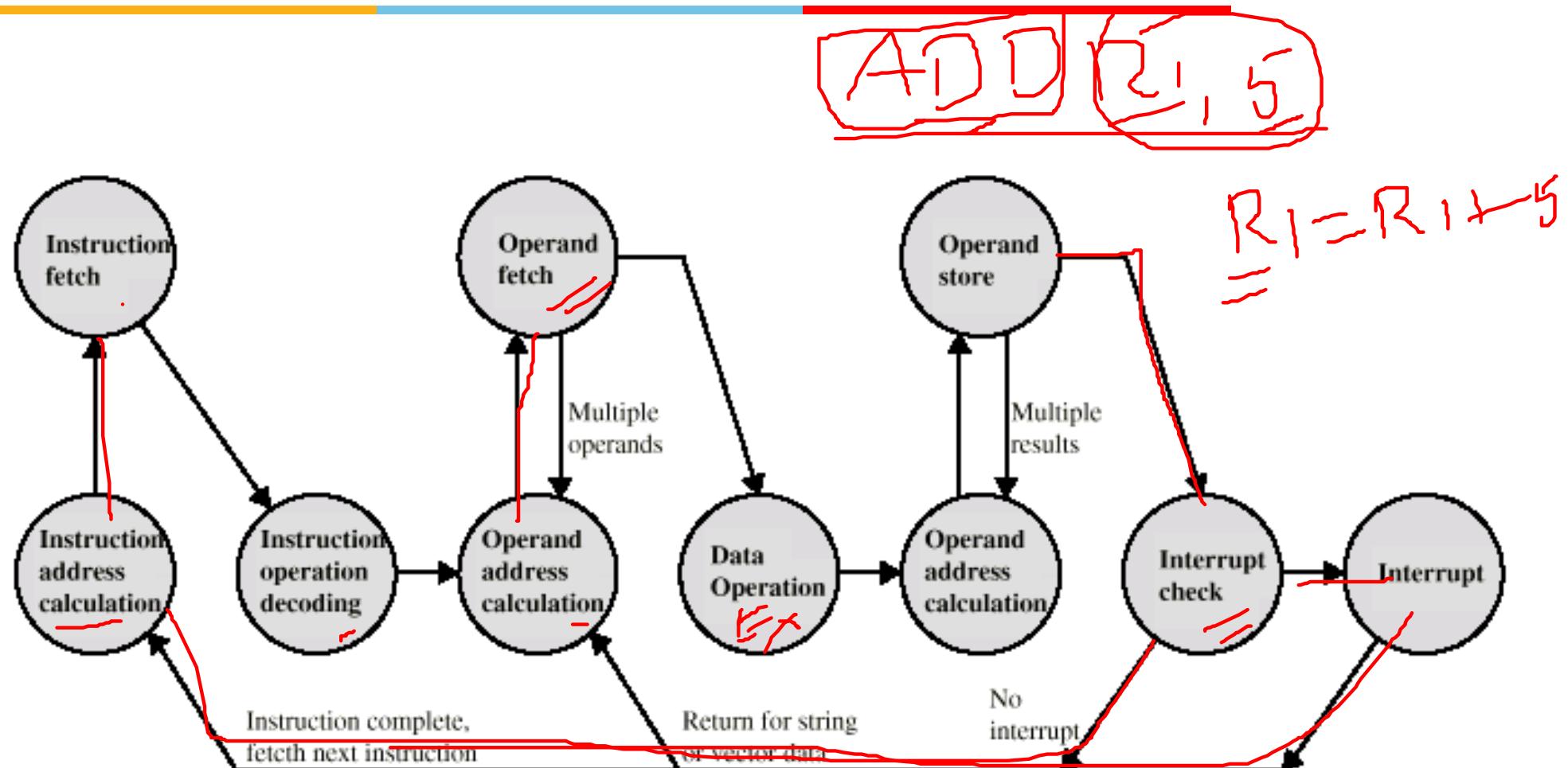
User Program



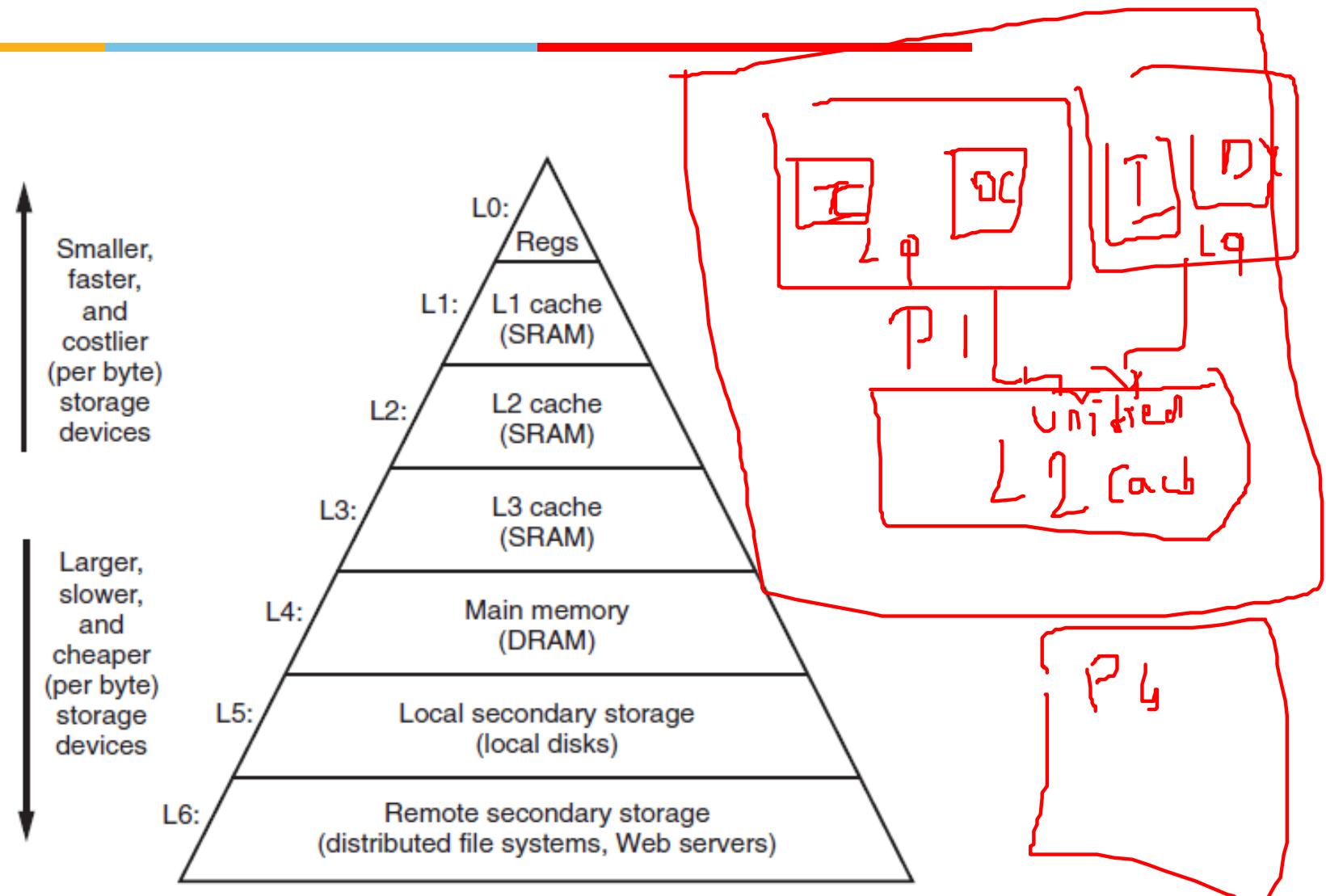
Interrupt Handler



# Instruction Cycle - State Diagram



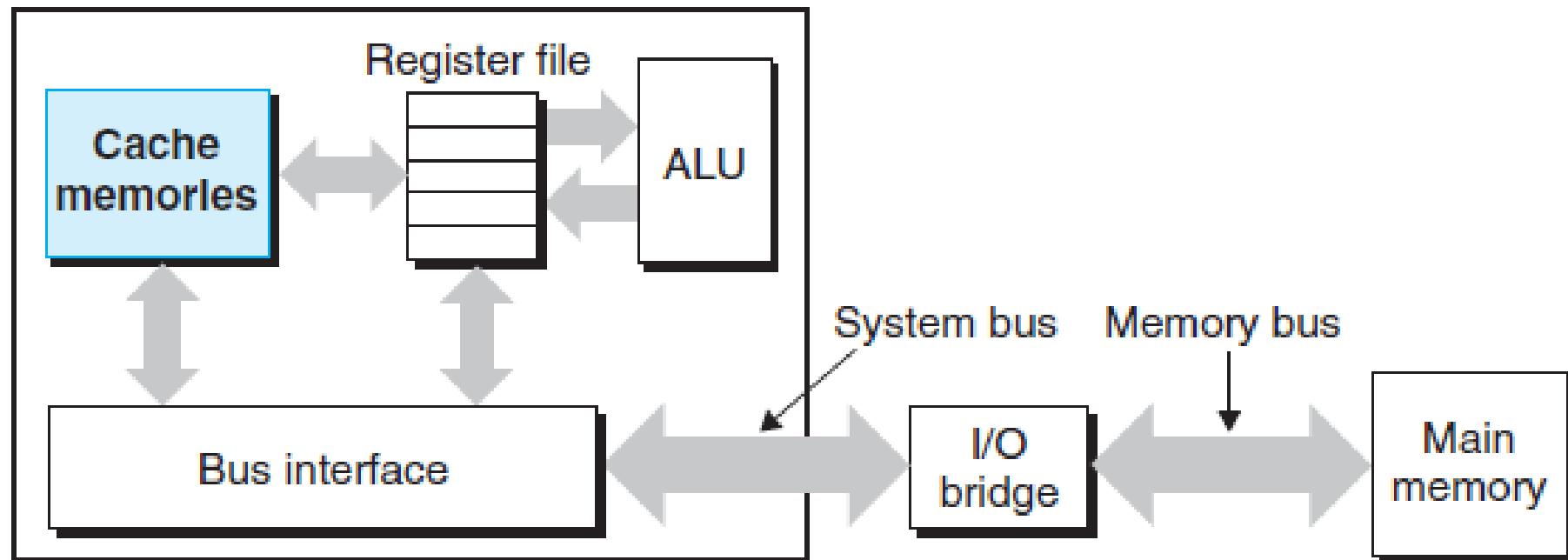
# Memory Hierarchy



An example of a memory hierarchy.

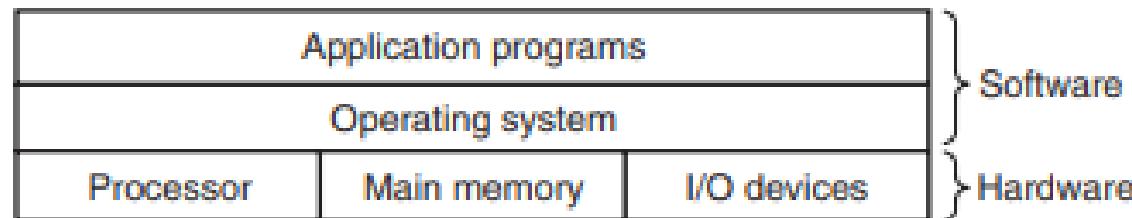
# Role of Cache Memory

CPU chip



# Operating System

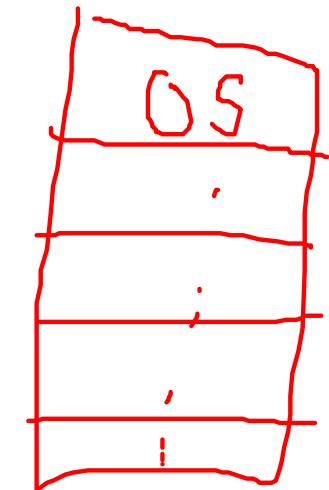
- collection of software/ Program that acts as an intermediary between an user of a computer and the computer hardware.
- is a program that helps to run all the other programs
- Three main functions:
  - Resource management
  - Establish an user interface
  - Execute and provide services for application software



Layered view of a  
computer system.

# Main objectives

- Convenience
- Efficiency
- Ability to evolve and offer new services
- Maximize System performance
- Protection and access control
- Footprint of OS should be small



# Important Note

---

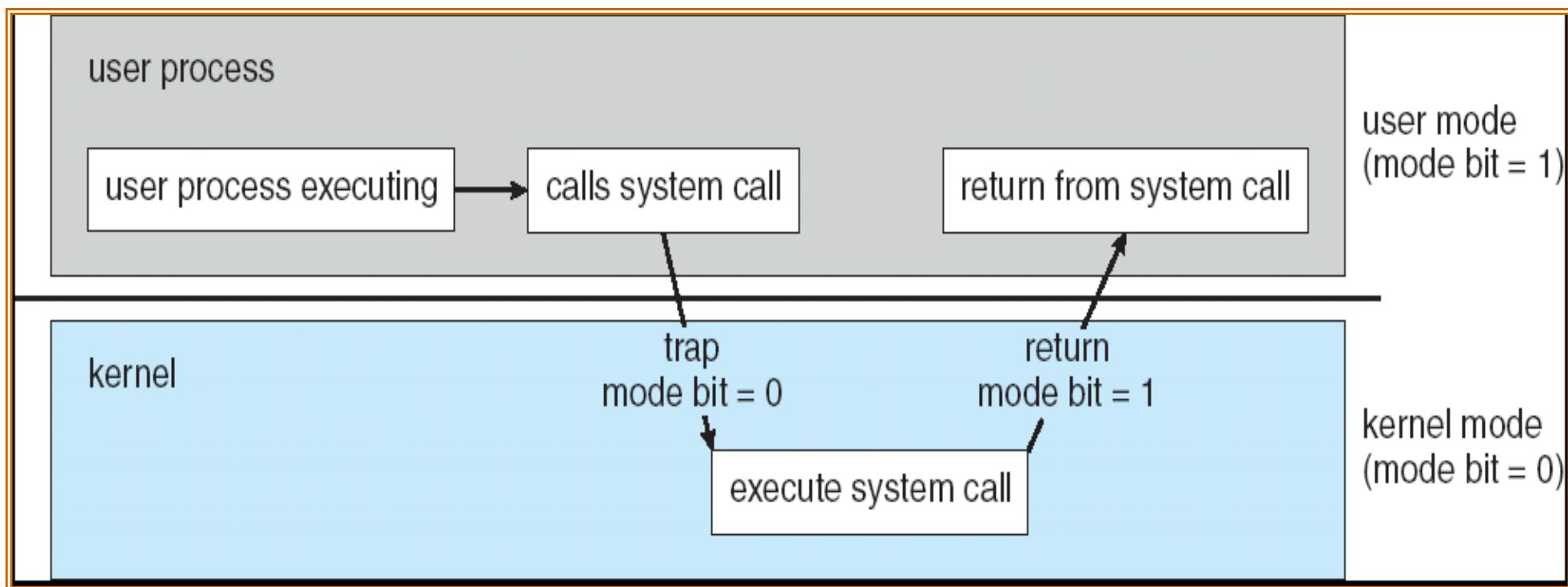
- "The one program running at all times on the computer" is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

# Operating System Operations

- Dual-mode operation
  - User mode
  - Kernel mode ( also known as System Mode / Supervisor mode/ privileged mode )
- User mode(1):
  - user program executes in user mode
  - certain areas of memory are protected from user access
  - certain privileged instructions may not be executed
- Kernel Mode (0)
  - privileged instructions may be executed
  - protected areas of memory may be accessed

Printf()  
{ can't }  
}

# Transition from user to kernel mode





# Lab Activity

Screenshot of a web browser showing the BITS-Pilani Virtual Lab interface.

The browser tabs are:

- DH News: Latest & Breaking News, L... (closed)
- Inbox (27,947) - lucy.gudino@pil... (closed)
- e-Learning Portal (closed)
- Wilp CS-IS Lab

The address bar shows: Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

The main content area displays the "BITS - Pilani Virtual Lab" logo and a sidebar with navigation icons:

- Change Lab (monitor icon)
- View Slots (house icon)
- Book Slot (calendar icon)
- Resources (book icon)

The main content area shows the "Resources" section under "Computer Organization And Software Systems".

Breadcrumbs: Home / Computer Organization And Software Systems

Available resources:

- LabCapsule1-Introduction To CPU-OS Simulator
- LabCapsule2-Instruction Set Of CPU-OS Simulator
- LabCapsule3-Cache Memory
- LabCapsule4-Pipeline
- LabCapsule5-Process Management
- LabCapsule6-Multithreading
- LabCapsule7-Synchronization
- LabCapsule8-Deadlock

A message bubble from "Customer Support" says: "Welcome to our site, if you need help simply reply to this message, we are online and ready to help." (just now)

A text input field says: "Type here and press enter.."

A teal speech bubble icon in the bottom right corner has a red notification badge with the number 1.

# Contd...

Screenshot of a web browser showing the BITS-Pilani Virtual Lab interface.

The browser tabs are:

- DH News: Latest & Breaking News, L... (closed)
- Inbox (27,947) - lucy.gudino@pil... (closed)
- e-Learning Portal (closed)
- Wilp CS-IS Lab

The address bar shows: Not secure | bitscsis.vlabs.platifi.com/index1.html#!/resources

The main content area displays:

## BITS - Pilani Virtual Lab

### Resources

Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator

On the left sidebar, the "Resources" option is selected, indicated by a teal background.

- Lab Sheet 1.1
- Lab Sheet 1.2
- Lab Sheet 1.3
- Lab Sheet 1.4

The bottom right corner shows a teal circular notification icon with a red '1'.

The taskbar at the bottom includes icons for File Explorer, Task View, Edge, Google Chrome, File Manager, Task Scheduler, Word, and PowerPoint.

The system tray shows the date and time: 08:37 28-02-2021, along with icons for battery, signal, and language (ENG).

# Contd...

Screenshot of a web browser showing the BITS-Pilani Virtual Lab interface.

The browser tabs are:

- DH News: Latest & Breaking News, L x
- Inbox (27,947) - lucy.gudino@pil... x
- e-Learning Portal x
- Wilp CS-IS Lab x

The main content area shows the "BITS - Pilani Virtual Lab" homepage. The sidebar on the left includes:

- Change Lab (Icon: Monitor)
- View Slots (Icon: House)
- Book Slot (Icon: Calendar)
- Resources (Icon: Book)

The main content area displays the following navigation path:

Home / Computer Organization And Software Systems / LabCapsule1-Introduction To CPU-OS Simulator / Lab Sheet 1.1

Two resources are listed:

- Lab Sheet 1.1\_Introducti... (Icon: Document)
- Lab Video (Icon: Camera)

The bottom taskbar shows various application icons and system status:

- Windows Start button
- File Explorer
- Edge browser
- Google Chrome
- File Manager
- Calculator
- OneDrive
- Word document
- PowerPoint document
- System tray icons for battery, signal, and volume
- Language: ENG
- Date and Time: 08:37 28-02-2021
- A notification bubble in the bottom right corner indicates 1 unread message.



# Computer Organization and Software Systems

## CONTACT SESSION 2

Dr. Lucy J. Gudino  
WILP & Department of CS & IS



**BITS** Pilani  
Pilani Campus

# Today's Class

Contact Hour	List of Topic Title	Text/Ref Book/external resource
3	<b>Performance Assessment</b> MIPS Rate Amdahl's Law	Class Slides
4	<b>Memory Organization</b> Storage Technologies Random Access Memory Disk Storage Solid State Disks Storage Technology Trends	T1, R2



# Performance Assessment

**BITS Pilani**  
Pilani Campus

# Units

- Kilo- (K) = 1 thousand =  $10^3$  and  $2^{10}$
- Mega- (M) = 1 million =  $10^6$  and  $2^{20}$
- Giga- (G) = 1 billion =  $10^9$  and  $2^{30}$
- Tera- (T) = 1 trillion =  $10^{12}$  and  $2^{40}$
- Peta- (P) = 1 quadrillion =  $10^{15}$  and  $2^{50}$
- Exa - (E) = 1 quintillion =  $10^{18}$  and  $2^{60}$

Byte = a unit of storage

- $1KB = 2^{10} = \underline{1024 \text{ Bytes}} \Rightarrow 1024B$
- $1MB = 2^{20} = 1,048,576 \text{ Bytes}$
- Main memory (RAM) is measured in MB / GB
- Disk storage is measured in GB for small systems, TB for large systems.

# Examples

Hertz = clock cycles per second (frequency)

- 1MHz = 1,000,000Hz
- Processor speeds are measured in MHz or GHz.

# Units...

- Milli- (m) = 1 thousandth =  $10^{-3}$
- Micro- ( $\mu$ ) = 1 millionth =  $10^{-6}$
- Nano- (n) = 1 billionth =  $10^{-9}$
- Pico- (p) = 1 trillionth =  $10^{-12}$
- Femto- (f) = 1 quadrillionth =  $10^{-15}$



# Examples

- Millisecond = 1 thousandth of a second
  - Hard disk drive access times are often 10 to 20 milliseconds.
- Nanosecond = 1 billionth of a second
  - Main memory access times are often 50 to 70 nanoseconds.
- Micron (micrometer) = 1 millionth of a meter
  - Circuits on computer chips are measured in microns.

# Important Terms

- **Execution time** : The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution
- **Throughput or bandwidth** : number of tasks completed per unit time.

# Example

What changes to a computer system will increase throughput, decrease execution time, or both?

1. Replacing the processor in a computer with a faster version
2. Adding additional processors of same type to a system, that is, it uses multiple processors for separate tasks

# Contd...

- Relationship between Performance and execution time of Computer X

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

- if the performance of X is greater than the performance of Y, we have

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

$$\frac{1}{\text{Execution time}_x} > \frac{1}{\text{Execution time}_y}$$

$$\text{Execution time}_y > \text{Execution time}_x$$

# Contd...

- Quantitative performance analysis
  - Computer X is "n" times faster than Computer Y

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

$$\frac{1}{E_X} > \frac{1}{E_Y}$$

$$\frac{E_Y}{E_X}$$

(1)

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

- If performance of X is  $n$  times better than Y, then the execution time on Y is  $n$  times longer than it is on X

# Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B? //

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

$$= \frac{15}{10} \Rightarrow 1.5$$

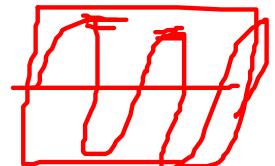
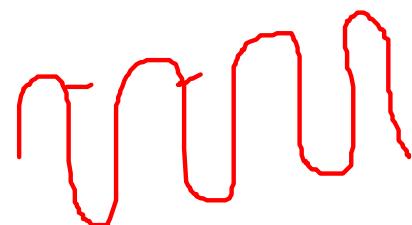
- Computer A is therefore 1.5 times faster than B.

# CPU performance and its factors

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

- CPU execution time for a program:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \underline{\text{Clock cycle time}}$$



$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\underline{\text{Clock rate}}}$$

# Example

- Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

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

## Computer A

Execution TimeA = 10s

Clock RateA =  $2 \times 10^9$  Hz

CPU Clock CycleA = ?

$$10s = \frac{CC_A}{2 \times 10^9}$$

$$2 \times 10^9 = CC_A$$

## Computer B

Execution TimeB = 6s

CPU Clock CyclesB = 1.2xClock CycleA

Clock Rate B = ?

$$G = \frac{1.2 \times CC_A}{CR_B}$$

$$\begin{aligned} CR_B &= \frac{1.2 \times 2 \times 10^9}{6} \\ &= 0.2 \times 10^9 \\ &= 4.0 \times 10^8 \end{aligned}$$

Let's first find the number of clock cycles required for the program on A:

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

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU time for B can be found using this equation:

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4 \text{ GHz}$$

# Instruction Performance

- CPI: Clock cycles Per Instruction
  - Average number of clock cycles per instruction for a program or program fragment.

$$\text{CPU clock cycles} = \underline{\text{Instructions for a program}} \times \underline{\text{Average clock cycles per instruction}}$$

# Example

---

Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

# Solution

Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

- The number of processor clock cycles for each computer

$$\text{CPU clock cycles}_A = \underline{I} \times 2.0$$

$$\text{CPU clock cycles}_B = \underline{I} \times 1.2$$

- Execution time for each computer

$$\text{Execution time} = \text{CPU clock cycles} \times \text{Clock cycle time}$$

$$\text{Execution time}_A = I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

$$\text{Execution time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

- Comparison:

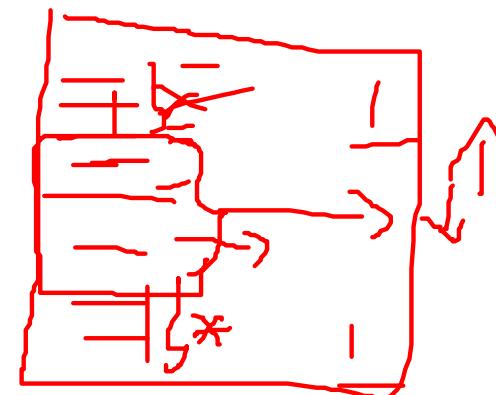
$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \cancel{I} \text{ ps}}{500 \cancel{I} \text{ ps}}$$

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \cancel{I} \text{ ps}}{500 \cancel{I} \text{ ps}} = \underline{\underline{1.2}}$$

# Amdahl's Law

- proposed by Gene Amdahl in 1967
- deals with the potential speedup of a program using multiple processors compared to a single processor

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$



# Amdahl's Law

$$\text{Speedup} = \frac{\text{Performance after enhancement}}{\text{Performance before enhancement}} = \frac{\text{Execution time before enhancement}}{\text{Execution time after enhancement}}$$

$$S = \frac{1}{(1-f) + \frac{f}{k}} \rightarrow P$$

f  
 ↗  
 ↘

**S=Speedup,**  
**f=fraction of time enhancement,**  
**k=speedup of the faster component**

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

↗  
 ↗  
 ↙ ↘  
 ↗ no of time

# Amdahl's Law

$$S = \frac{1}{(1-f) + f/k}$$

If 90% of a program is speeded up to run 10 times faster  $f=0.9$  and  $k=10$

Overall speedup is  $1/(1-0.9)+(0.9/10)=$   
 ~~$S = 1/(0.1+0.09) = 1/(0.19) = 5.26$~~

Making 80% of a program run 20% faster

$$f=0.80 \text{ and } k=1.2 \quad K = 100 + \frac{10}{0.8} \Rightarrow 1 + \frac{12}{10} \Rightarrow 1.2$$

$$1/(1-0.8)+(0.8/1.2)=$$

$$1/(0.2+0.8/1.2)=1/(0.2+0.66)=1/0.866=1.154$$

# Example

On a large system CPU upgrade makes it faster by 50% for INR 10,000. A disk drive upgrade of INR 7000 speeds it up by 150%. Evaluate the speedups? Processes spend 70% in CPU and 30% waiting Disk drives.

~~Processor upgrade~~

**Disk Drive upgrade**

$$f = 0.70, \quad S = \frac{1}{(1 - 0.7) + 0.7/1.5} = 1.304$$

$$f = 0.30, \quad S = \frac{1}{(1 - 0.3) + 0.3/2.5} = 1.219$$

30% improvement

22% Improvement

CPU-30 % improvement -faster by 50%  
---so 1% increment is INR 10000/30=INR 333

$$\begin{array}{r} 150 \\ \downarrow \\ 1.5 + 1 \Rightarrow 2.5 \end{array}$$

DISK DRIVE- 22% improvement – speeds up 150%---so a 1% increment is INR 7000/22=INR=318

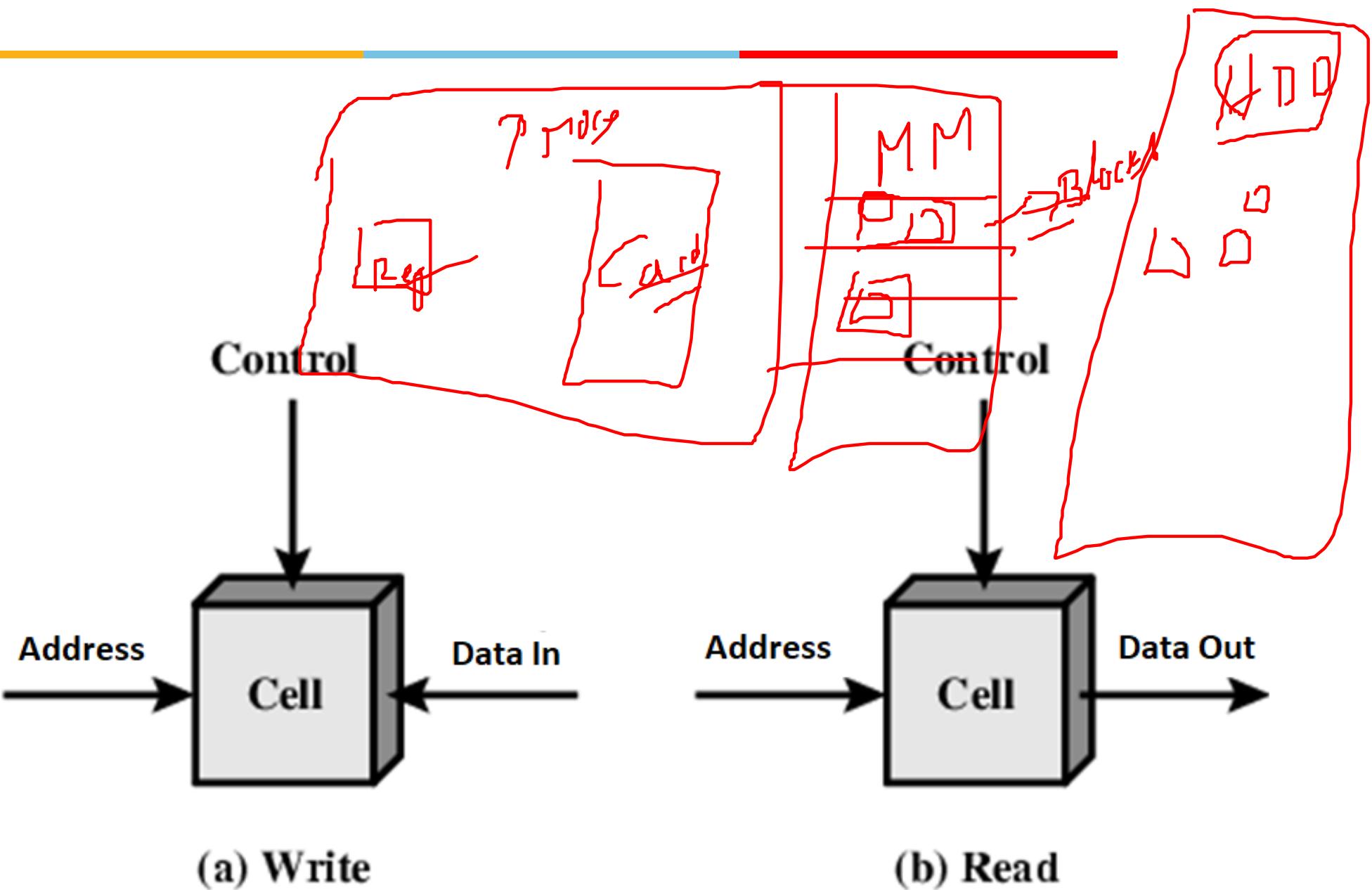
Each 1% of improvement for the processor costs INR333, and for the disk a 1% improvement costs INR318. "Is cost/performance the most important metric?"



# Memory Organization

**BITS** Pilani  
Pilani Campus

# Semiconductor Memory

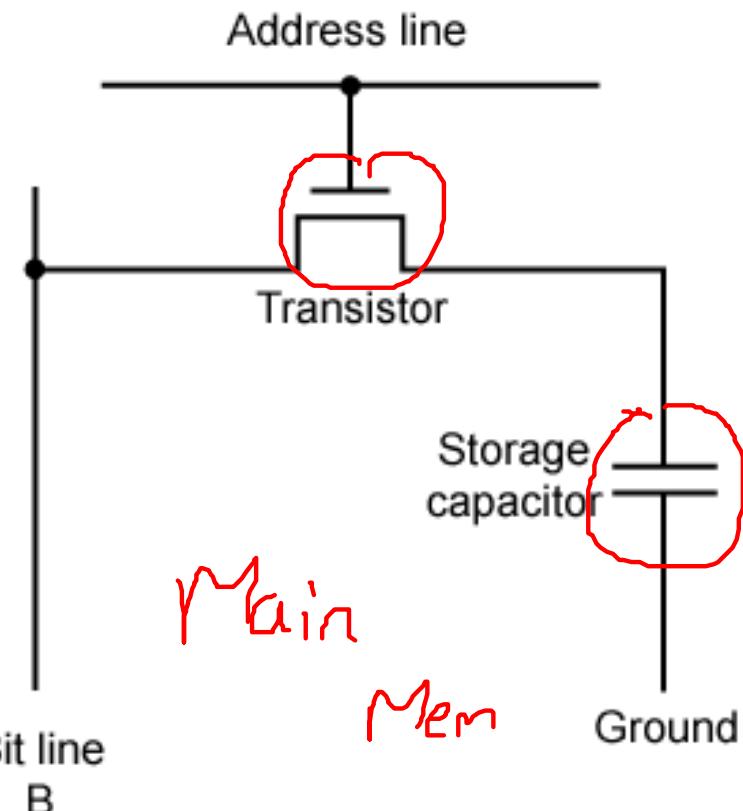
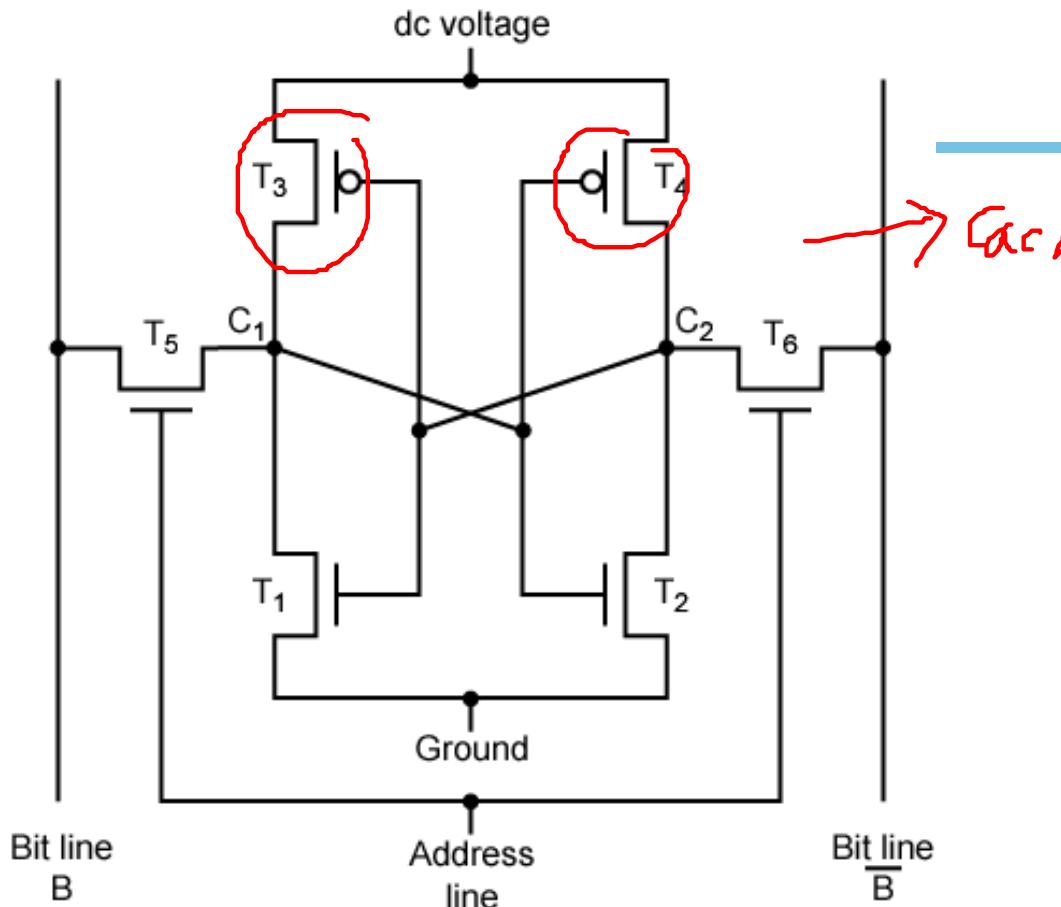


# Random-Access Memory (RAM)

---

- Key features
  - RAM is traditionally packaged as a chip.
  - Basic storage unit is normally a cell (one bit per cell).
  - Multiple RAM chips form a memory.
- RAM comes in two varieties:
  - ✓ SRAM (Static RAM)
  - ✗ DRAM (Dynamic RAM)
- SRAM and DRAM are volatile memories
  - Lose information if powered off.

# SRAM vs DRAM Summary



	Trans. per bit	Access time	Needs refresh?	Needs EDC?	Cost	Applications
SRAM	4 to 6	1X	No	Maybe	100x	Cache
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers

# Read Only Memory

- Permanent Storage and Nonvolatile Memories
- Read Only Memory Variants:
  - Read-only memory (**ROM**): programmed during production
  - Programmable ROM (**PROM**): can be programmed once
  - Erasable PROM (**EPROM**): can be bulk erased (UV, X-Ray)
  - Electrically erasable PROM (**EEPROM**): electronic erase capability
  - Flash memory: EEPROMs. with partial (block-level) erase capability
    - Wears out after about 100,000 erasing
- Firmware

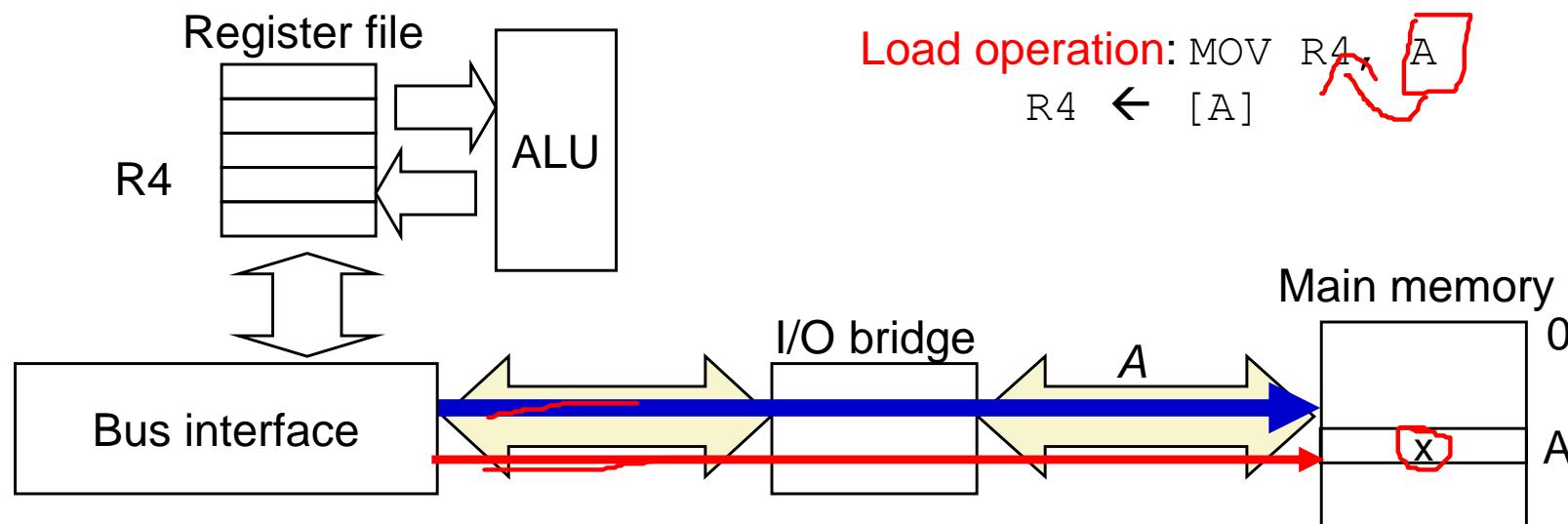
# Applications

---

- Storing fonts for printers
- Storing sound data in musical instruments
- Video game consoles
- Implantable Medical devices.
- High definition Multimedia Interfaces(HDMI)
- BIOS chip in computer
- Program storage chip in modem, video card and many electronic gadgets, controllers for disks, network cards, ....

# Memory Read Operation (1)

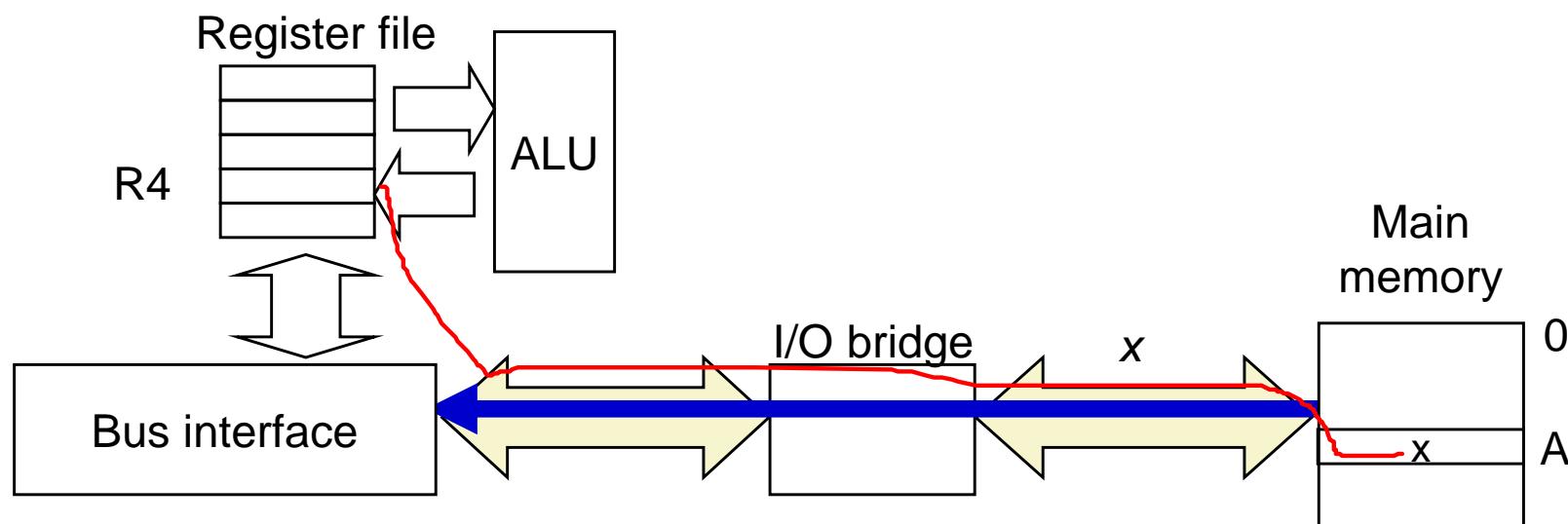
CPU places address  $A$  and then read control signal on the memory bus



# Memory Read Operation (2)

Main memory reads A from the memory bus, retrieves word x, and places it on the bus

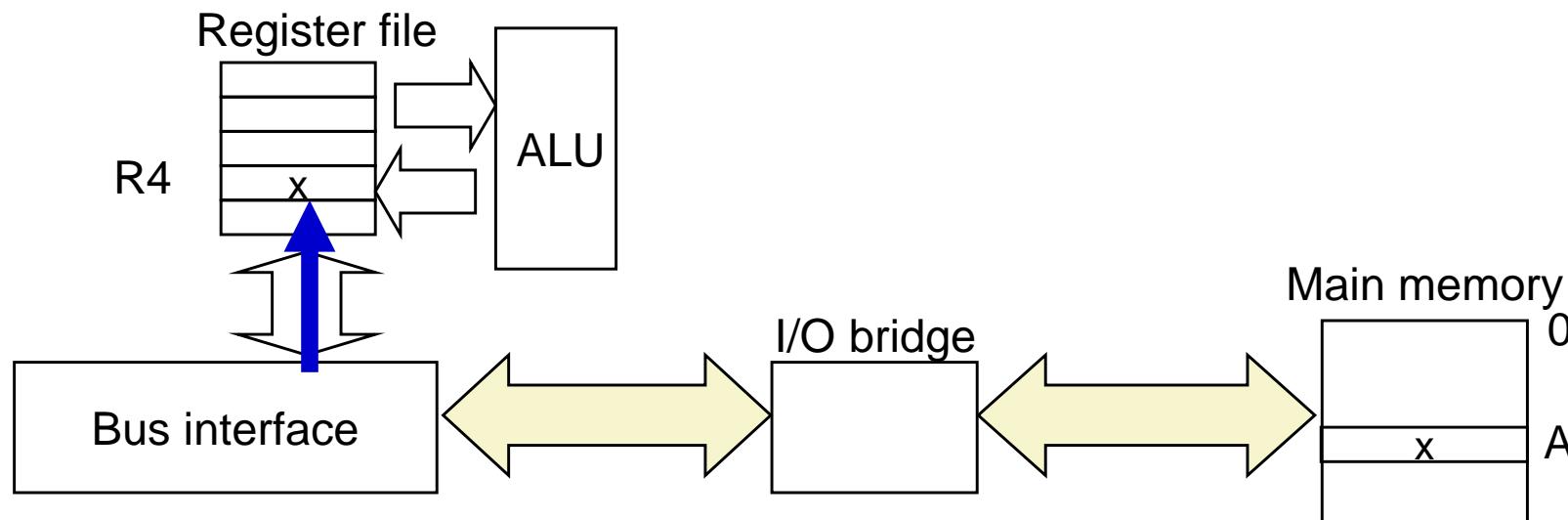
Load operation:  $\text{MOV } R4, A$   
 $R4 \leftarrow [A]$



# Memory Read Operation (3)

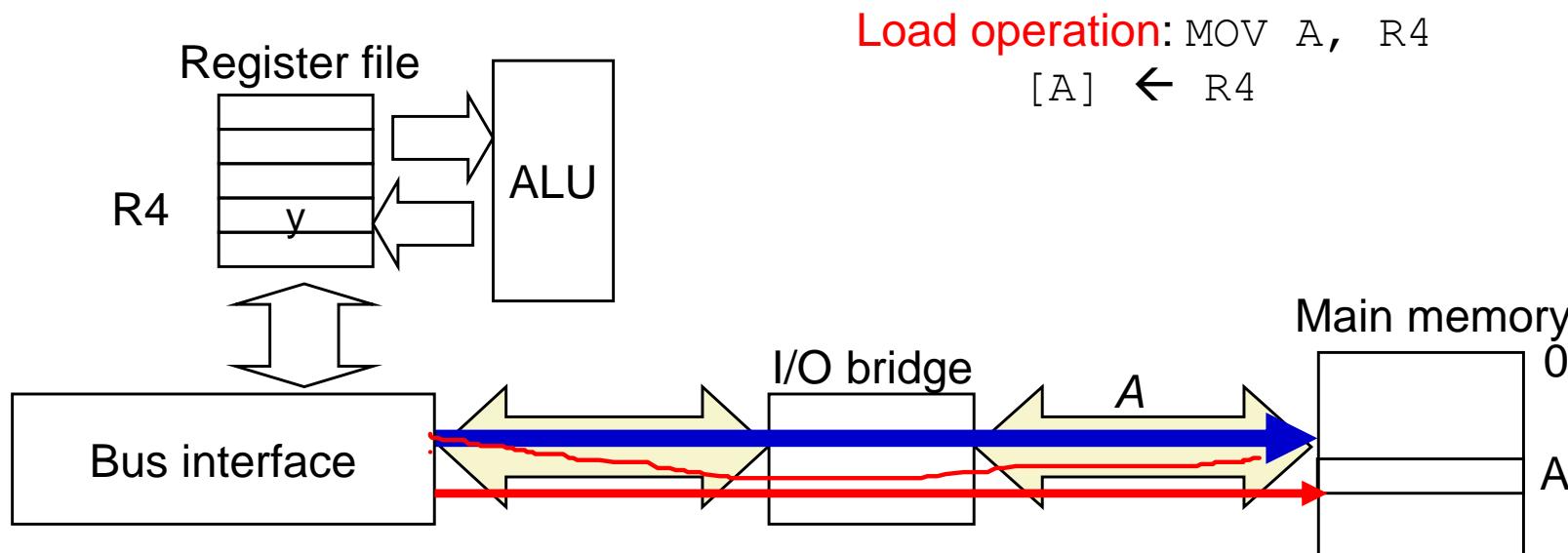
CPU read word  $x$  from the bus and copies it into register R4.

Load operation:  $\text{MOV } R4, A$   
 $R4 \leftarrow [A]$



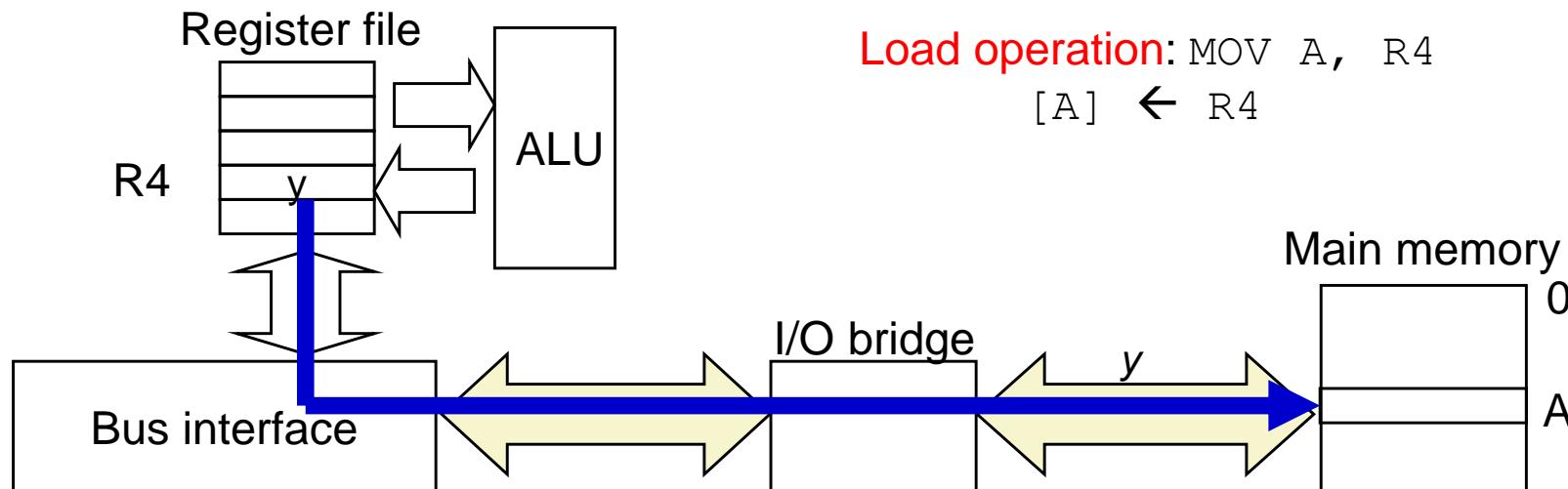
# Memory Write Operation (1)

CPU places address  $A$  and **WRITE** control signal on bus.  
 Main memory reads them and waits for the corresponding data word to arrive.



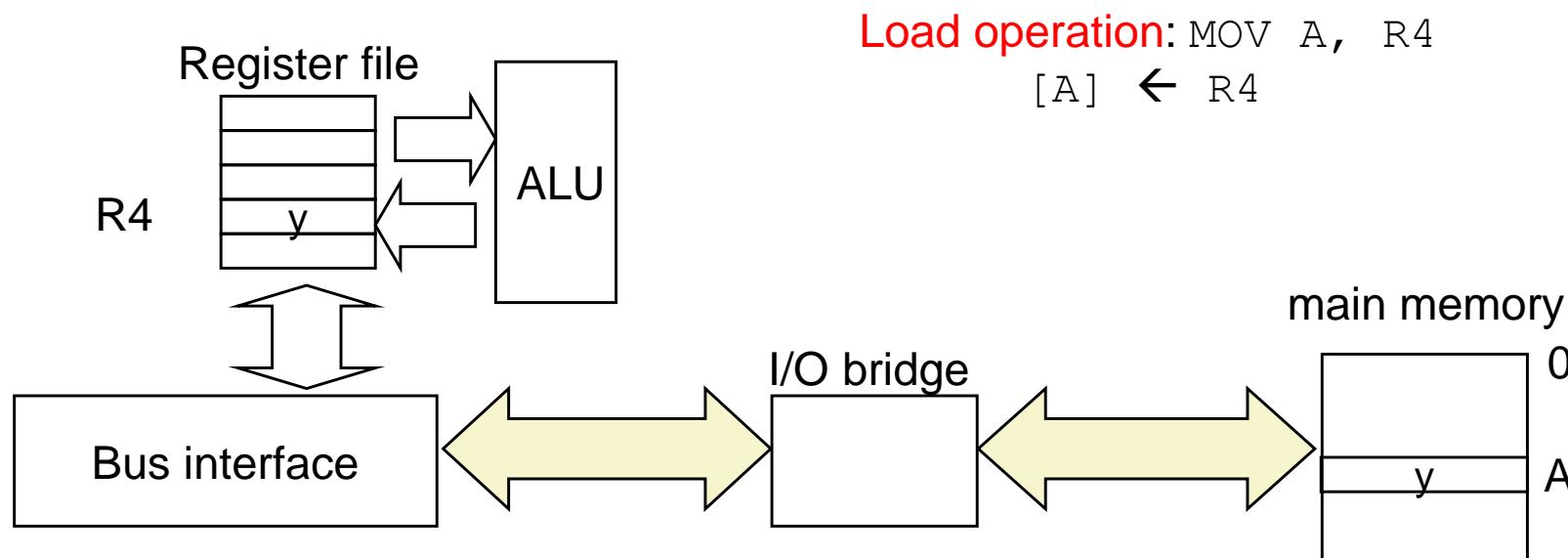
# Memory Write Operation (2)

CPU places data word  $y$  on the bus



# Memory Write Operation (3)

Main memory reads data word  $y$  from the bus and stores it at address  $A$ .



# Magnetic Disk Drive

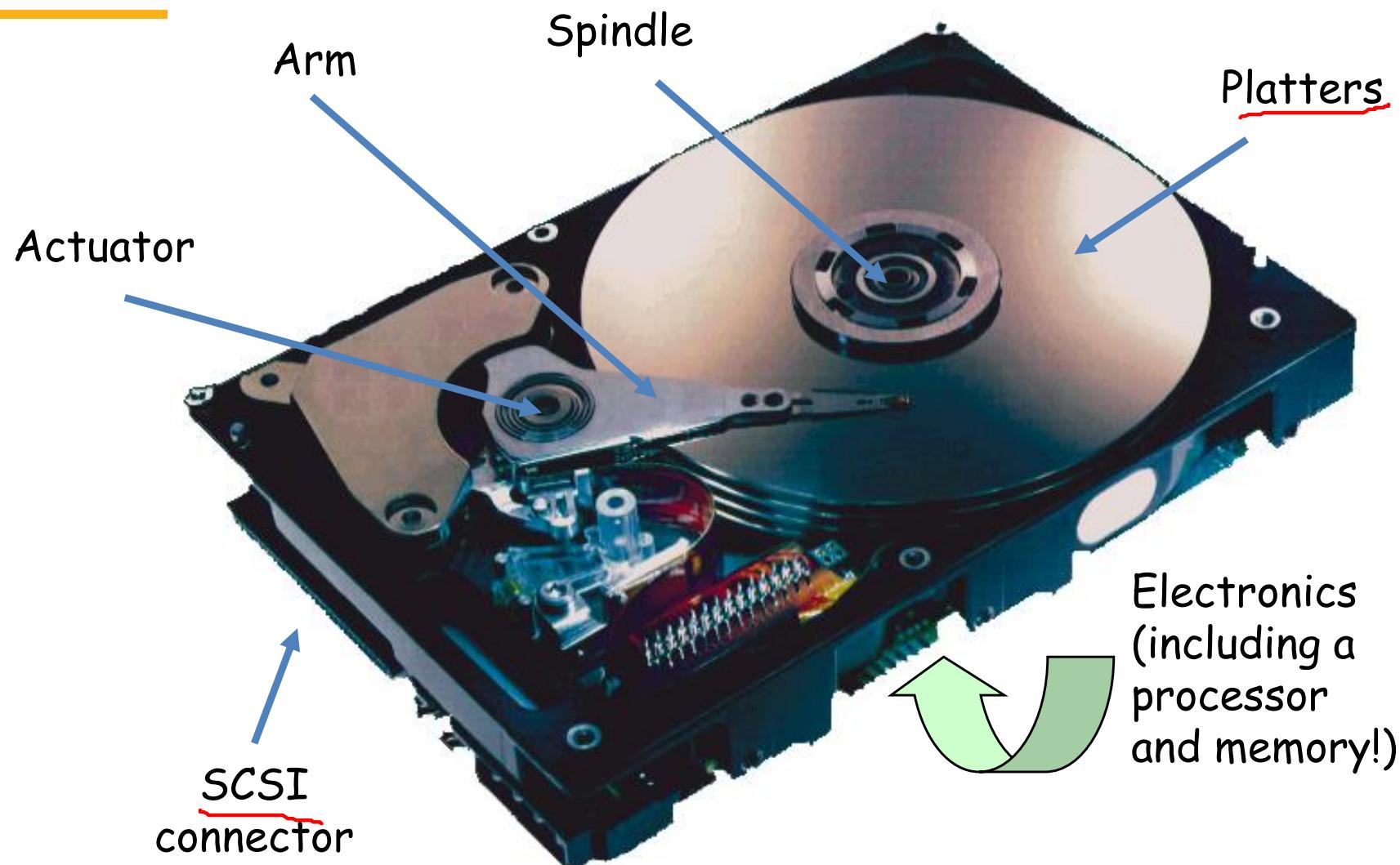
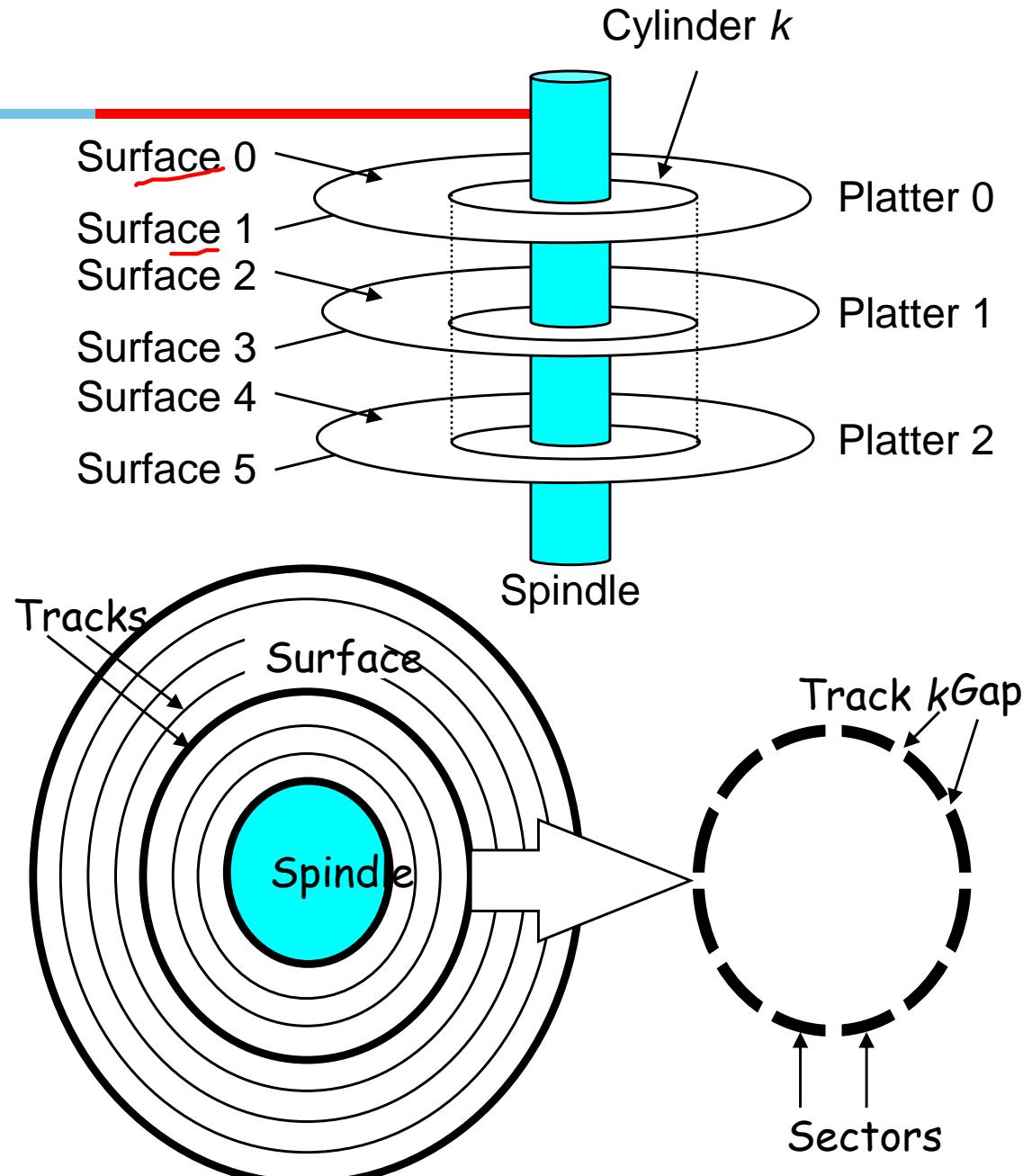


Image courtesy of Seagate Technology

# Disk Geometry

- Disks consist of **platters**, each with two **surfaces**.
- Each **surface** consists of concentric rings called **tracks**
- Aligned tracks form a **cylinder**
- Each track consists of **sectors** separated by **gaps**.



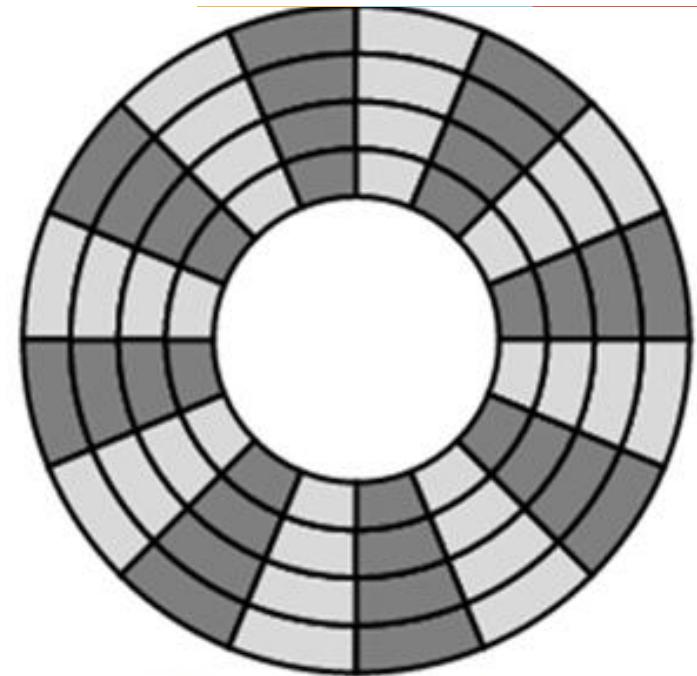
# Disk Capacity

---

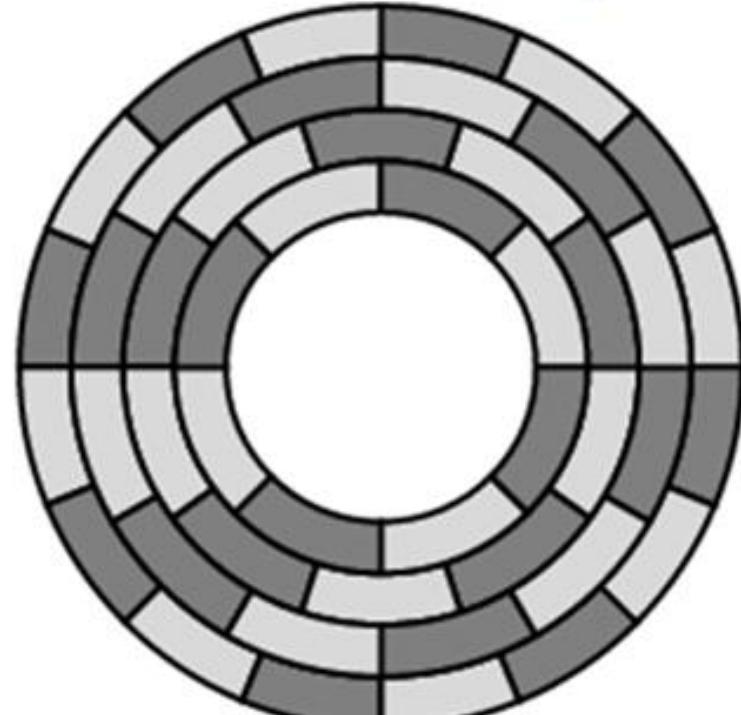
- **Capacity:** maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB /TB), where  $1\text{ GB} = 2^{30}\text{ Bytes}$ ,  $1\text{ TB} = 2^{40}\text{ Bytes}$ ,
- Capacity is determined by these technology factors:
  - **Recording density** (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - **Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
  - **Areal density** (bits/in<sup>2</sup>): product of recording and track density.

# Recording zones

- Modern disks partition tracks into disjoint subsets called **recording zones**
  - Each track in a zone has the same number of sectors, determined by the circumference of innermost track.
  - Each zone has a different number of sectors/track, outer zones have more sectors/track than inner zones.
  - So we use average number of sectors/track when computing capacity.



Without Recording Zones



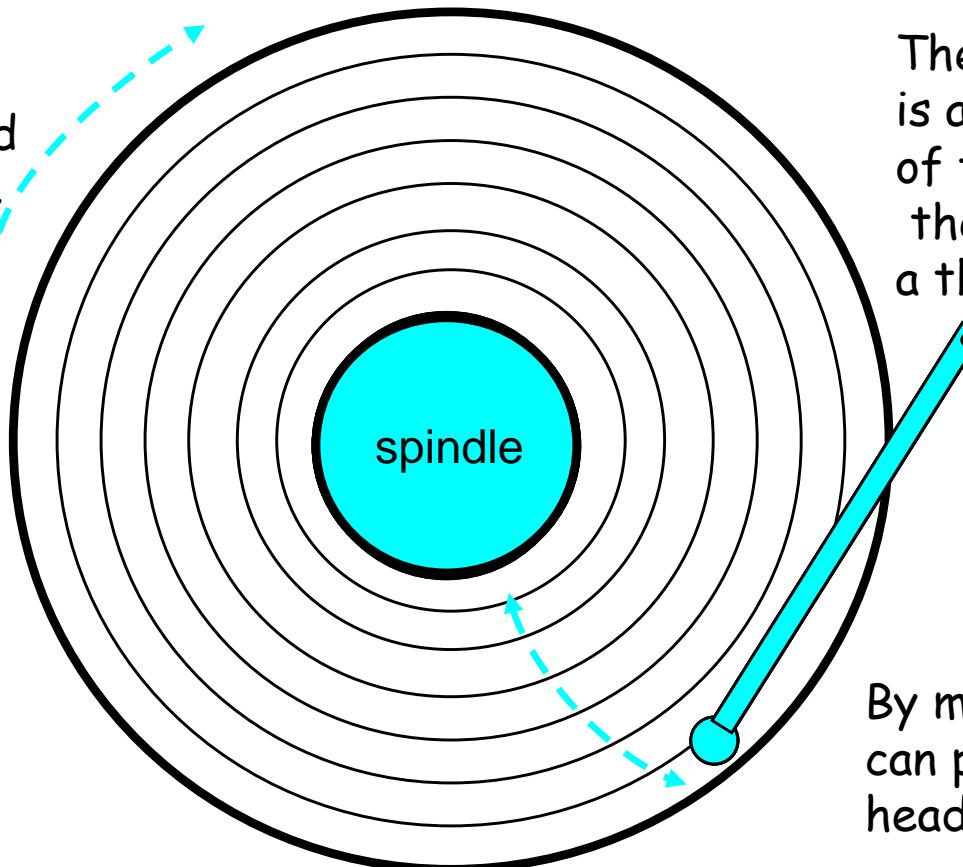
With Recording Zones

# Computing Disk Capacity

- Capacity =  $(\# \text{ bytes/sector}) \times (\text{avg. } \# \text{ sectors/track}) \times (\# \text{ tracks/surface}) \times (\# \text{ surfaces/platter}) \times (\# \text{ platters/disk})$
- Example:
  - 512 bytes/sector
  - 300 sectors/track (on average)
  - 20,000 tracks/surface
  - 2 surfaces/platter
  - 5 platters/disk
- Capacity =  $512 \times 300 \times 20000 \times 2 \times 5$   
 $= 30,720,000,000$   
 $= 1024 \times 1024 \times 1024$  (in GB)

# Disk Operation (Single-Platter View)

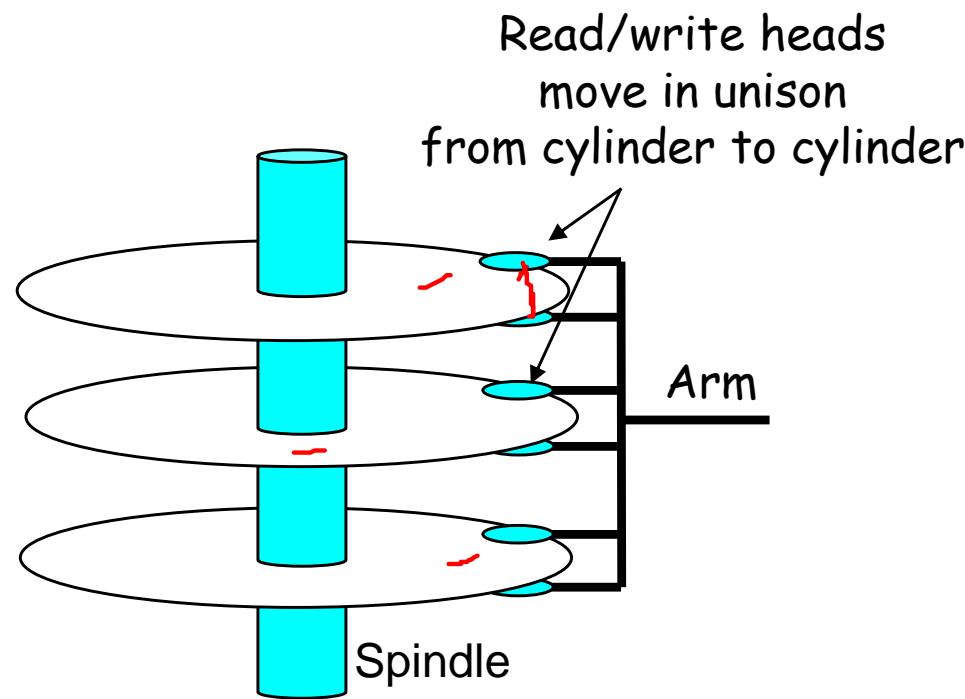
The disk surface spins at a fixed rotational rate



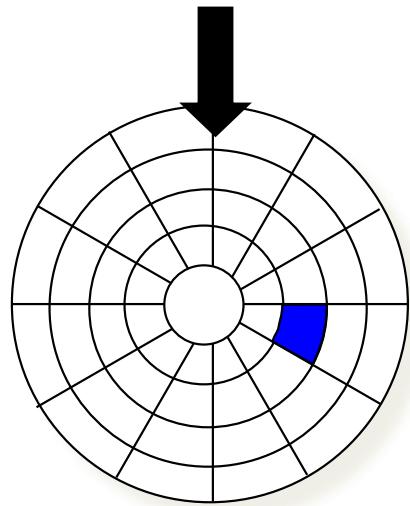
The read/write head is attached to the end of the arm and flies over the disk surface on a thin cushion of air.

By moving radially, the arm can position the read/write head over any track.

# Disk Operation (Multi-Platter View)

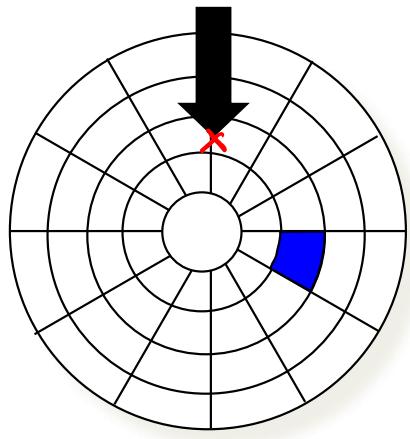


# Disk Access



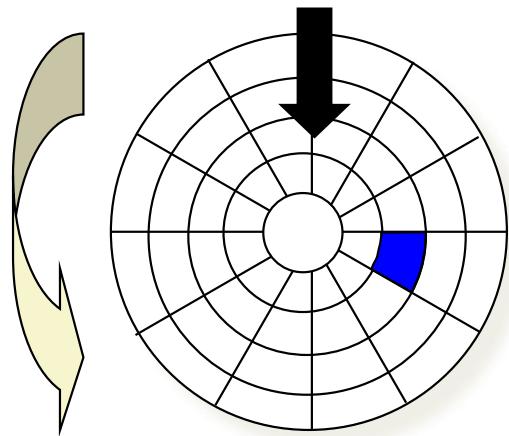
Need to access a sector  
colored in blue

# Disk Access



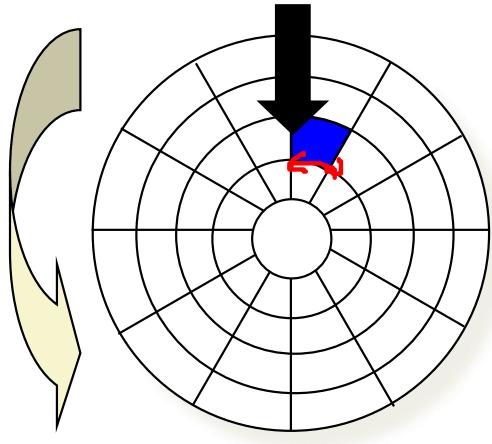
Head in position above a track

# Disk Access



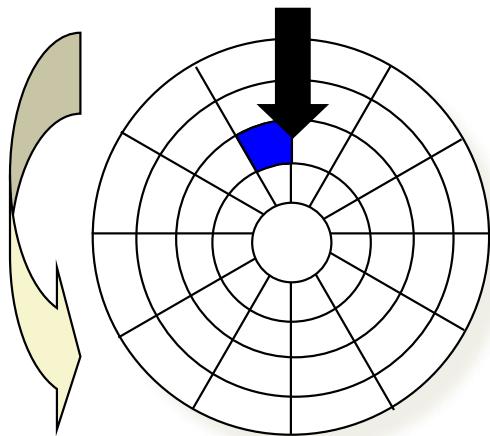
Rotate the platter in counter-clockwise direction

# Disk Access - Read



About to read blue sector

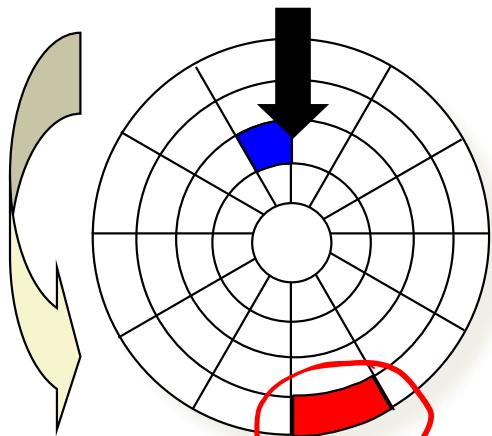
# Disk Access - Read



After BLUE  
read

After reading blue sector

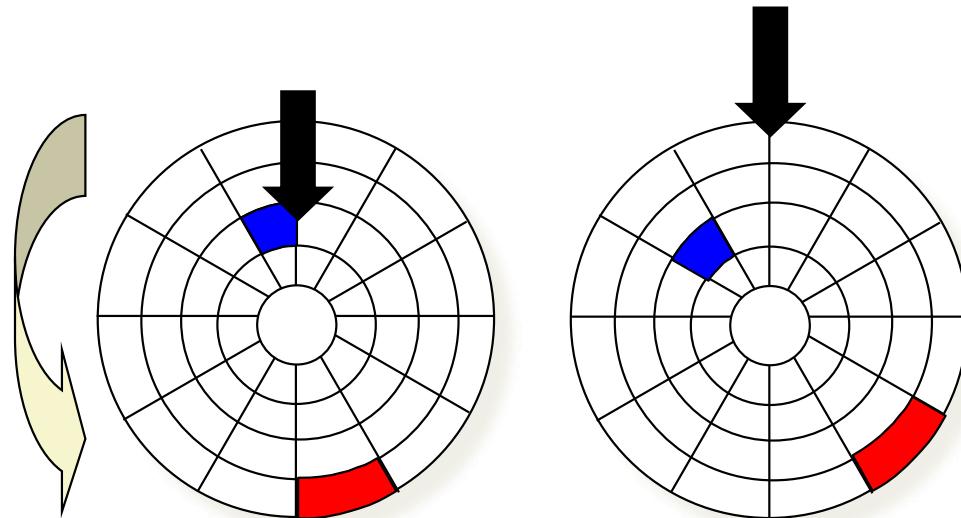
# Disk Access - Read



After BLUE  
read

Red request scheduled next

# Disk Access - Seek

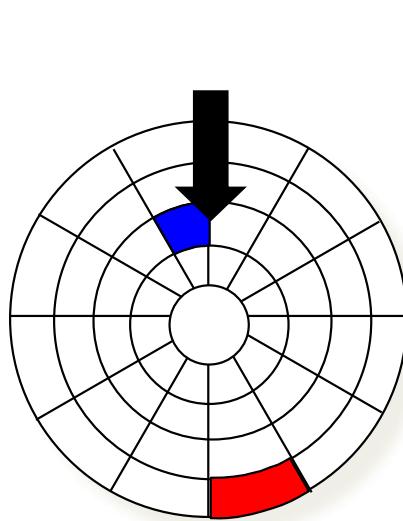


After BLUE  
read

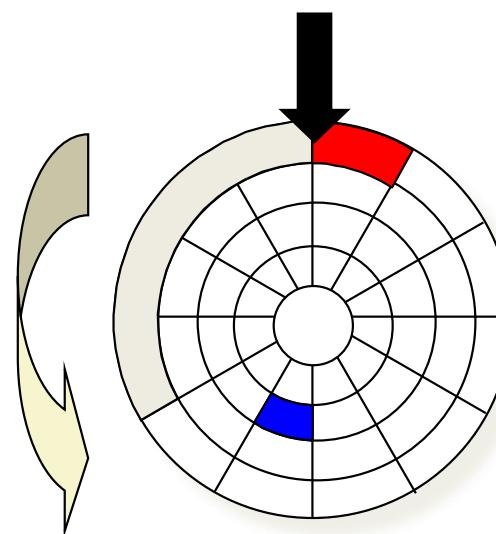
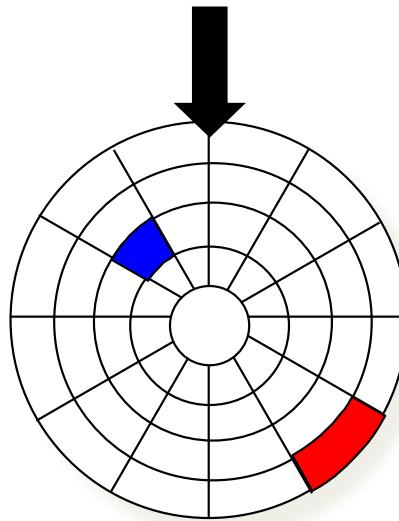
Seek for RED

Seek to red's track

# Disk Access - Rotational Latency



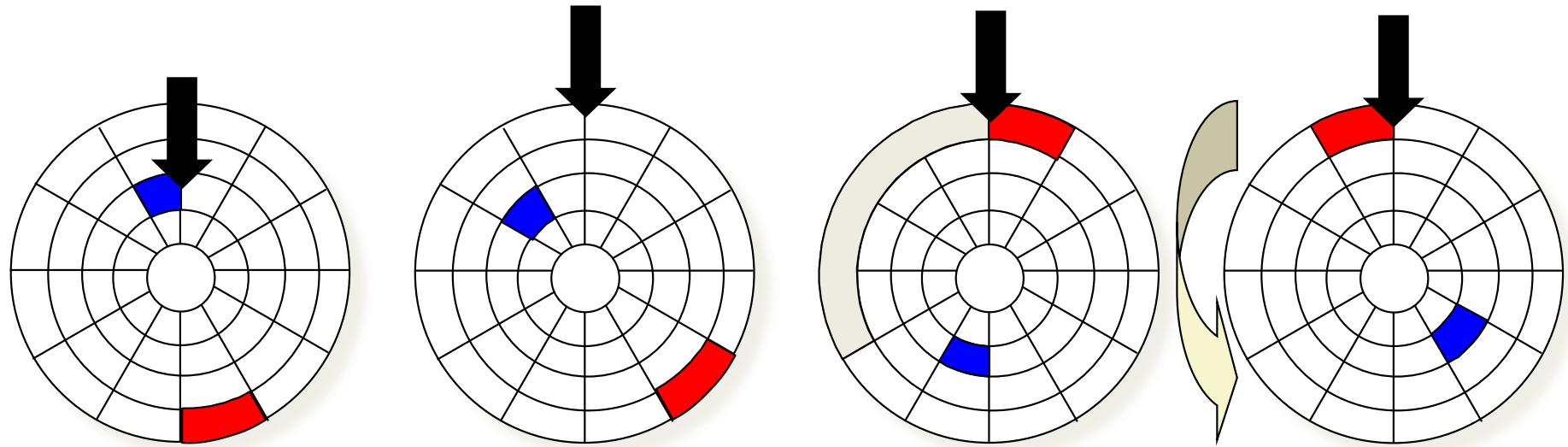
After BLUE Seek for RED  
read



Rotational latency

Wait for red sector to rotate around

# Disk Access - Read



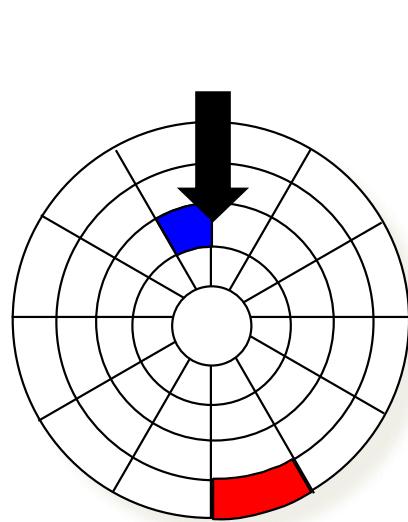
After **BLUE**  
read

Seek for **RED**

Rotational latency After **RED** read

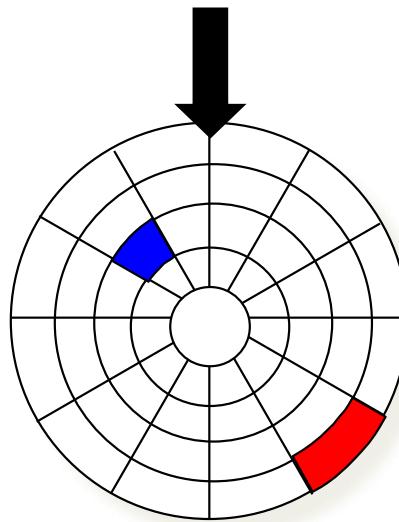
Complete read of red

# Disk Access - Access Time Components



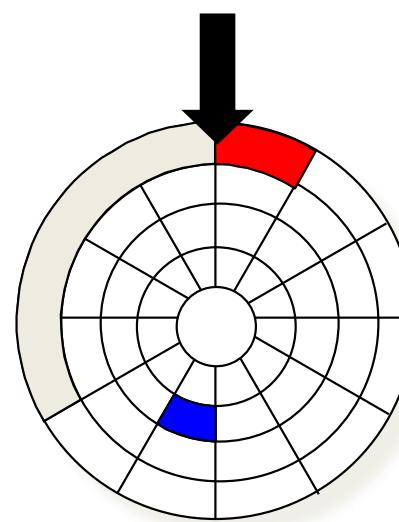
After **BLUE**  
read

↑  
Data transfer



Seek for **RED**

↑  
Seek



Rotational latency After **RED** read

↑  
Rotational latency

**S T S** -   
**B D C**   
↑  
Data transfer

# Disk Access Time

- Average time to access some target sector given by :
  - $T_{\text{access}} = T_{\text{avg seek}} + T_{\text{avg rotation}} + T_{\text{avg transfer}}$
- **Seek time (Tavg seek)**
  - Time to position heads over cylinder containing target sector.
  - Typical Tavg seek is 3–9 ms
- **Rotational latency (Tavg rotation)**
  - Time waiting for first bit of target sector to pass under r/w head.
  - $T_{\text{avg rotation}} = \frac{1}{2r}$ , where r is rotation Speed in revolution per Second
  - Typical Tavg rotation =  $7200 \text{ RPMs} = \frac{7200}{60} \text{ RPS}$
- **Transfer time (Tavg transfer)**
  - Time to read the bits in the target sector.
  - $T_{\text{avg transfer}} = b/rN$ , where b is the number of bytes to be transferred and N is the average number of bytes on a track

# Disk Access Time Example

Given:

- Rotational rate = 7,200 RPM
- Average seek time = 9 ms.
- Avg # sectors/track = 400.
- 512 bytes per sector

$$\frac{60}{2 \times 7200} = \frac{60}{14400} = 0.00416 \text{ sec}$$

Derived:

- Tavg rotation =  $\frac{1}{2r} = \frac{1}{2} \times (60 \text{ secs}/7200 \text{ RPM})$   
 $= 0.00416 = 4.16 \text{ ms.}$
- Tavg transfer =  $b/rN$   
 $= \frac{512}{7200} \times \frac{1}{400 \times 512}$   
 $= 0.02 \text{ ms.}$
- Taccess =  $9 \text{ ms} + 4.16 \text{ ms} + 0.02 \text{ ms} = 13.18 \text{ ms}$

# Contd..

## Important points:

- Access time dominated by seek time and rotational latency.
- First bit in a sector is the most expensive, the rest are free.
- SRAM access time is about 4 ns/doubleword,  
DRAM about 60 ns
  - Disk is about 40,000 times slower than SRAM,
  - 2,500 times slower than DRAM.



# Computer Organization and Software Systems

## CONTACT SESSION 3

Dr. Lucy J. Gudino  
WILP & Department of CS & IS



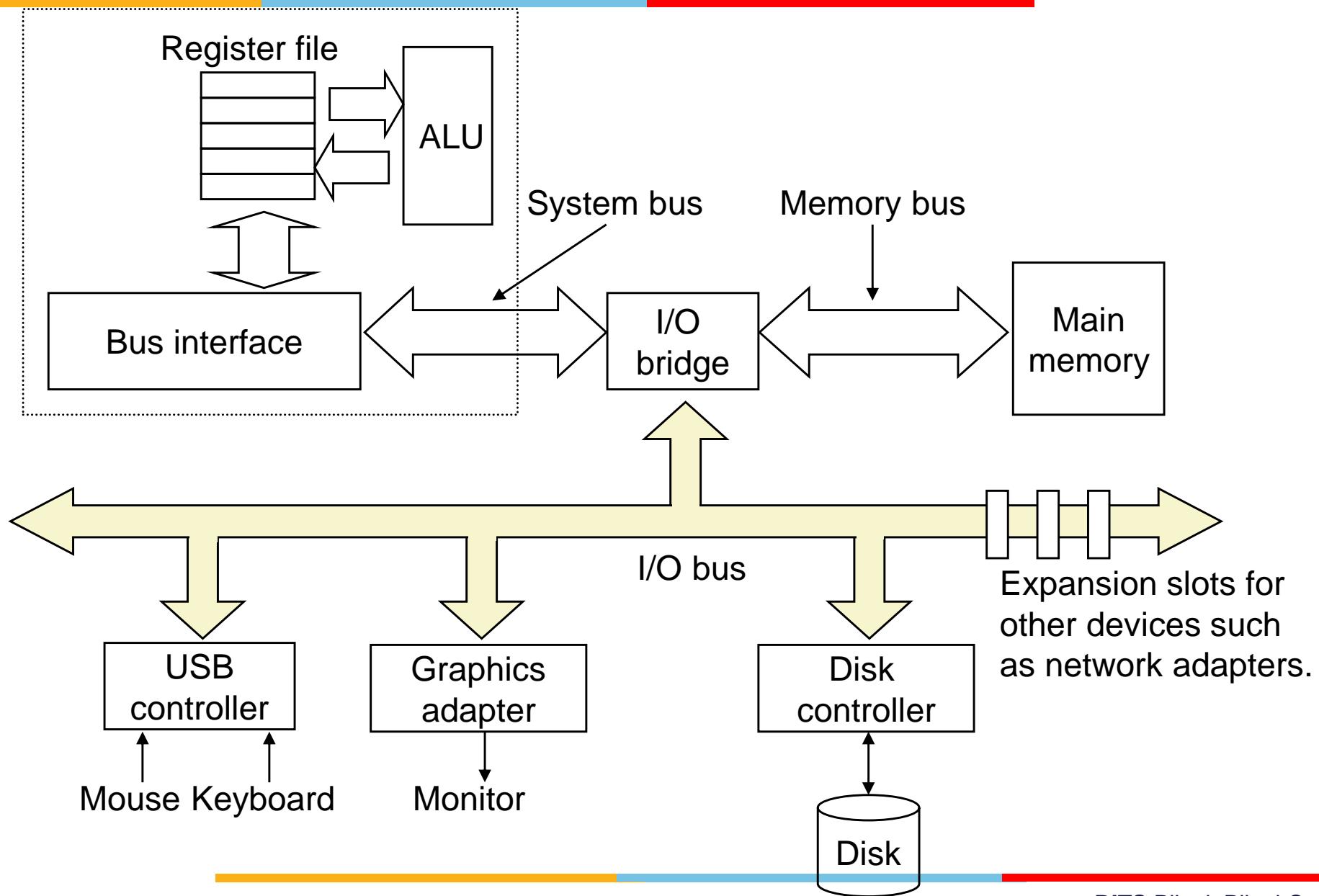
**BITS** Pilani  
Pilani Campus

# Logical Disk Blocks

- Modern disks present a simpler abstract view of the complex sector geometry:
  - The set of available sectors is modeled as a sequence of logical blocks (0, 1, 2, ...B-1)
- Mapping between logical blocks and actual (physical) sectors
  - Maintained by hardware/firmware device called disk controller.
  - Converts requests for logical blocks into (surface, track,sector) triples.
- Allows controller to set aside spare cylinders for each zone.
  - Accounts for the difference in "formatted capacity" and "maximum capacity".

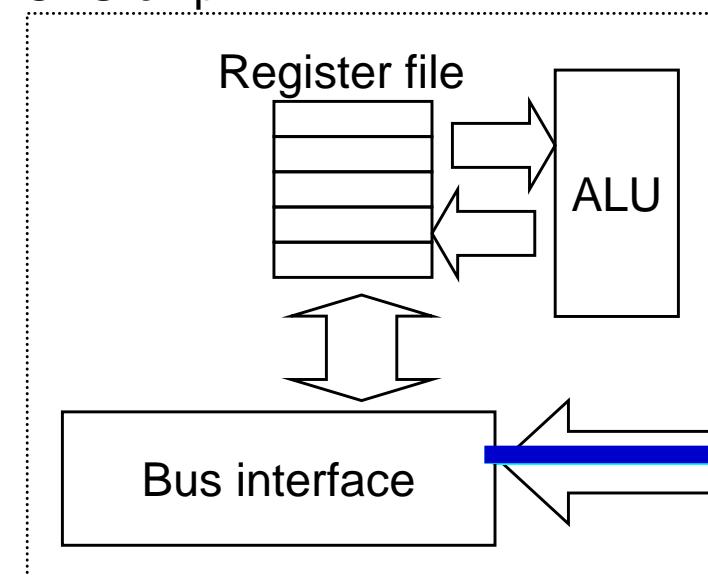
# I/O Bus

CPU chip

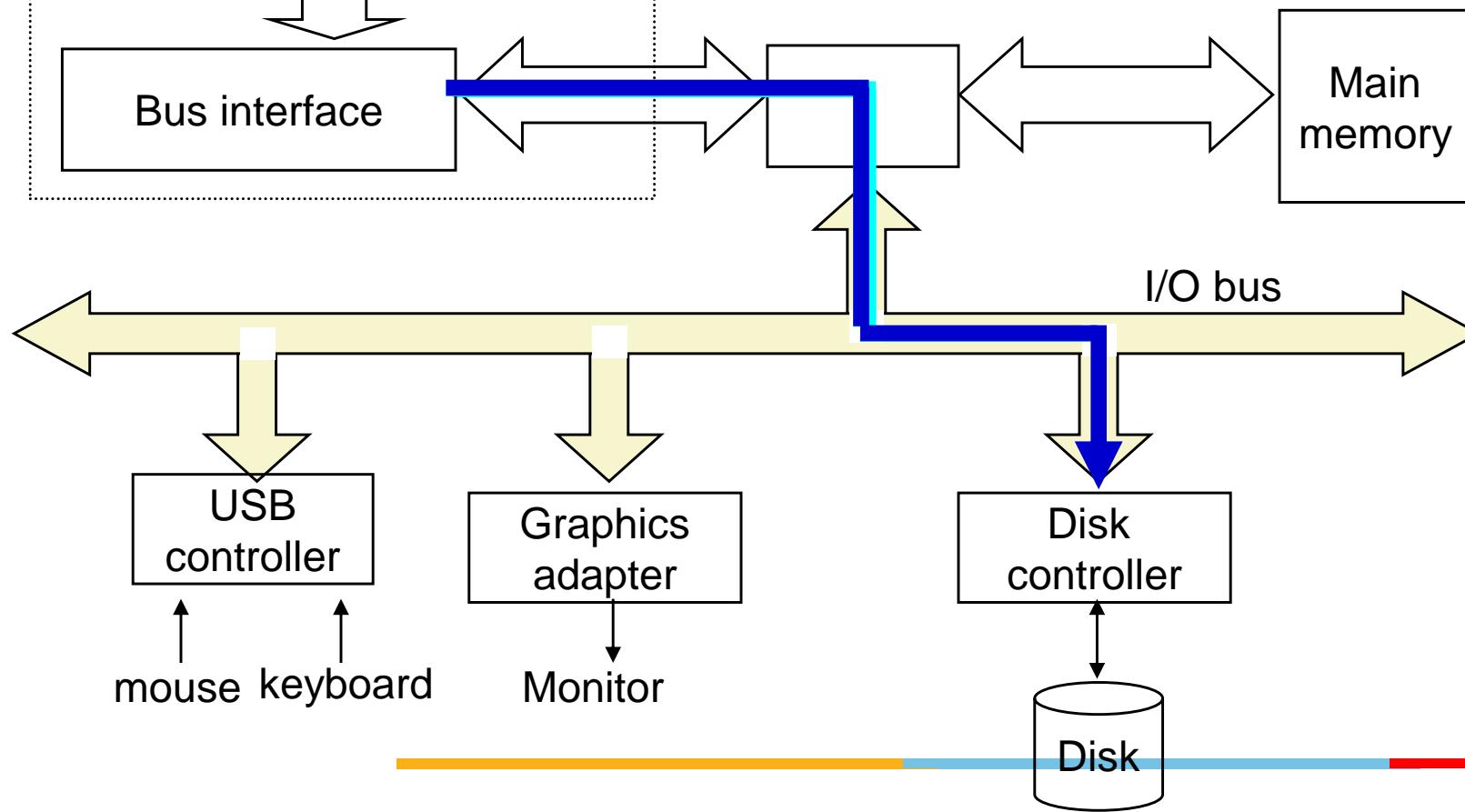


# Reading a Disk Sector (1)

CPU chip

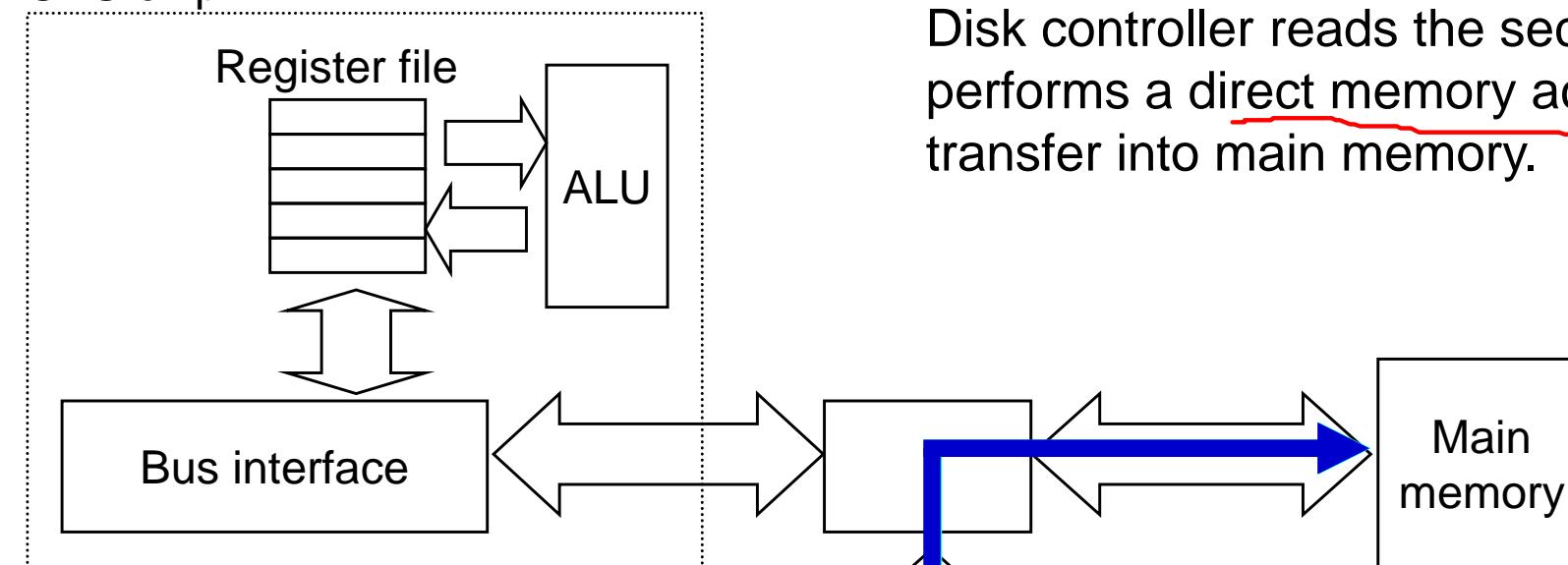


CPU initiates a disk read by writing a command, logical block number, and destination memory address to a **port** (address) associated with disk controller.

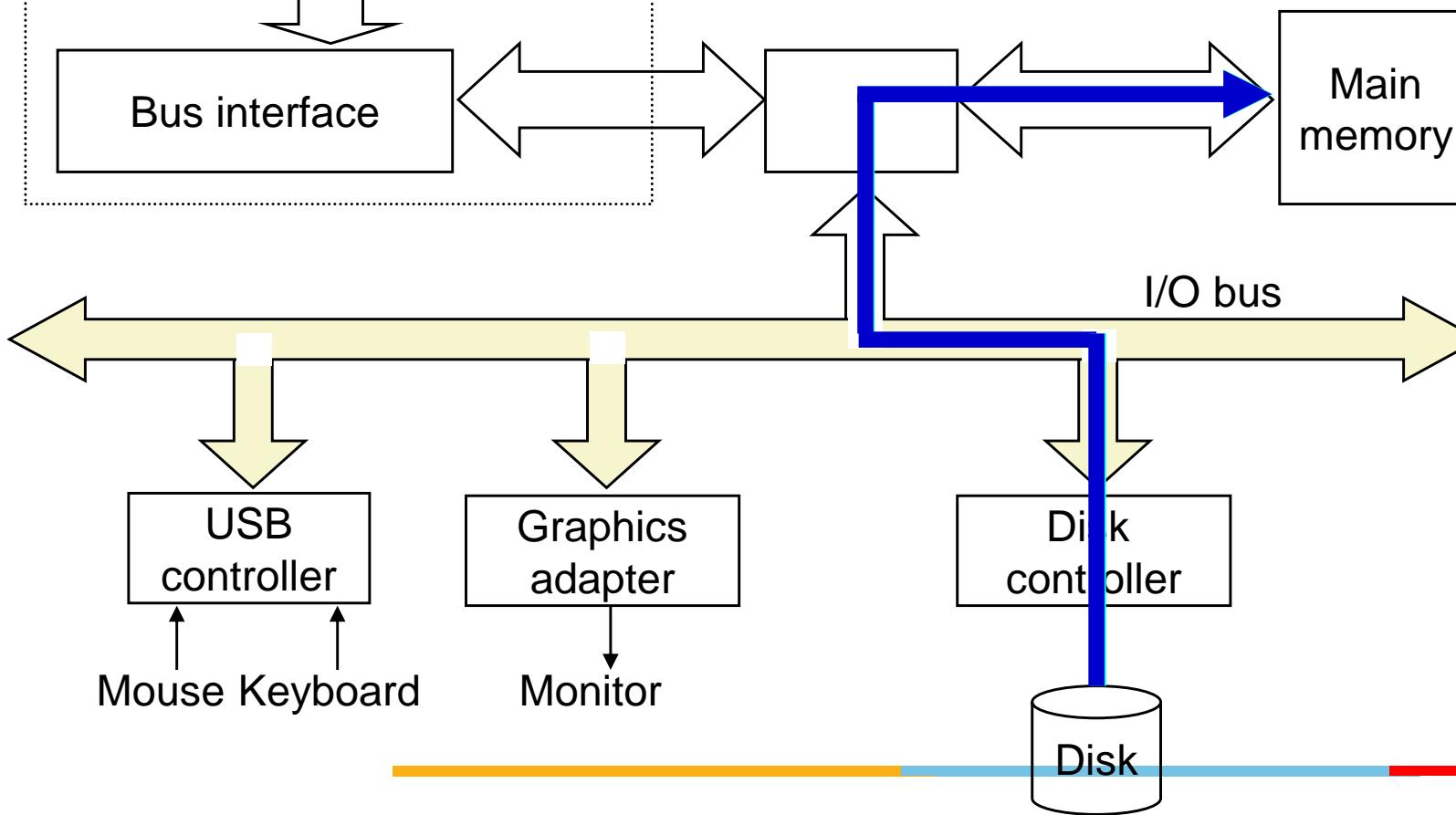


# Reading a Disk Sector (2)

CPU chip

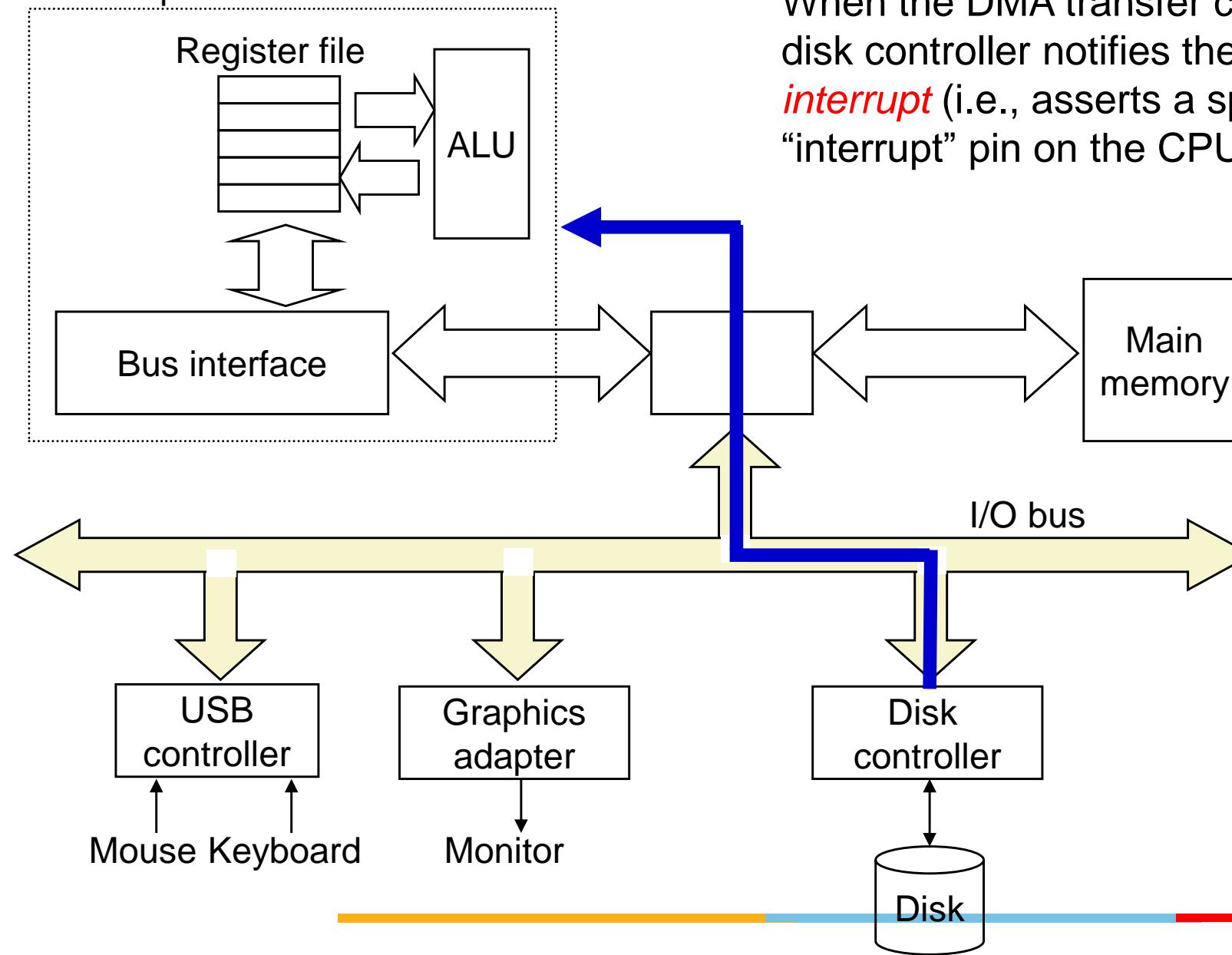


Disk controller reads the sector and performs a direct memory access (**DMA**) transfer into main memory.



# Reading a Disk Sector (3)

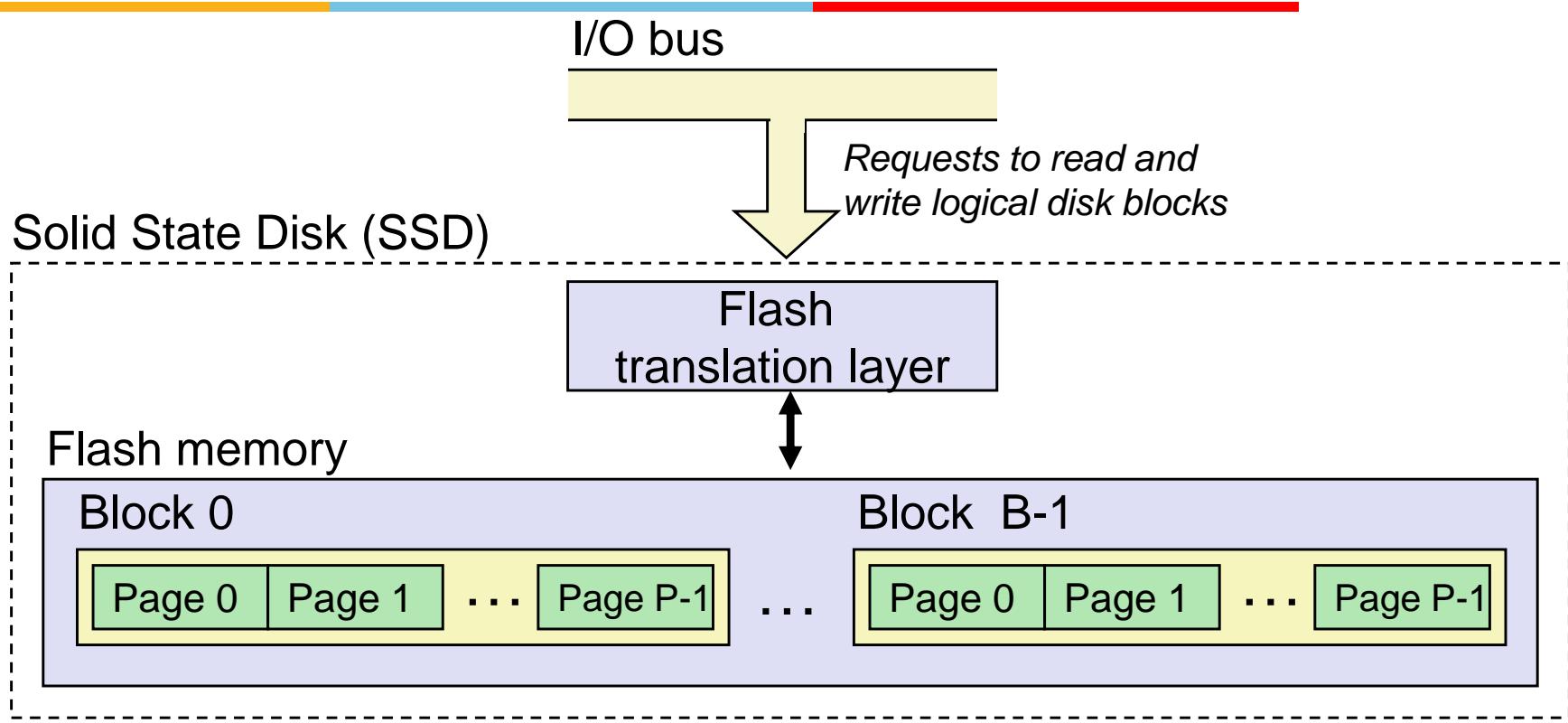
CPU chip



# Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
5-6	<p><b>Memory Organization (Contd..)</b></p> <ul style="list-style-type: none"> <li>• Solid State Devices</li> <li>• Locality           <ul style="list-style-type: none"> <li>• Locality of Reference to Program Data</li> <li>• Locality of instruction fetches</li> </ul> </li> <li>• Memory Hierarchy</li> <li>• Cache Memories           <ul style="list-style-type: none"> <li>• Generic Cache Memory Organization</li> <li>• Direct-Mapped Caches</li> <li>• <u>Fully Associative Caches</u></li> </ul> </li> </ul>	T1

# Solid State Disks (SSDs)



- Pages: 512B to 4KB, Blocks: 32 to 128 pages
- Data read/written in units of pages.
- Page can be written only after its block has been erased
- A block wears out after about 100,000 repeated writes.

# SSD Performance Characteristics

Sequential read tput*	550 MB/s	Sequential write tput	470 MB/s
Random read tput	365 MB/s	Random write tput	303 MB/s
Avg seq read time	50 us	Avg seq write time	60 us

Tput → Throughput

- Sequential access faster than random access
  - Common theme in the memory hierarchy
- Random writes are somewhat slower
  - Erasing a block takes a long time (~1 ms)
  - Modifying a block page requires all other pages to be copied to new block
  - In earlier SSDs, the read/write gap was much larger.

Source: Intel SSD 730 product specification.

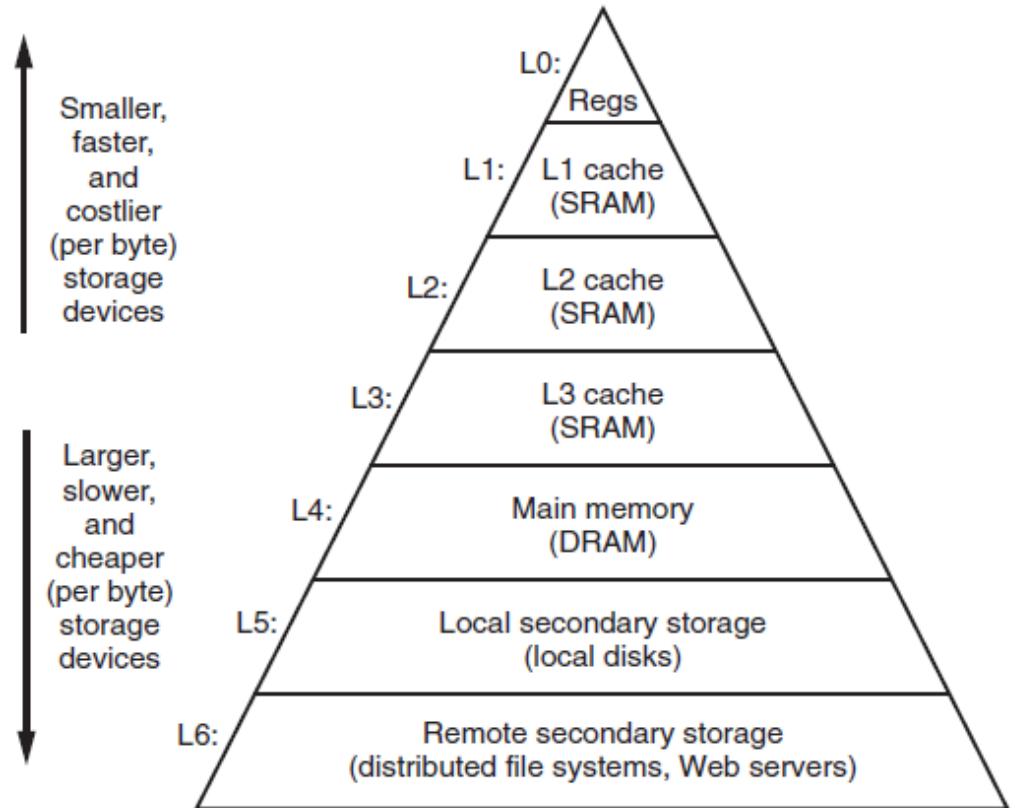
# SSD Tradeoffs vs Rotating Disks

---

- Advantages
    - No moving parts → faster, less power, more rugged
  - Disadvantages
    - Have the potential to wear out
      - Mitigated by "wear leveling logic" in flash translation layer
      - E.g. Intel SSD 730 guarantees 128 petabyte ( $128 \times 10^{15}$  bytes) of writes before they wear out
    - In 2015, about 30 times more expensive per byte
  - Applications
    - MP3 players, smart phones, laptops
    - Beginning to appear in desktops and servers
-

# The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?



An example of a memory hierarchy.

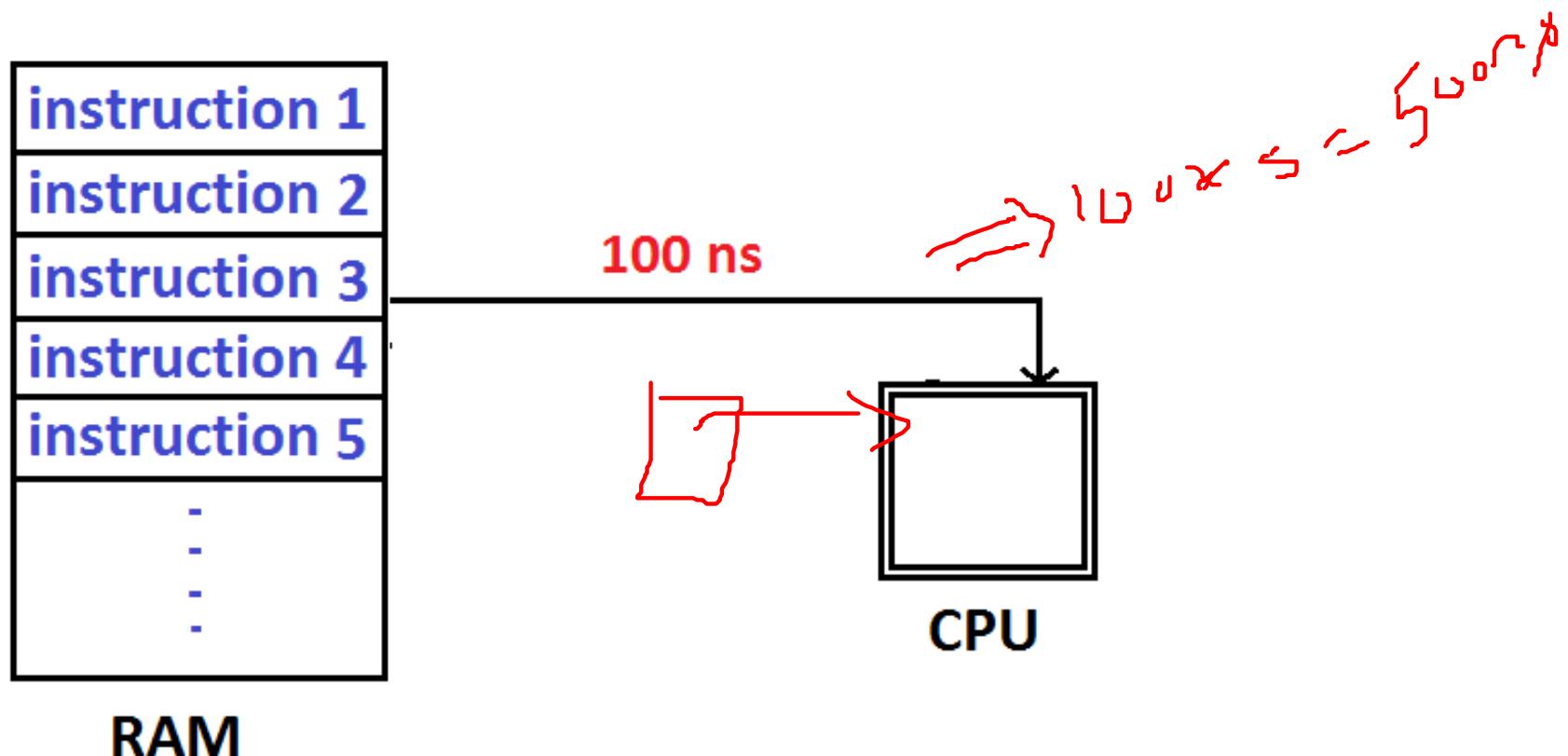
- Faster access time , ----- cost per bit
- Greater capacity, ----- cost per bit
- Greater capacity, -----access time



# Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - "RAM"
- External memory
  - Backing store

# Performance enhancement - Motivation



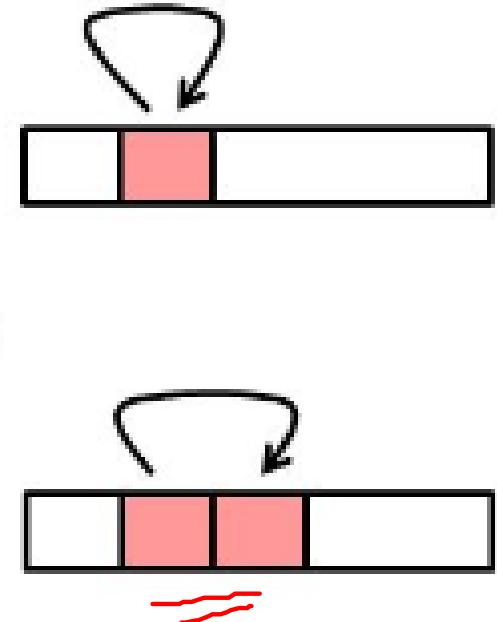
# Performance enhancement - Motivation



## Locality of Reference

During the course of the execution of a program, memory references tend to cluster

- **Temporal locality:** Locality in time
  - If an item is referenced, it will tend to be referenced again soon
- **Spatial locality:** Locality in space
  - If an item is referenced, items whose addresses are close by will tend to be referenced soon.



# Example

$n = 10$

```
product = 1; q  
{ for ( i = 0; i < n-1; i++)  
    product = product * a[i] ;
```

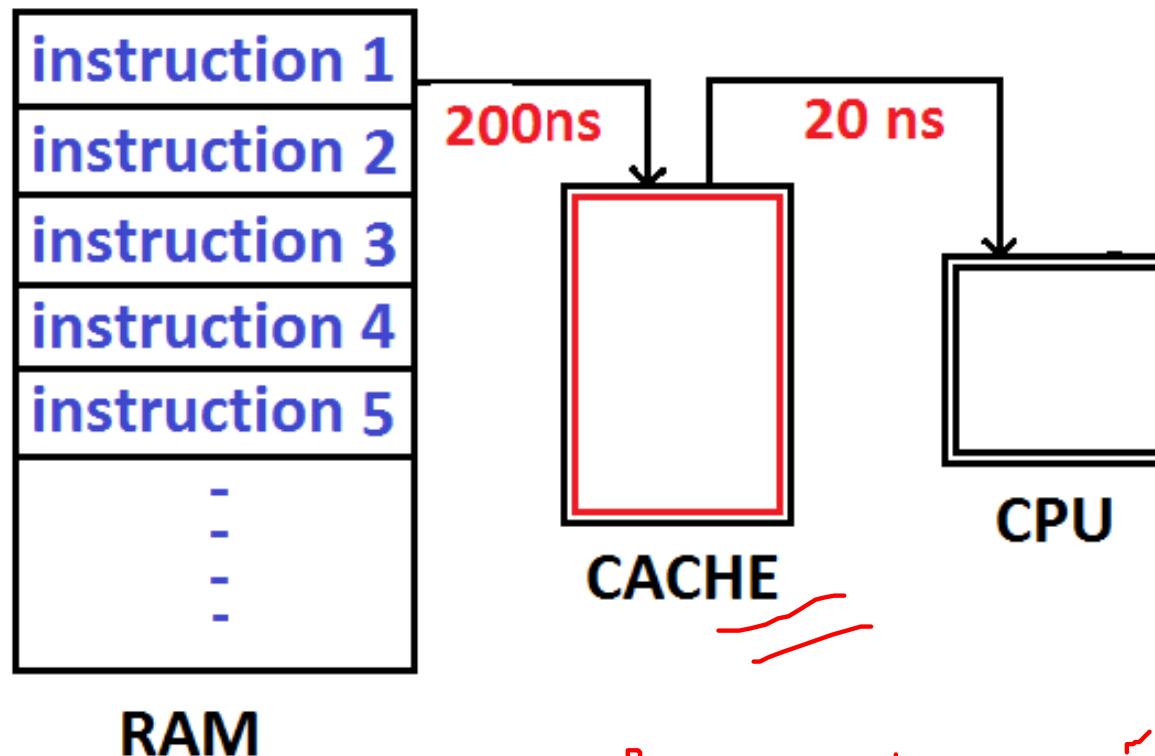
Data :

- Access array elements in succession - spatial locality
- Reference to “product” in each iteration - Temporal locality

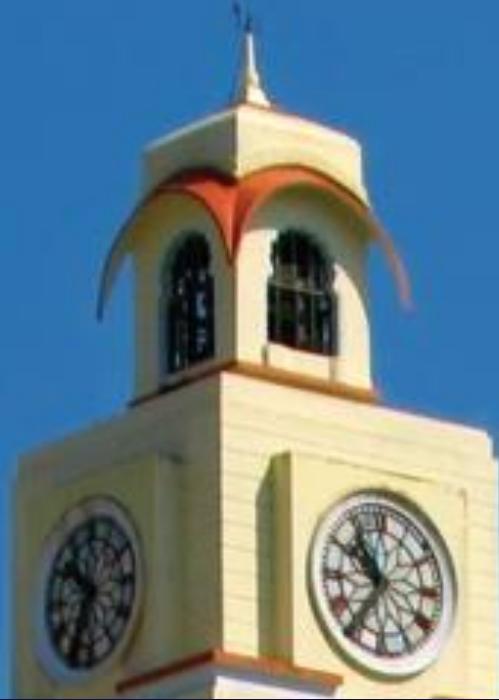
Instructions :

- Reference instructions in sequence : Spatial locality
- Looping through : Temporal locality

# Performance enhancement - Motivation



$$\cancel{200} + \cancel{20} \times 5 \rightarrow \cancel{200} + \cancel{10}$$
$$300 \cancel{7}$$

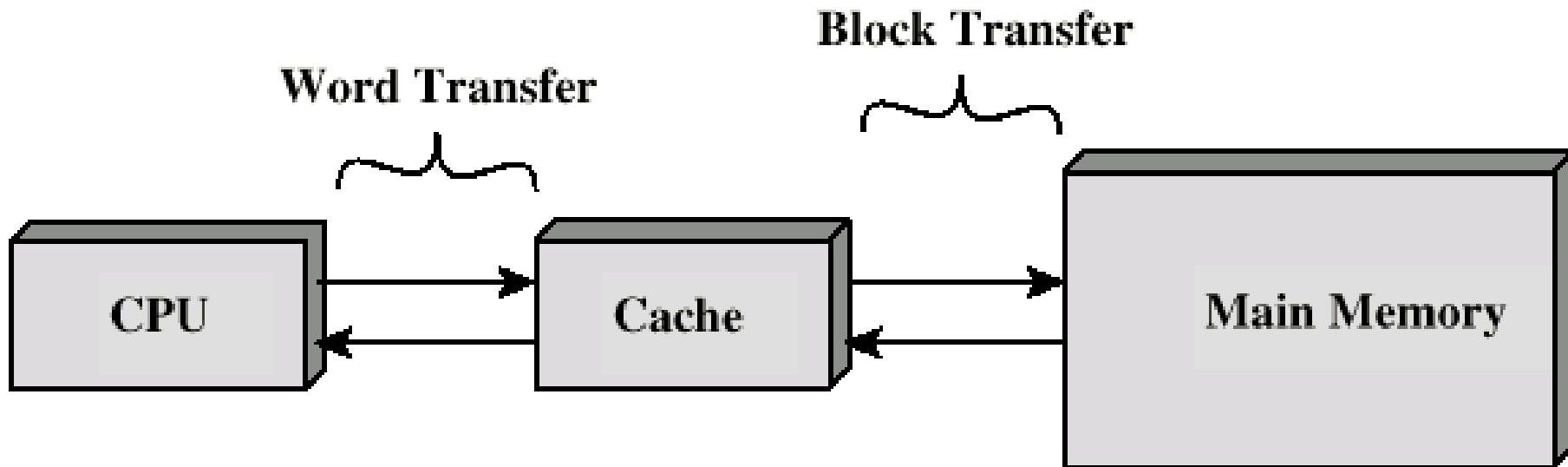


# Cache

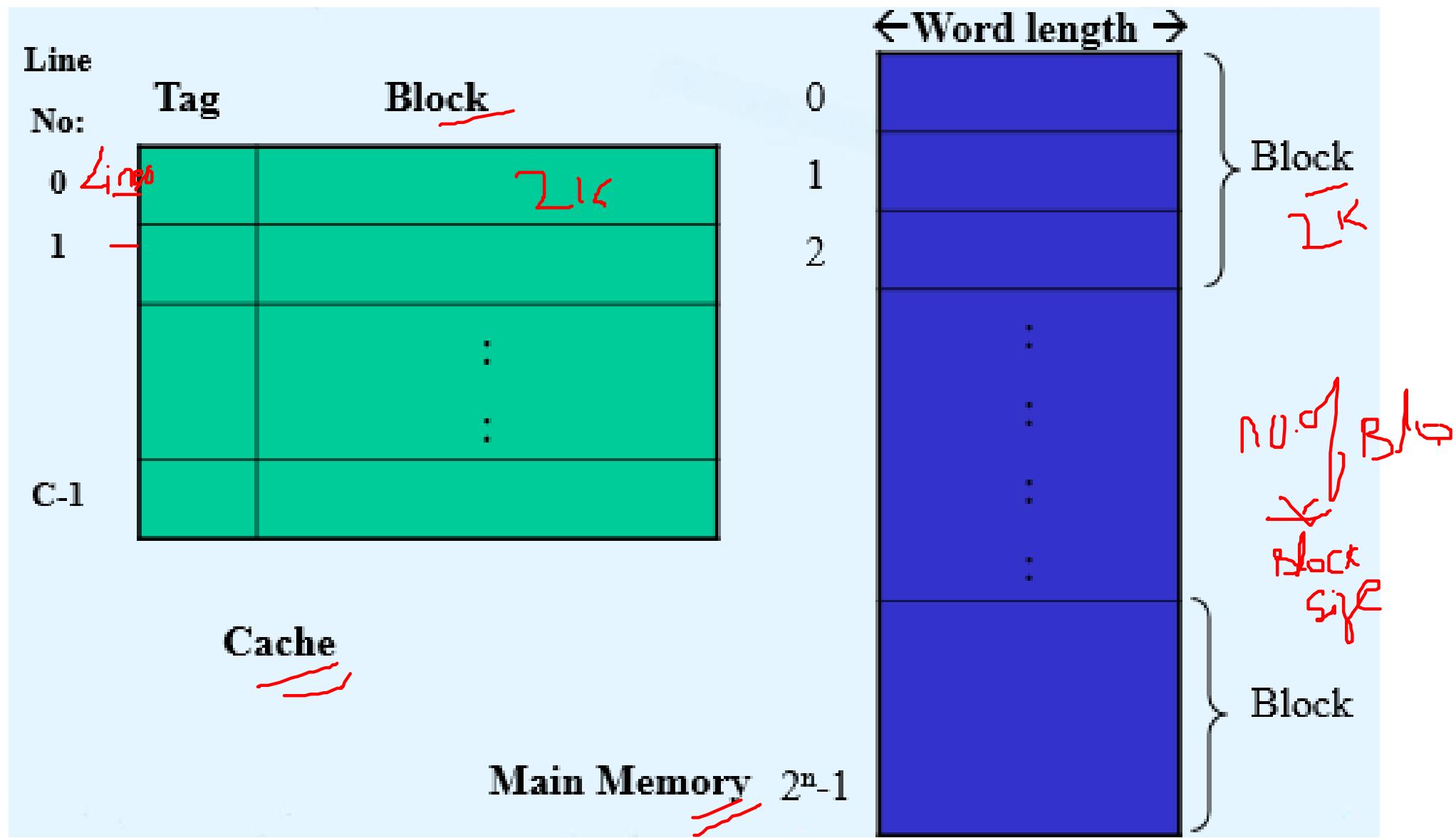
**BITS Pilani**  
Pilani Campus

# Cache

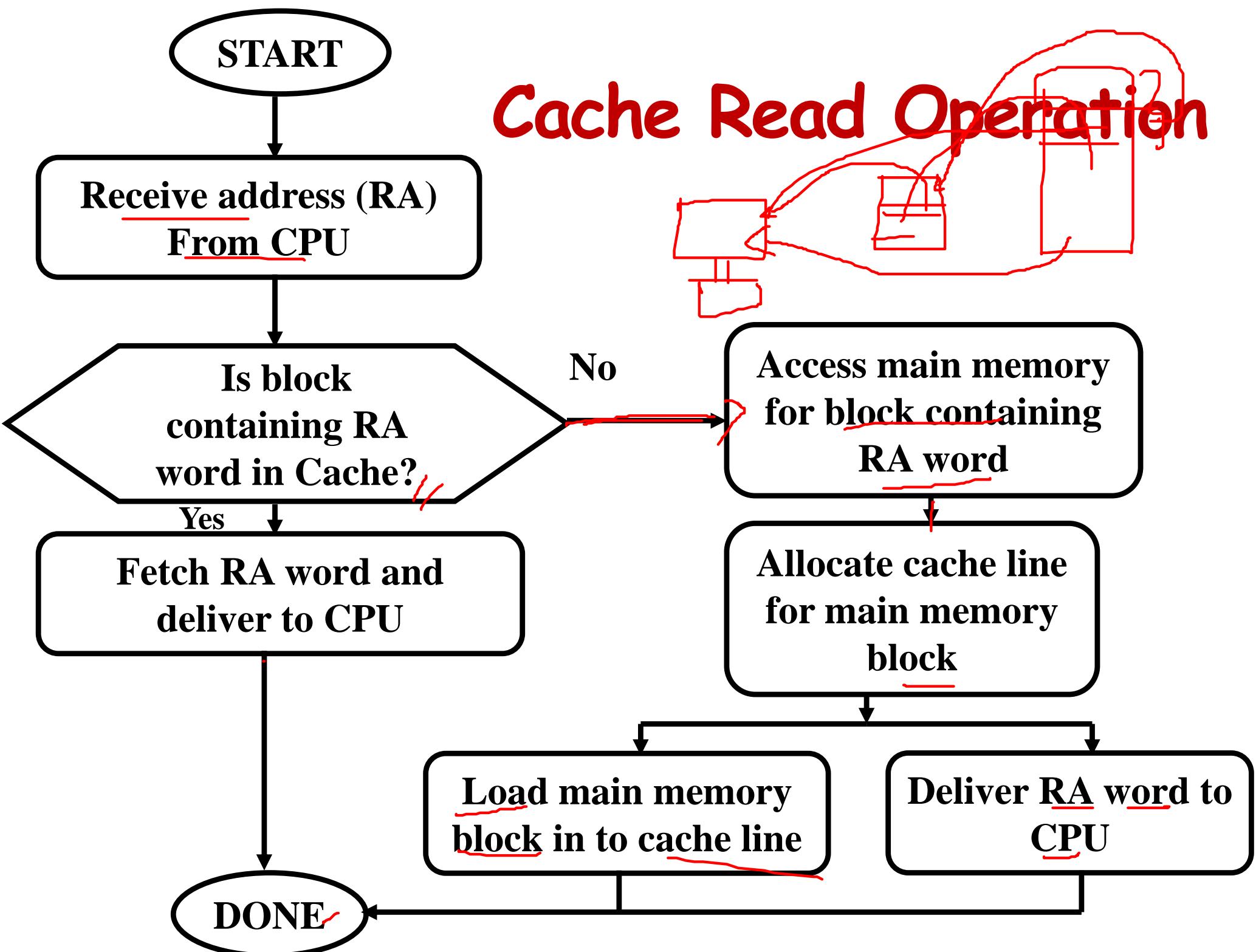
- Small, fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or separate module



# Cache and Main Memory Structure



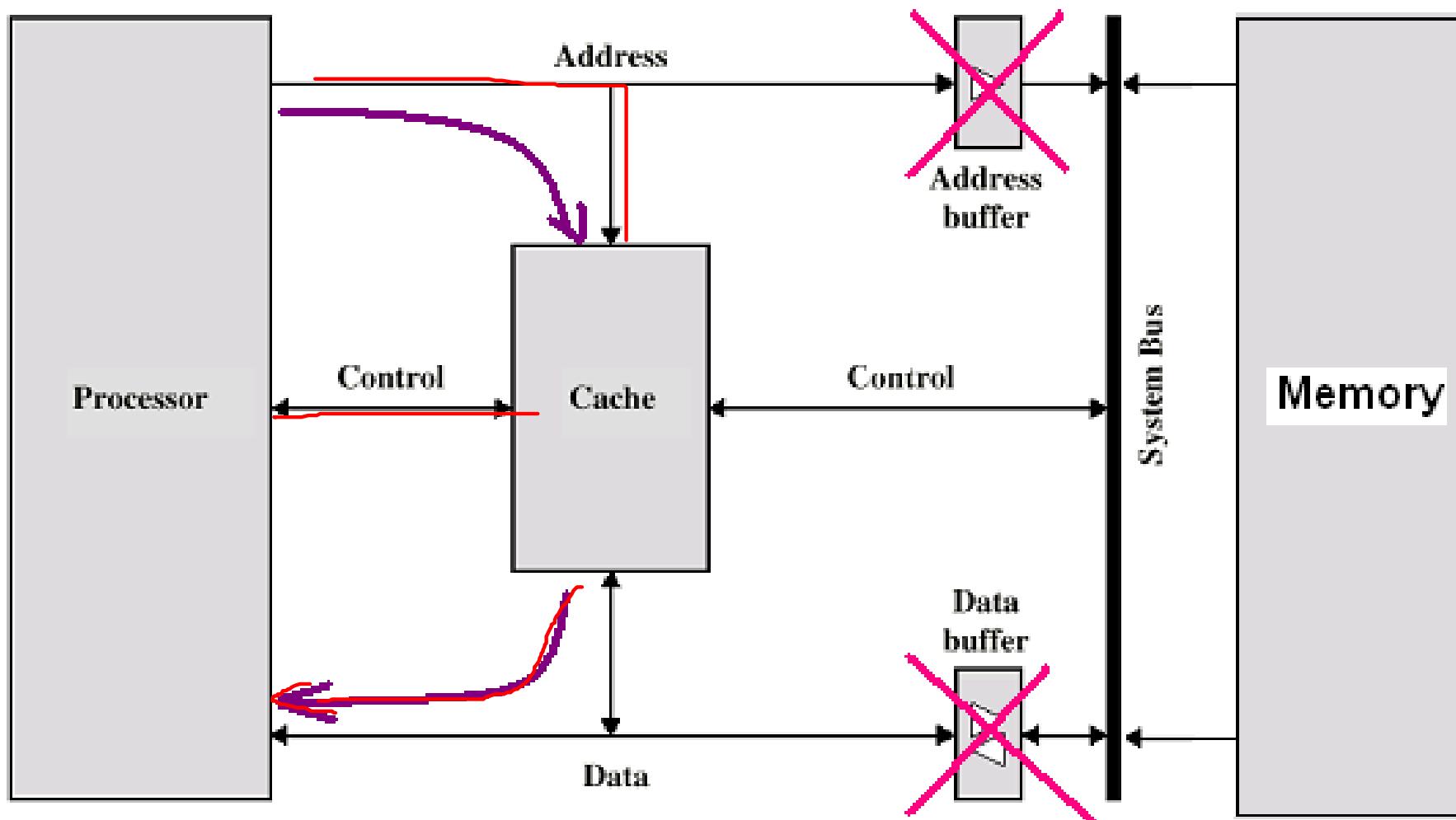
# Cache Read Operation



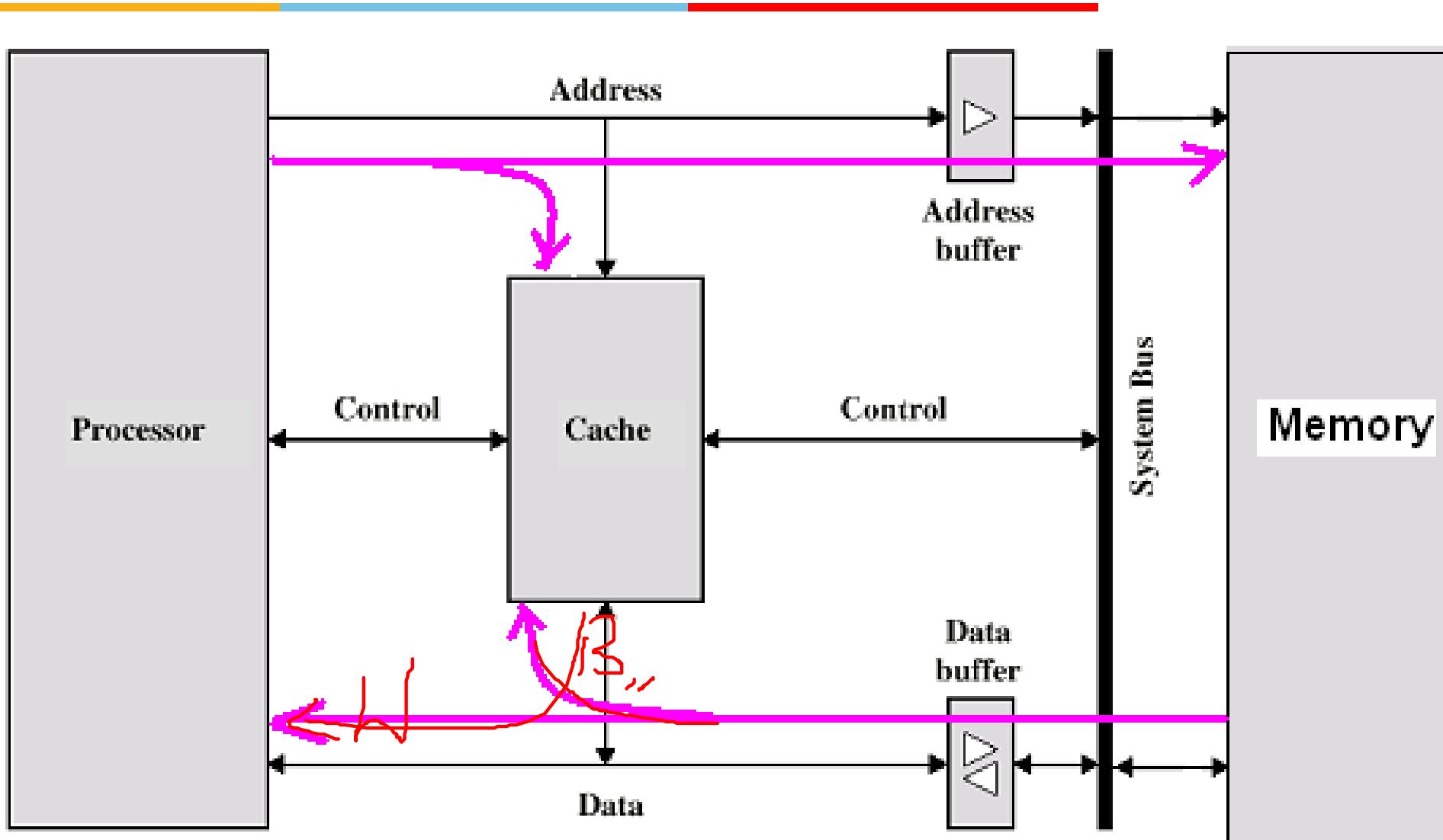
# Performance of cache

- Hit ratio : Number of Hits / total references to memory
- Hit ✓
- Miss ✓

# Read Hit



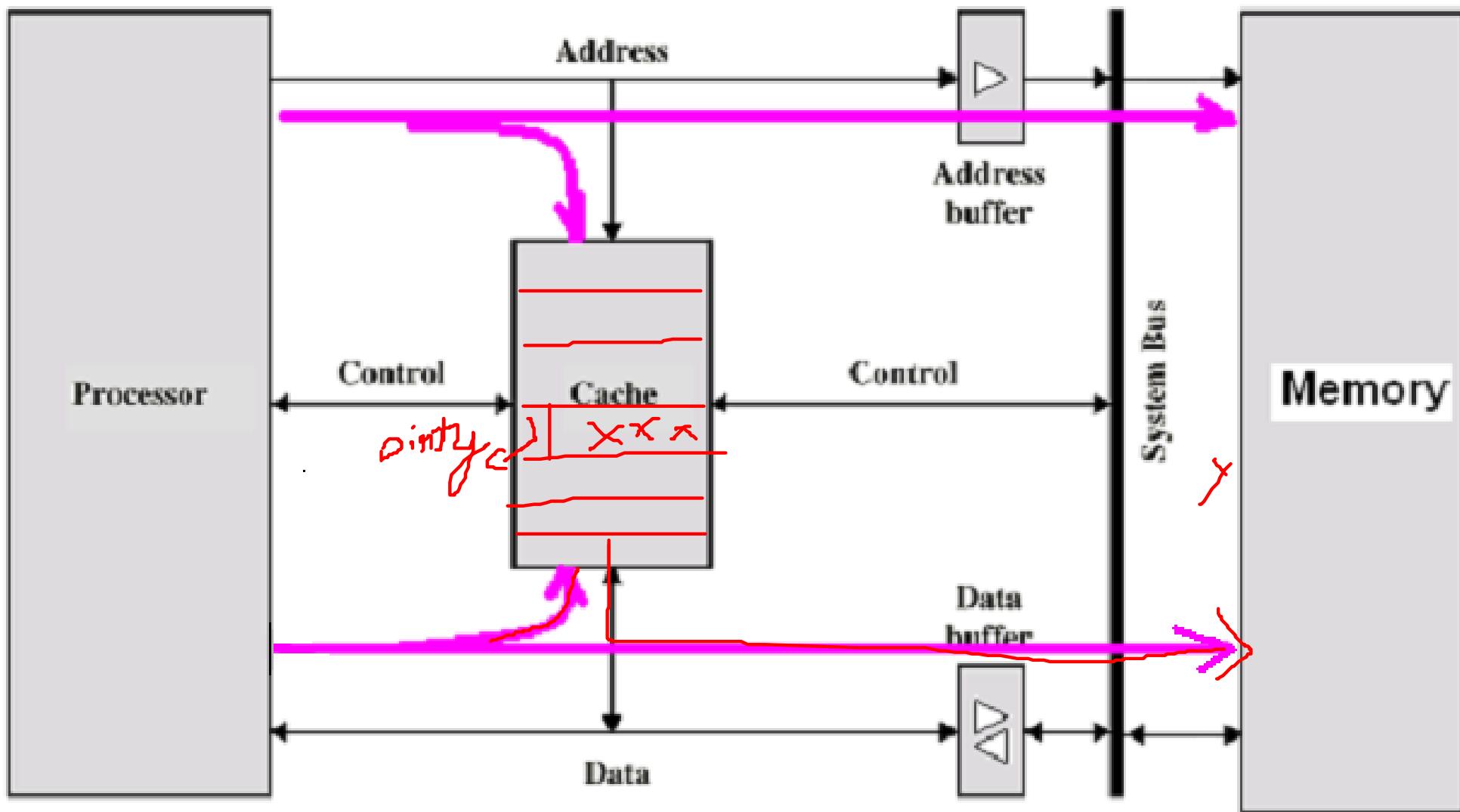
# Read Miss



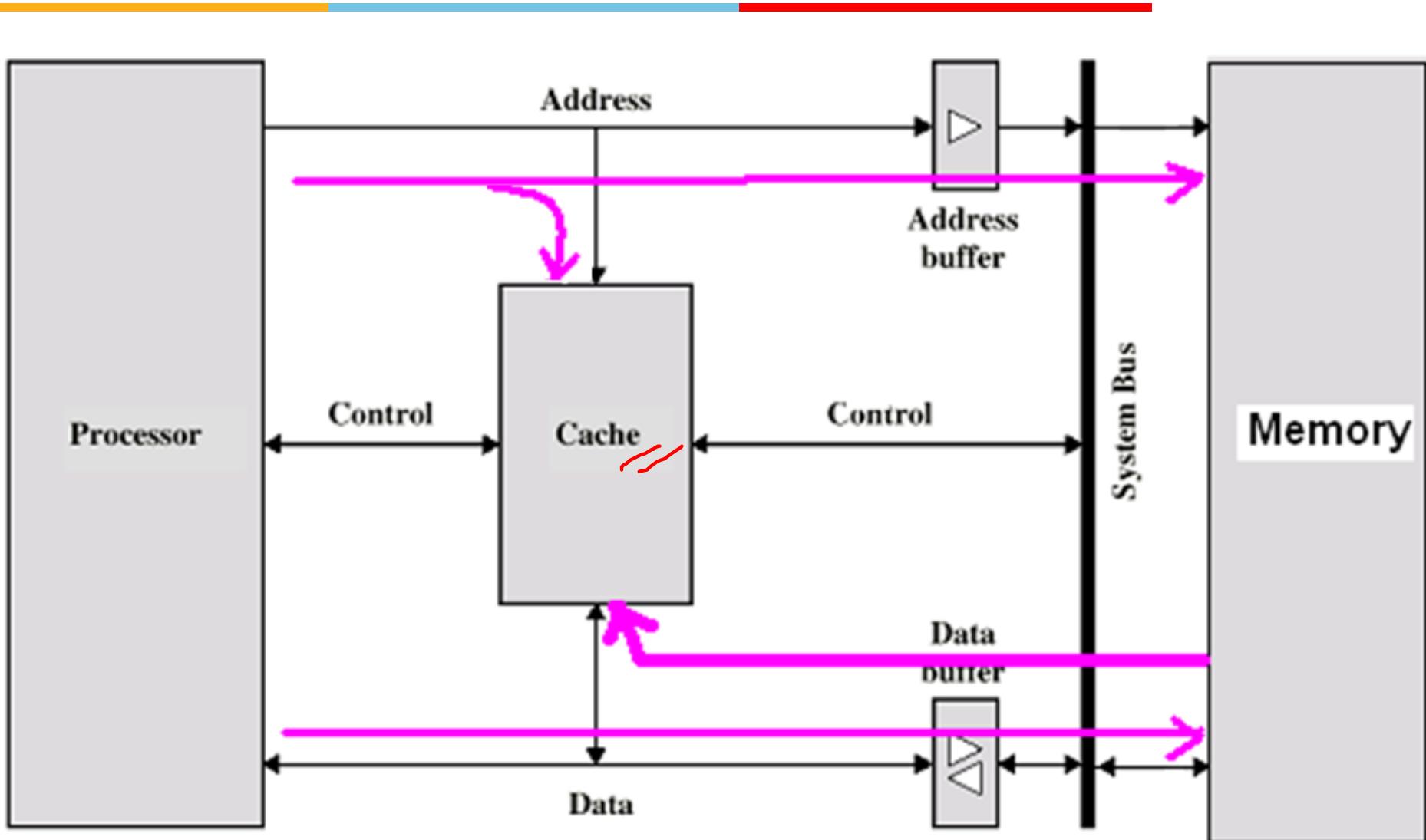
Miss Penalty

BITS Pilani, Pilani Campus

# Write hit



# Write miss



# Mapping Function

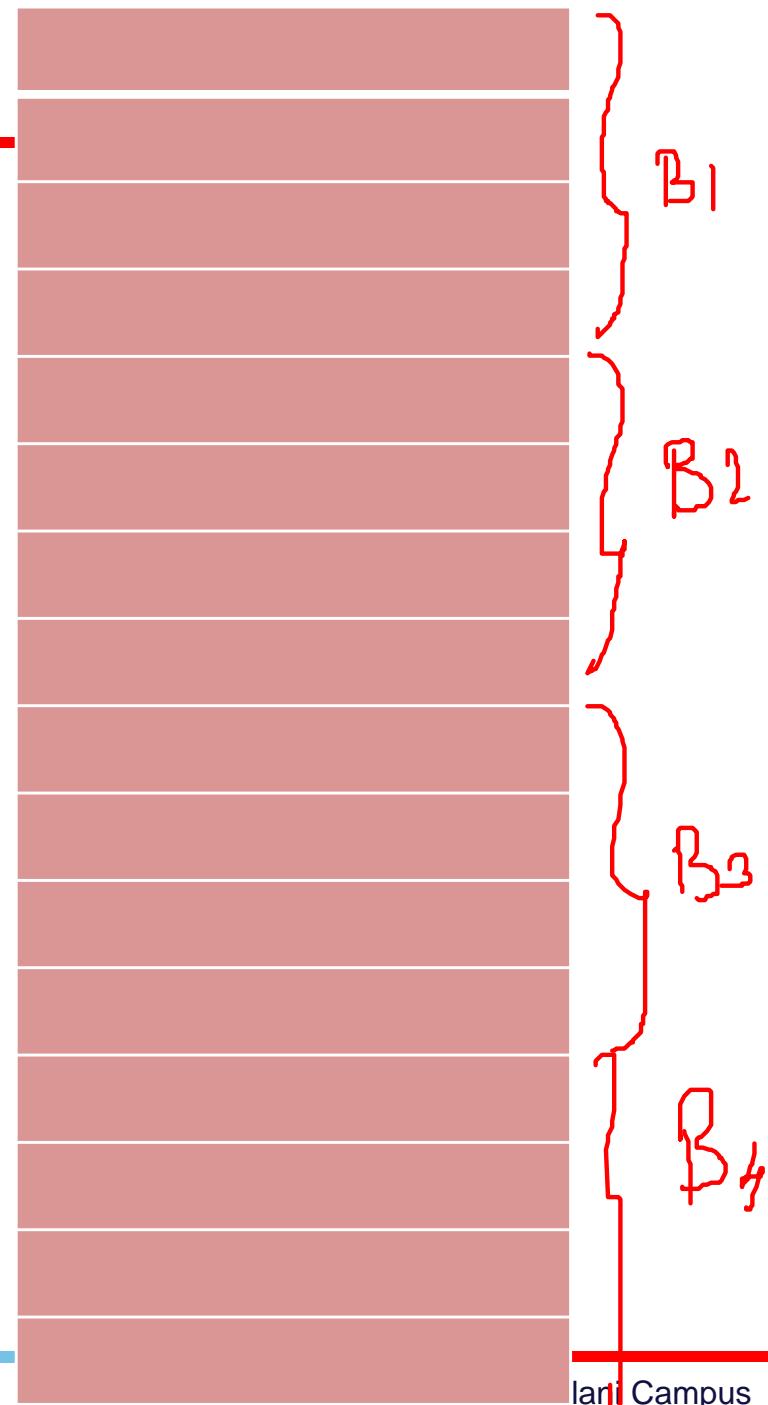
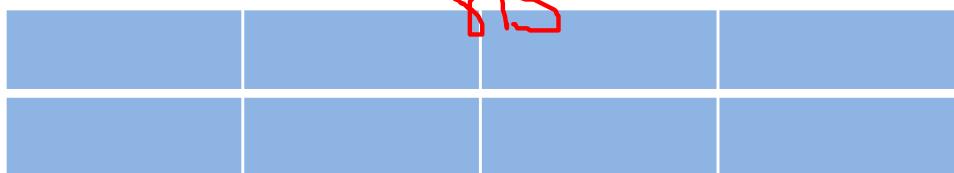
- How memory blocks are mapped to cache lines
- Three types
  - Direct mapping
  - Associative mapping
  - Set Associative mapping

# Direct Mapped Cache



- 16 Bytes main memory
  - How many address bits are required?
- Memory block size is  $\frac{16}{4} = 4$  bytes
- Cache of 8 Byte
  - How many cache lines?
  - cache contains 2 lines (4 bytes per Line)

$$\frac{16}{4} \Rightarrow 4$$



# Direct Mapped Cache

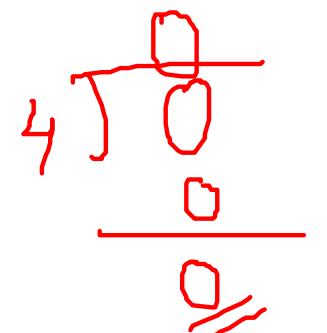
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place

$$i = j \bmod m$$

where  $i$  = cache line number

$j$  = main memory block no.

$m$  = no.of lines in the cache



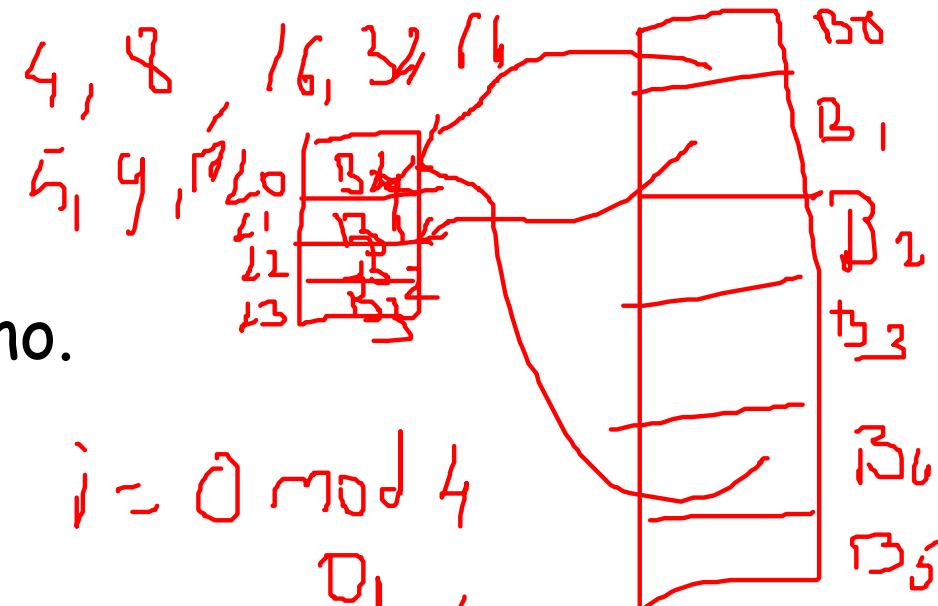
$$\equiv$$

$$i = 0 \bmod 4$$

$$= 1 \bmod 4$$

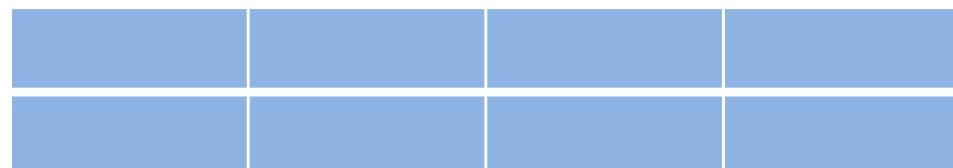
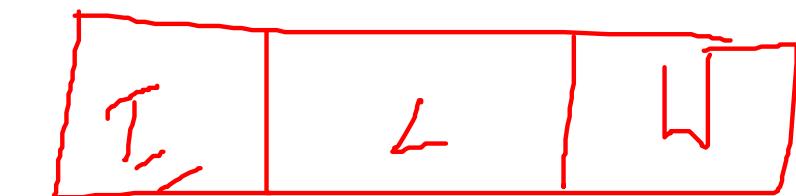
$$2 \bmod 4$$

$$4 \bmod 4$$



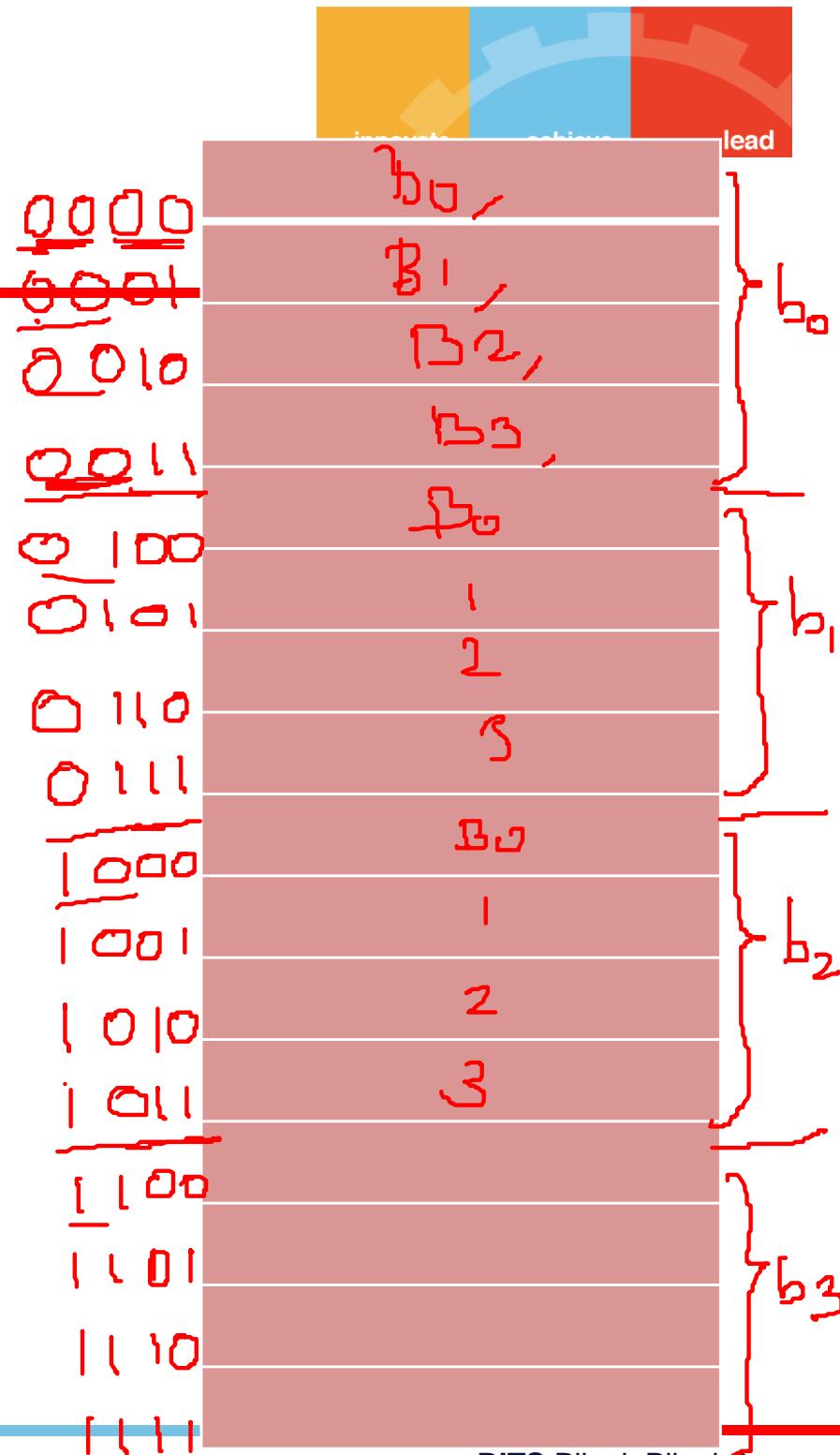
# Direct Mapped Cache

$$i = j \bmod m$$

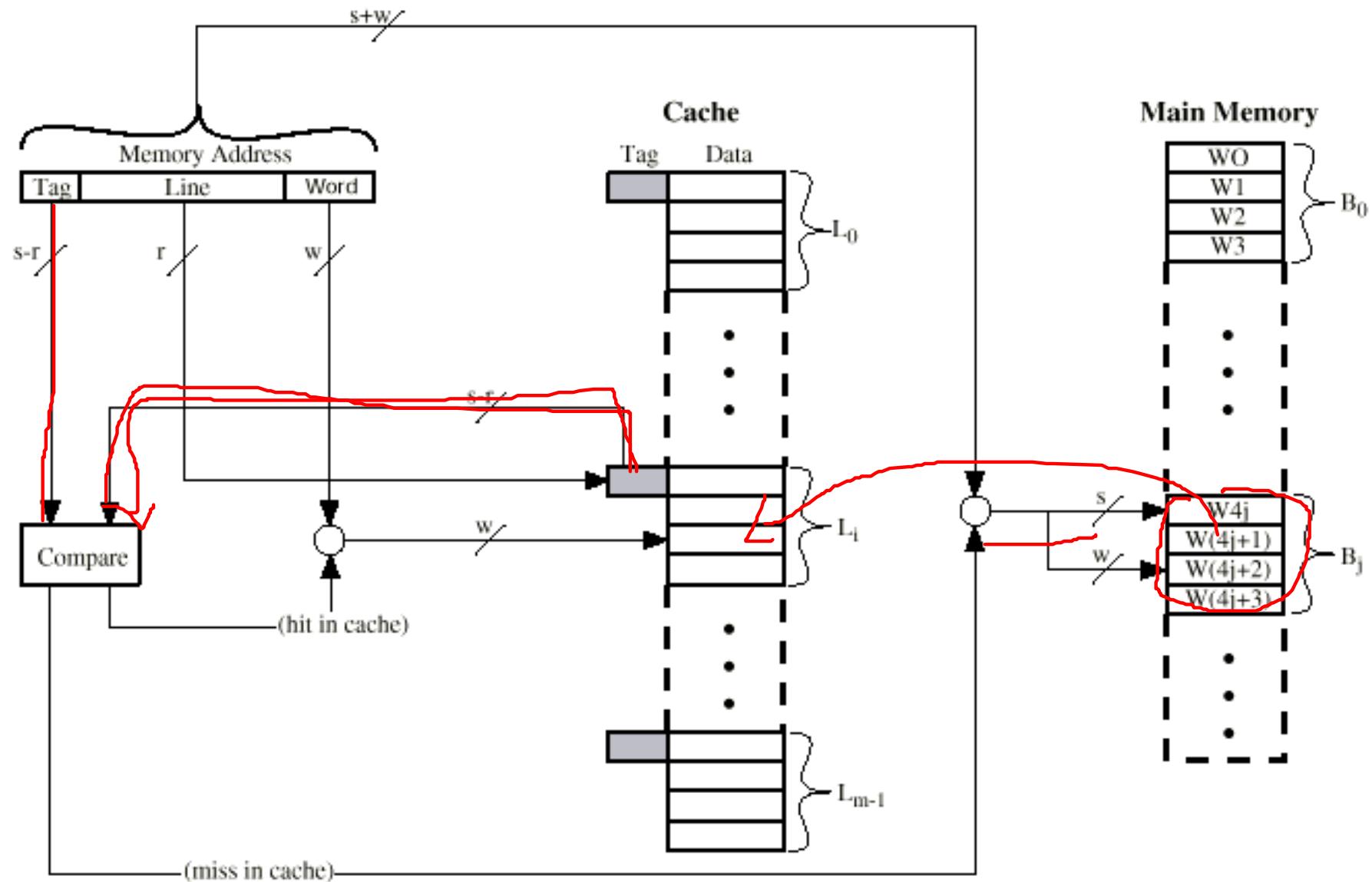


- Address is split in three parts:

- Tag
- Line
- Word



# Direct Mapped Cache Organization



# Direct mapped cache- Summary

Address length =  $(s+w)$  bits

Number of addressable units =  $2^{s+w}$  words or bytes

Block size = line size =  $2^w$  words or bytes

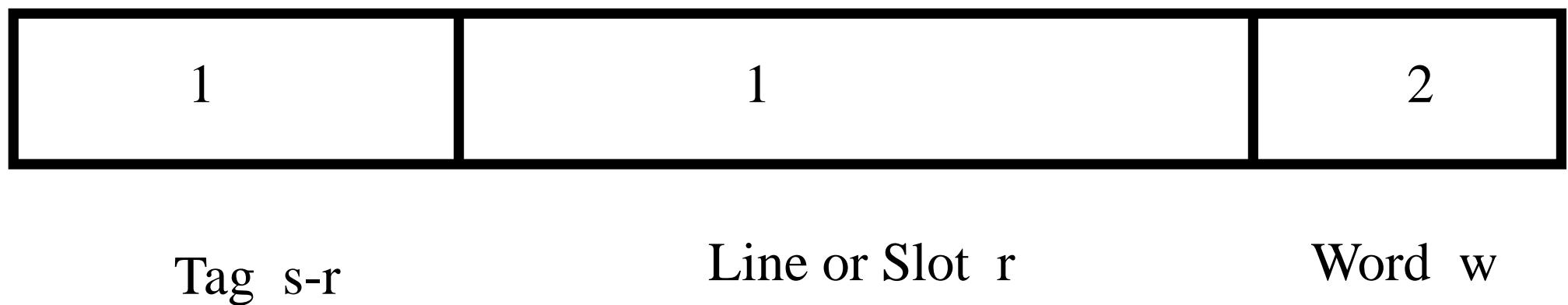
Number of blocks in main memory =  $2^{s+w} / 2^w = 2^s$

Number of lines in cache =  $m = 2^r$

Size of tag =  $(s-r)$  bits

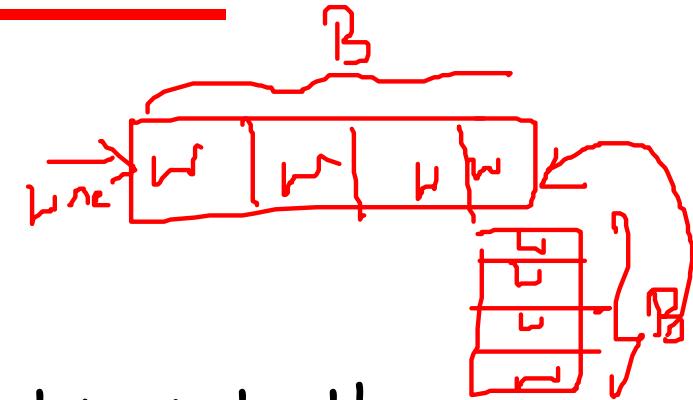


# Direct mapped cache- Summary



# Direct mapped cache-pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high



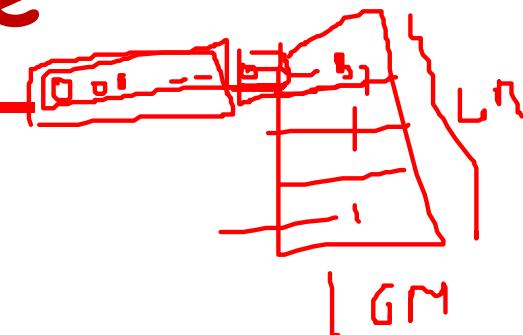
0, L, 0, 4, 1, 0, 4  
L 0

# Problem 1 : Direct Mapped Cache

Given :

$$\begin{aligned} L_1 &= 2^10 \\ L_2 &= 2^{12} \\ L_M &= 2^20 \end{aligned}$$

$$4 = 2^2$$



- Cache of 64KByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

- Number of bits required to address the main memory  $\Rightarrow 24$  bits,
- Number of blocks in main memory  $\Rightarrow 16M \rightarrow 16 \times 1M \rightarrow 2^4 + 2^{20} \Rightarrow 1024$
- Number of cache lines  $\Rightarrow 16K \rightarrow 16 \times 1K \rightarrow 2^4 \times 2^{10} \Rightarrow 4096$
- Number of bits required to identify a word (byte) in a block  $\Rightarrow 2$  bits
- Number of bits to identify a block  $\Rightarrow 22$  bits
- Tag, Line, Word



# Solution 1

Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

a) Number of bits required to address the main memory

24 bits

b) Number of blocks in main memory

4M

c) Number of cache lines

16 k

# Solution 1

Given :

- Cache of 64kByte, Cache block of 4 bytes
- 16MBytes main memory

Find out

d) Number of bits required to identify a word (byte) in a block?

2 bits

e) Number of bits required to identify a block

22 bits

f) Tag, Line, Word

Tag s-r	Line r	Word w
8	14	2

# Problem 2

Consider a machine with a byte addressable main memory of  $2^{16}$  bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?
- b. Into what line would bytes with each of the following addresses be stored?

0001 0001 0001 1011  
1100 0011 0011 0100  
1101 0000 0001 1101  
1010 1010 1010 1010

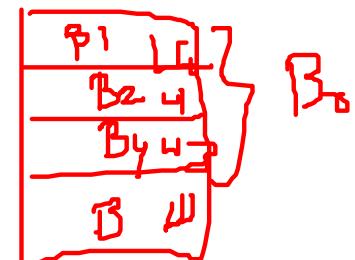
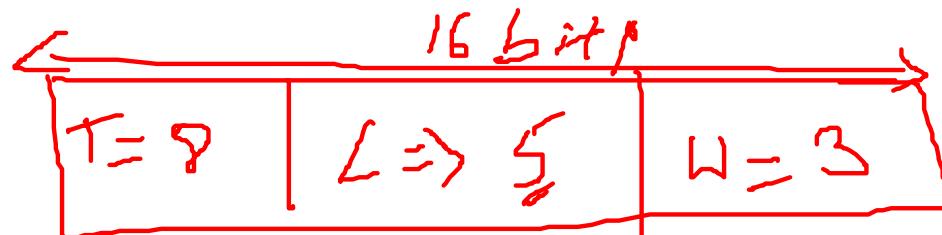
- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?
- d. How many total bytes of memory can be stored in the cache?
- e. Why is the tag also stored in the cache?

11

# Solution 2

Consider a machine with a byte addressable main memory of  $2^{16}$  bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- a. How is a 16-bit memory address divided into tag, line number, and byte number?



- b. Into what line would bytes with each of the following addresses be stored?

0001	0001	0001	1011	$\Rightarrow L_2$
1100	0011	0011	0100	$\Rightarrow L_6$
1101	0000	0001	1101	$\Rightarrow L_3$
1010	1010	1010	1010	$\Rightarrow L_{21}$

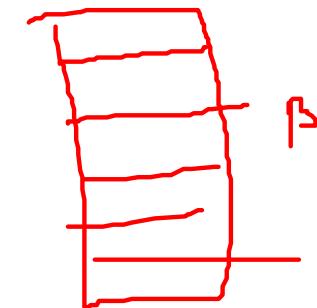
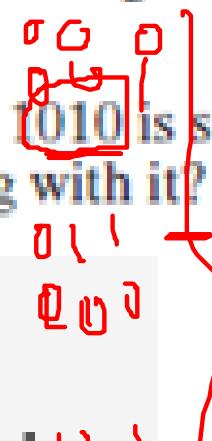
# Solution 2

Consider a machine with a byte addressable main memory of  $2^{16}$  bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

- c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

0001 1010 0001 1000  
 0001 1010 0001 1001  
**0001 1010 0001 1010**  
 0001 1010 0001 1011  
 0001 1010 0001 1100  
 0001 1010 0001 1101  
 0001 1010 0001 1110  
 0001 1010 0001 1111

: given in the problem



# Solution 2

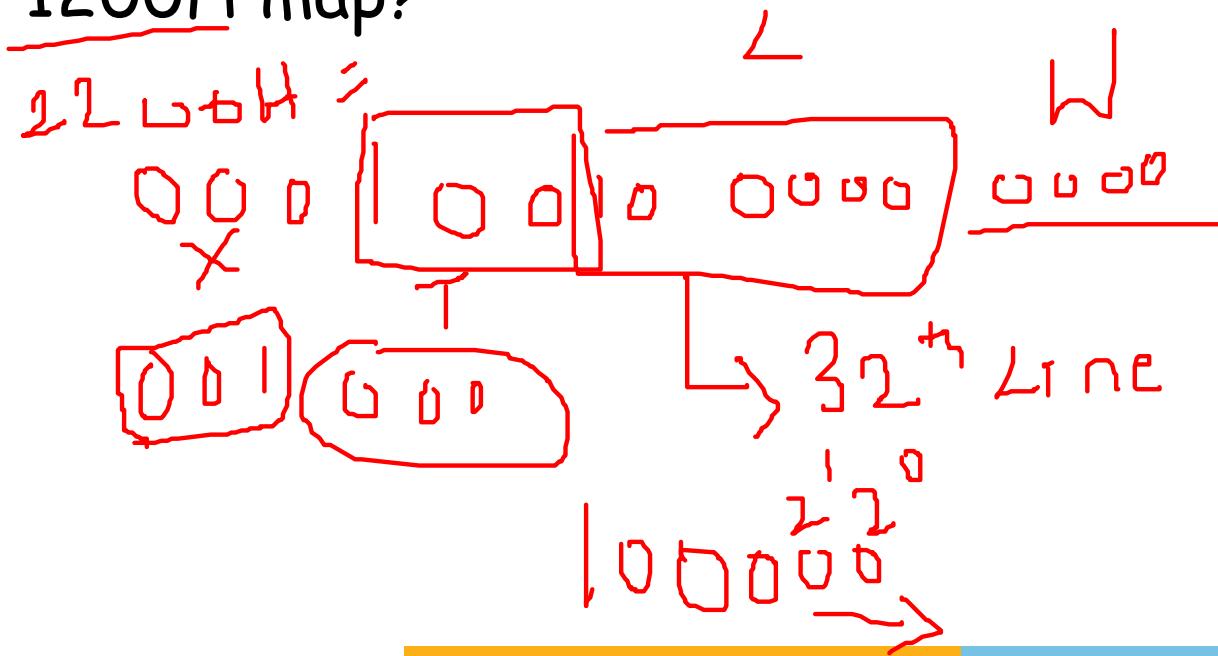
Consider a machine with a byte addressable main memory of  $2^{16}$  bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

d. How many total bytes of memory can be stored in the cache?

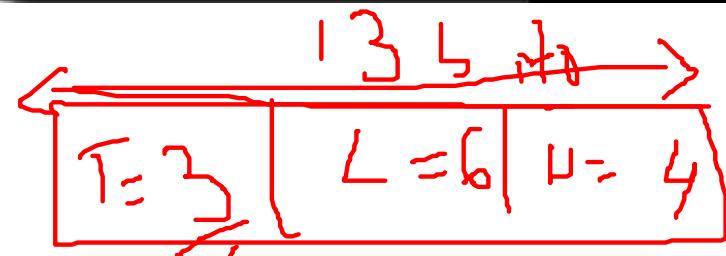
e. Why is the tag also stored in the cache?

# Problem 3

Consider a direct-mapped cache with 64 cache lines and a block size of 16 bytes and main memory of 8K (Byte addressable memory). To what line number does byte address 1200H map?



	Hexadecimal	Binary	Decimal
0		0000	0
1		0001	1
2		0010	2
3		0011	3
4		0100	4
5		0101	5
6		0110	6
7		0111	7
8		1000	8
9		1001	9
A		1010	10
B		1011	11
C		1100	12
D		1101	13
E		1110	14
F		1111	15





# Computer Organization and Software Systems

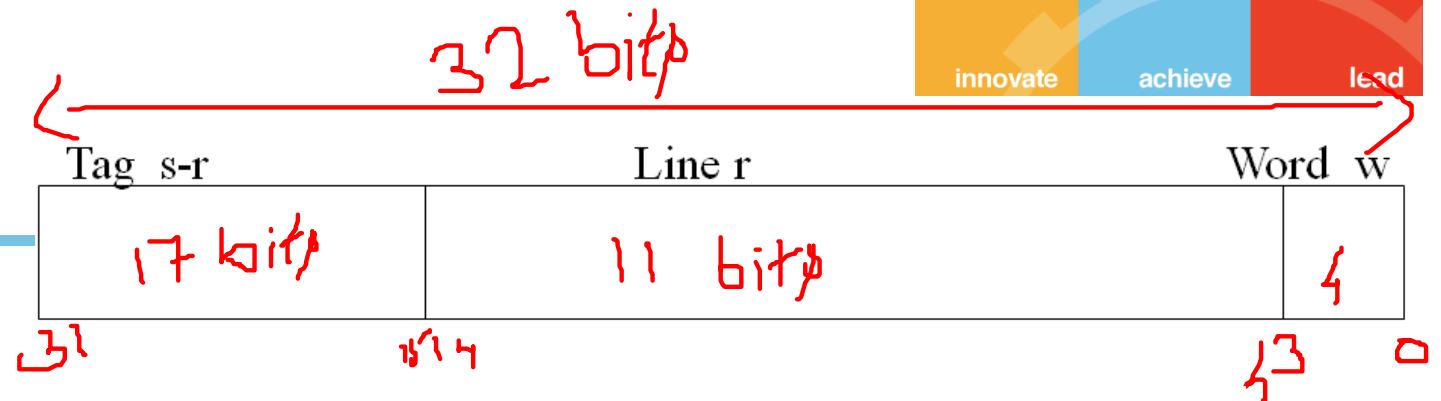
## CONTACT SESSION 4



**BITS** Pilani  
Pilani Campus

Dr. Lucy J. Gudino  
WILP & Department of CS & IS

# Problem 4



The system uses a L1 cache with direct mapping and 32-bit address format is as follows:

bits 0 - 3 = offset (word)

bits 4 - 14 = index bits (Line)

bits 15 - 31 = tag

- a) What is the size of cache line?  $\Rightarrow 2^4 \Rightarrow 16 \text{ Bytes}$
- b) How many Cache lines are there?  $\Rightarrow 2^{11} \Rightarrow 2K \text{ Lines} \Rightarrow 2 \times 2^{10}$
- c) How much space is required to store the tags in the L1 cache?  $\Rightarrow 17 \times 2K \Rightarrow 34K \text{ bits}$
- d) What is the total Capacity of cache including tag storage?  $\Rightarrow 34 \text{ Kbits} + 2K \times 16 \text{ bytes}$   
 ~~$34 \text{ Kbits} + 32 \text{ K Bytes}$~~

# Problem 5

- 16 Bytes main memory,  
Memory block size is 4 bytes,  
Cache of 8 Byte (cache is 2  
lines of 4 bytes each )
- Block access sequence :
- Find out hit ratio.

~~0 2 0 2 2 0 0 2 0 0 0 2 1~~

~~X X ✓ X X ✓ X X ✓ X X~~  
~~= = = = = = = = = = = =~~  
~~4 / 13 => 30%.~~

$$\begin{aligned}
 i &= j \bmod m \\
 &= 0 \bmod 2 \\
 &= 0 \\
 i &= 2 \bmod L = 6 \\
 &= 0 \bmod 2 = 0 \\
 &= 2 \bmod 2 = 0 \\
 \end{aligned}$$

$L_0$    
 $L_1$    
 $4 \text{ B}$   
 $i \bmod L = 1$

# Problem 6

$$\begin{aligned}1K &= 2^{10} \\1M &= 2^{20} \\1G &= 2^{30}\end{aligned}$$


---

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache.

Answer the following questions:

- a) What is the number of bits needed to address the main memory?  $\Rightarrow 20 \text{ bits}$
- b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

# Problem 6

---

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache.

Answer the following questions:

- a) What is the number of bits needed to address the main memory?

20 bits

- a) If the cache hit ratio is 95%, what is the average access time for a memory reference?
-

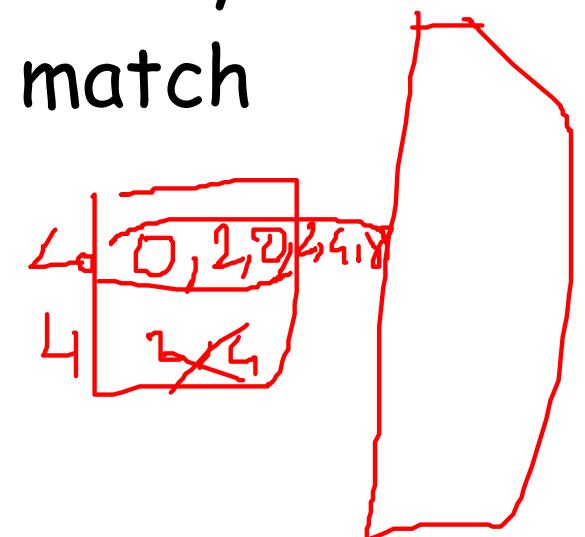
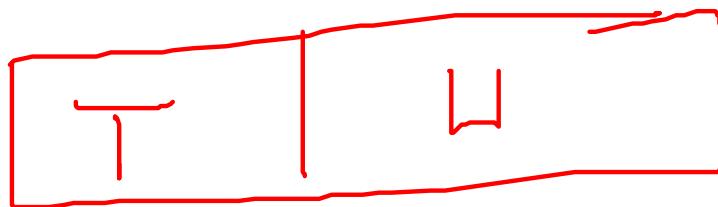
# Solution 6

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

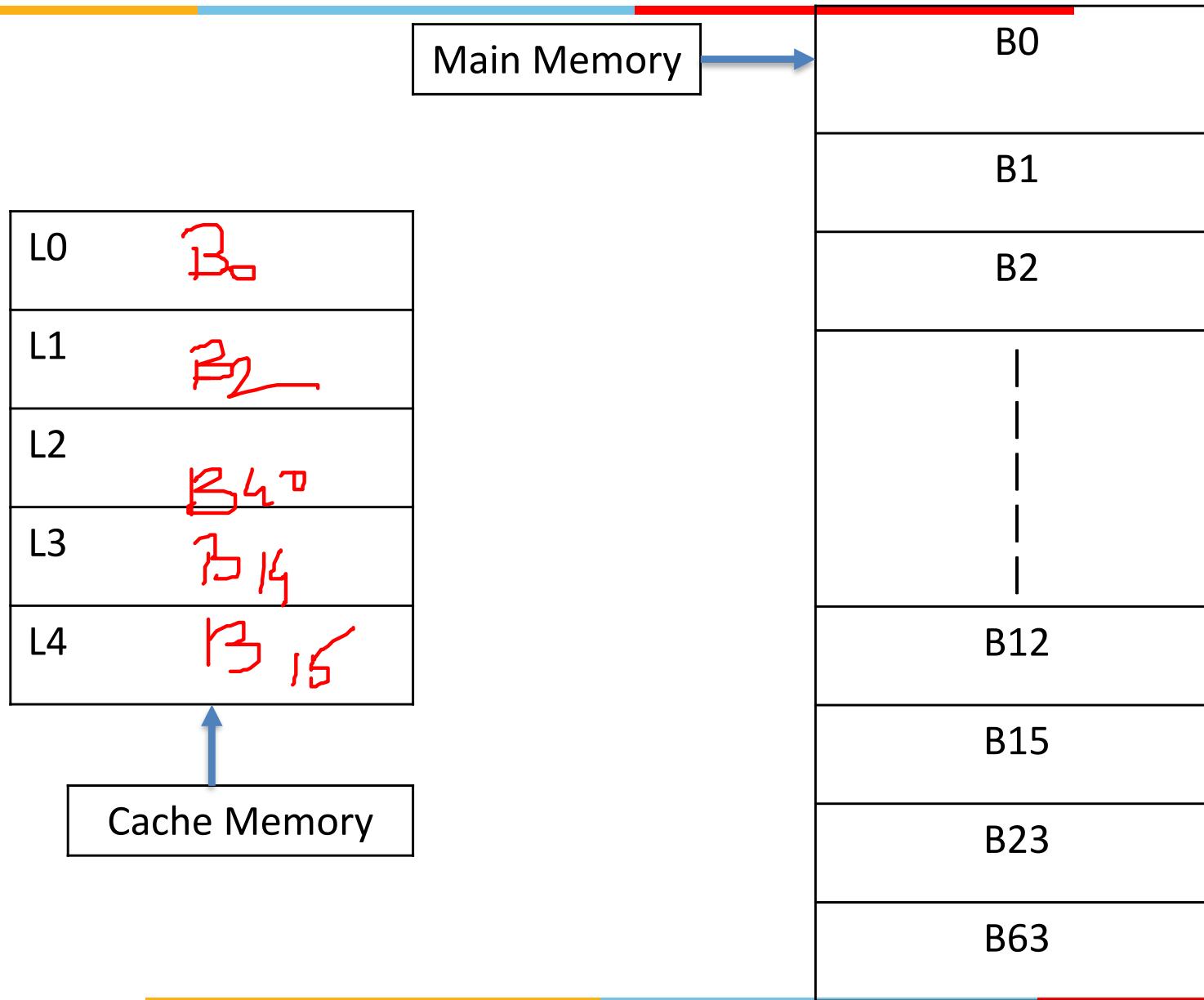
$$\begin{aligned}\text{Avg access time} &= \text{hit ratio} * \text{cache access} + \\ &\quad (1 - \text{hit ratio}) * (\text{cache access} + \text{memory access}) \\ &= .95 * 0.1 \text{ microsec} + .05 * (1 + 0.1) \text{ microsec} \\ &= .095 + .055 \text{ microsec} \\ &= 0.15 \text{ microseconds}\end{aligned}$$

# Associative Mapping

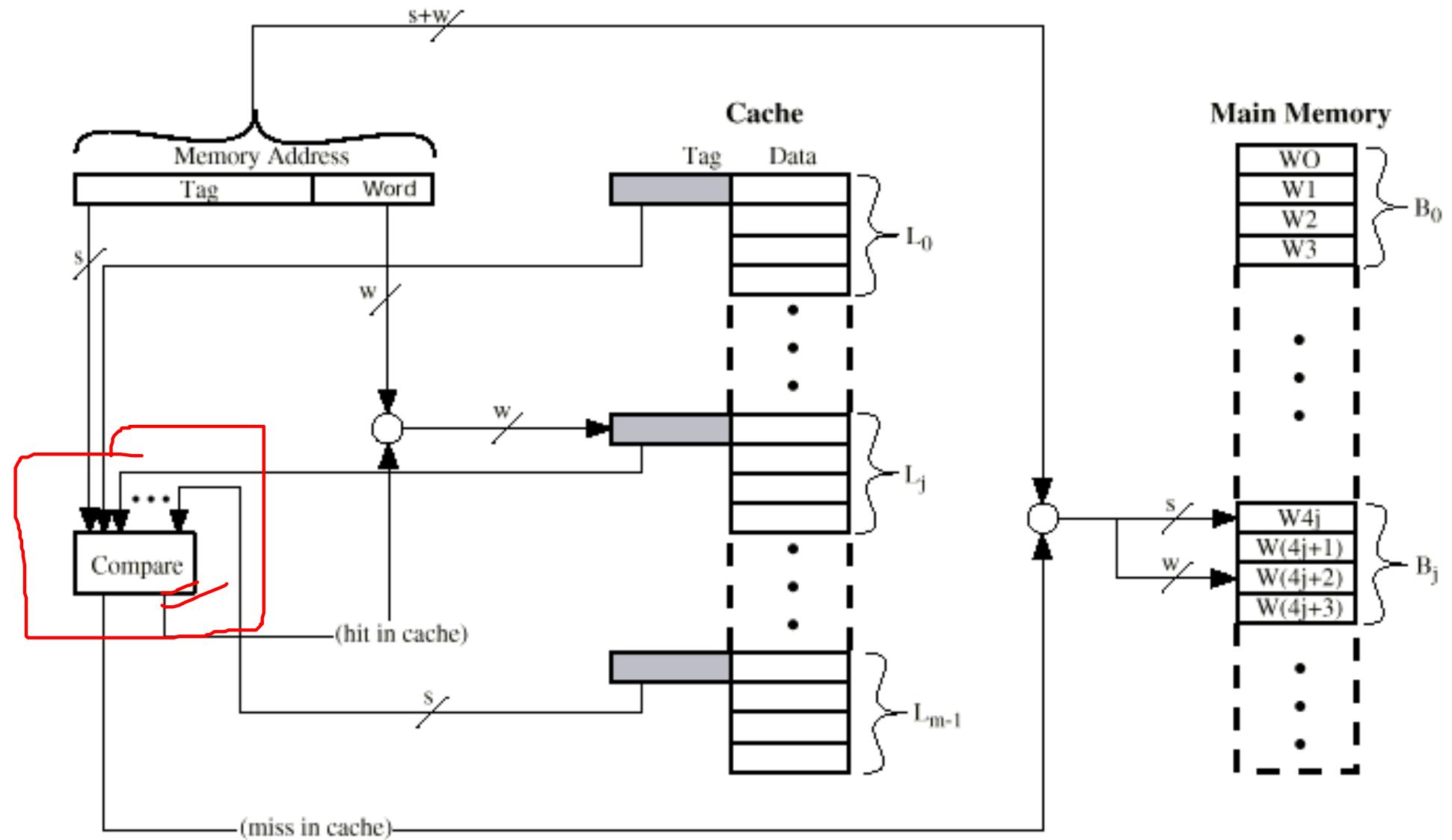
- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive



# Associative Mapping Cache Organization



# Associative Cache Organization



# Associative Mapping Summary

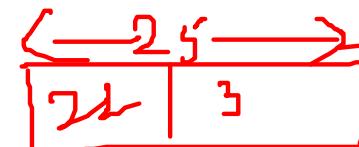
- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{s+w}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{s+w}/2^w = 2^s$
- Number of lines in cache = undetermined
- Size of tag =  $s$  bits

# Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory  $\Rightarrow 32M \Rightarrow 2^5 * 2^{20} \Rightarrow 25 \text{ bits}$
- b) Number of blocks in main memory  $\Rightarrow 32M / 8 \Rightarrow 4M$
- c) Number of cache lines  $\Rightarrow 128K / 8 \Rightarrow 16K$
- d) Number of bits required to identify a word (byte) in a block?  $\Rightarrow 3 \text{ bits}$
- e) Tag, Word 

# Problem 7

---

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory = 25 bits
- b) Number of blocks in main memory
- c) Number of cache lines
- d) Number of bits required to identify a word (byte) in a block?
- e) Tag, Word

# Problem 7

---

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory=25 bits
- b) Number of blocks in main memory= $2^{22}$  blocks or  
4M blocks
- c) Number of cache lines
- d) Number of bits required to identify a word (byte) in a block?
- e) Tag, Word

# Problem 7

---

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory=25 bits
- b) Number of blocks in main memory= $2^{22}$  blocks or  
4M blocks
- c) Number of cache lines= 16k Lines
- d) Number of bits required to identify a word (byte) in a block?
- e) Tag, Word

# Problem 7

---

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory=25 bits
- b) Number of blocks in main memory= $2^{22}$  blocks or  
4M blocks
- c) Number of cache lines= 16k Lines
- d) Number of bits required to identify a word (byte) in a block?= 3bits
- e) Tag, Word

# Problem 7

---

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

- a) Number of bits required to address the memory=25 bits
- b) Number of blocks in main memory= $2^{22}$  blocks or  
4M blocks
- c) Number of cache lines= 16k Lines
- d) Number of bits required to identify a word (byte) in a block?= 3bits
- e) Tag, Word = 22 bits, 3bits

# Problem 8

---

Cache of 64kByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = \_\_\_\_\_

Number of lines in the cache memory = \_\_\_\_\_

Word bits = \_\_\_\_\_

Tag bits = \_\_\_\_\_

# Problem 8

---

Cache of 64kByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = 24bits

Number of lines in the cache memory = 16K lines

Word bits = 2 bits

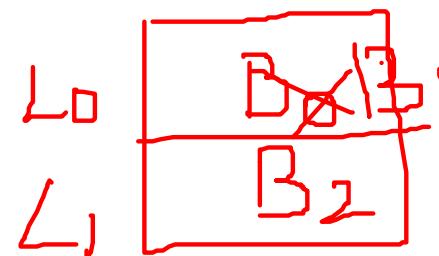
Tag bits = 22 bits

# Problem 9

- 16 Bytes main memory, Memory block size is 4 bytes, Cache of 8 Byte (cache is 2 lines of 4 bytes each ) and associative mapping
- Block access sequence :

0 2 0 2 2 0 0 2 0 0 0 2 1  
 ✗ ✗ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✗

- Find out hit ratio.



$$10/13 \Rightarrow 75\%$$

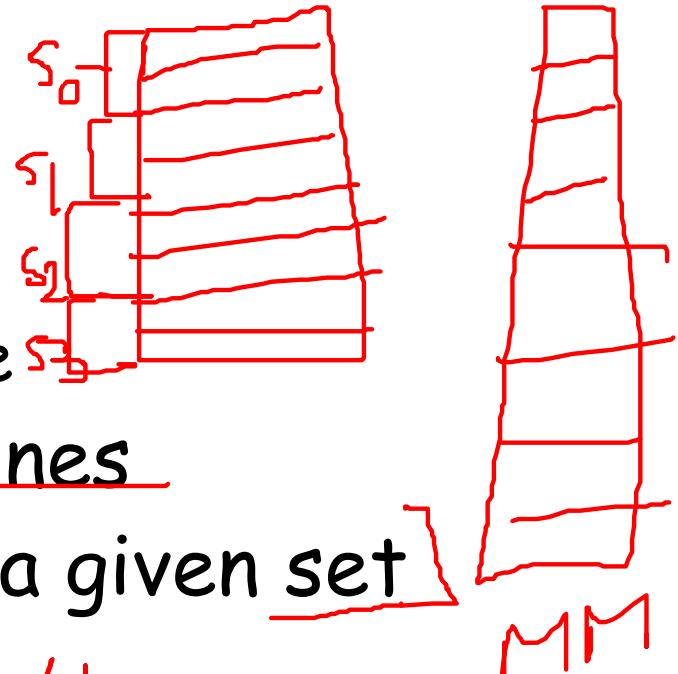
# Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
7-8	<ul style="list-style-type: none"> <li>• <b>Memory Organization (Contd..)</b></li> <li>• Cache Memories (Contd..)           <ul style="list-style-type: none"> <li>• Set Associative Caches</li> <li>• Issues with Writes <i>(marked with a red underline)</i></li> <li>• Performance Impact of Cache Parameters</li> <li>• Writing Cache friendly Codes</li> </ul> </li> <li>• Replacement Algorithms <i>(marked with a red underline)</i></li> </ul>	T1, R1

# Set Associative Mapping

- m line Cache is divided into a number of sets ( $v$  sets each with  $k$  lines)
- $m = v * k$
- $i = j \text{ modulo } v$   
where  $i = \text{cache set number}$   
 $j = \text{main memory block number}$   
 $m = \text{number of lines in the cache}$
- Each set contains ' $k$ ' number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set 2-way Set<sub>AB</sub>
  - 2 way set associative mapping
  - A given block can be in one of 2 lines in only one set

Tag	Set	Word
-----	-----	------



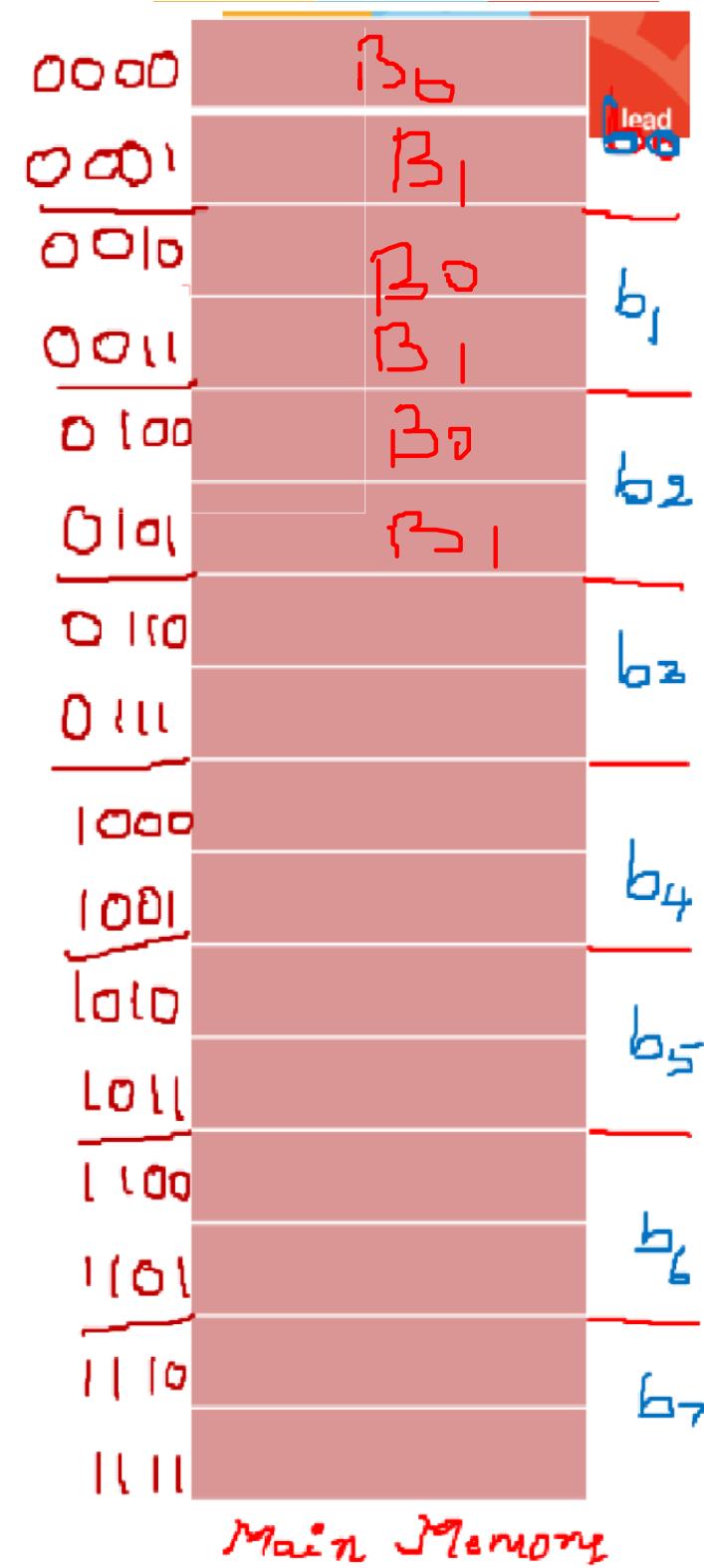
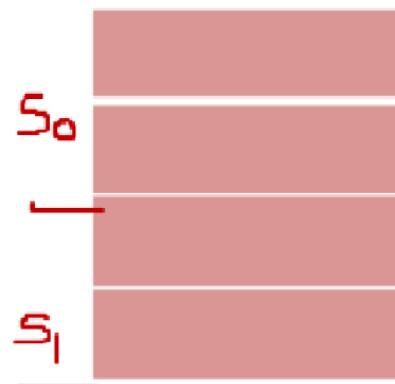
2-way Set<sub>AB</sub>  
4-way  
K-way

# Example

$$16 \geq 2^4$$

- 16 Bytes main memory, Block Size is 2 Bytes,
- Cache of 8 Bytes, 2 way set associative cache

• # address bits  $\Rightarrow$  4 bits  
 • Cache line size  $\Rightarrow$  2 Bytes  
 • # main memory blocks  $\Rightarrow$  8 Block Cache Memory  
 • # Number of cache lines  $\Rightarrow$  4 Cache  
 • # lines per set  $\Rightarrow$   $4/2 \Rightarrow 2$   
 • # of sets  $\Rightarrow$  2 set  
 K-Way 2-Way  
 LRU



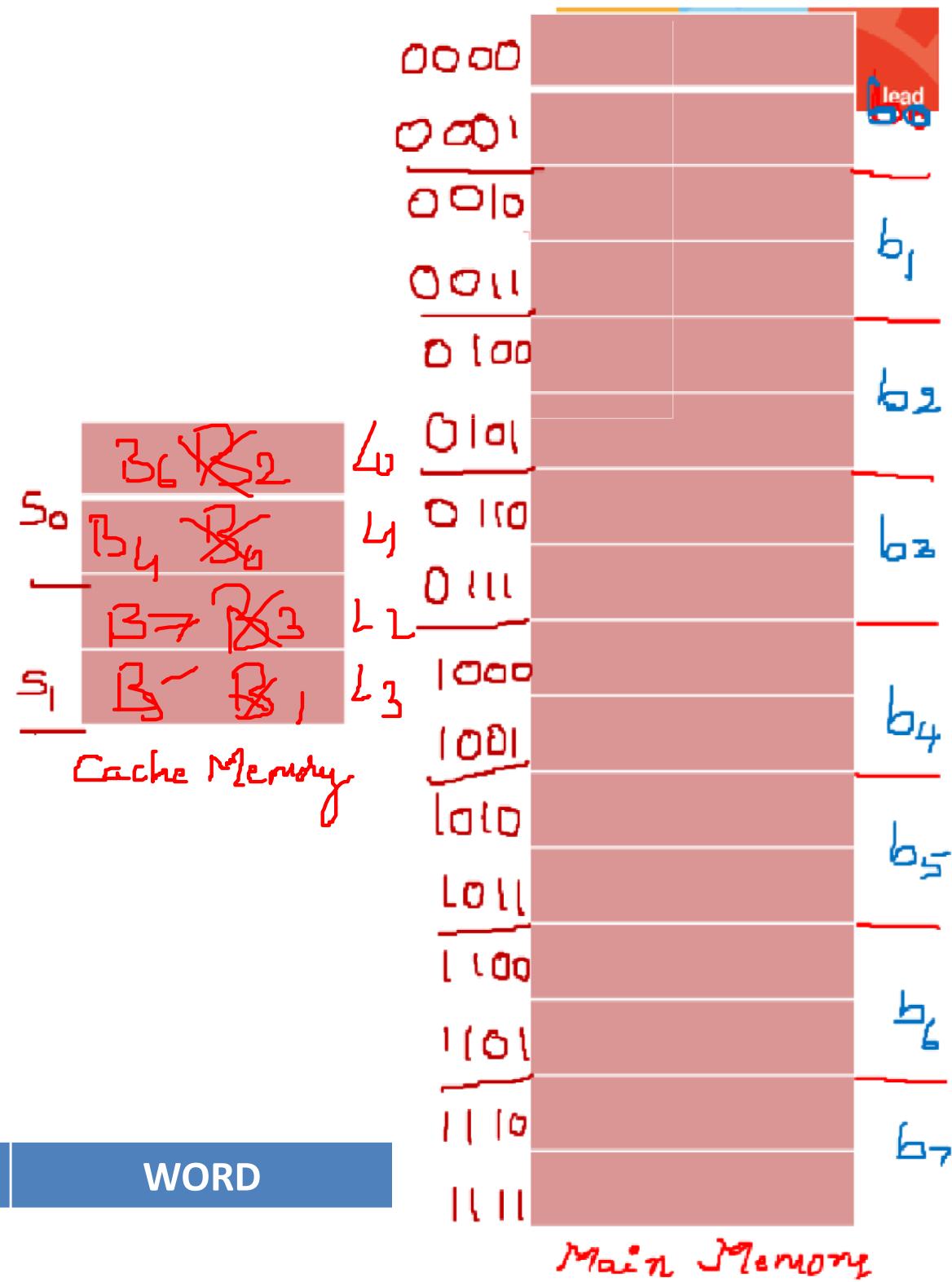
# Example - Mapping Function

$i = j \text{ modulo } v$	Set #
0%2	0
1%2	1
2%2	0
3%2	1
4%2	0
5%2	1
6%2	0
7%2	1

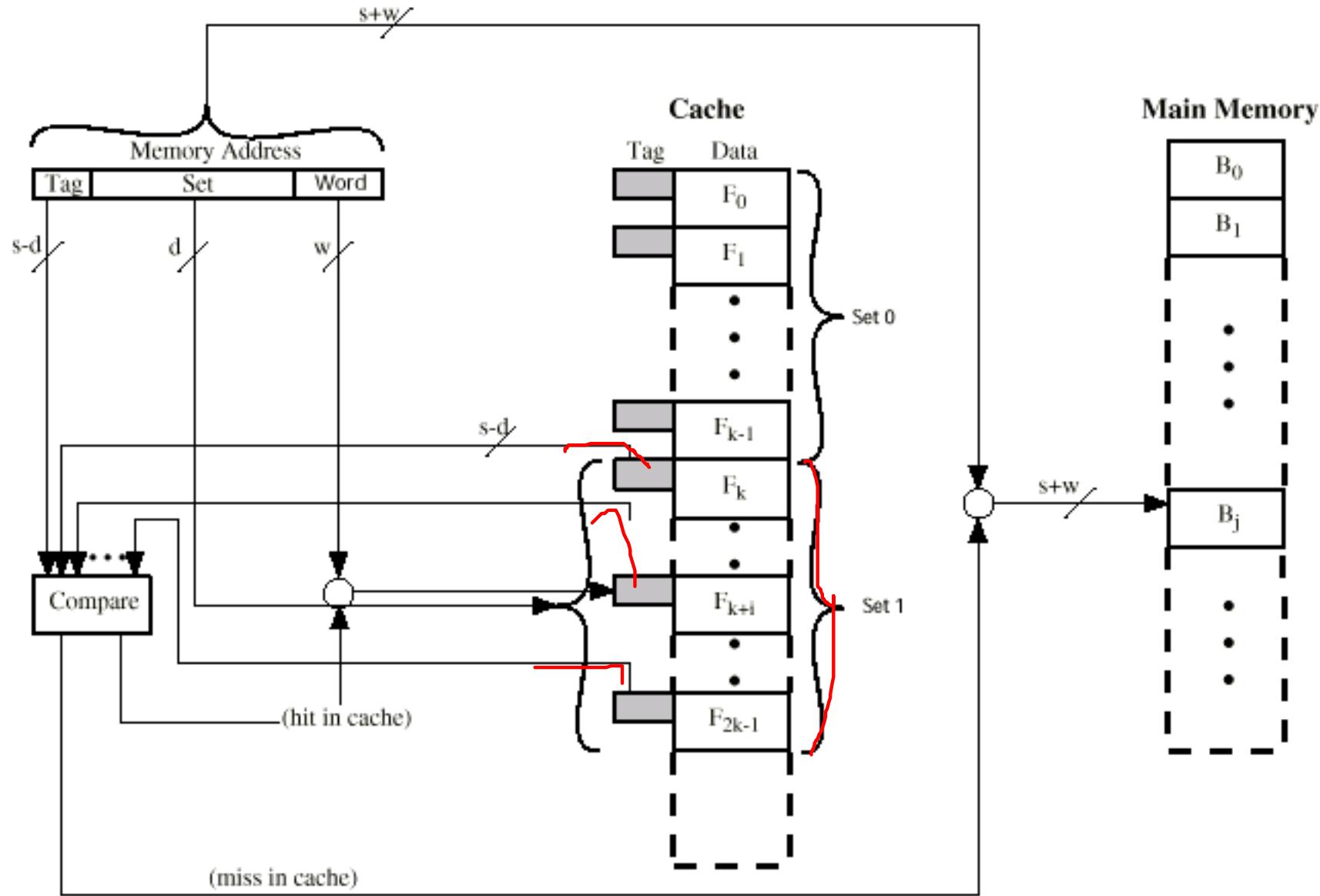
TAG

SET

WORD



# Set Associative Cache Organization



# Set Associative Mapping Summary

Address length =  $(s + w)$  bits

Number of addressable units =  $2^{s+w}$  words or bytes

Block size = line size =  $2^w$  words or bytes

Number of blocks in main memory =  $2^d$

Number of lines in set =  $k$

Number of sets =  $v = 2^d$

Number of lines in cache =  $kv = k * 2^d$

Size of tag =  $(s - d)$  bits

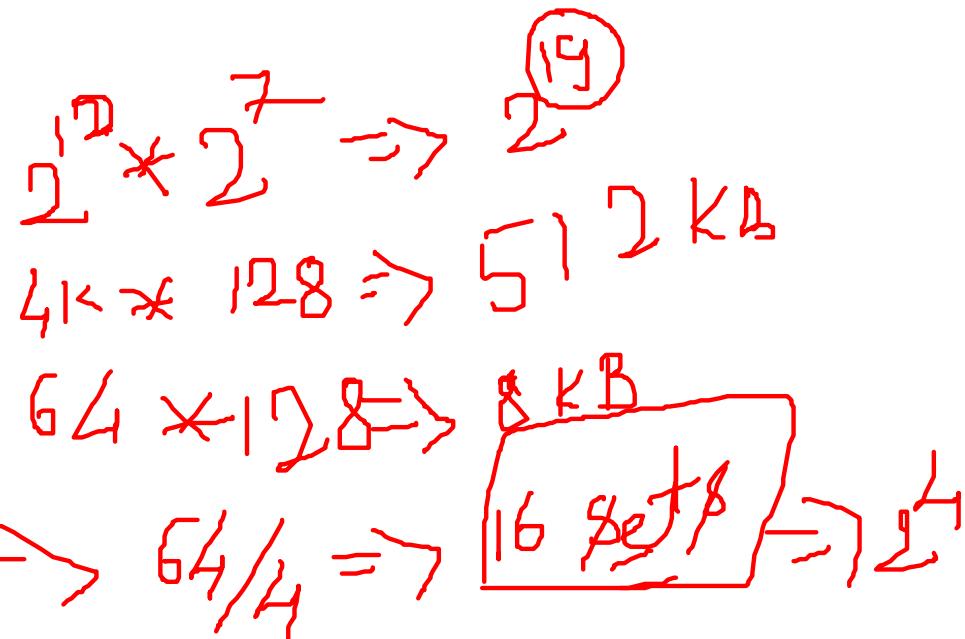
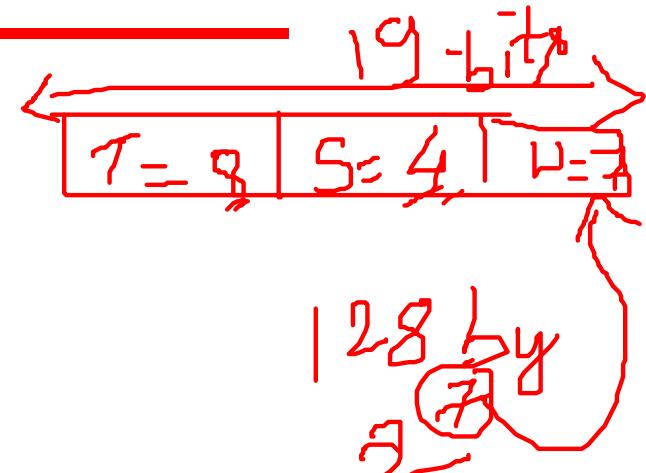
# Problem 1

4-way //

A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 bytes each. Show the format of main memory addresses.

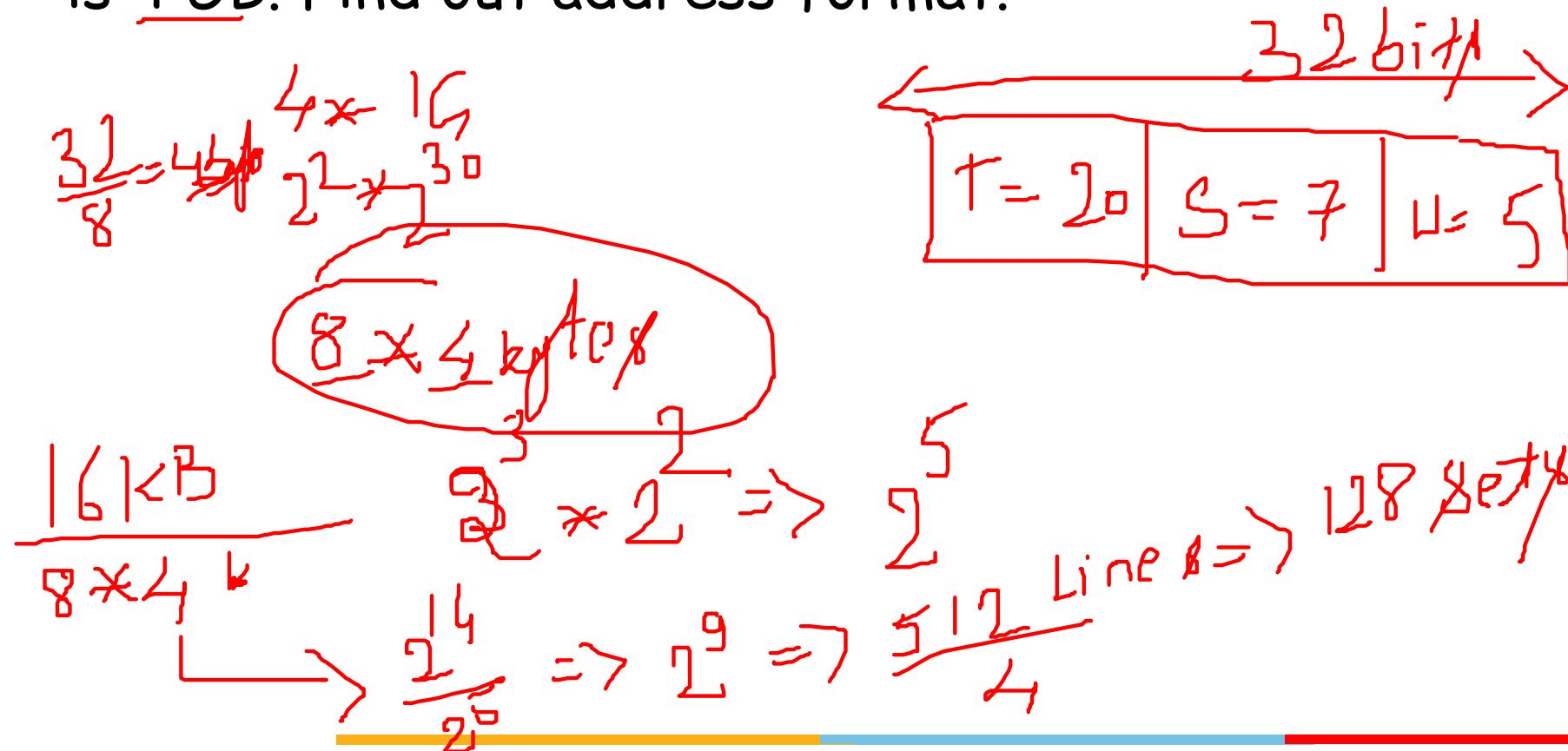
Find out

- Total main memory capacity  $\Rightarrow 4K \times 128 = 512KB$
- Total cache memory capacity  $\Rightarrow 64 \times 128 = 8KB$
- Total number of sets in the cache  $\Rightarrow 64 / 4 = 16$  sets
- Number of bits for TAG, SET and word



## Problem 2(Byte Addressable)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.



## Problem 2(Word Addressable)



A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

1G B    30 bits

# Replacement Algorithms (1/3)

## Direct mapped cache

- No choice
- Each block maps to one line and replace that line

# Replacement Algorithms (2/3)

- Needed in Associative & Set Associative mapped cache
- Hardware implemented algorithm (speed)
- Methods:
  - Least Recently Used (LRU)
  - Least Frequently Used (LFU)
  - First In First Out (FIFO)
  - Random

# Replacement Algorithms (3/3)

- **Least Recently used (LRU):** Replace the block in the set that has been in the cache longest with no reference to it
  - e.g. 2 way set associative
  - Uses "USE" bits
  - Most effective method
- **Least frequently used:** Replace block which has had fewest hits
  - Uses counter with each line
- **First in first out (FIFO):** Replace block that has been in cache longest
  - Round robin or circular buffer technique
- **Random**



## Problem 2

Consider a reference pattern that accesses the sequence of blocks 0, 4, 0, 2, 1, 8, 0, 1, 2, 3, 0, 4.  
Assuming that the cache uses associative mapping,  
find the hit ratio with a cache of four lines

- a) LRU
- b) LFU
- c) FIFO

# Problem 2 - LRU

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0 <sub>0</sub>	0 <sub>0</sub>	0 <sub>2</sub>	0 <sub>1</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0 <sub>6</sub>	0 <sub>6</sub>	0 <sub>6</sub>	0 <sub>6</sub>	0 <sub>10</sub>	0 <sub>10</sub>
L1	4 <sub>1</sub>	4 <sub>1</sub>	4 <sub>1</sub>	4 <sub>1</sub>	(4 <sub>1</sub> )	8 <sub>5</sub>	8 <sub>5</sub>	8 <sub>5</sub>	(8 <sub>5</sub> )	3 <sub>9</sub>	3 <sub>9</sub>	3 <sub>9</sub>
L2			2 <sub>3</sub>	2 <sub>3</sub>		2 <sub>3</sub>	2 <sub>3</sub>	2 <sub>3</sub>	2 <sub>8</sub>	2 <sub>8</sub>	2 <sub>9</sub>	2 <sub>8</sub>
L3				1 <sub>4</sub>	1 <sub>4</sub>	1 <sub>4</sub>	1 <sub>7</sub>	1 <sub>7</sub>	1 <sub>7</sub>	1 <sub>7</sub>	1 <sub>7</sub>	4 <sub>11</sub>
H/M	M	M	H	M	M	M	H	H	H	M	H	M

$$\begin{aligned}
 L_0 &= 0 \\
 L_1 &= 3 \\
 L_2 &= 2 \\
 L_3 &= 4
 \end{aligned}$$

5 / 12, 42-1.

# Problem 2 - LFU

Ref	0	4	0	2	1	8	0	1	2	3	0	4
L0	0 <sub>1</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0 <sub>2</sub>	0 <sub>2</sub>	0 <sub>2</sub>	0 <sub>3</sub>	0 <sub>3</sub>	0 <sub>3</sub>	0 <sub>3</sub>	0 <sub>4</sub>	0 <sub>4</sub>
L1		4 <sub>1</sub>	4 <sub>1</sub>	4 <sub>1</sub>	4 <sub>1</sub>	8 <sub>1</sub>	8 <sub>1</sub>	8 <sub>1</sub>	8 <sub>1</sub>	3 <sub>1</sub>	3 <sub>1</sub>	4 <sub>1</sub>
L2			4 <sub>1</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2 <sub>2</sub>	2 <sub>2</sub>	2 <sub>2</sub>				
L3				1 <sub>1</sub>	1 <sub>1</sub>	1 <sub>1</sub>	1 <sub>2</sub>					
H/M			H				H	H	H	M	H	M

$$L_0 = 0$$

$$L_1 = 4$$

$$L_2 = 2$$

$$L_3 = 1$$

$$5/12$$

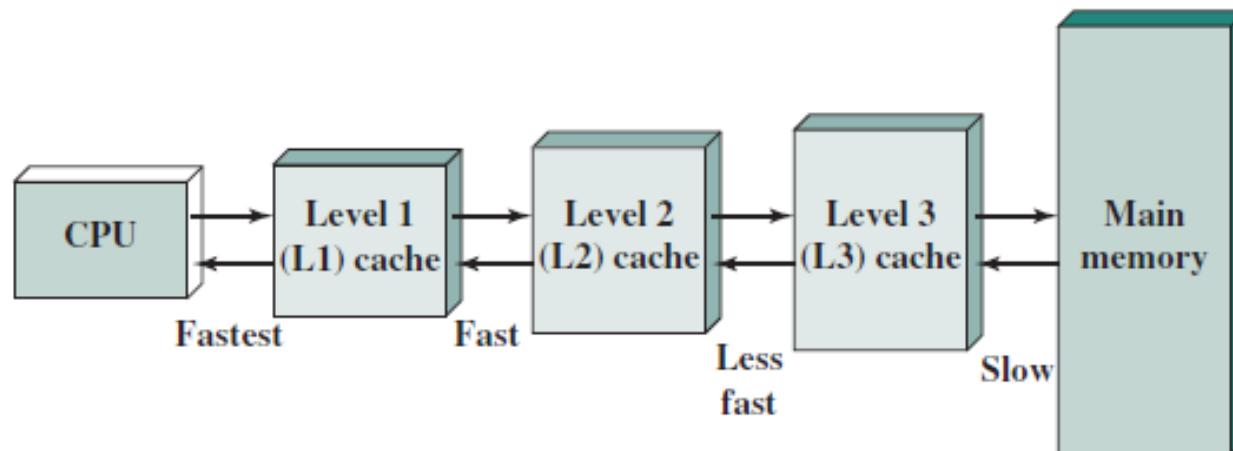
# Problem 2 - FIFO

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0	0	0	0	0	8	8	8	8	8	8	8
L1	4	4	4	4	4	0	0	0	0	0	0	0
L2		2	2	2	2	2	2	2	2	3	3	3
L3				1	1	1	1	1	1	1	1	4
H/M			H					H	H	M	H	

4/12

# Issues with Writes

- Multiple copies of data exist:
  - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from memory or not)



(b) Three-level cache organization

Cache and Main Memory

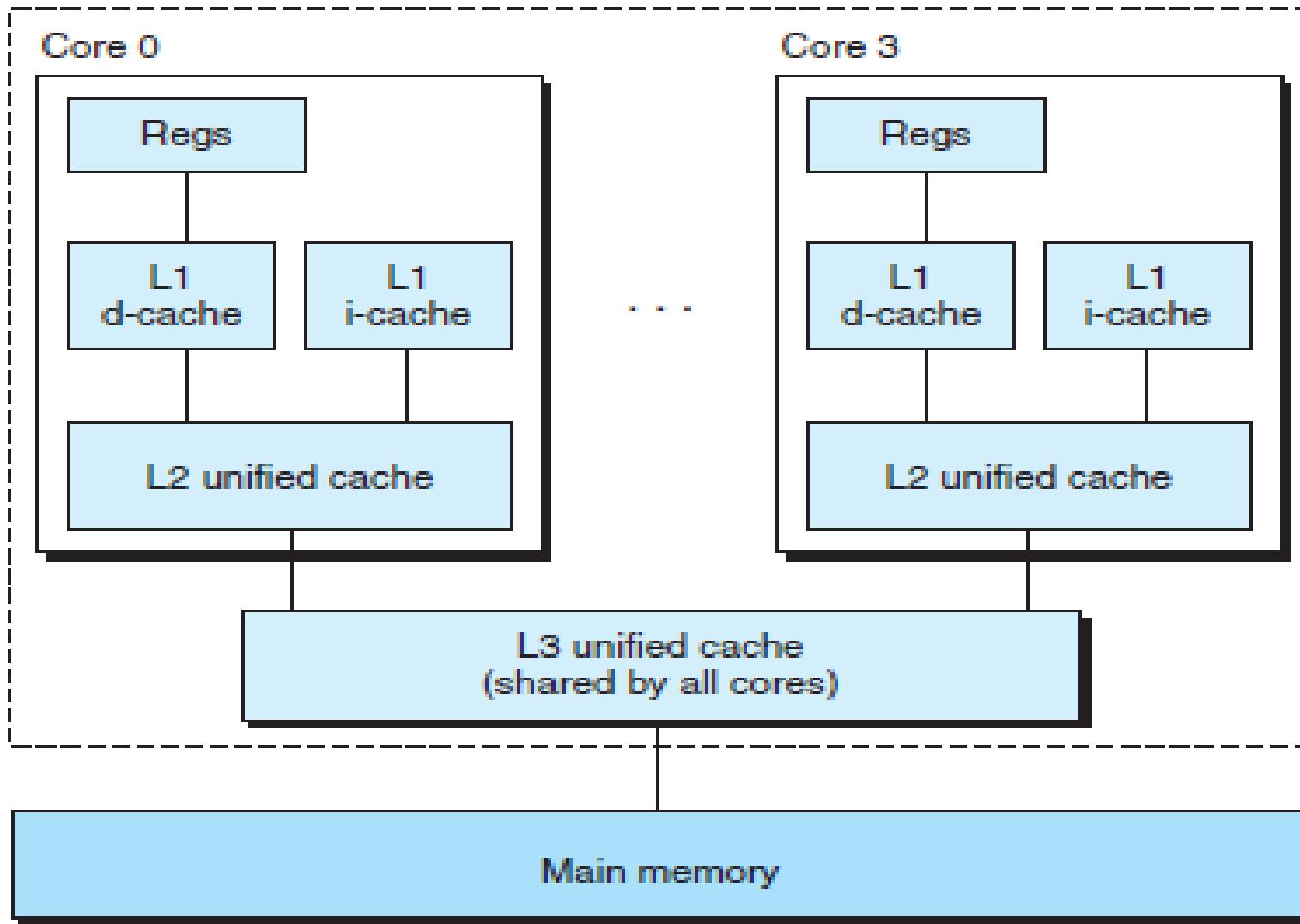
# Issues with Writes

- What to do on a write-miss?
  - Write-allocate (load into cache, update line in cache)
    - Good if more writes to the location follow
  - No-write-allocate (writes straight to memory, does not load into cache)  
*White outside*
- Typical
  - Write-through + No-write-allocate
  - Write-back + Write-allocate

# Intel Core i7 Cache Hierarchy



Processor package



# Intel Core i7 Cache Hierarchy

Cache type	Access time (cycles)	Cache size ( $C$ )	Assoc. ( $E$ )	Block size ( $B$ )	Sets ( $S$ )
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	11	256 KB	8	64 B	512
L3 unified cache	30–40	8 MB	16	64 B	8192

Characteristics of the Intel Core i7 cache hierarchy.

# Performance Impact of Cache Parameters

---

- Associativity :
  - higher associativity → more complex hardware,
  - Higher Associativity → Lower miss rate
  - Higher Associativity → reduces average memory access time (AMAT)
- Cache Size
  - Larger the cache size → Lower miss rate
  - Larger the cache size → reduces average memory access time (AMAT)
- Block Size:
  - Smaller blocks do not take maximum advantage of spatial locality.

# Revisiting Locality of reference

```

1 int sumvec(int v[N])
2 {
3     int i, sum = 0;
4
5     for (i = 0; i < N; i++)
6         sum += v[i];
7     return sum;
8 }
```

Does this function have good locality?

N=8

Address	0	1	2	3	4	5	6	7
Contents	v0	v1	v2	v3	v4	v5	v6	v7
Access Order	1	2	3	4	5	6	7	8

# Stride k - reference pattern

Byte Addressable memory and word length is 1 byte

innovate

achieve

lead

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 1



Address difference  
Stride = -----  
Word Length

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 2

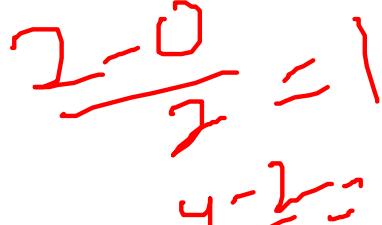
# Stride k - reference pattern



Byte Addressable  
memory and word  
length is 2 bytes

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Stride 1



i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Address difference  
Stride = -----  
Word Length

Stride 2

# Revisiting Locality of reference

```

1 int sumarrayrows(int a[M][N])
2 {
3     int i, j, sum = 0;
4
5     for (i = 0; i < M; i++)
6         for (j = 0; j < N; j++)
7             sum += a[i][j];
8
9     return sum;
}

```

Does this function have good locality?

M = 2, N=3						
Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	2	3	4	5	6

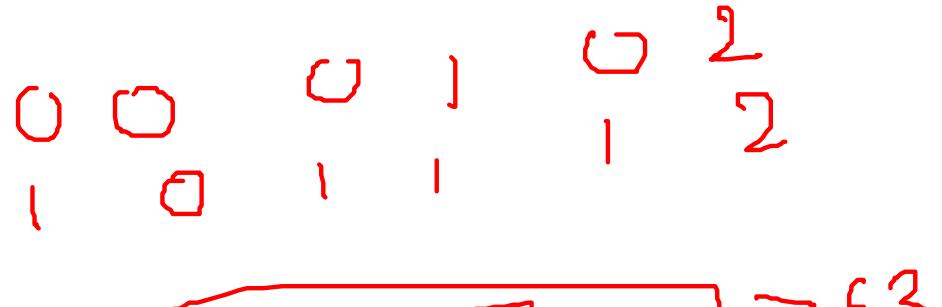
# Revisiting Locality of reference

```

1 int sumarraycols(int a[M][N])
2 {
3     int i, j, sum = 0;
4
5     for (j = 0; j < N; j++)
6         for (i = 0; i < M; i++)
7             sum += a[i][j];
8
9     return sum;
}

```

Does this function have good locality?



M = 2, N=3						
Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	3	5	2	4	6



# COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 5



**BITS** Pilani  
Pilani Campus

Pruthvi Kumar K R

# Revisiting Locality of reference

```

1 int sumarrayrows(int a[M][N])
2 {
3     int i, j, sum = 0;
4
5     for (i = 0; i < M; i++)
6         for (j = 0; j < N; j++)
7             sum += a[i][j];
8
9 }
```

Does this function have good locality?

*Stride  
=*

M = 2, N=3

Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	2	3	4	5	6

# Revisiting Locality of reference

```

1 int sumarraycols(int a[M] [N])
2 {
3     int i, j, sum = 0;
4         2
5     for (j = 0; j < N; j++)
6         for (i = 0; i < M; i++)
7             sum += a[i] [j];
8
9     return sum;
}

```

Does this function have good locality?

S3

$M = 2, N=3$						
Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	3	5	2	4	6

# Writing Cache Friendly Code

- Make the common case go fast
  - Focus on the inner loops of the core functions
- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

# Example 1

```
int sumarrayrows(int a[4][4])
{
    int i, j, sum = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            sum += a[i][j];
    return sum;
}
```

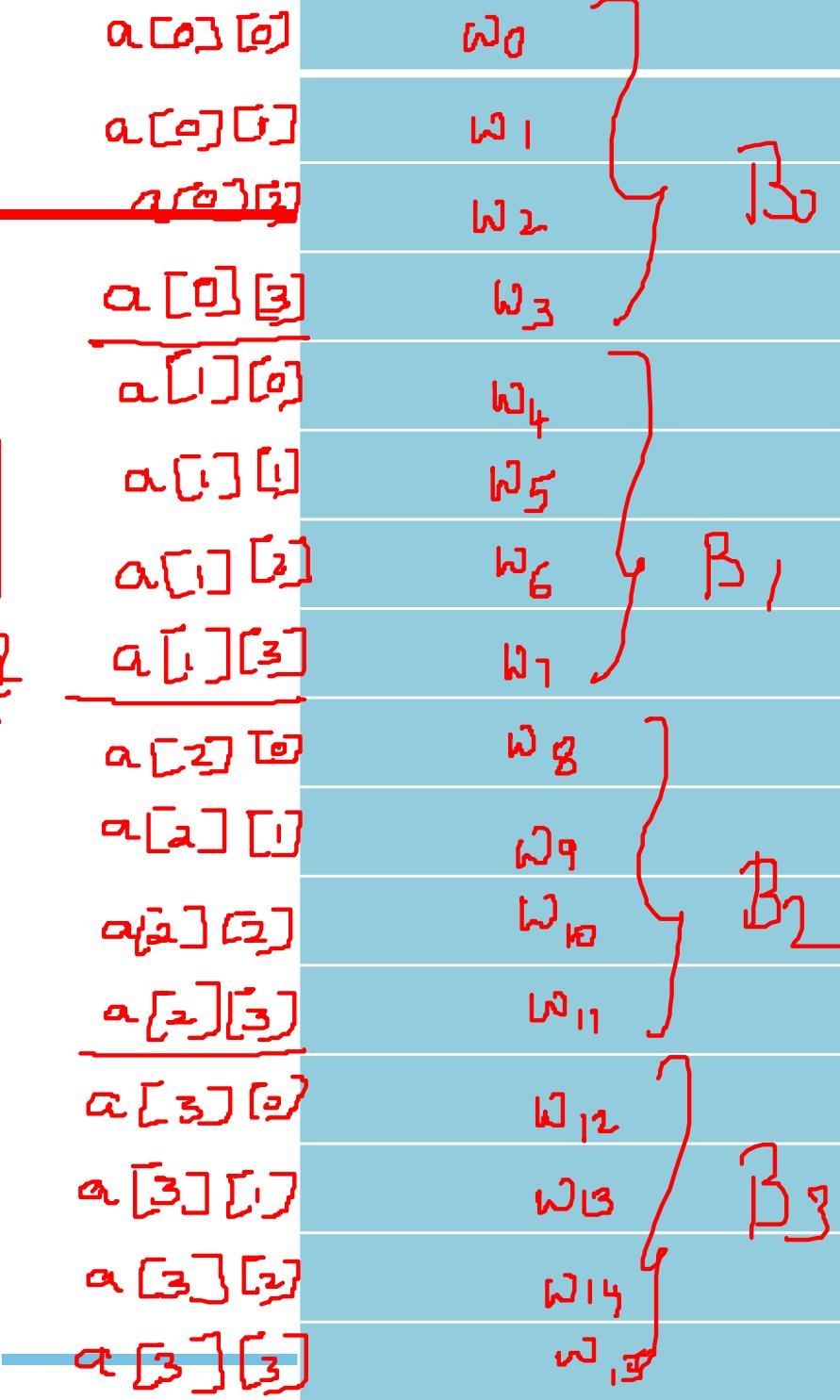
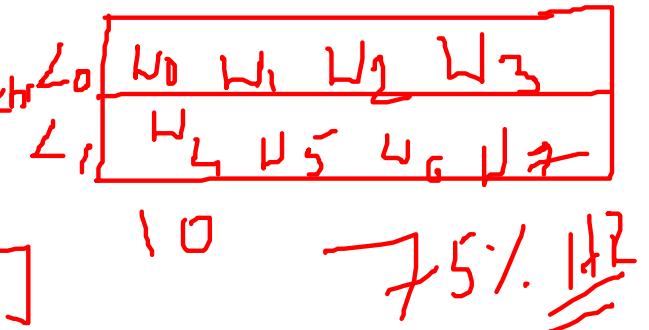
Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

# Example 1(Contd..)

```
int sumarrayrows(int a[4][4])
{
    int i, j, sum = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            sum += a[i][j];
    return sum;
}
```

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0	M	H	H	H
i = 1	M	H	H	H
i = 2	M	H	H	H
i = 3	M	H	H	H



## Example 2:

```
int sum_array(int a[4][4])
{
    int i, j, sum = 0;
    for (j = 0; j < 4; j++)
        for (i = 0; i < 4; i++)
            sum += a[i][j];
    return sum;
}
```

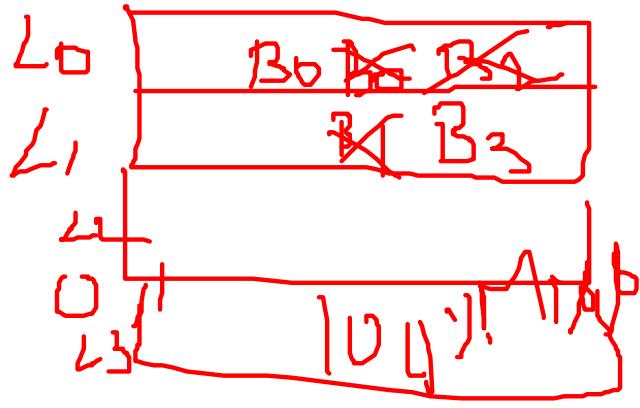
Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

# Example 1(Contd..)

```
int sum_array(int a[4][4])
{
    int i, j, sum = 0;
    for (j = 0; j < 4; j++)
        for (i = 0; i < 4; i++)
            sum += a[i][j];
    return sum;
}
```

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0	M	M	M	M
i = 1	M	M	M	M
i = 2	M	M	M	M
i = 3	M	M	M	M



a[0][0]	w <sub>0</sub>
a[0][1]	w <sub>1</sub>
a[0][2]	w <sub>2</sub>
<u>a[0][3]</u>	
<u>a[1][0]</u>	w <sub>3</sub>
a[1][1]	w <sub>4</sub>
a[1][2]	w <sub>5</sub>
a[1][3]	w <sub>6</sub>
<u>a[2][0]</u>	w <sub>7</sub>
a[2][1]	w <sub>8</sub>
a[2][2]	w <sub>9</sub>
a[2][3]	w <sub>10</sub>
<u>a[3][0]</u>	w <sub>11</sub>
a[3][1]	w <sub>12</sub>
a[3][2]	w <sub>13</sub>
a[3][3]	w <sub>14</sub>
<u>a[3][3]</u>	w <sub>15</sub>

# Home Work - Which one is better ?

---

Program 1:

```
for (int i = 0; i < n; i++) {  
    z[i] = x[i] - y[i];  
    z[i] = z[i] * z[i];  
}
```

Program 2:

```
for (int i = 0; i < n; i++) {  
    z[i] = x[i] - y[i];  
}  
for (int i = 0; i < n; i++) {  
    z[i] = z[i] * z[i];  
}
```



# CISC Instruction Set (Intel x86 as an example)

**BITS** Pilani  
Pilani Campus

# Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
9-10	<ul style="list-style-type: none"> <li>• <b>Instruction Set Architecture - CISC Vs RISC</b></li> <li>• CISC Instruction Set (Intel x86 as an example)           <ul style="list-style-type: none"> <li>• Machine Instruction Characteristics</li> <li>• Types of Operands</li> <li>• Types of Operations</li> <li>• Addressing Modes</li> </ul> </li> <li>• Instruction Formats</li> </ul>	T1

# Introduction

$A \leftarrow B_x, C_x, SI, DI$

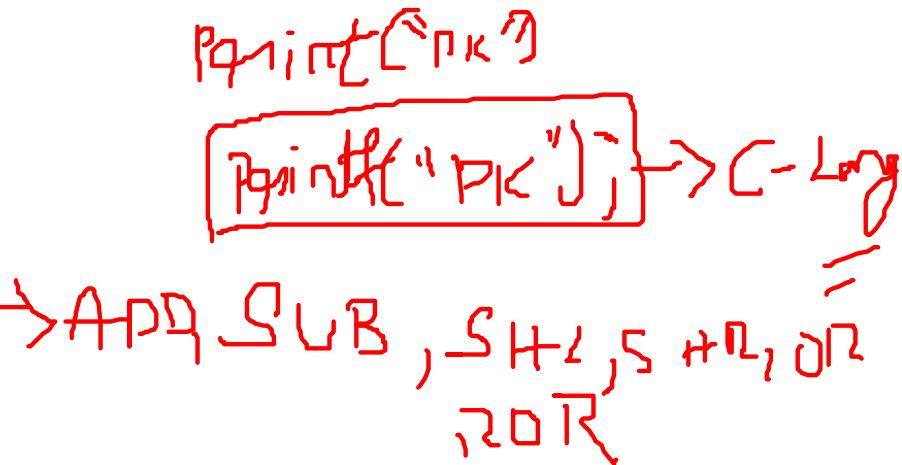
$A, B$

- What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU

- Elements of an Instruction

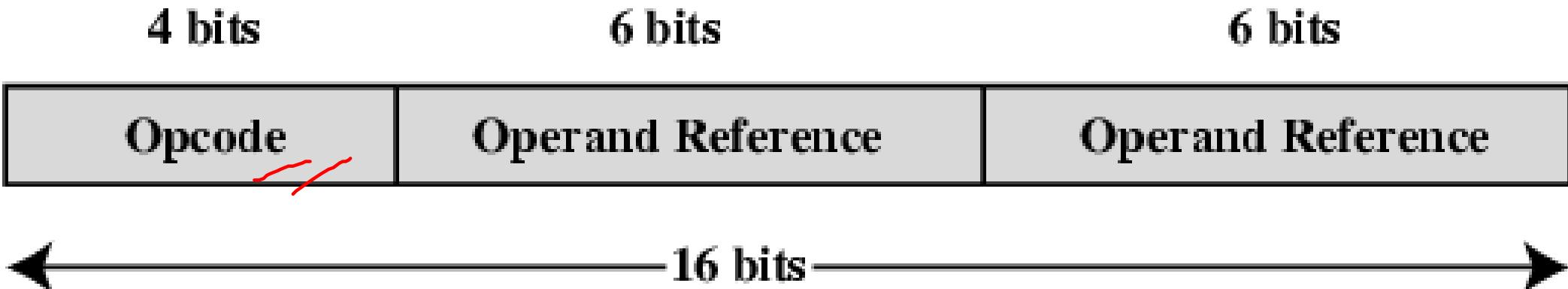
- Operation code (Op code)
- Source Operand reference
- Result Operand reference
- Next Instruction Reference



- Source and Destination Operands can be found in four areas
  - Main memory (or virtual memory or cache)
  - CPU register
  - I/O device
  - Immediate

# Simple Instruction Format

8086  
||



- During instruction execution, an instruction is read into an instruction register (IR) in the processor.
- The processor must be able to extract the data from the various instruction fields to perform the required operation.
- Opcodes are represented by abbreviations, called **mnemonics**

Example: ADD AX, BX → Add instruction

# Instruction Types

- Data processing : Arithmetic and logic instructions
- Data storage (main memory) : Movement of data into or out of register and or memory locations
- Data movement (I/O) : I/O instructions
- Program flow control : Test and branch instructions

# Number of Addresses (1/2)

- 3 addresses

- Result, Operand 1, Operand 2

- $c = a + b$ ; add c, a, b  $\Rightarrow 4 + 6 + 6 + 6 \Rightarrow 22$  bits

- May be a forth - next instruction (usually implicit)

- Needs very long words to hold everything

- 2 addresses

- One address doubles as operand and result

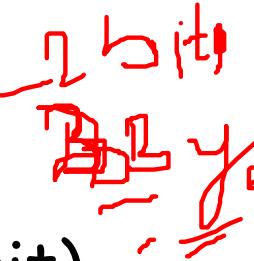
- $a = a + b$ : add a, b  $\Rightarrow 4 + 6 + 6 \Rightarrow 16 \Rightarrow 2$  bytes

- Reduces length of instruction

- The original value of a is lost.

$$1 B = 8 \text{ bits}$$

$$3 B = 24$$



$$a \leftarrow a + b$$

# Number of Addresses (2/2)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

ADD A'  $\rightarrow$  2 bytes

AC  $\leftarrow AC + A$

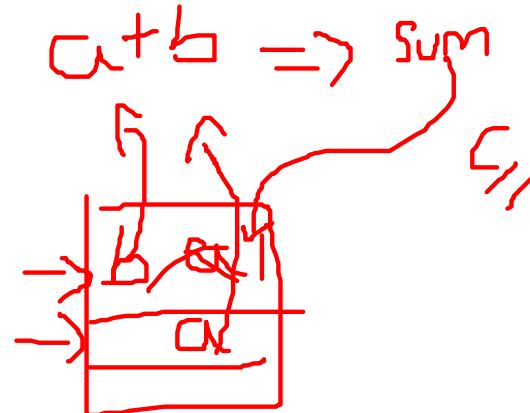
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g.  $c = a + b$

push a

push b

add,

pop c



# Example

Execute  $Y = \frac{A - B}{C + (D \times E)}$

innovate

achieve

lead

<u>Instruction</u>	<u>Comment</u>
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

<u>Instruction</u>	<u>Comment</u>
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

(c) One-address instructions

<u>Instruction</u>	<u>Comment</u>
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Two address instructions

# How Many Addresses

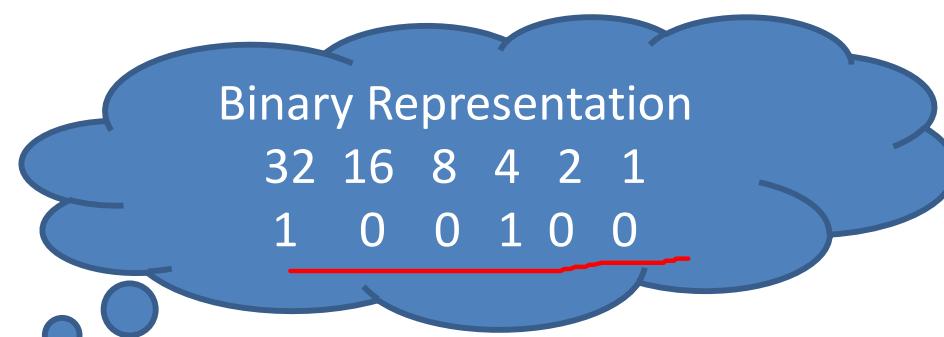
- Fewer addresses
  - More Primitive instructions, shorter length instructions
  - Less complex instructions, hence requires less complex hardware
  - More instructions per program
    - Longer programs
    - More complex programs
    - Longer execution time
- Multiple address instructions
  - Lengthy instructions
  - More registers
    - Inter-register operations are quicker
  - Fewer instructions per program

# Instruction set Design Decisions

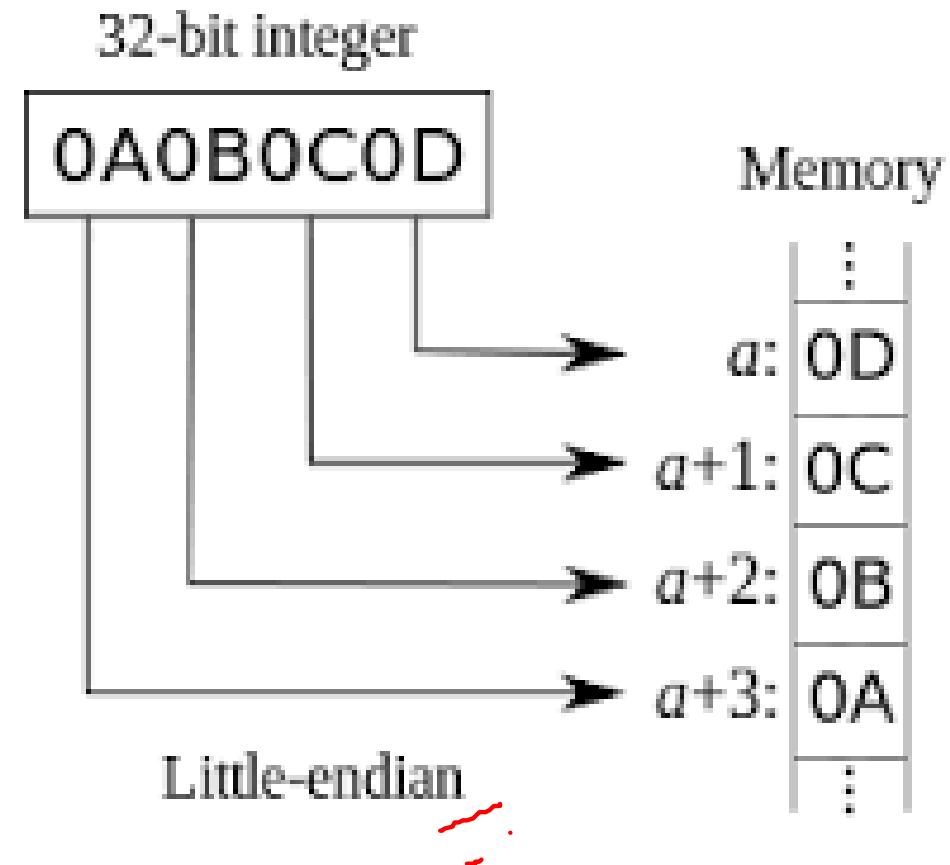
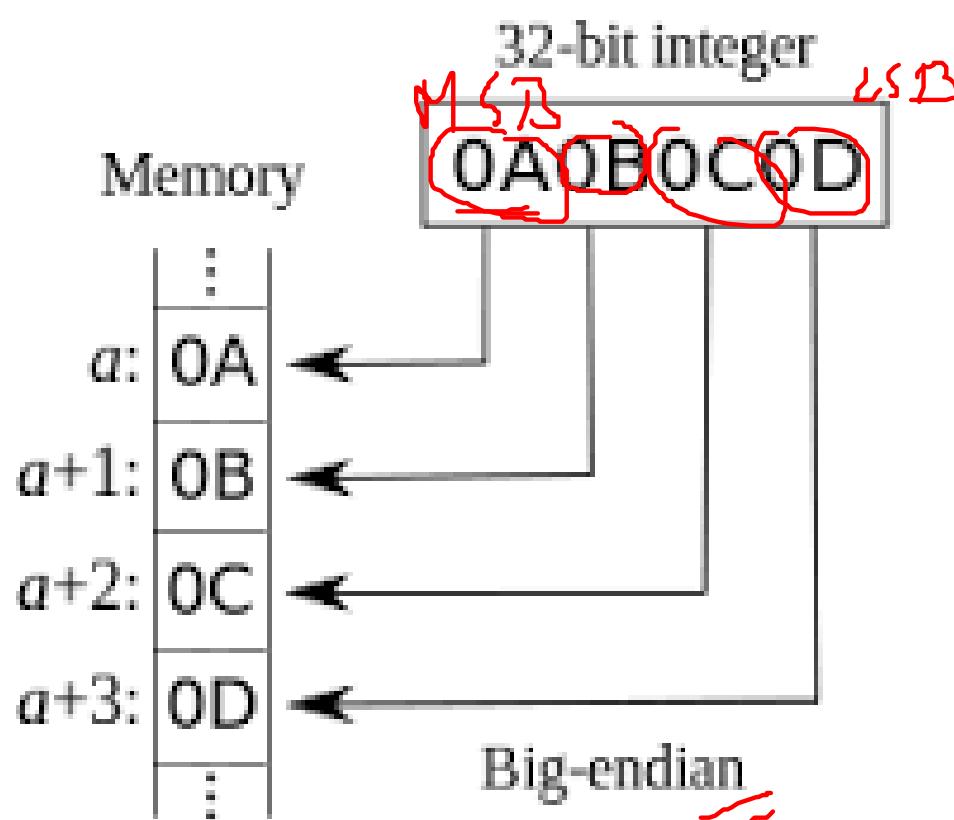
- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes

# Types of Operand

- Machine instructions operate on data
- General categories of data
  - Addresses
  - Numbers       $36 \rightarrow 0011\ 0110$       3 bytes      b bits
  - Binary integer or binary fixed point, floating point, decimal
- Characters
  - ASCII etc.
- Logical Data
  - Bits or flags
- Packed Decimal
  - 36 : 0011 0110



# Byte Ordering



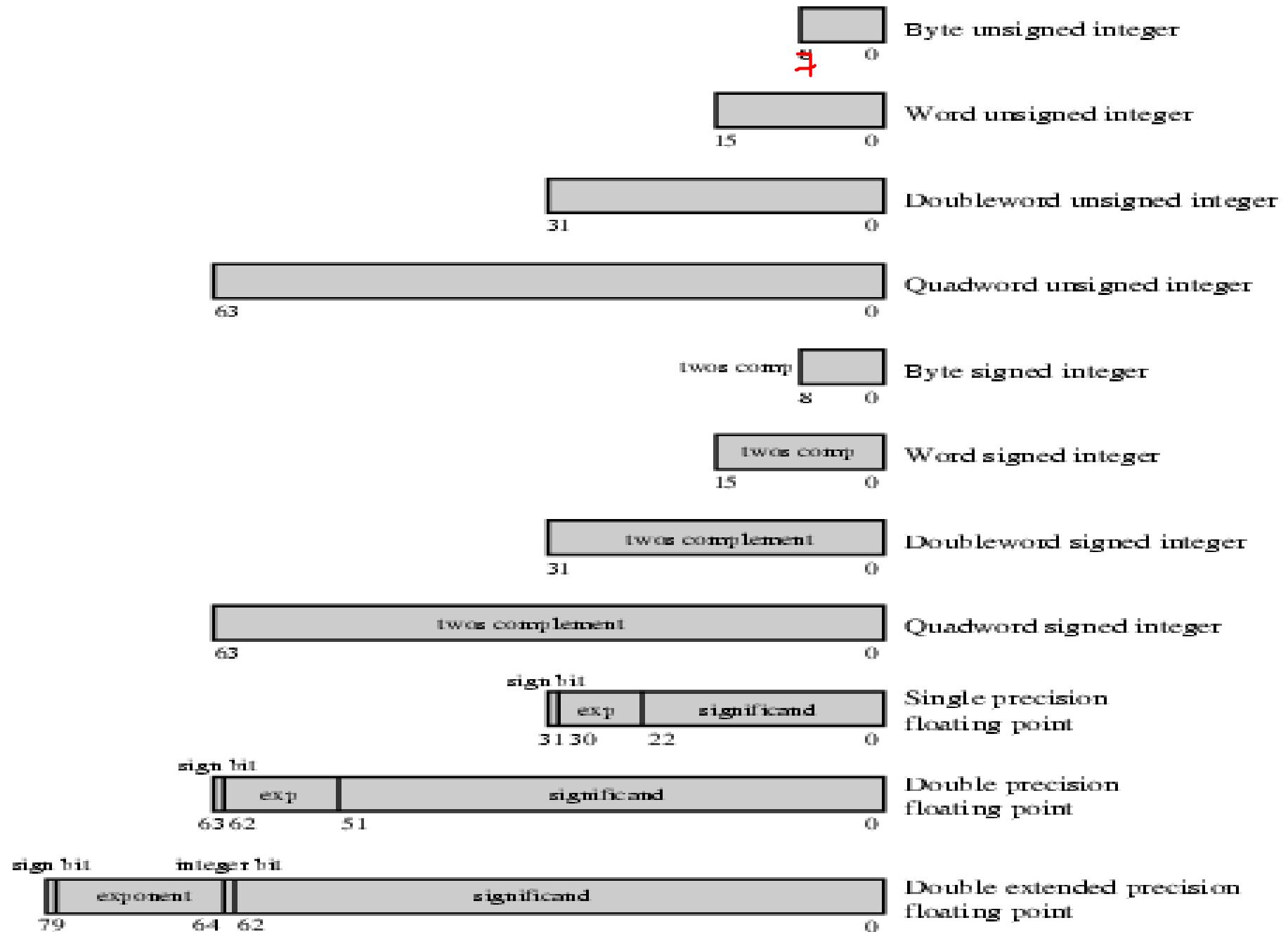
# x86 Data Types

- General - Byte, Word, double word, quadword, double quad word - arbitrary binary contents
- Integer - signed binary using two's complement representation
- Ordinal - unsigned integer
- Unpacked BCD - One digit per byte
- Packed BCD - 2 digits per byte
- Near Pointer - 32 bit offset within segment
- Far pointer -
- Bit field : A contiguous sequence of bits in which the position of each bit is considered as an independent unit.
- Bit and Byte String
- Floating Point



Data Type	Description
General	Byte, word (16 bits), doubleword (32 bits), quadword (64 bits), and double quadword (128 bits) locations with arbitrary binary contents.
Integer	A signed binary value contained in a byte, word, or doubleword, using two's complement representation.
Ordinal	An unsigned integer contained in a byte, word, or doubleword.
Unpacked binary coded decimal (BCD)	A representation of a BCD digit in the range 0 through 9, with one digit in each byte.
Packed BCD	Packed byte representation of two BCD digits; value in the range 0 to 99.
Near pointer	A 16-bit, 32-bit, or 64-bit effective address that represents the offset within a segment. Used for all pointers in a nonsegmented memory and for references within a segment in a segmented memory.
Far pointer	A logical address consisting of a 16-bit segment selector and an offset of 16, 32, or 64 bits. Far pointers are used for memory references in a segmented memory model where the identity of a segment being accessed must be specified explicitly.
Bit field	A contiguous sequence of bits in which the position of each bit is considered as an independent unit. A bit string can begin at any bit position of any byte and can contain up to 32 bits.
Bit string	A contiguous sequence of bits, containing from zero to $2^{32} - 1$ bits.
Byte string	A contiguous sequence of bytes, words, or doublewords, containing from zero to $2^{32} - 1$ bytes.
Floating point	See Figure 10.4.
Packed SIMD (single instruction, multiple data)	Packed 64-bit and 128-bit data types

# x86 Numeric Data Formats



# Types of Operation

- Data Transfer
- Arithmetic
- Logical
- Conversion
- I/O
- System Control
- Transfer of Control

# Data Transfer

- Specify
    - Source
    - Destination
    - Amount of data
  - Action:
    - Calculate the memory address, based on the address mode
    - If the address refers to virtual memory, translate from virtual to real memory address.
    - Determine whether the addressed item is in cache.
    - If not, issue a command to the memory module.
- 
- The diagram illustrates the components of a 32-bit CPU bus and the execution of a `Mov` instruction. The bus width is 32 bits, divided into 8-bit `AL` and 24-bit `BL`. The `AX` register is shown in the `AL` position. Addressing modes are indicated by curly braces: `AX` (direct), `BX` (base), `CX` (index), and `DX` (register). The `BL` register is shown in the `BL` position. The `Op code dest, src` field contains the `Mov` instruction. Below the bus, a stack diagram shows the state before and after the move operation. Red annotations include: `Mov AX, BL ✓` (correct), `Mov BL, AX ✗` (incorrect), `LA DA PA MU` (addressing modes), and a blank box for the source operand. The stack grows downwards.

# Arithmetic

- Add, Subtract, Multiply, Divide
- May include
  - Absolute value ( $|a|$ )
  - Increment ( $a++$ )
  - Decrement ( $a--$ )
  - Negate ( $-a$ )
- Signed Integer

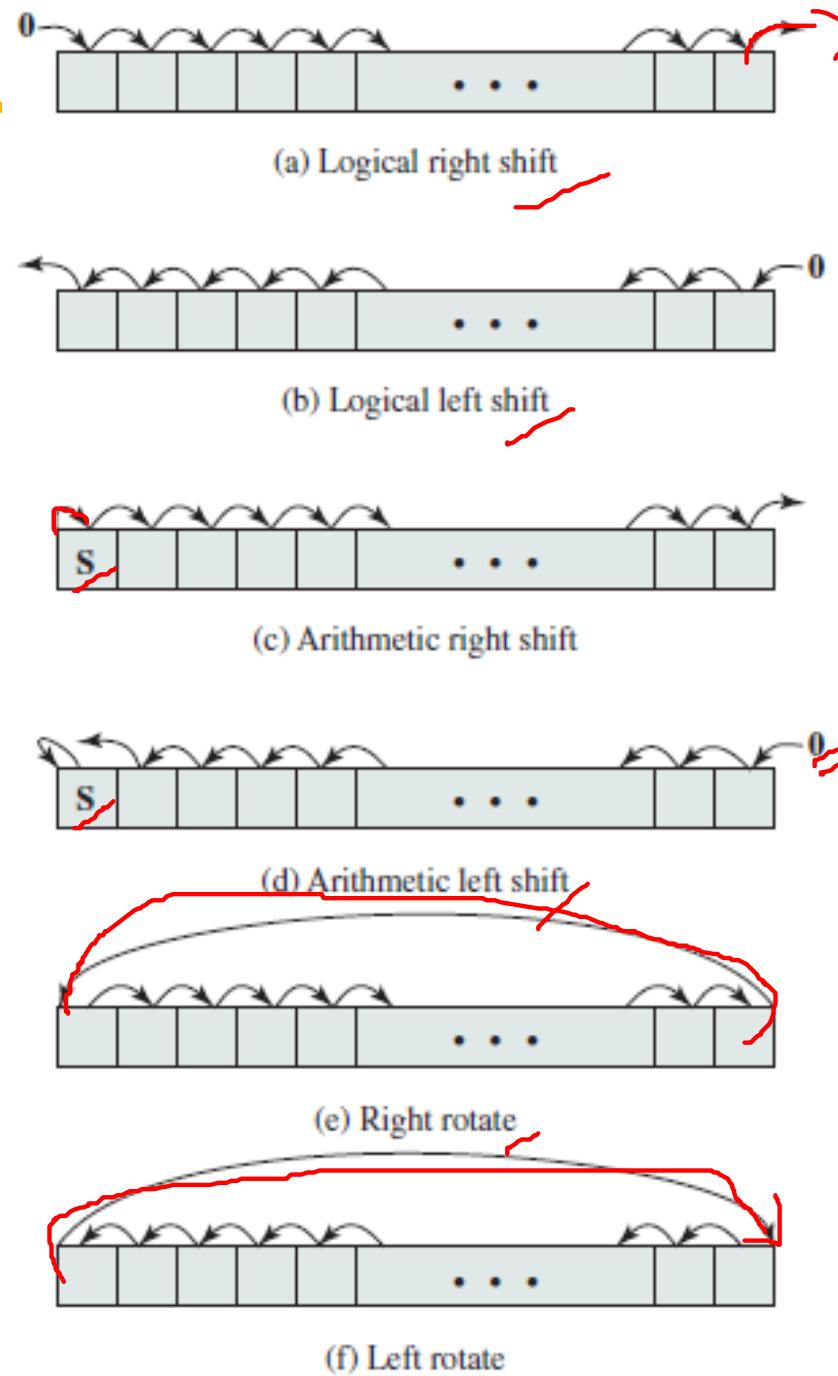
# Logical

- Bitwise operations
- AND, OR, NOT

## Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

# Shift and Rotate Operations



Input	Operation	Output
10101101	Logical right shift (3 bits)	10101101=> 00010101
10101101	Logical left shift (3 bits)	10101101=> <b>01101000</b>
10101101	Arithmetic right shift (3 bits)	10101101=> 11110101
10101101	Arithmetic left shift (3 bits)	10101101=> <b>11101000</b>
10101101	Right rotate (3 bits)	10101101=> <b>10110101</b>
10101101	Left rotate (3 bits)	10101101=> <b>01101101</b>

# Conversion

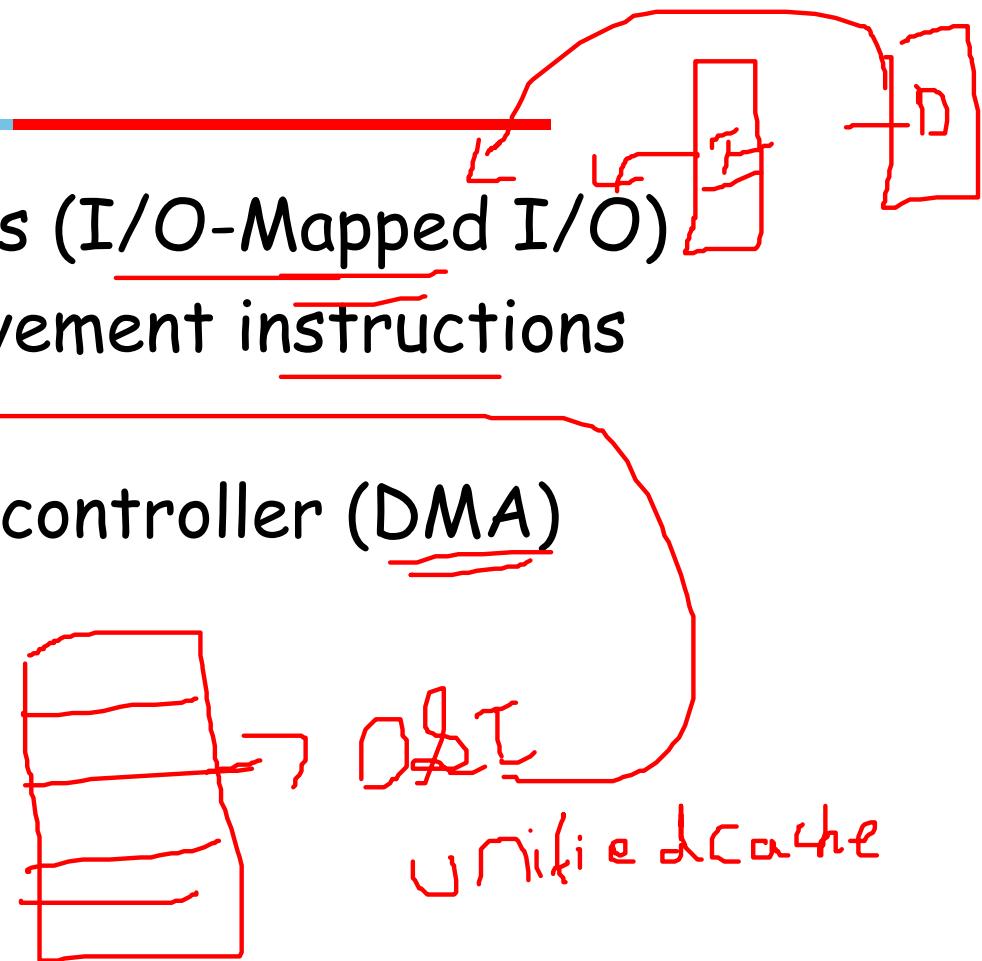
E.g. Binary to Decimal

$$\begin{array}{r} 129 \\ 11000000001 \end{array}$$

A hand-drawn diagram illustrating the conversion of the binary number 11000000001 to its decimal equivalent, 129. The binary digits are written vertically, with powers of 2 written above them: 128, 64, 32, 16, 8, 4, 2, 1. A red arrow points from the bottom right towards the binary digits, indicating the mapping between the two representations.

# Input/Output

- May be specific instructions (I/O-Mapped I/O)
- May be done using data movement instructions  
(memory mapped)
- May be done by a separate controller (DMA)



# Systems Control

- Privileged instructions
- CPU needs to be in specific state
  - User Mode  $\Rightarrow$  1
  - Kernel mode  $\Rightarrow$  0
- For operating systems use

# Transfer of Control

- Jump / Branch (Unconditional / Conditional)

– e.g. jump to x if result is zero

- Skip (Unconditional / Conditional)

○ skip (unconditional) : Increment to skip next ~~JNE~~ instruction

– e.g. increment and skip if zero

ISZ Register1

Branch xxxx

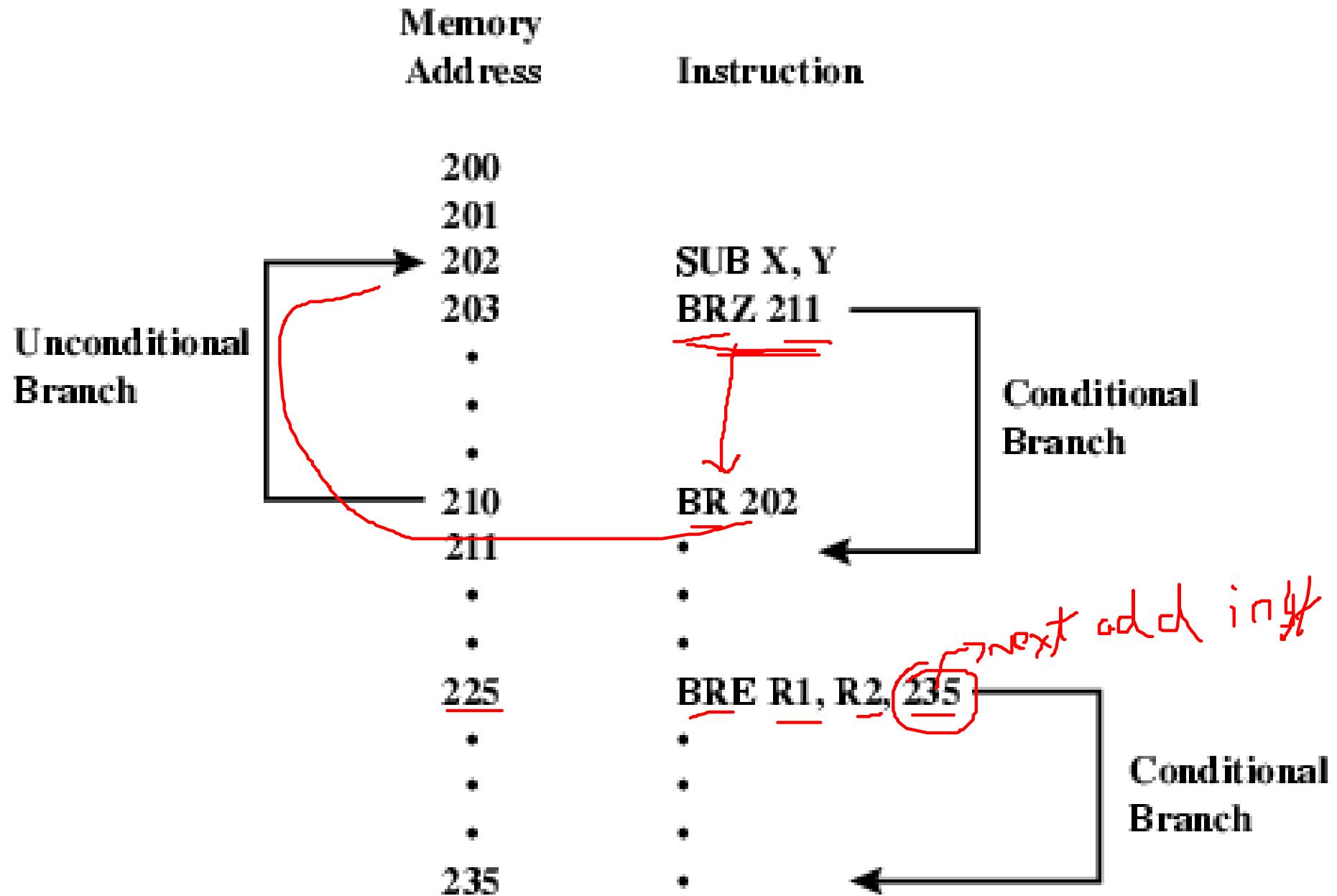
ADD A

- Subroutine call

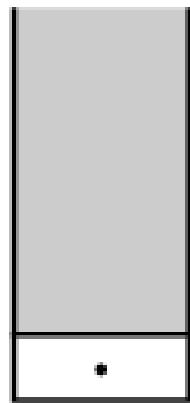
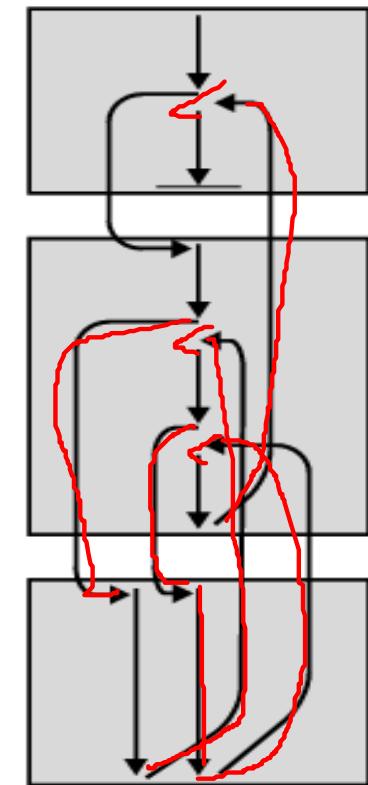
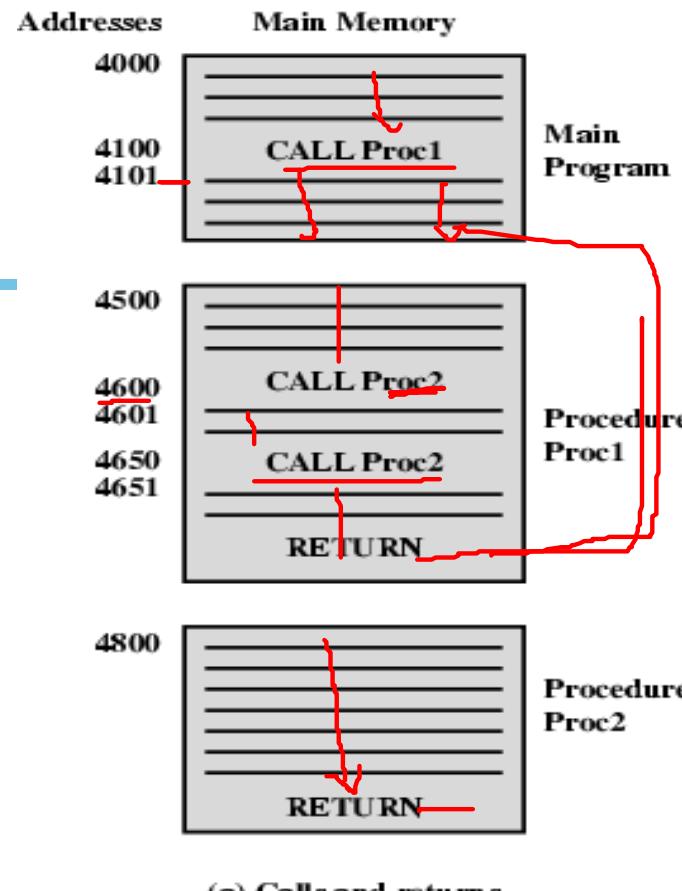
- interrupt call

JMP      JE  
JNE

# Branch / Jump Instruction



# Use of Stack



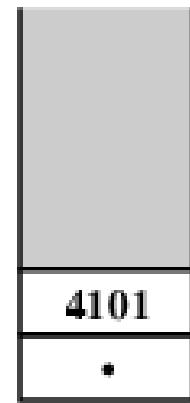
(a) Initial stack contents



(b) After CALL Proc1



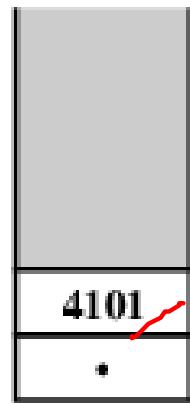
(c) Initial CALL Proc2



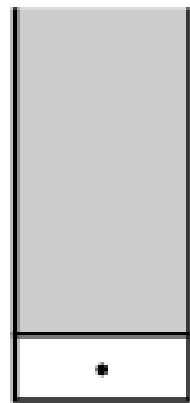
(d) After RETURN



(e) After CALL Proc2



(f) After RETURN



(g) After RETURN

# Addressing Modes

- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Stack



R2000  
MIPS



# COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 6

By  
Pruthvi Kumar K R



**BITS** Pilani  
Pilani Campus

# Addressing Modes

---

- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Stack

# Immediate Addressing

- Operand is specified in the instruction itself
- e.g. **ADD #5**
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

**ADD #5**

**ADD 5**

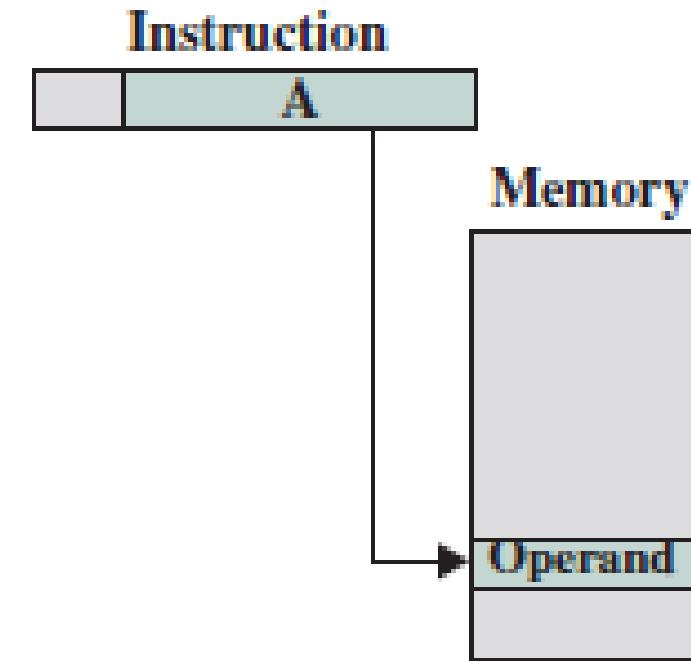
**5550**

**Instruction**



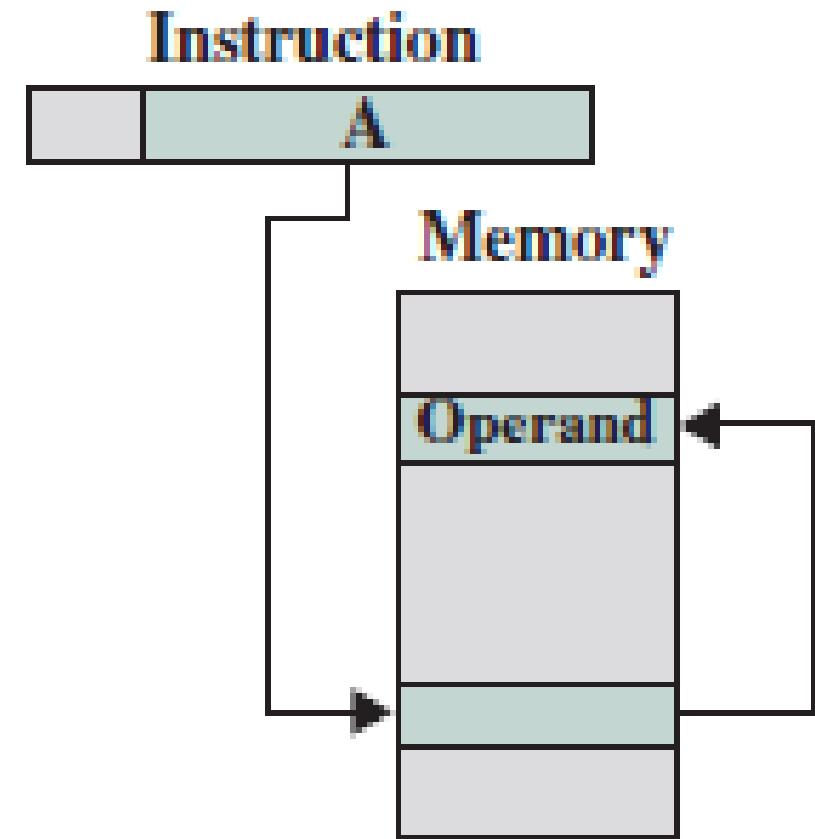
# Direct Addressing

- Address of the operand is specified in the instruction
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of memory cell whose address is A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- No additional calculations to work out effective address
- Limited address space



# Indirect Addressing

- Memory cell pointed to by address field of the instruction contains the address of (pointer to) the operand
- $EA = (A)$ 
  - Look in  $A$ , find address and look there for operand
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of  $A$  to accumulator



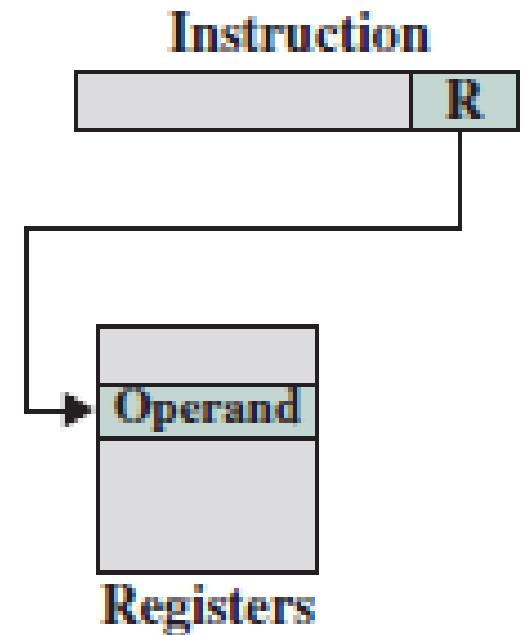
# Indirect Addressing...

- Large address space
- $2^n$  where n = word length
- May be nested, multilevel, cascaded
  - e.g. EA = (((A)))
- Multiple memory accesses to find operand
- Slower

# Register Addressing

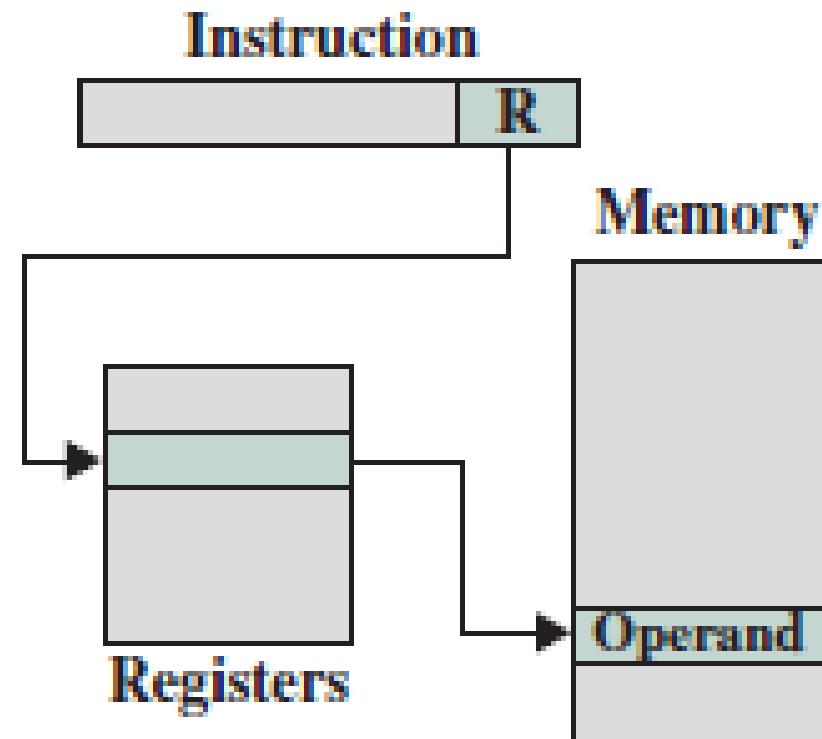
- Operand is held in register named in address field
- $EA = R$
- Limited number of registers
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No memory access hence Very fast execution but very limited address space
- Multiple registers helps in improving performance
  - Requires good assembly programming or compiler writing
  - C programming : register int a;

ADD A x,B x



# Register Indirect Addressing

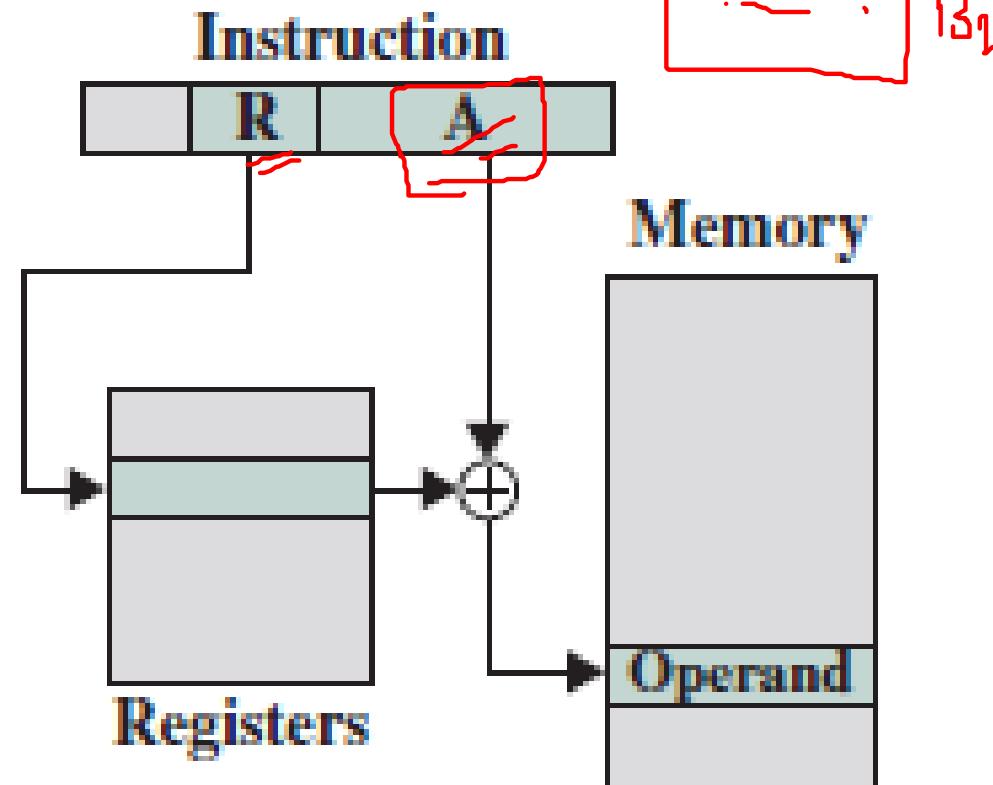
- Similar to indirect addressing
- $EA = (R)$
- Operand is in memory cell pointed to by contents of register R
- Large address space ( $2^n$ )
- One memory access compared indirect addressing





# Displacement Addressing

- $EA = A + (R)$
- Address field hold two values
  - $A$  = base value
  - $R$  = register that holds displacement
  - or vice versa
- Three variants:
  - Relative addressing
  - Base register addressing
  - Indexing



# Relative Addressing

- Also known as PC relative addressing
- A version of displacement addressing
- R = Program counter, PC
- $EA = \underline{A} + \underline{(PC)}$
- Relative addressing exploits the concept of locality
  - If most memory references are relatively near to the instruction being executed, then the use of relative addressing saves address bits in the instruction.

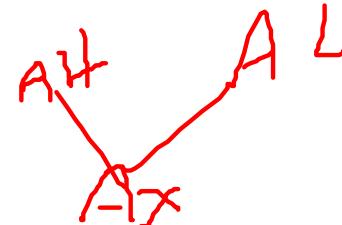
# Base-Register Addressing

- The referenced register "R" contains a main memory address
- address field contains a displacement A
- R may be explicit or implicit
- e.g. segment registers in 80x86

# Indexed Addressing

- The address field references a main memory address A
- The referenced register R contains a positive displacement from that address.
- $EA = A + R$
- Good for accessing arrays
  - $EA = A + R$
  - $R++$

# Auto Indexing

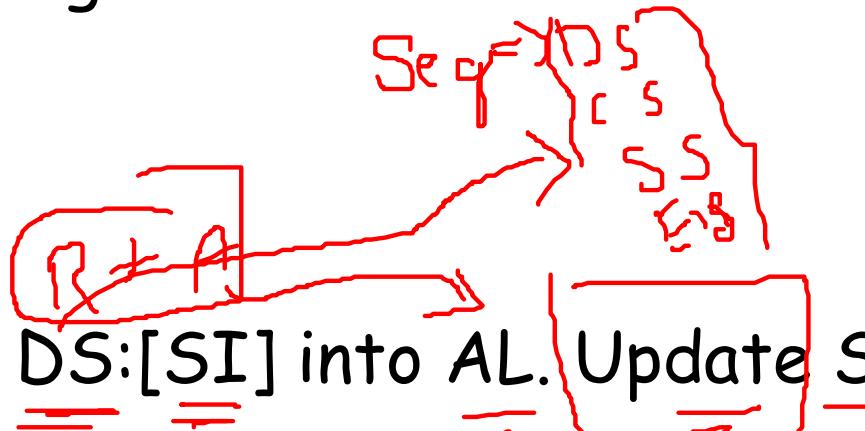


- Auto indexing incase certain registers are devoted exclusively to indexing

$$EA = A + (R)$$

$$(R) \leftarrow (R) + 1$$

Example: LODSB: Load byte at DS:[SI] into AL. Update SI.  
 $AL = DS:[SI]$



SI is incremented or decremented based on direction flag.

D = 0  $\rightarrow$  increment SI  
D = 1  $\rightarrow$  decrement SI

- Two types:

- Postindex
- Preindex

# Post-indexing

- indexing is performed after the indirection  
$$EA = (A) + (R)$$
- Steps:
  1. The contents of the address field are used to access a memory location containing a direct address.
  2. Address is then indexed by the register value
- Use:
  - for accessing one of a number of blocks of data of a fixed format

# Pre-indexing



- An address is calculated as with simple indexing

$$EA = (A + (R))$$

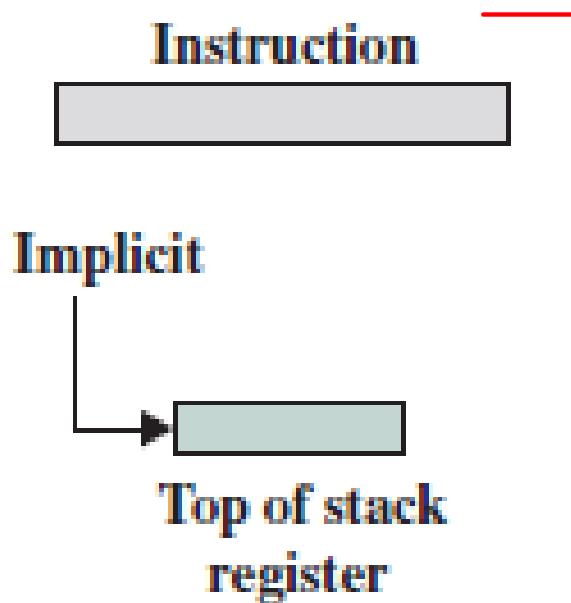
- Use:

- to construct a multiway branch table

\_\_\_\_\_

# Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD    Pop top two items from stack and add, push the result on stack top



# x86 Addressing Modes

L

Virtual or effective address is offset into segment

- Starting address plus offset gives linear address
- This goes through page translation if paging enabled

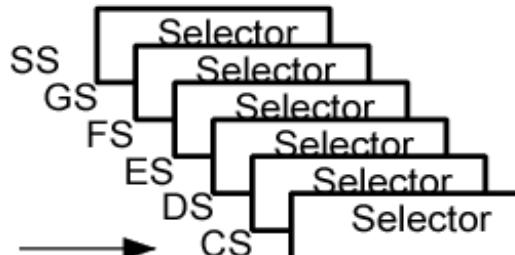
12 addressing modes available

- Immediate
- Register operand
- Displacement
- Base
- Base with displacement
- Scaled index with displacement
- Base with index and displacement
- Base scaled index with displacement
- Relative

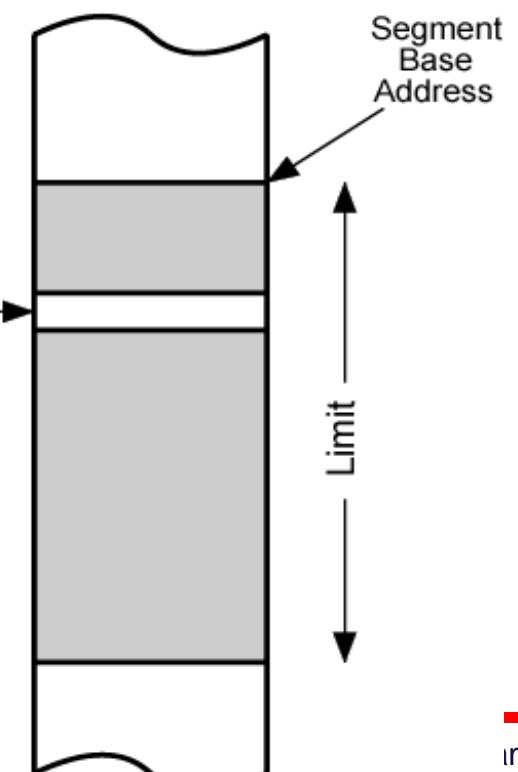
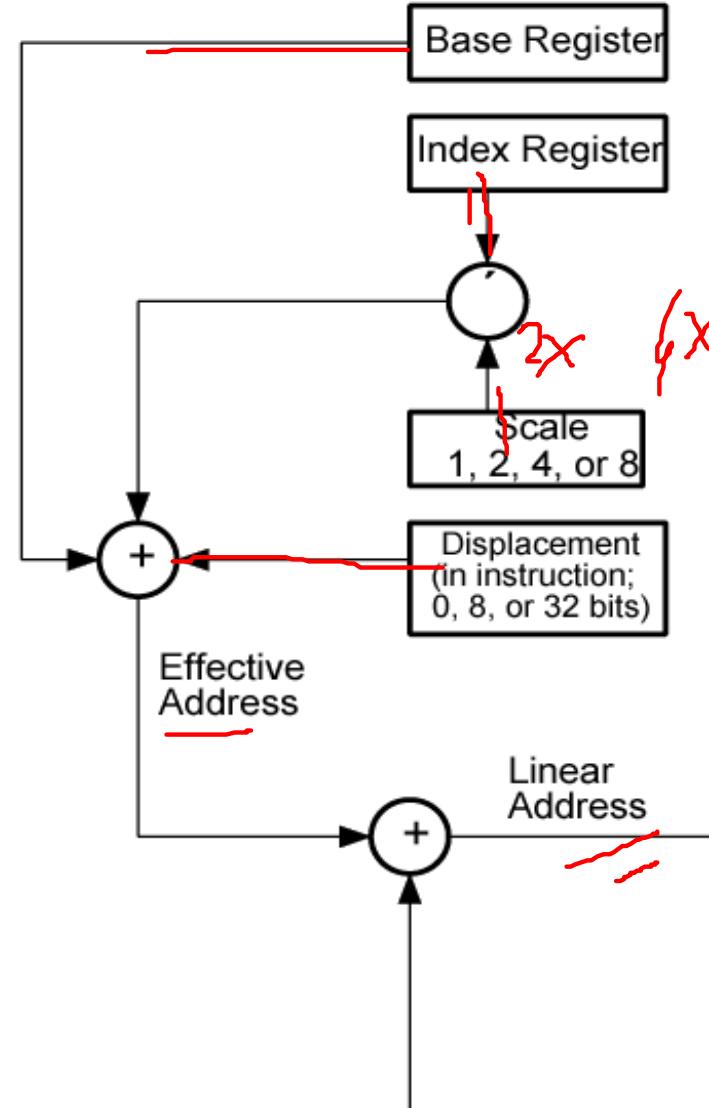
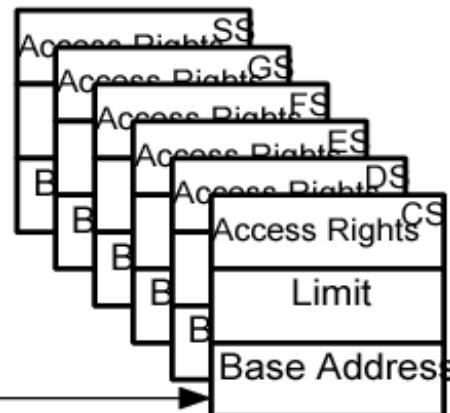
Scal e

# x86 Addressing Mode Calculation

Segment Registers

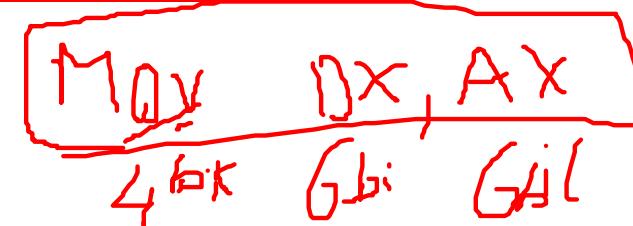


Descriptor Registers



# Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set



# Instruction Length

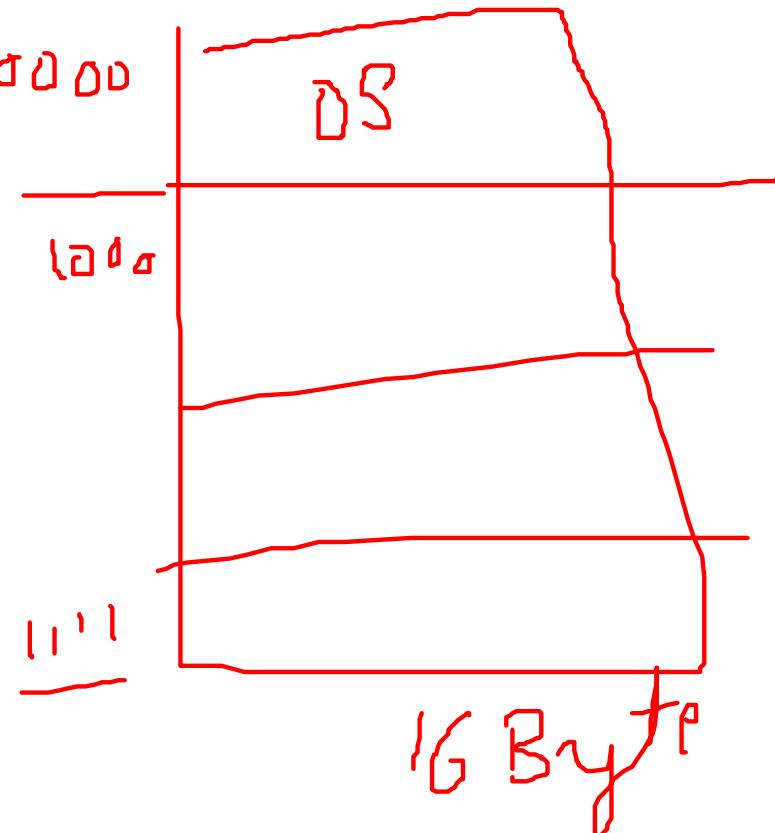
Affected by and affects:

- Memory size
- Memory organization
- Bus structure
- CPU complexity
- CPU speed

Trade off between powerful instruction repertoire and saving space

# Allocation of Bits

- Number of addressing modes
- Number of operands → 2, 0, 1
- Register versus memory
- Number of register sets
- Address range
- Address granularity





# MIPS-SINGLE CYCLE

**BITS** Pilani  
Pilani Campus

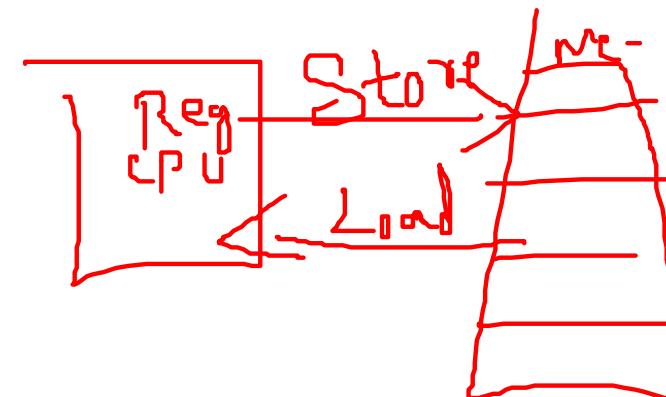
L/S  $\leftarrow$  I-type  $\rightarrow$

R-type  $\rightarrow$  ALU

I-type  $\rightarrow$  BRJ/JMP

# MIPS Architecture

- MIPS = Microprocessor without Interlocked Pipelined Stages.
- MIPS follows RISC principles and based on Harvard Architecture
- MIPS architecture is a register architecture
- MIPS ISA is Load/Store architecture
- R2000 is a 32-bit processor
- Uses fixed length instruction format.



# MIPS Register Set

## Types of Registers

- 32 general-purpose registers (\$0 - \$31)
- Program Counter (PC)
- Two special Purpose register (HI and LO) 

*32 bit*  
 $\times$   
*32 bit*
- Used to hold the results of integer *multiply* and *divide* instruction.
- *integer multiply* operation, HI and LO register hold the 64-bit result.
- *integer divide* operation, the 32-bit quotient is stored in the LO and the remainder in the HI register.

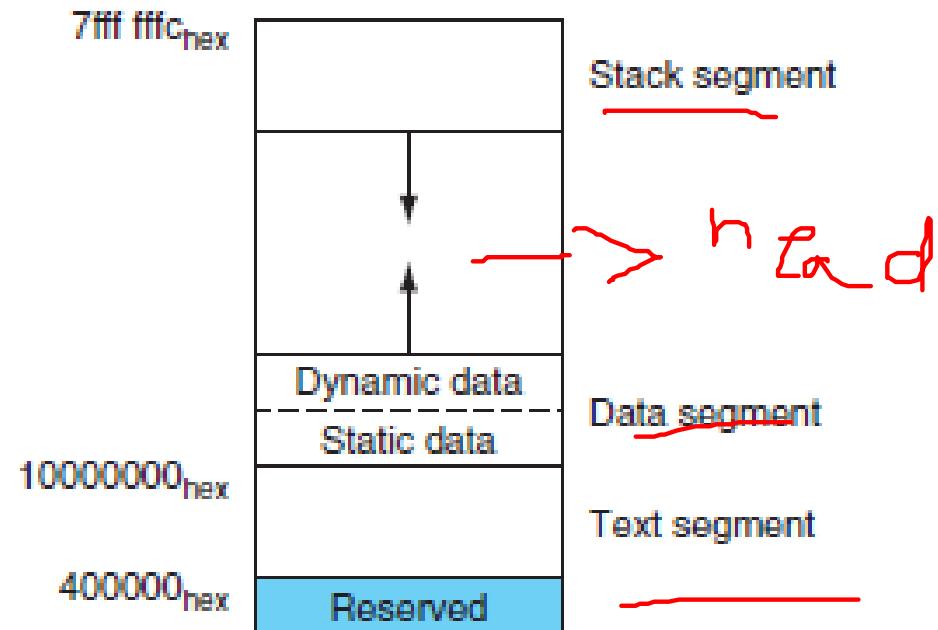
# General-Purpose Register Usage Convention



Register name	Number	Intended usage
<u>Zero</u>	0	Constant 0
<u>at</u>	1	Reserved for assembler
v0,v1	2,3	Values for function results and <u>Exp</u> evaluation
a0,a1,a2,a3	4-7	<u>Arguments</u> 1-4
t0-t7	8-15	Temporary (not preserved across call)
s0-s7	16-23	Saved temporary( <u>preserved</u> across call)
t8,t9	24,25	Temporary (not preserved across call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to Global area
sp	29	Stack Pointer
fp	30	Frame pointer(if needed);Otherwise, a saved register \$s8
ra	31	Return address(used by a procedure call)

# Memory Usage

- A program's address space consists of three parts:
  - Code/Text segment
  - Data segment
  - Stack segment
  -

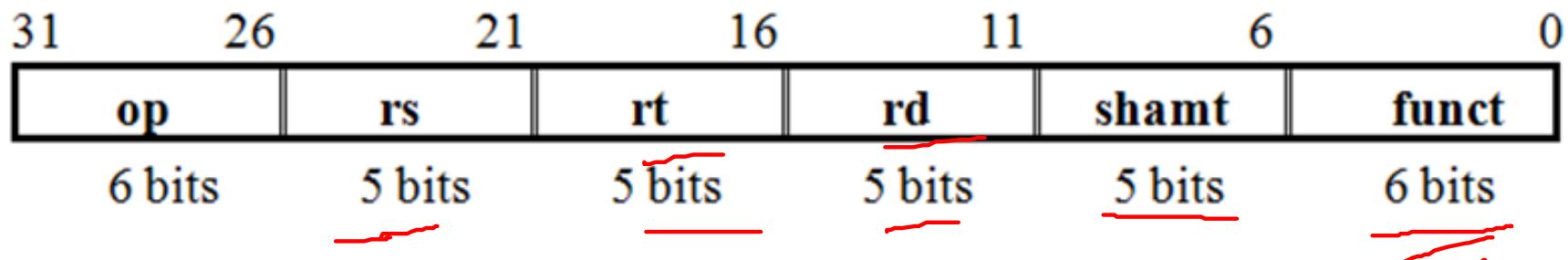


# Instruction Format

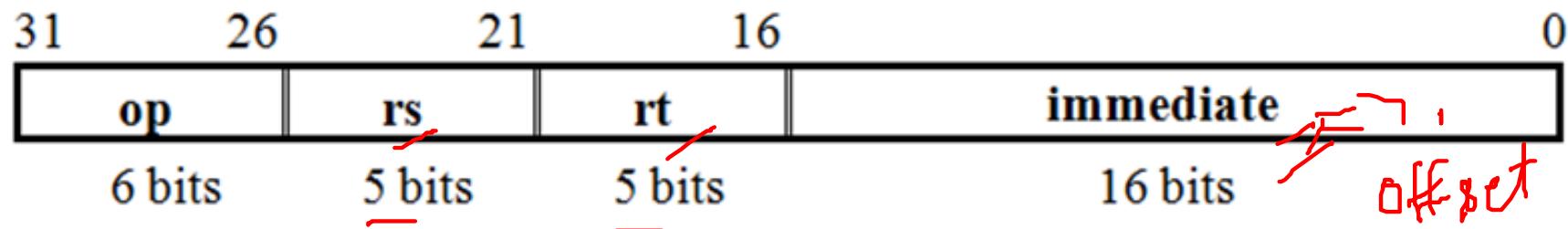
- Fixed-length instruction format
- 32-bits long
- Three different instruction formats:
  1. Immediate (I-type) → LIS
  2. Jump (J-type) → CBR / JMP
  3. Register (R-type) → ALU
- Meaning of various fields in the instruction format
  - op: Opcode
  - rs: The first register source operand
  - rt: The second register source operand
  - rd: The register destination operand
  - shamt / sa: shift amount
  - funct: function code

# Instruction Format

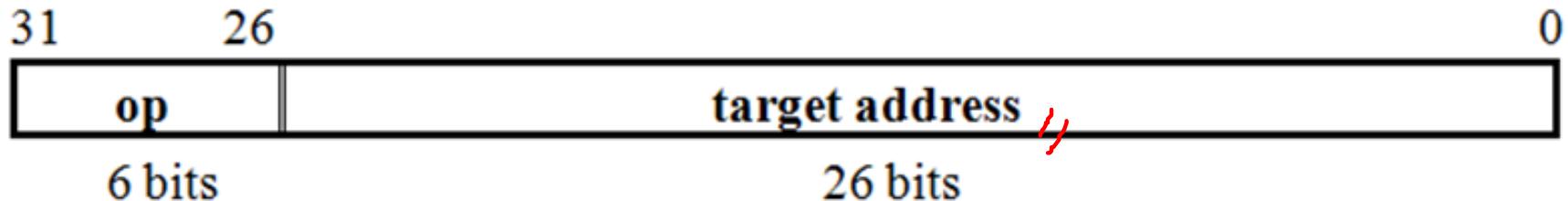
R-Type:



I-Type:



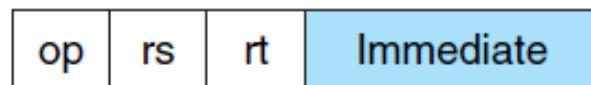
J-Type:



# Addressing modes

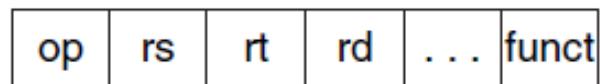
- Immediate Addressing Mode

1. Immediate addressing



$\rightarrow I\text{-type}$

- Register Addressing Mode

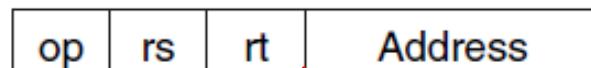


Registers

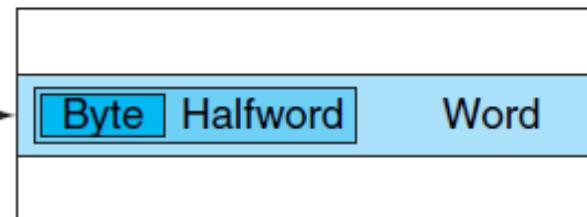
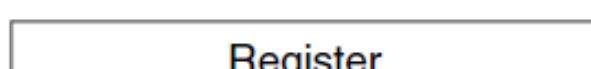
Register

$R\text{-type}$

- Base or Displacement Addressing Mode



Memory

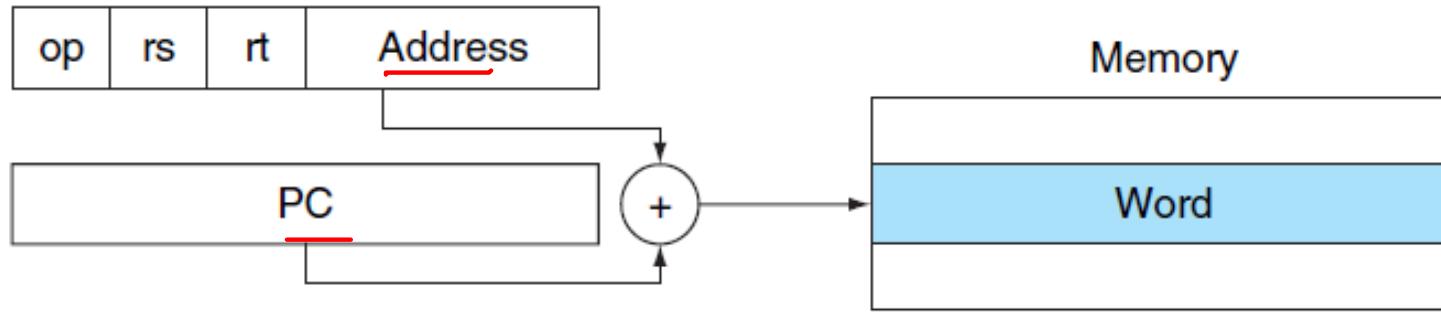


$R\text{-type}$

# Addressing modes

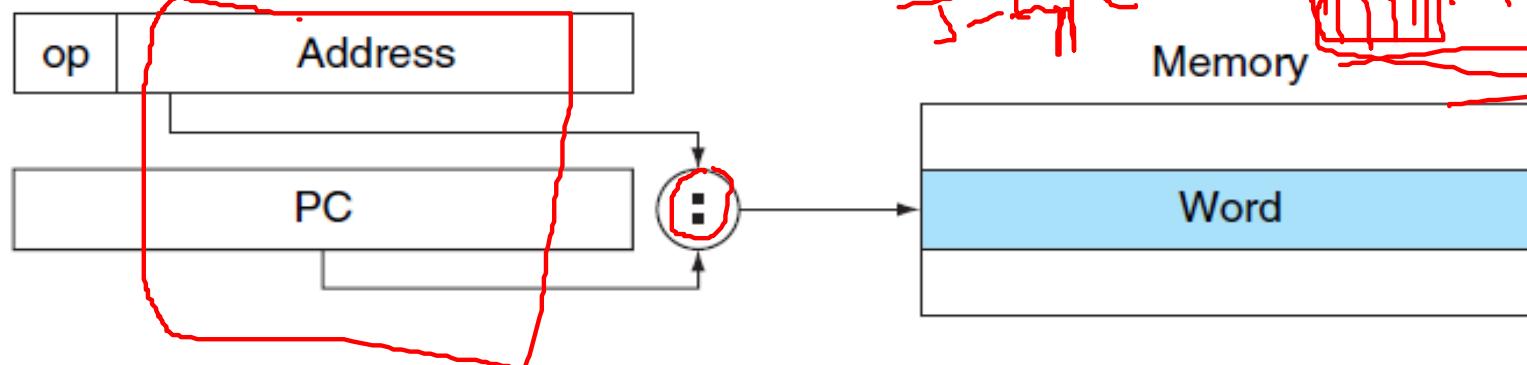
- PC- Relative Addressing Mode

*J - type P*



- Pseudo Direct Addressing Mode

*J - type P*



# MIPS Instruction Set

- R { Arithmetic Instructions ( ADD, SUB etc.)
- I { Logical Instructions (OR, AND, SHIFT, etc. )
- F { Data Transfer Instructions (Load and Store)
- J { Decision Making Instructions ( J, BNE, BEQ, etc.)
- S { Stack Related Instructions (PUSH and POP)

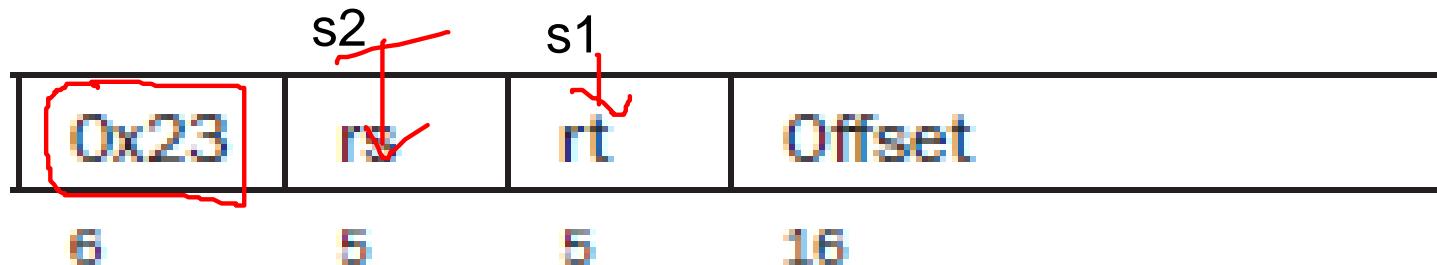
# A Basic MIPS Implementation

- Basic implementation includes a subset of core MIPS instruction set
  - Memory reference instruction
    - lw and sw
  - Arithmetic and logical instructions
    - add, sub, and, or and slt
  - Branch instructions
    - beq and j

# Memory Reference Instruction - lw → I = type

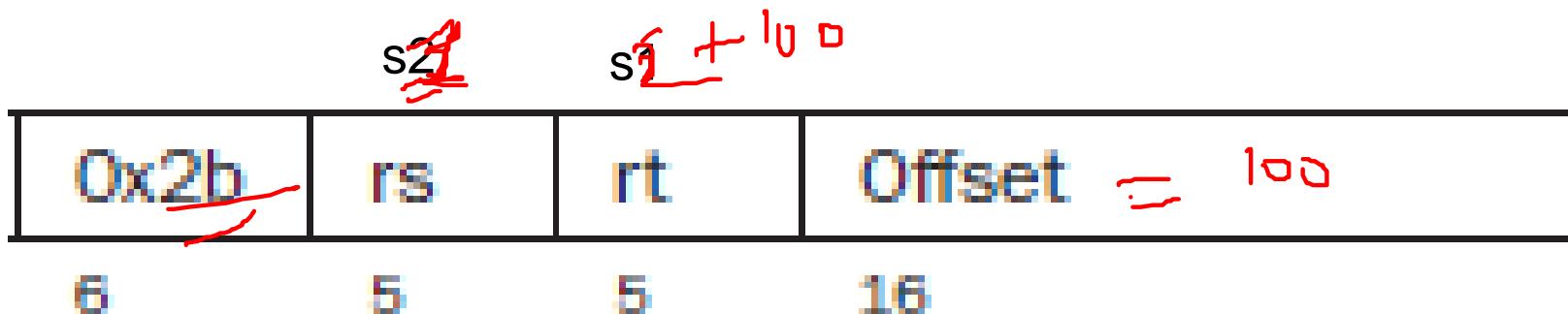
- copies data from memory to register
- Format : lw reg, address
- Example : lw \$s1, 100(\$s2)
- $\$s1 \leftarrow \text{memory}[100 + \$s2]$
- Alignment restriction
- MIPS is Big endian
- Spilling registers

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	-	1	2	3	4	5	6
							7



# Memory Reference Instruction - sw

- Store Instruction
- copies data from register to memory
- Format : sw reg, address
- Example : sw \$s1, 100(\$s2)  
memory[100 +\$s2]  $\leftarrow$  \$s1



# Arithmetic Instructions :add

add des, src1, src2

Addressing mode: register

Example: add \$s3, \$s1, \$s2

meaning : \$s3 = \$s1 + \$s2

opcode : 0 , function: 0x20

sa :0

R type

\$s1      \$s2      \$s3

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

# Arithmetic Instructions :sub

**sub des, src1, src2**

Addressing mode: register

Example: sub \$s3, \$s1, \$s2

meaning :  $\$s3 = \$s1 - \$s2$

opcode : 0 , function: 0x22

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x22
6	5	5	5	5	6

# Logical instructions :and

and des, src1, src2

Addressing mode: register

Example: and \$s3, \$s1, \$s2

meaning :  $\$s3 = \$s1 \& \$s2$  (bit by bit)

opcode : 0 , function: 0x24

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

# Logical instructions :or

or des, src1, src2

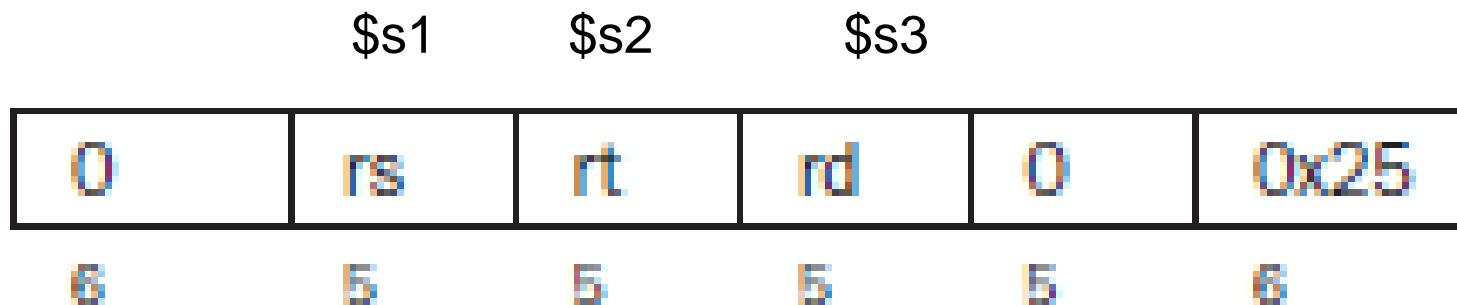
Addressing mode: register

Example: or \$s3, \$s1, \$s2

meaning :  $\$s3 = \$s1 \mid \$s2$  (bit by bit)

opcode : 0 , function: 0x25

sa :0



# Logical instructions :slt

set on less than : slt

Format: slt des, src1, src2

set des = 1, if src1 < src2

Example:

slt \$s3, \$s1, \$s2



\$s1	\$s2	\$s3			
0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

Set register rd to 1 if rs is less than rt, and to 0 otherwise

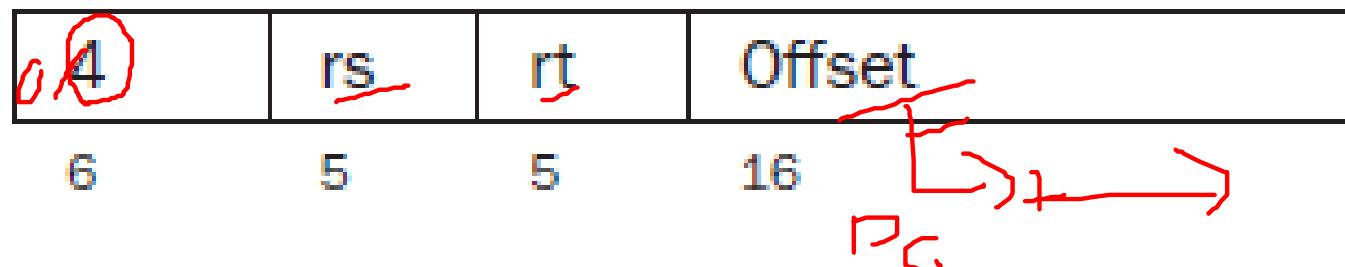
# Branch Instructions: beq and j

beq : branch if equal

- format : beq \$s1, \$s2, label

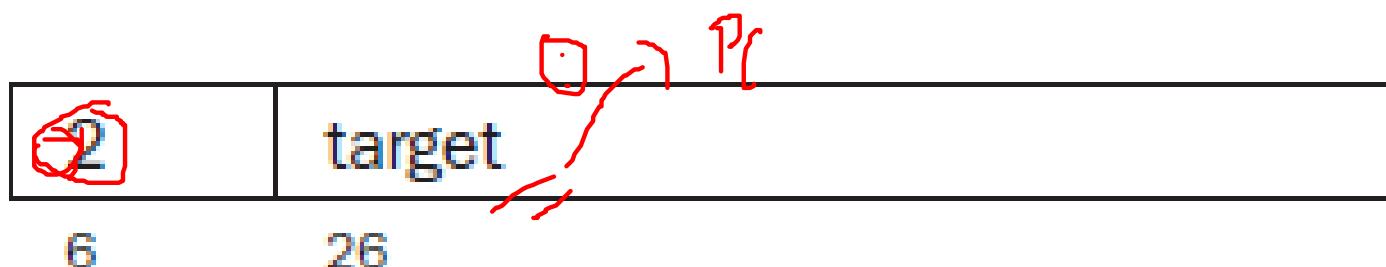
- If  $\$s1 = \$s2$  then branch to address specified as Label

$\begin{array}{r} 100 \\ - 100 \\ \hline \end{array}$



j : jump unconditional

- format : j label



[Back](#)

- and \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x24
6	5	5	5	5	6

## Summary

- lw \$s1, 100(\$s2)

0x23	rs	rt	Offset
6	5	5	16

- sw \$s1, 100(\$s2)

0x2b	rs	rt	Offset
6	5	5	16

- add \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

- sub \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x22
6	5	5	5	5	6

- or \$s1, \$s2, \$s3

0	rs	rt	rd	0	0x25
6	5	5	5	5	6

- slt \$t0, \$s1, \$s2

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

- beq \$s1, \$s2, label

4	rs	rt	Offset
6	5	5	16

- j label

2	target
6	26



# COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 7

Dr. Lucy J. Gudino  
WILP & Department of CS & IS



**BITS** Pilani  
Pilani Campus



# MIPS Single Cycle

**BITS Pilani**  
Pilani Campus

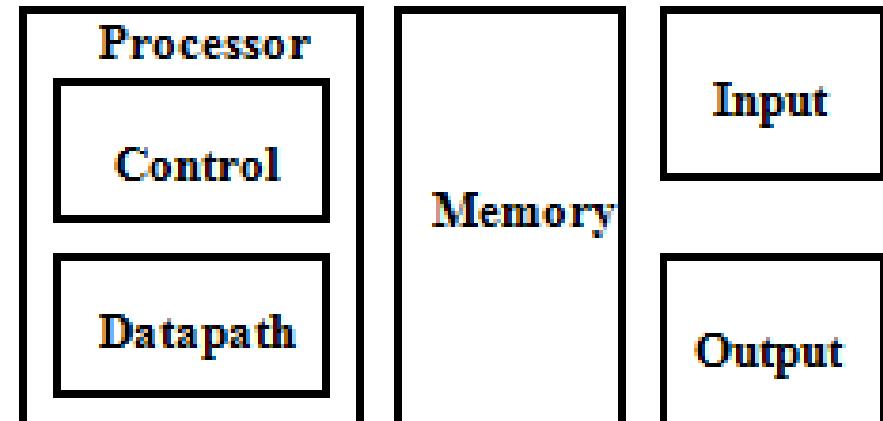


# Data Path Design

**BITS Pilani**  
Pilani Campus

# Introduction

- The Five Classic Components of a Computer
  - Datapath: The component of the processor that performs data processing operations
  - Control: The component of the processor that commands the datapath, memory and I/O devices according to the instructions of the memory
  - Memory
  - Input device
  - Output device



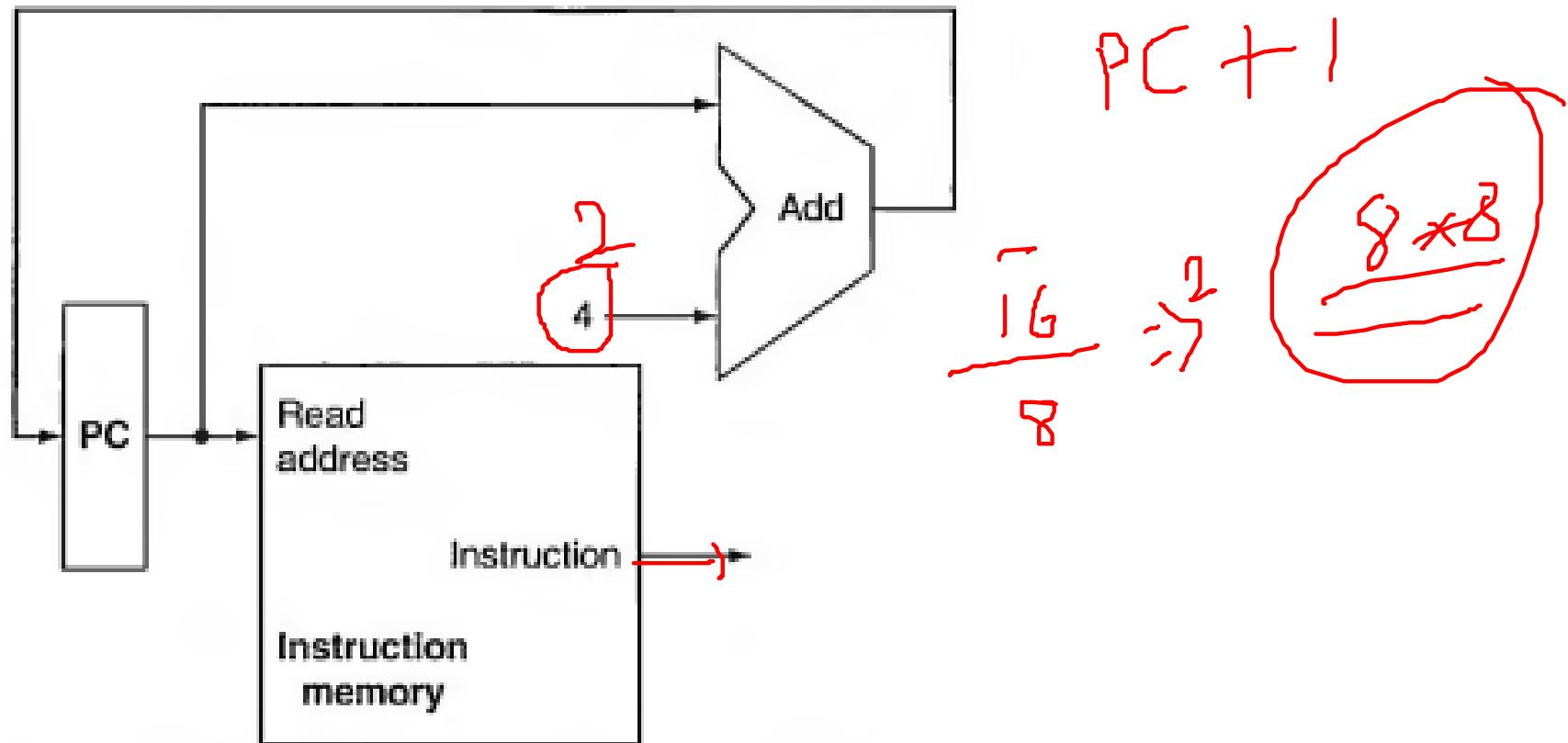
# Building a datapath

---

- Datapath elements :memory - instruction/data, register file, ALU, Adders, Multiplexers....
- Demonstrate Datapath implementation for the following instruction:
  - lw and sw
  - add, sub, and, or and slt
  - beq and j
- Instruction Cycle = Fetch + Execute

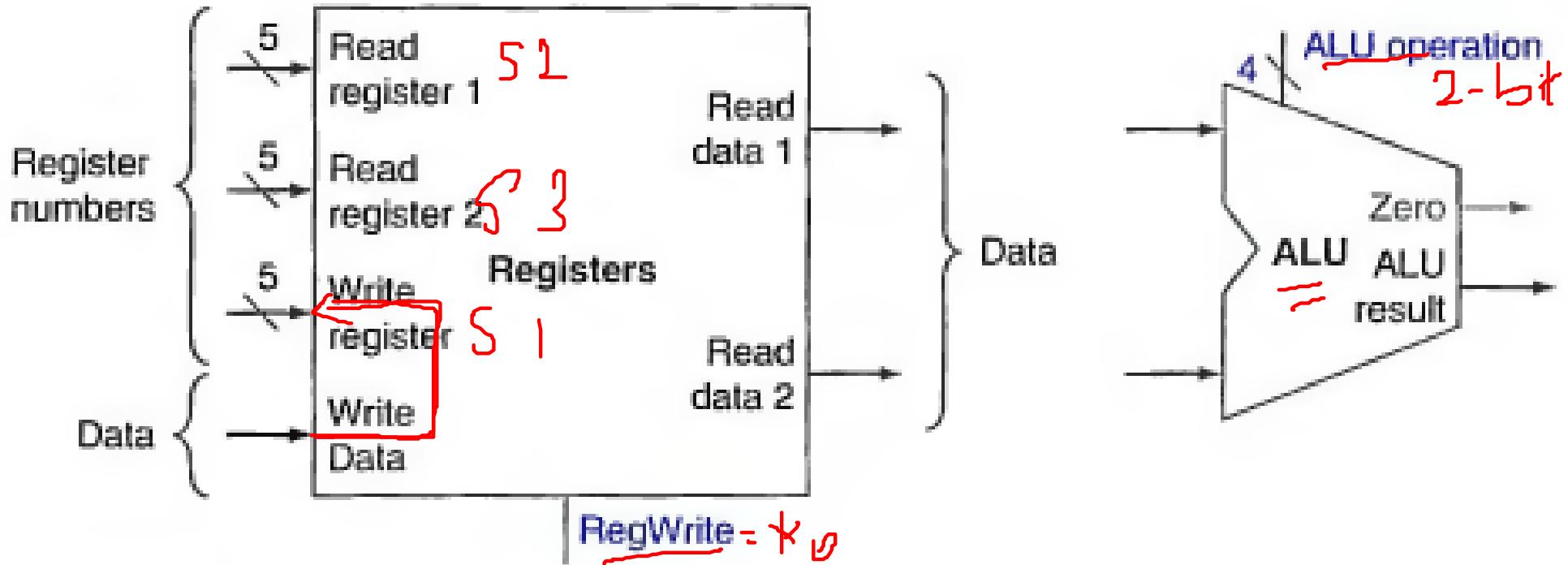
# Do this...

write a datapath used for fetching instructions and incrementing the program counter



# Data Path for R-type instruction

Instruction  
Summary



a. Registers

b. ALU

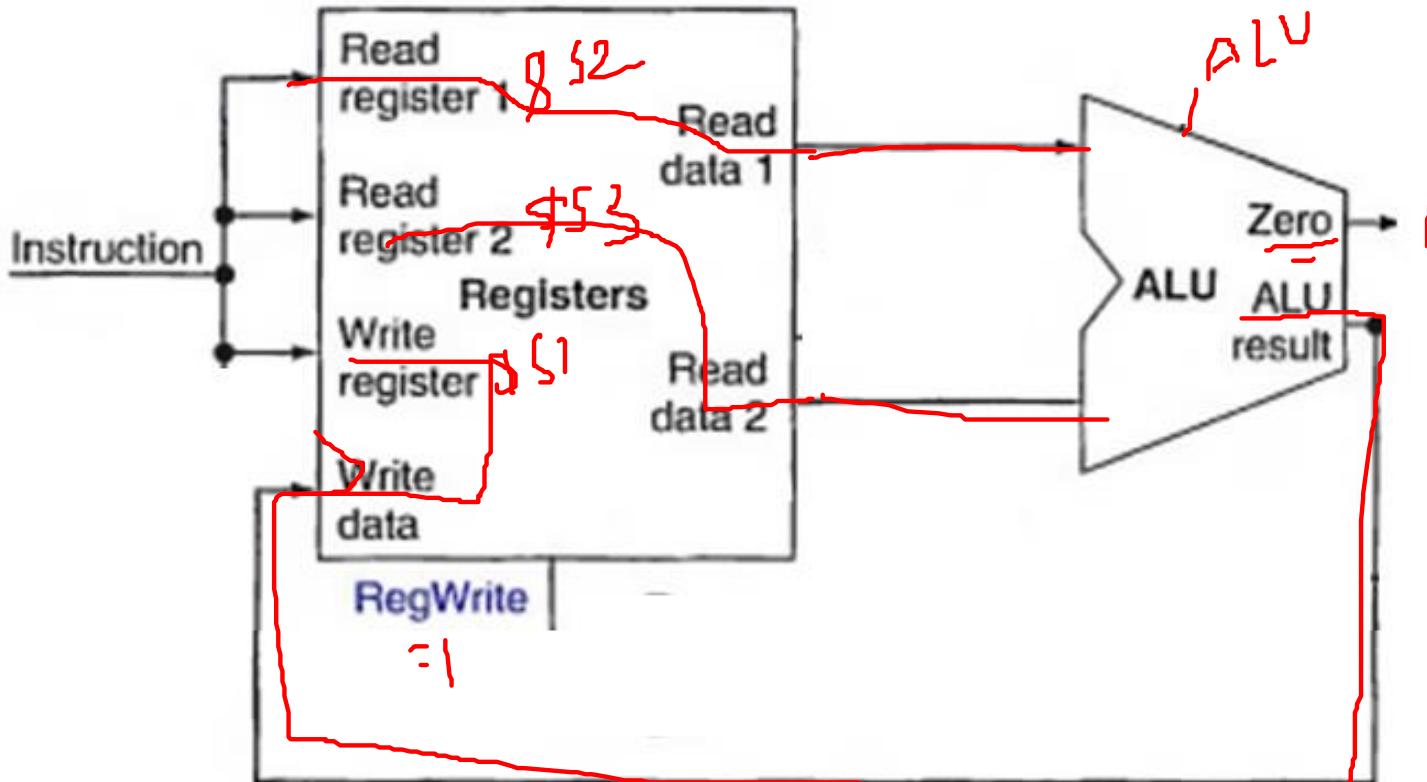
Examples:  
add, sub, and, or, slt  $\rightarrow$  rs, rt, rd

add \$s1, \$s2, \$s3



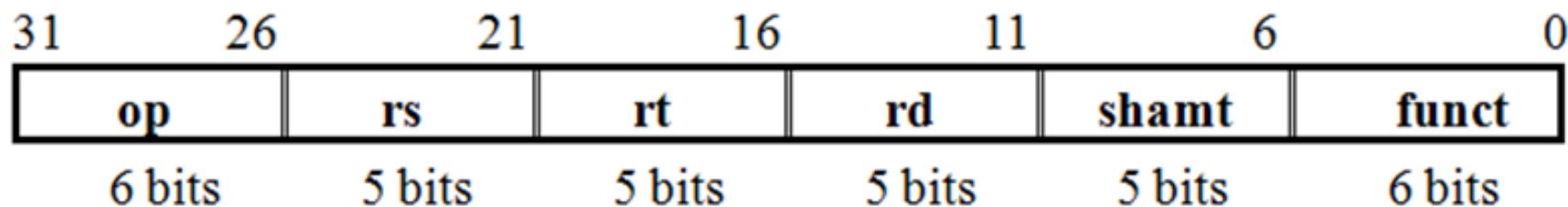
# Data Path for R-type instruction...

Instruction Summary

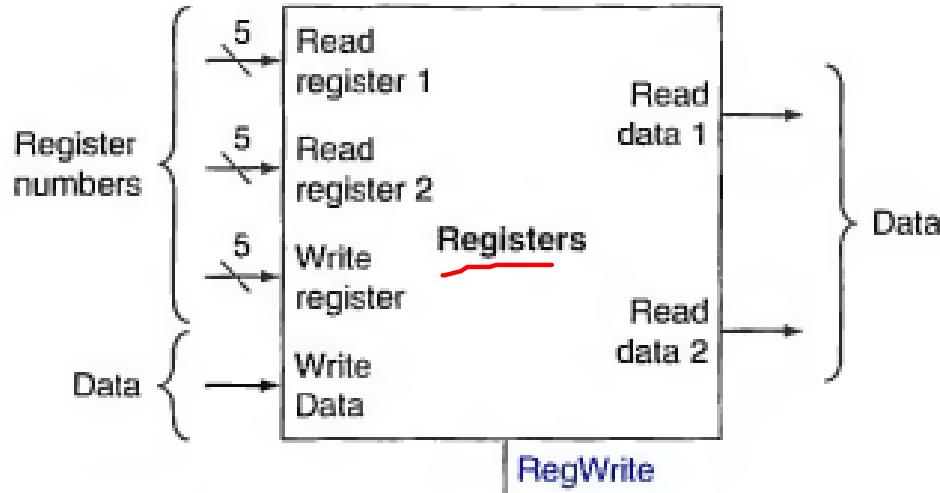


- add \$s1, \$s2, \$s3

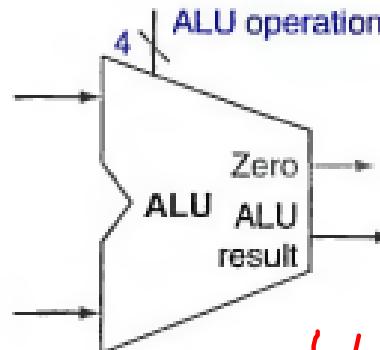
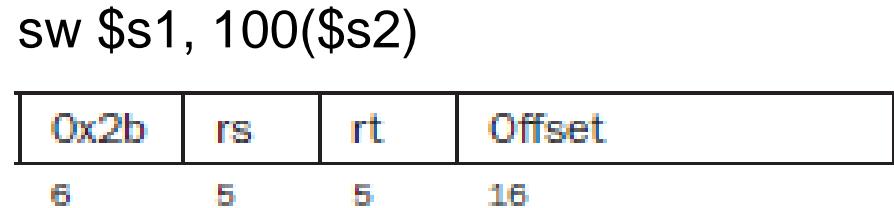
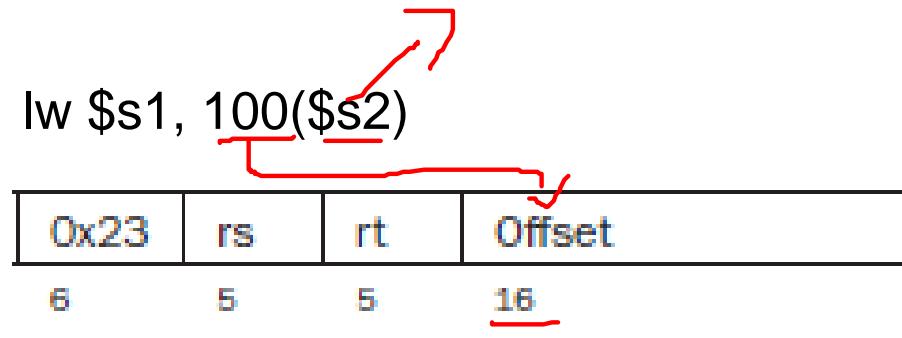
The datapath for the memory instructions and the R-type instructions.



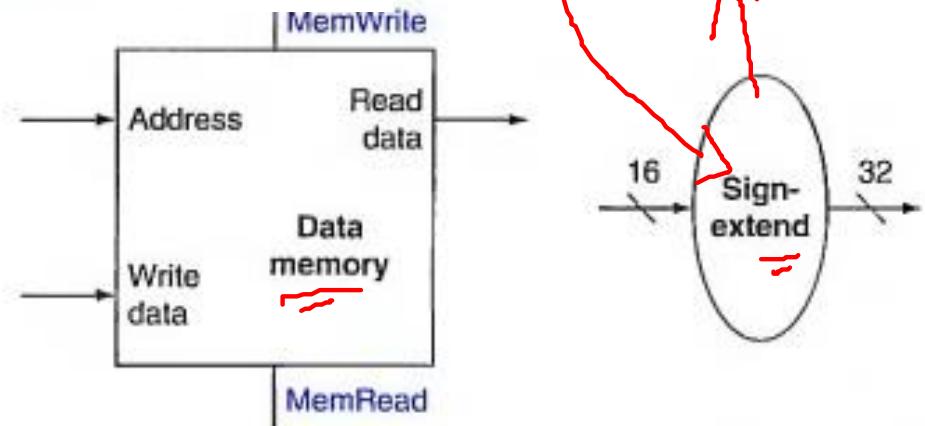
# Data Path for lw and sw instructions



a. Registers



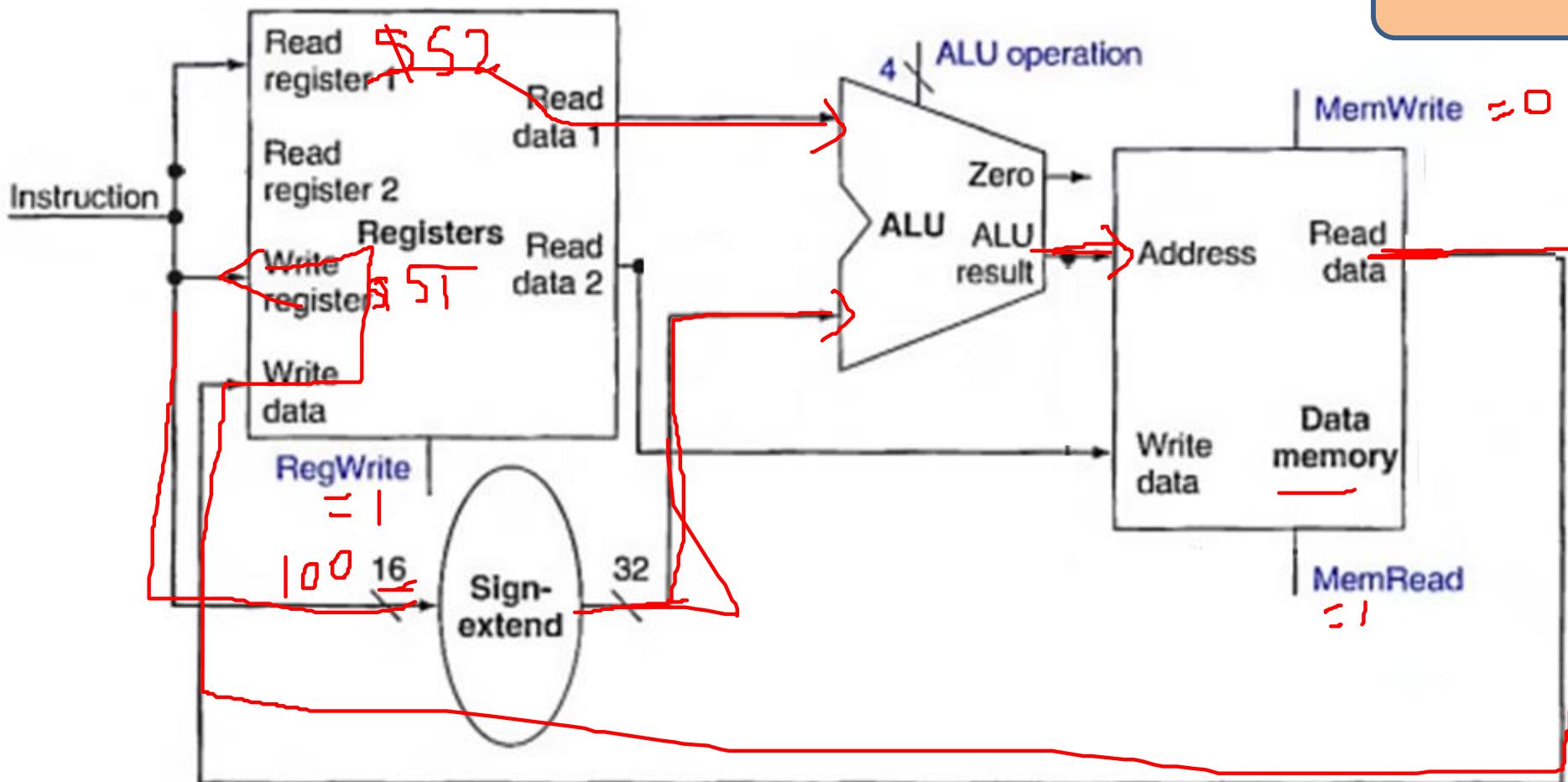
b. ALU



a. Data memory unit

Instruction Summary

# Data Path for lw instructions



lw \$s1, 100(\$s2)

~~s2~~ ~~s1~~

0x23	rs	rt	Offset
------	----	----	--------

6

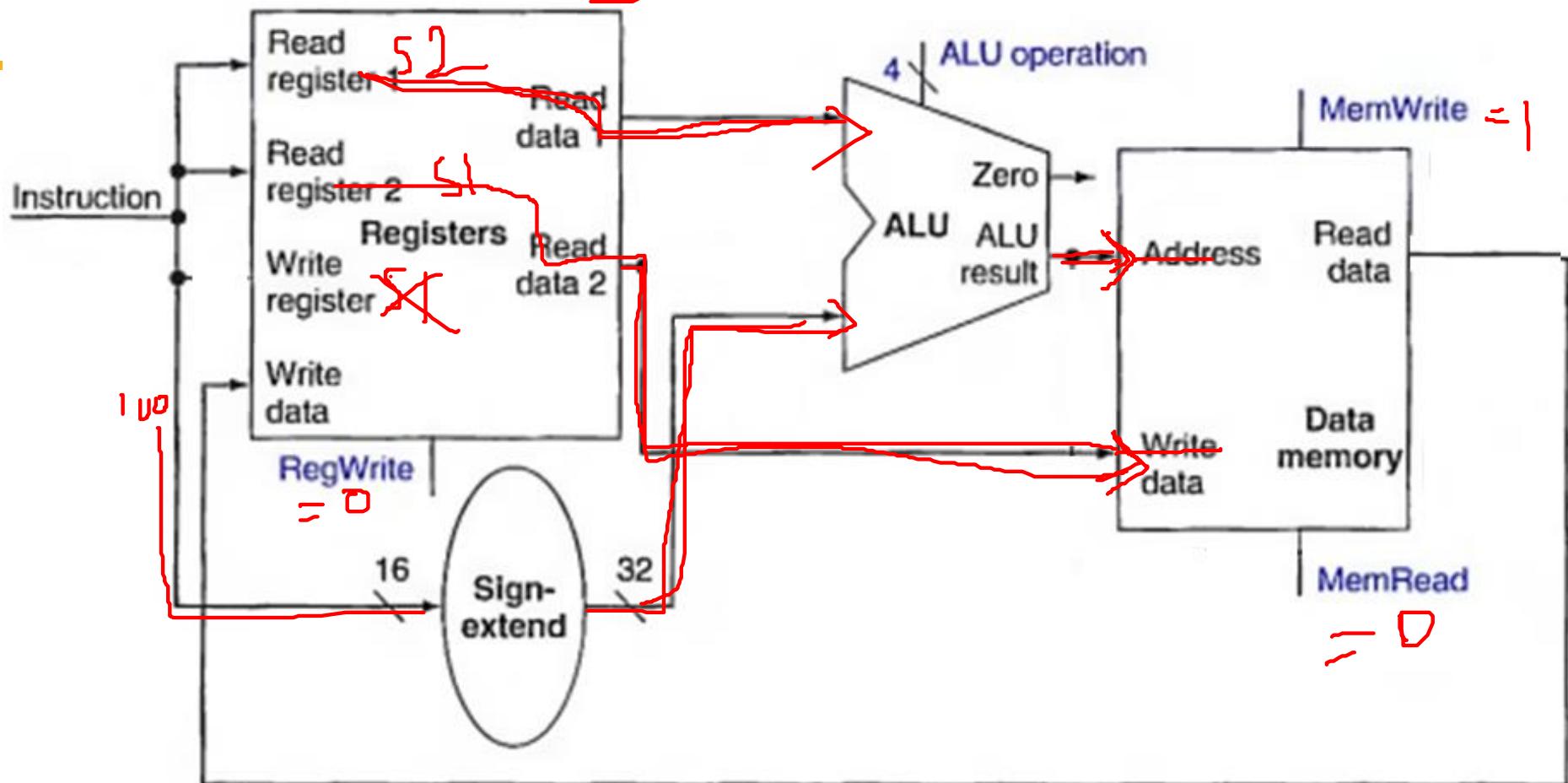
5

5

16

\$S1 ← mem[100 + 9S2]

# Data Path for sw instructions



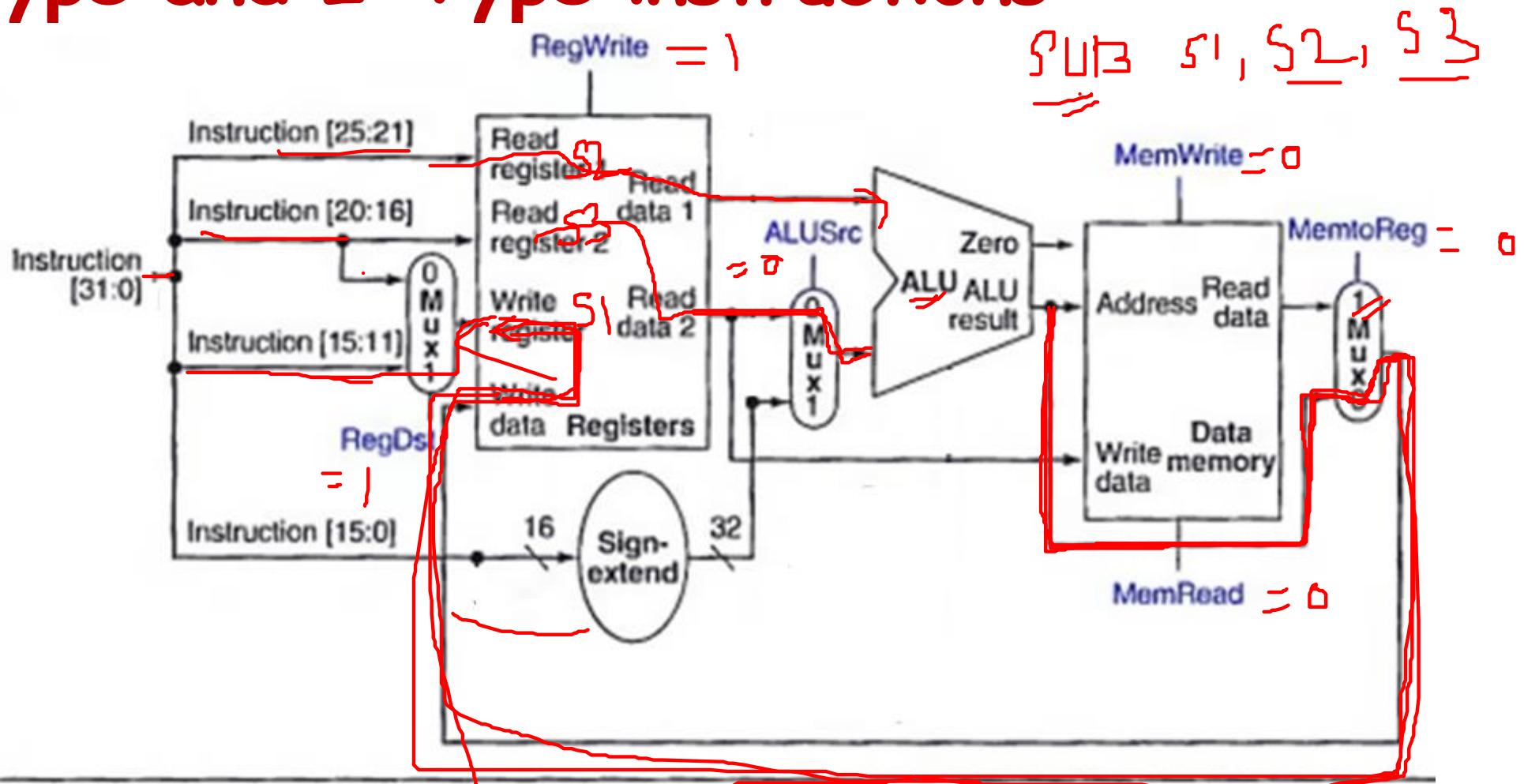
`sw $s1, 100($s2)`

s2 s1

0x2b	rs	rt	Offset
------	----	----	--------

6 5 5 16

# Combined Datapath for R-Type and I-Type instructions



The datapath with all necessary multiplexors and all control lines identified.

R-Type

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

I-Type

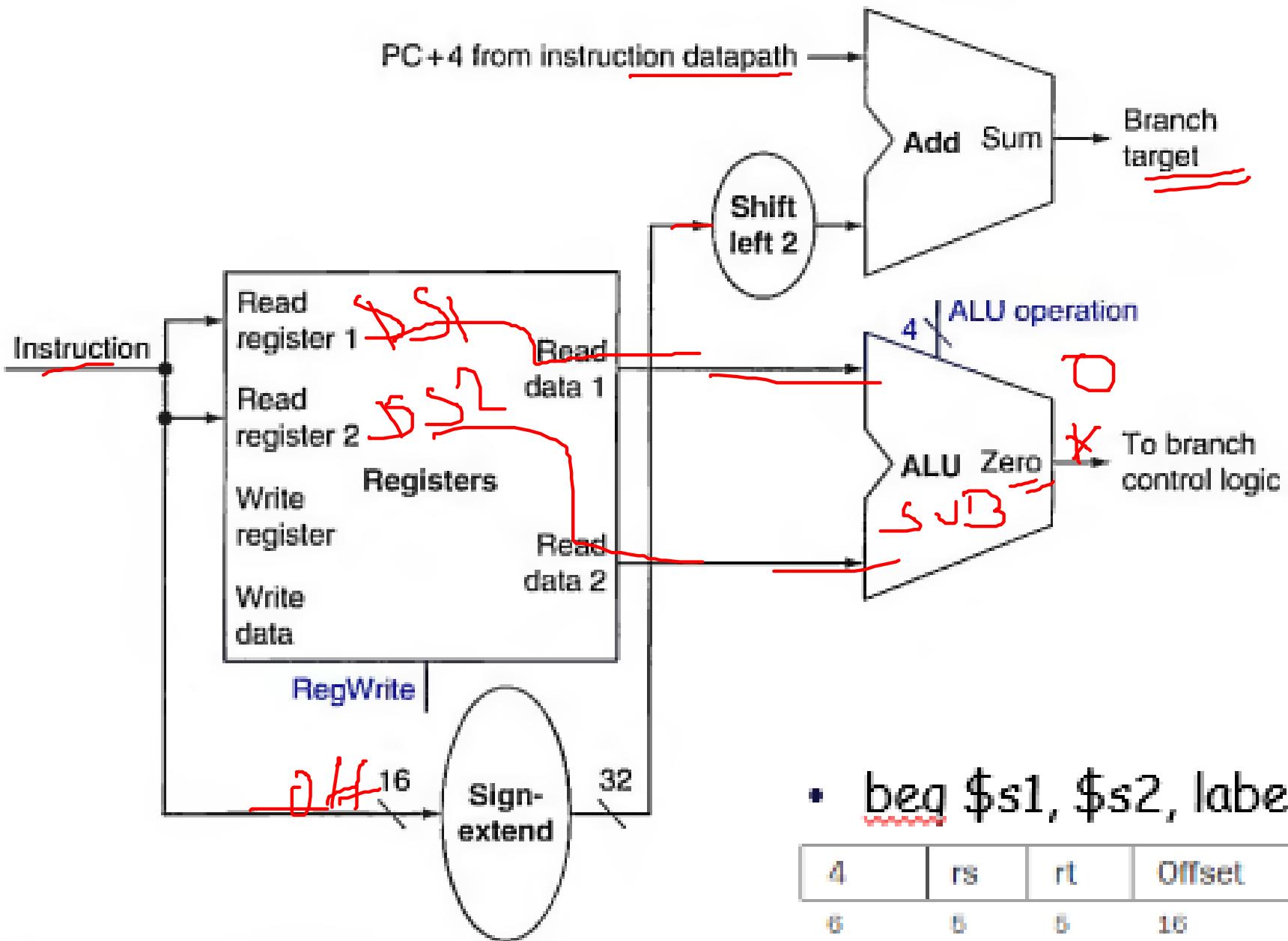
0x2b	rs	rt	Offset
6	5	5	16

# beq instruction

---

- beq \$s1, \$s2, offset
- Two important points to be noted
  - The instruction set architecture defines that the base for the branch address calculation is the address of the instruction following the branch
  - the architecture also states that the offset field is shifted left 2 bits
- branch taken or branch not taken

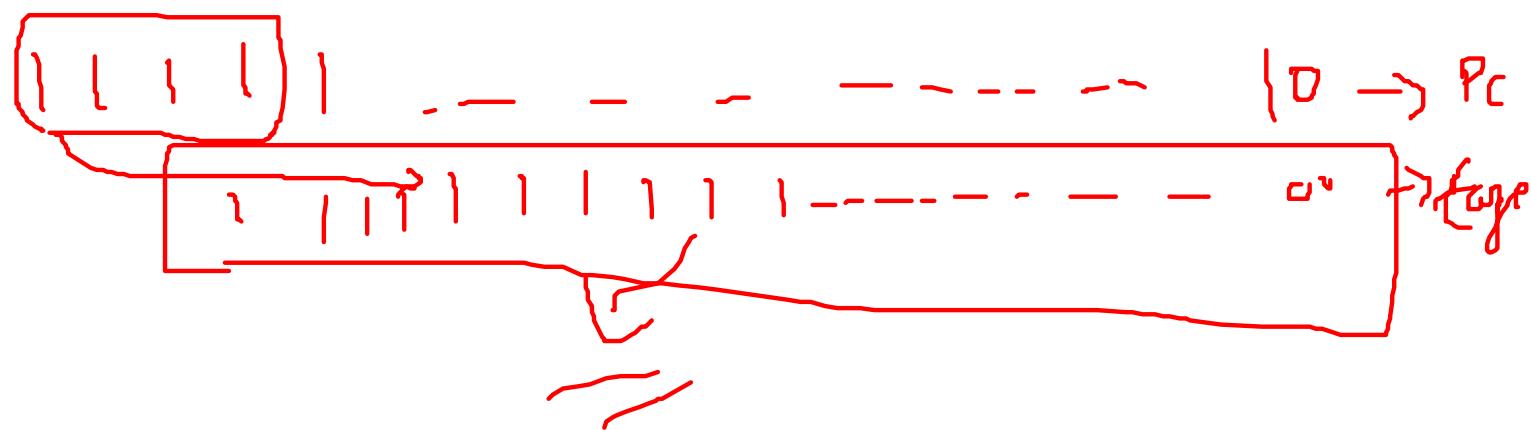
# Datapath for beq instruction



# Jump instruction

j <26 bit jump offset>

the jump instruction operates by replacing the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits



- j label

2	target
6	26

# Creating a single cycle datapath

---

Simplest datapath executes all instructions in one clock cycle

- No element in the datapath can be used more than once
- Any element needed more than once must be duplicated

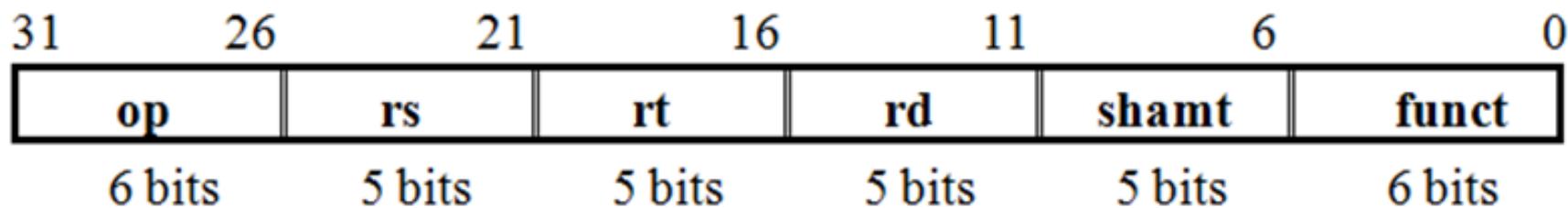
# Control unit design

[Previous](#)

## The ALU control

- used by load and store instructions
- used by r-type instruction : AND, OR, add, sub and slt
  - opcode : zero
  - operation performed based on “function field”

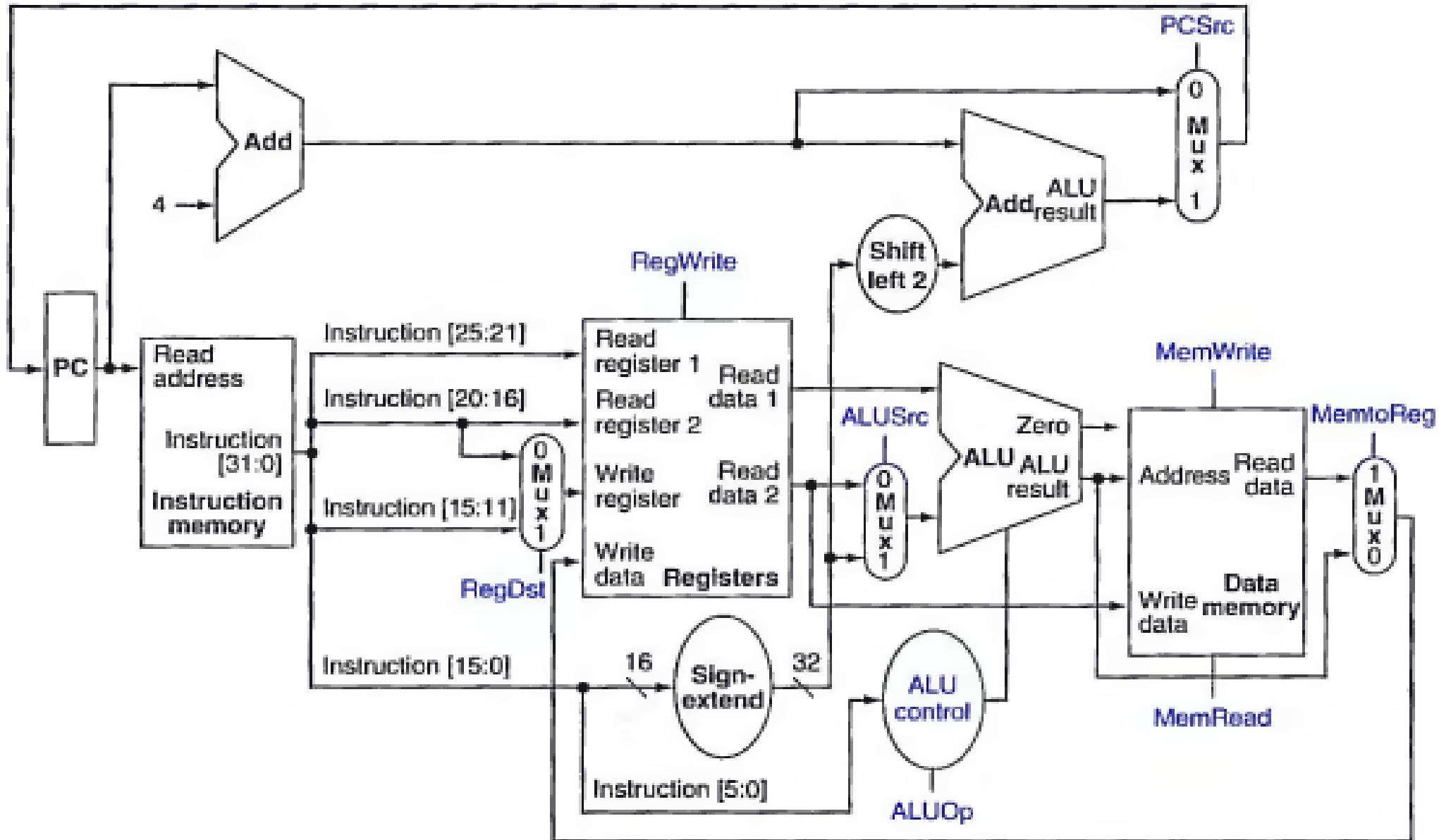
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



# Contd...

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01 → beq (sub)
    - 10 → add, subtract, AND, OR and slt
- Main Control unit generates 7 control signals : RegDst, RegWrite, ALUSrc, PCSsrc, MemRead, MemWrite and MemtoReg

# Datapath



The datapath with all necessary multiplexors and all control lines identified.

# Effects of 7 control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

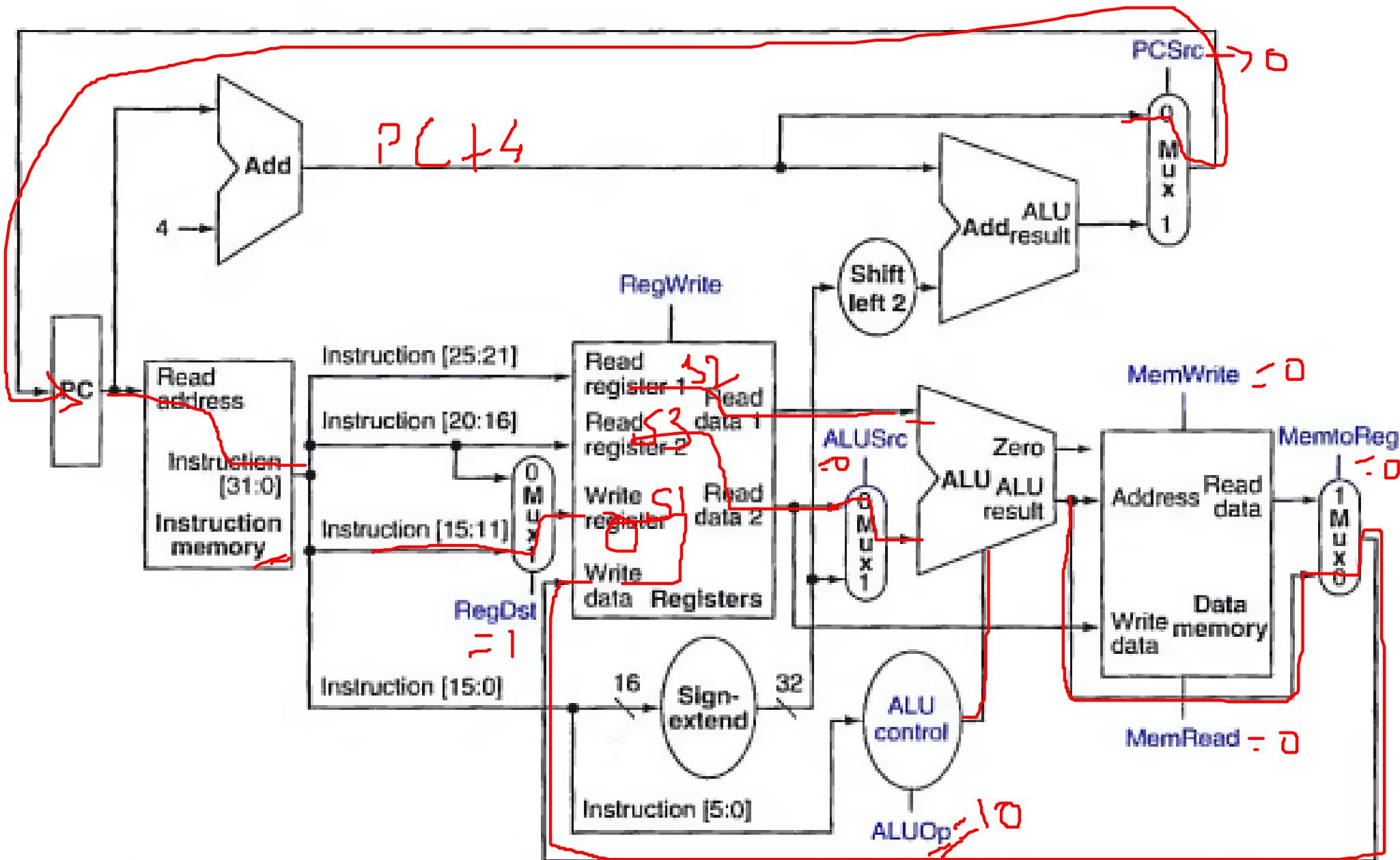
# Major observations

- opcode is always contained in bits 31:26
- The two registers to be read are always specified by the rs and rt fields at positions 25:21 and 20:16
  - r-type
  - branch equal
  - store
- The base register for load and store instruction is always in bit positions 25:21 i.e. rs
- The 16 bit offset for branch equal, load and store is always in positions 15:0
- The destination register is in one of two places.
  - load instruction → rt 20:16
  - r type instruction → rd 15:11

# Execution steps: add \$s1, \$s2, \$s3

1. The instruction is fetched from Int<sub>Mem</sub>, and the PC is incremented to PC + 4
2. Two registers S2 and S3 are read from the register file
3. The ALU operates on the data read from the register file, based on the OP code
4. The result from the ALU is written into the S1 register in register file

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1, \$s2, \$s3



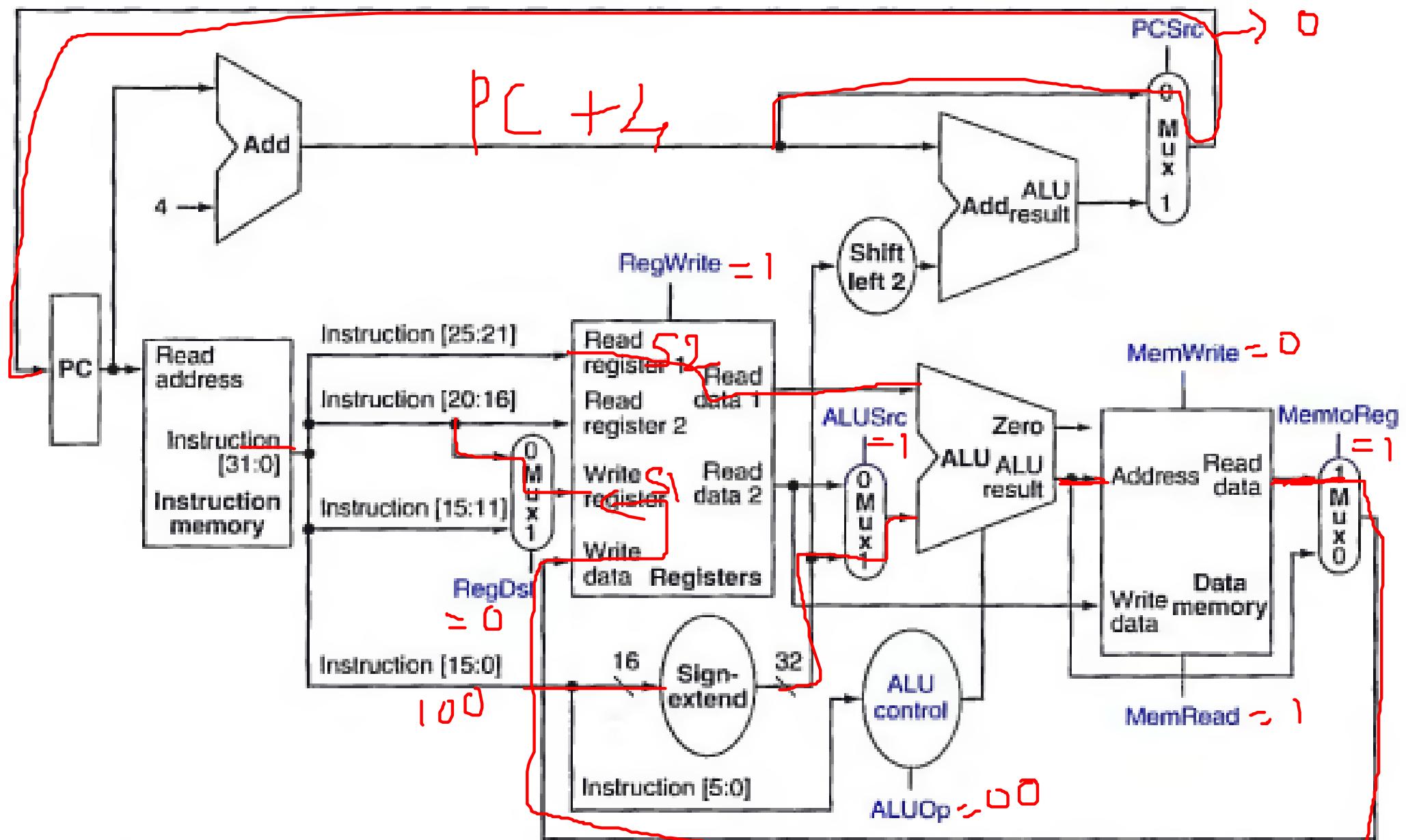
The datapath with all necessary multiplexors and all control lines identified.

# Execution steps: lw \$s1, 100(\$s2)



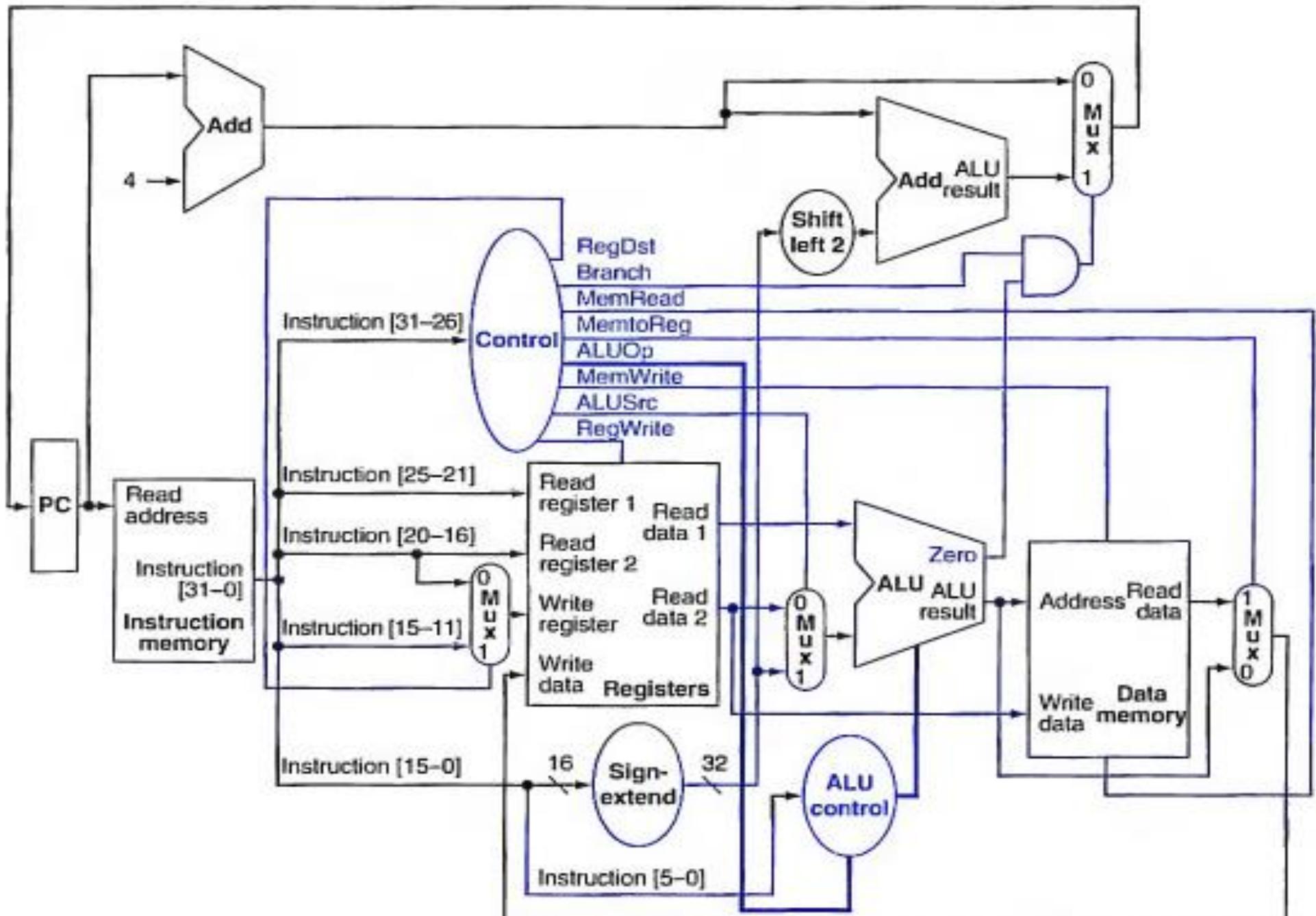
1. The instruction is fetched, and the PC is incremented
2. A register  $s_2$  value is read from the register file
3. The ALU computes the sum of the value read from the register file and the sign extended lower 16 bits of the instruction
4. the sum from the ALU is used as the address for the data memory
5. The data from the memory unit is written into  $s_1$  in register file

Name	Format	Example					Comments
lw	I	35	18	17	100	lw	\$s1, 100(\$s2)



The datapath with all necessary multiplexors and all control lines identified.

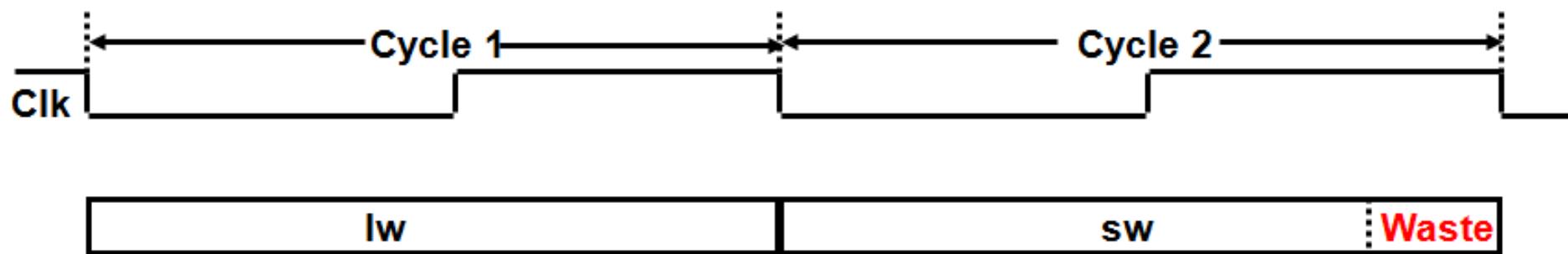
# Complete Data Path



# Single Cycle Advantages & Disadvantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the **slowest** instruction
- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle but is simple and easy to understand

=





# Control Unit Design

**BITS Pilani**  
Pilani Campus



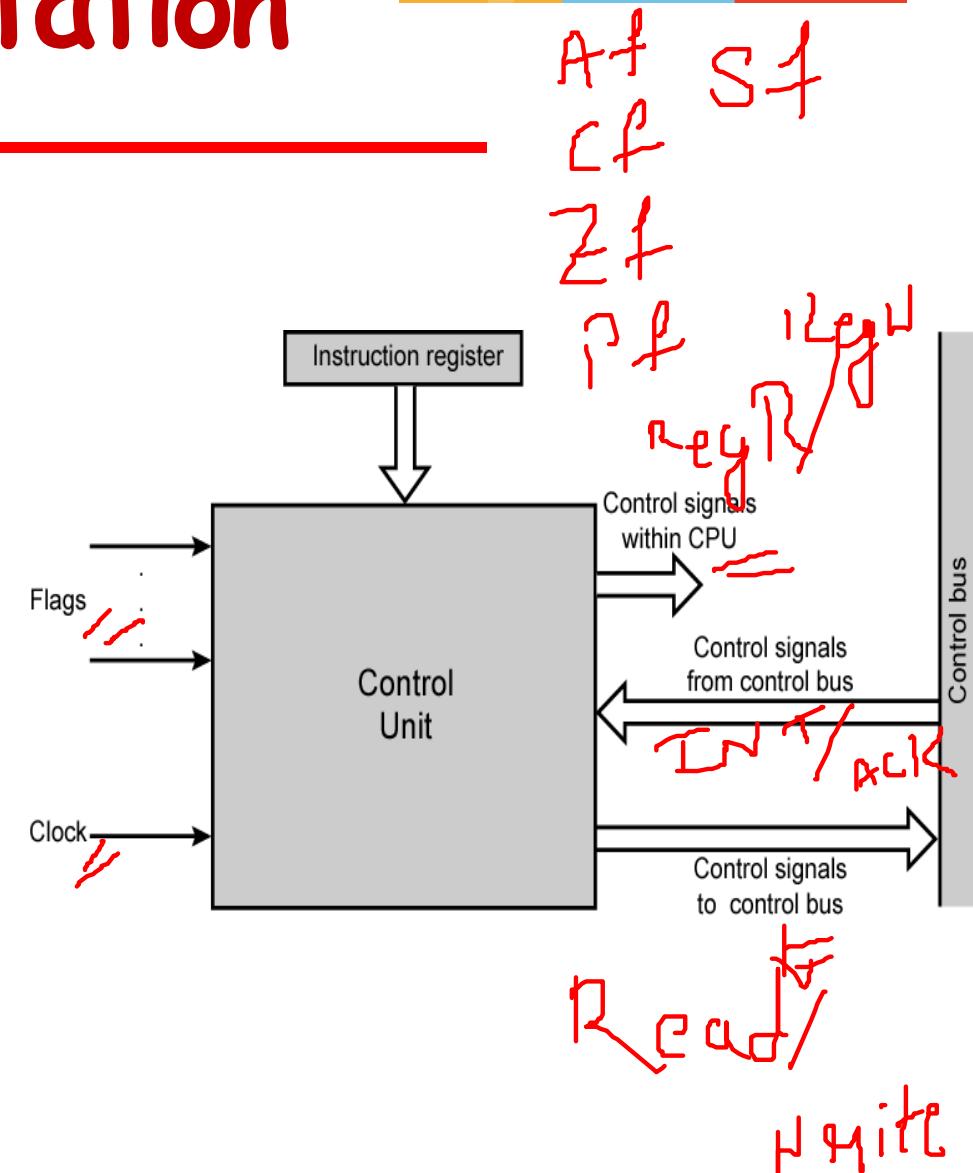
# Control Unit implementation

Hardwired control unit (RISC)

Microprogrammed control unit(CISC)

# Hardwired Implementation

- Control unit inputs
  - Flags and control bus
    - Each bit means something
  - Instruction register
    - Op-code causes different control signals for each different instruction
    - Unique logic for each op-code
  - Clock



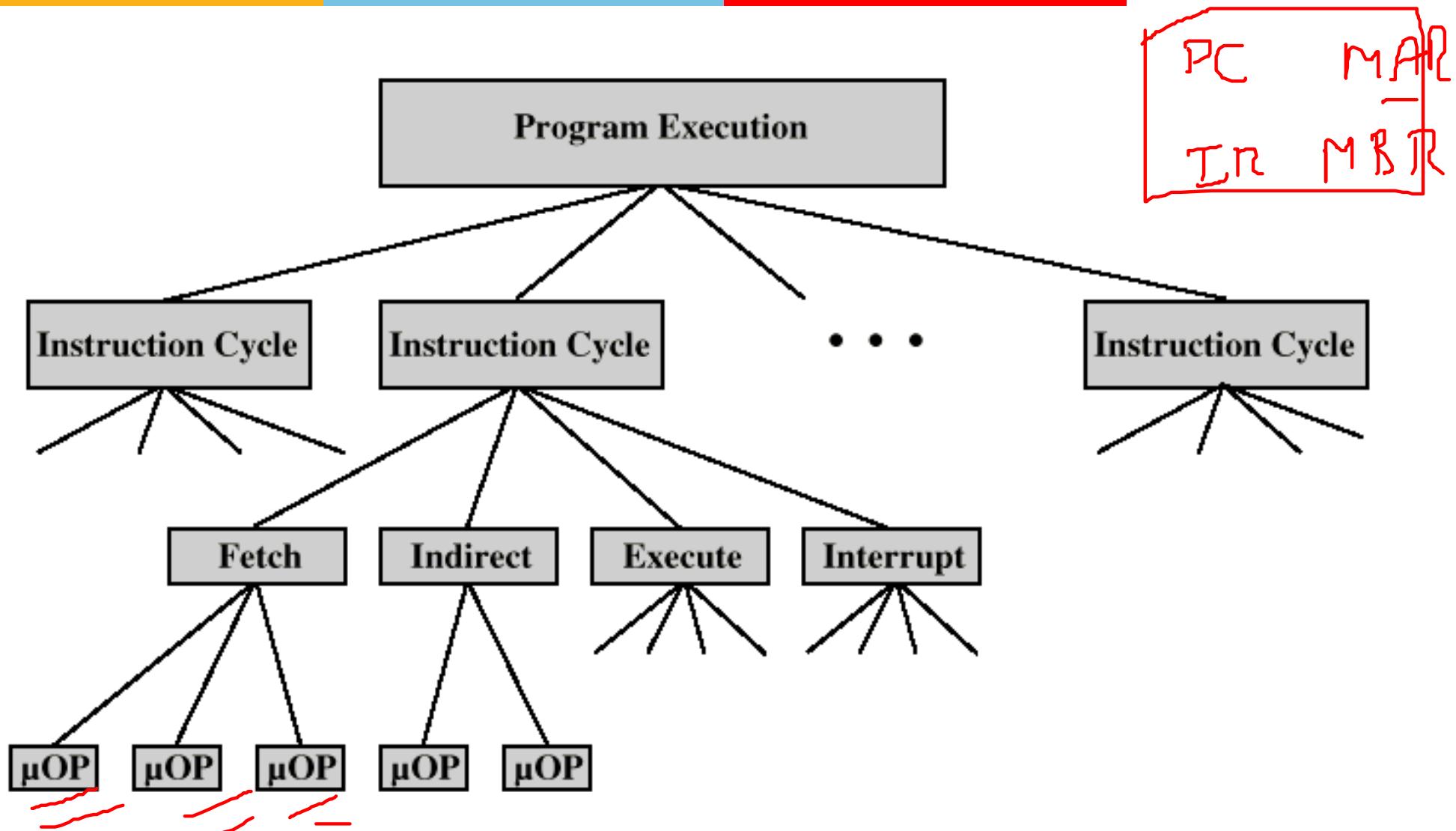
# Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

# Microprogrammed Control Unit

- A computer executes a program
- Fetch/execute cycle/interrupt cycle...

# Constituent Elements of Program Execution



# Example: ADD R1,X

ADD R1,X - add the contents of location X to R1, and stores the result in R1

Fetch Sequence :

t1: MAR  $\leftarrow$  (PC)

t2: MBR  $\leftarrow$  (memory)

PC  $\leftarrow$  (PC) +1

t3: IR  $\leftarrow$  (MBR)

~~A) DIRX~~

OR

t1: MAR  $\leftarrow$  (PC)

t2: MBR  $\leftarrow$  (memory)

t3: PC  $\leftarrow$  (PC) +1

IR  $\leftarrow$  (MBR)

# Example: ADD R1,X

Execute Cycle (ADD):

t4:  $\text{MAR} \leftarrow (\text{IR}_{\text{address}})$

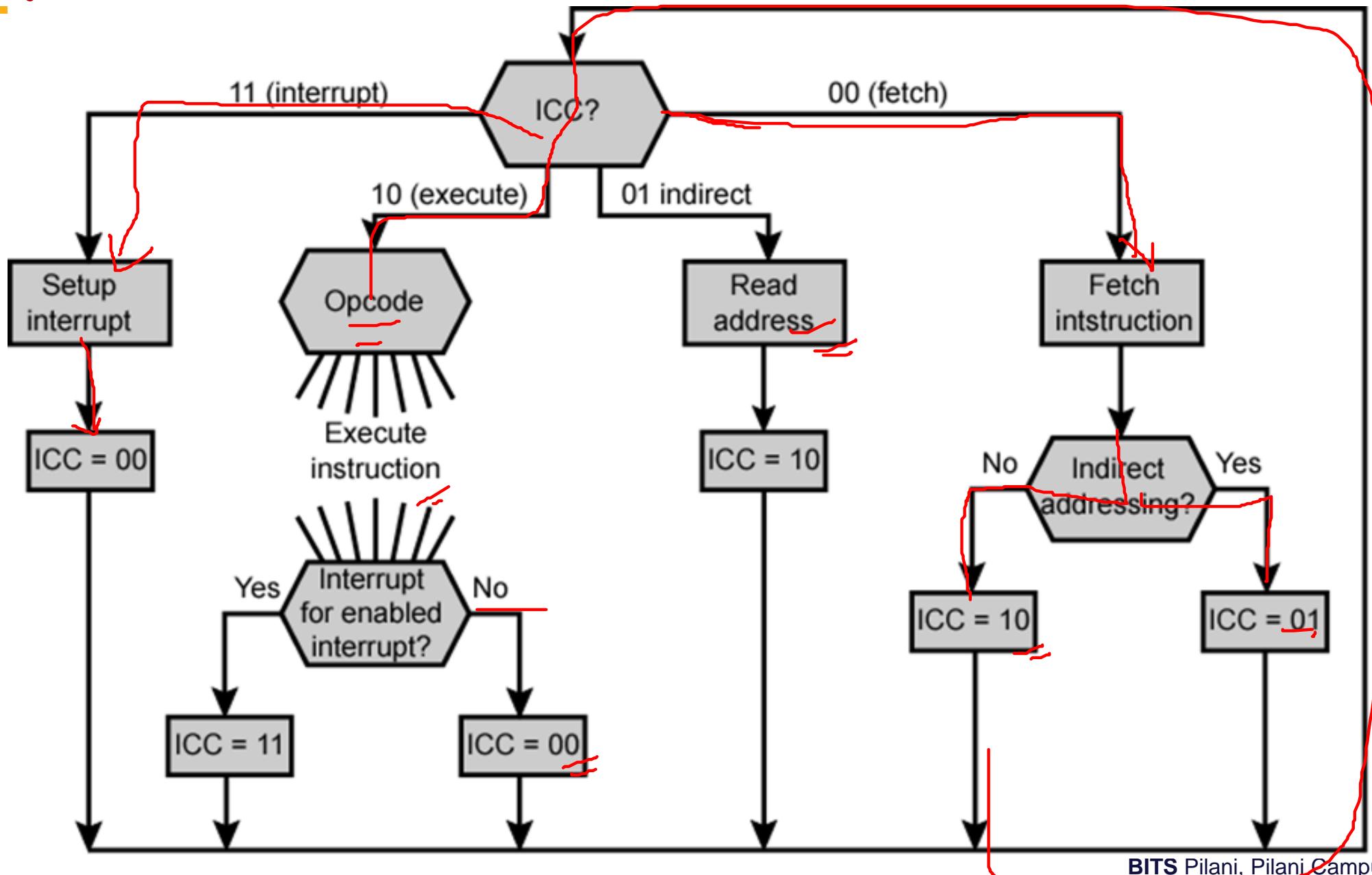
t5:  $\text{MBR} \leftarrow (\text{memory})$

t6:  $\underline{\underline{R1}} \leftarrow \underline{\underline{R1}} + (\cancel{\text{MBR}})$

# Instruction Cycle

- Different phases : Instruction fetch, indirect, execute and interrupt
- Each phase decomposed into sequence of elementary micro-operations
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

# Flowchart for Instruction Cycle



# Functions of Control Unit

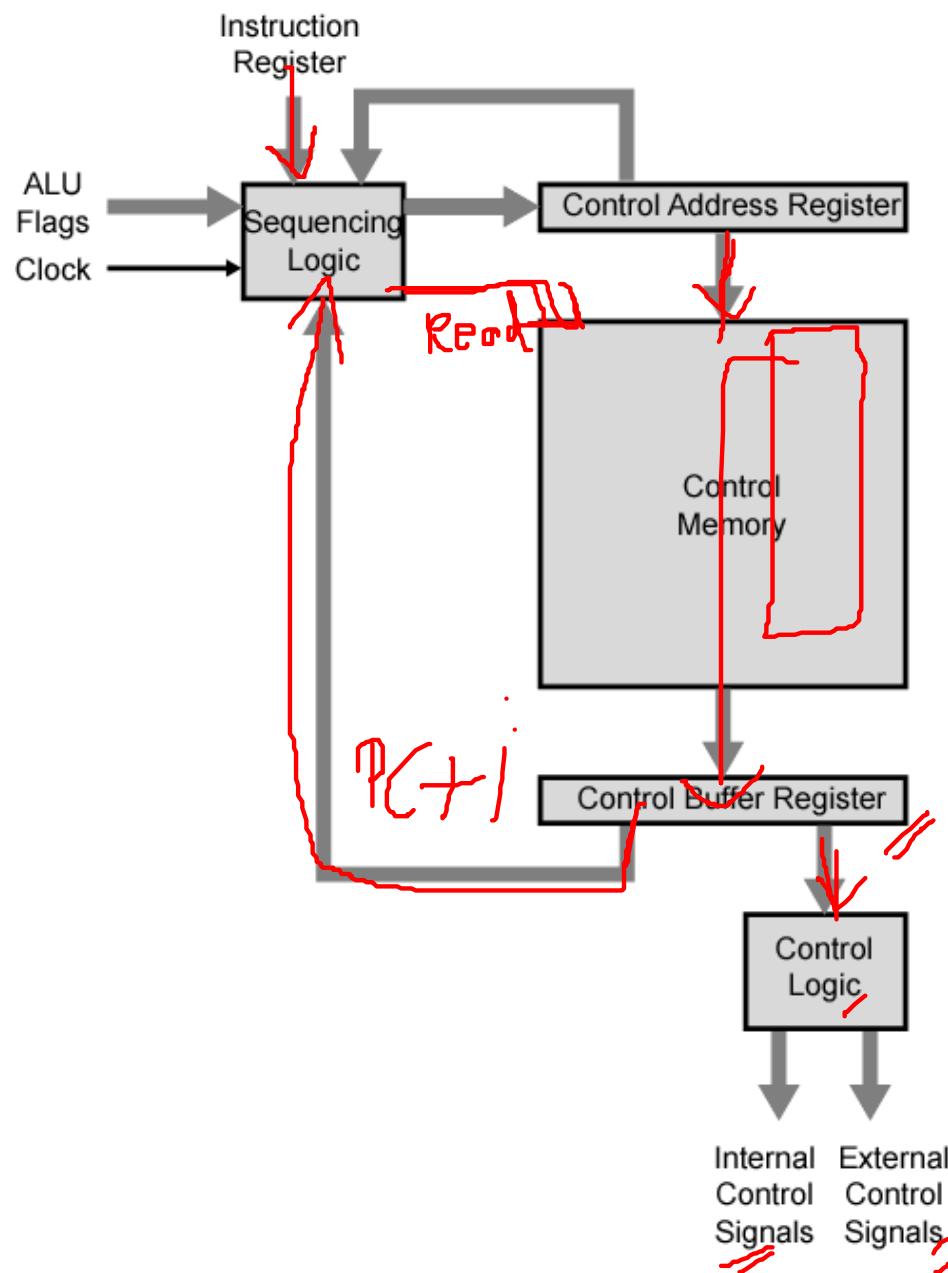
- The control unit performs two basic tasks:
  - Sequencing
    - Causing the CPU to step through a series of micro-operations
  - Execution
    - Causing the performance of each micro-op

Example: ADD R1, X

t1: MAR  $\leftarrow$  (PC)  
t2: MBR  $\leftarrow$  (memory)  
t3: PC  $\leftarrow$  (PC) + 1  
IR  $\leftarrow$  (MBR)

t4: MAR  $\leftarrow$  (IR<sub>address</sub>)  
t5: MBR  $\leftarrow$  (memory)  
t6: R1  $\leftarrow$  R1 + (MBR)

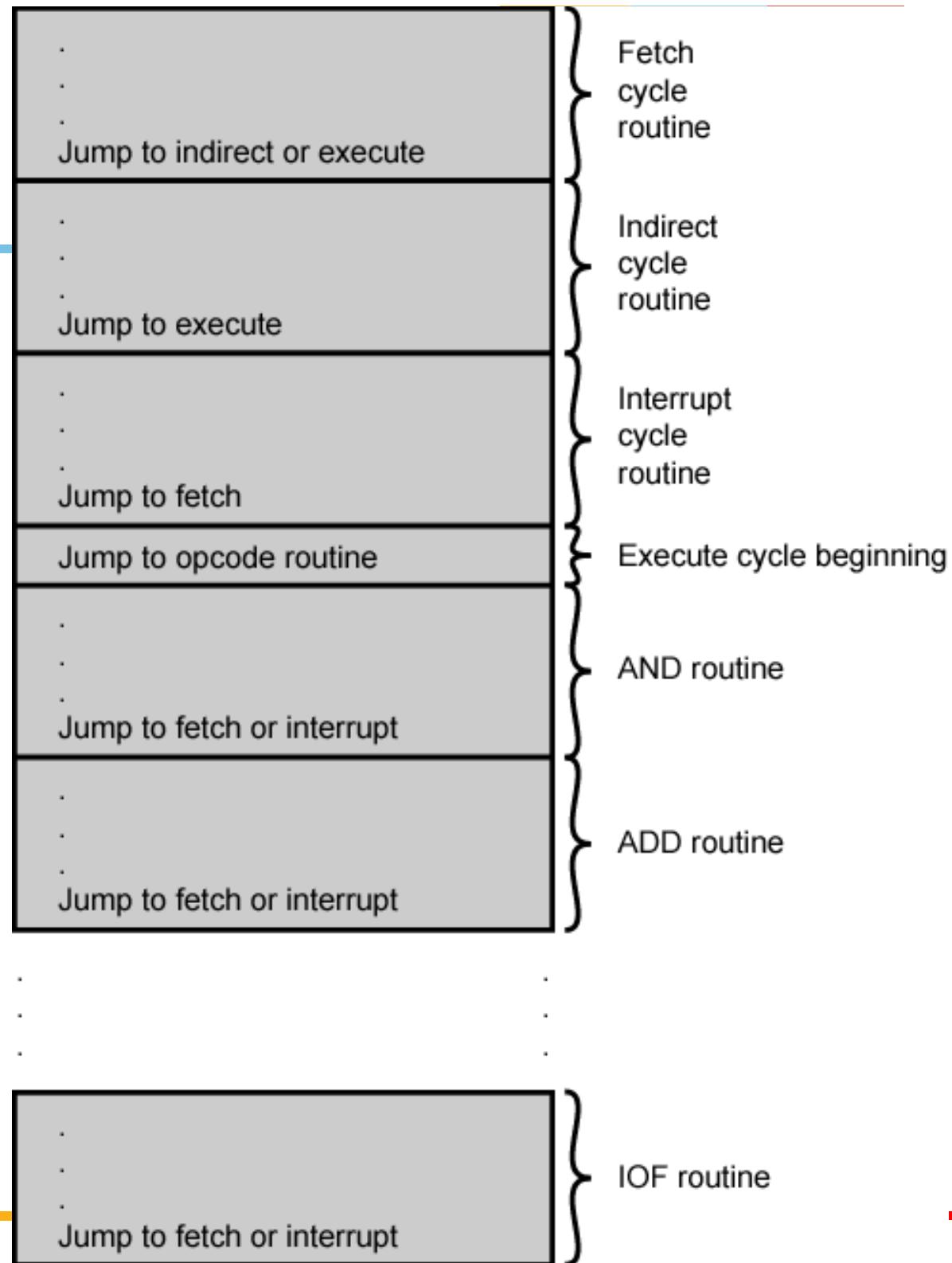
# Microprogrammed Control Unit



# Control Signals

- Inputs to the control unit
  - Clock
    - One micro-instruction (or set of parallel micro-instructions) per clock cycle
  - Instruction register
    - Op-code for current instruction
    - Determines which micro-instructions are performed
  - Flags
    - State of CPU
    - Results of previous operations
  - From control bus
    - Interrupts
    - Acknowledgements

# Organization of Control Memory





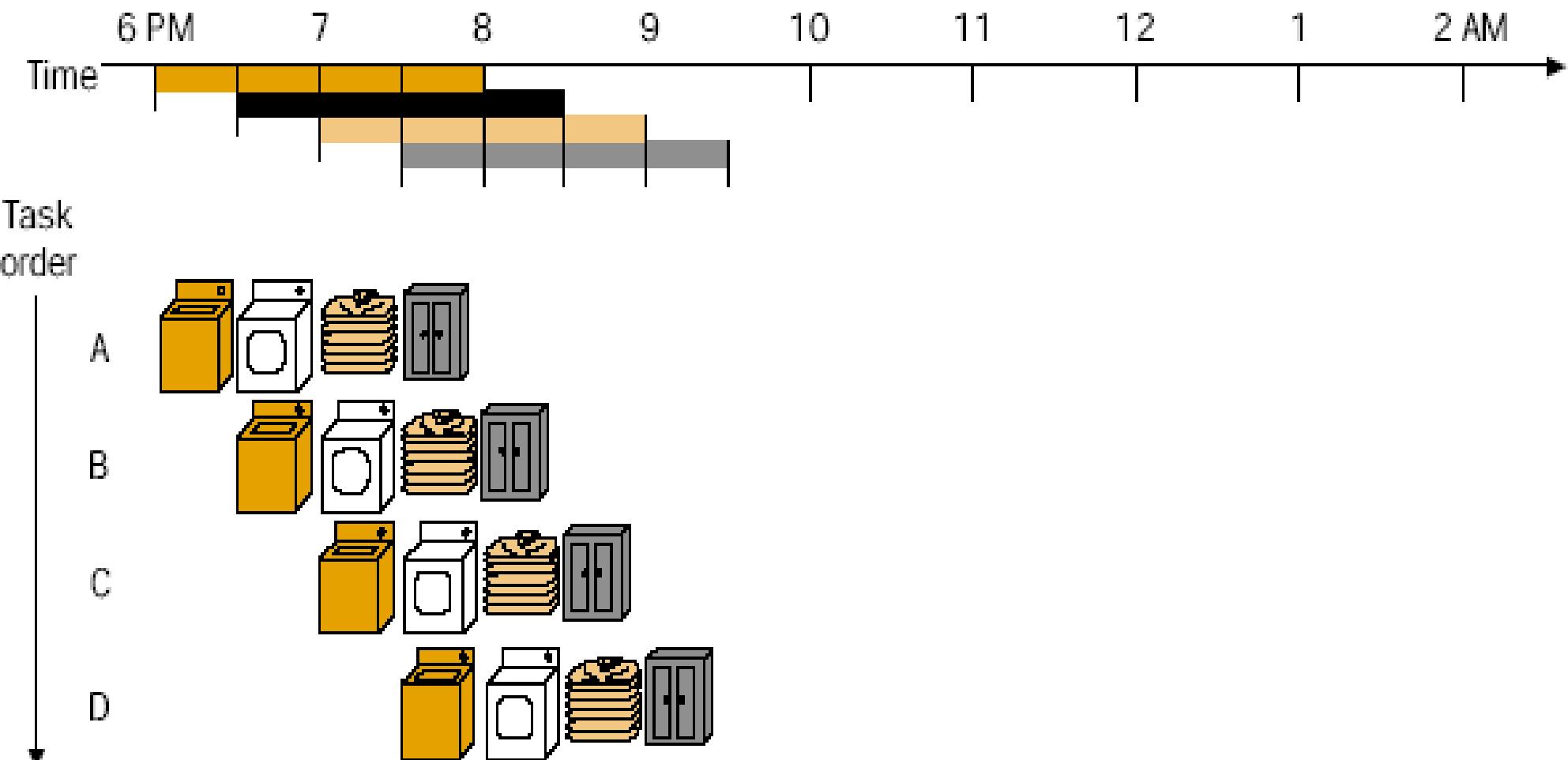
# Pipeline

**BITS Pilani**  
Pilani Campus

# Laundry System



# Laundry System.....



# Pipelining

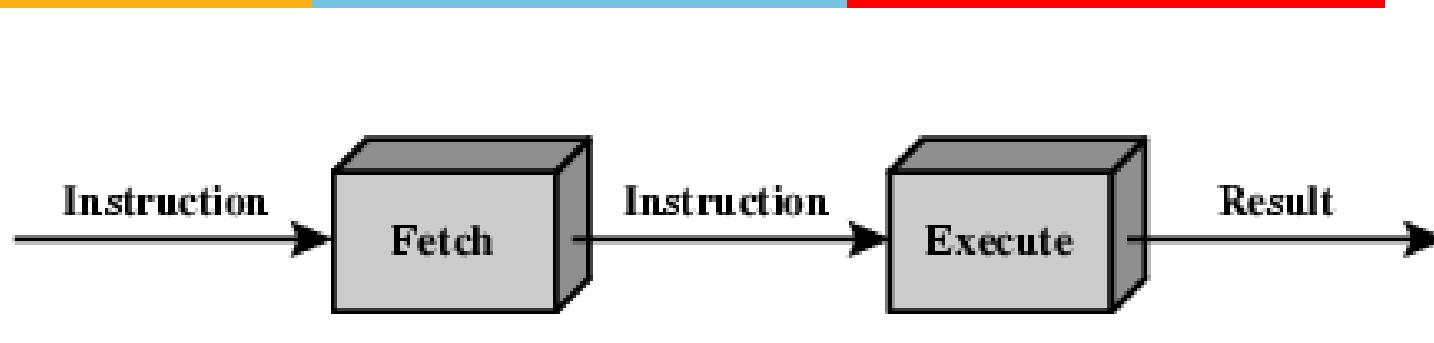
- An overlapped parallelism: overlapped execution of multiple operations
- Pipelining
  - Subdivide the input task into a sequence of subtasks
  - Specialized hardware stage for each task
  - Concurrent operation of all the stages

# Two segment instruction pipeline

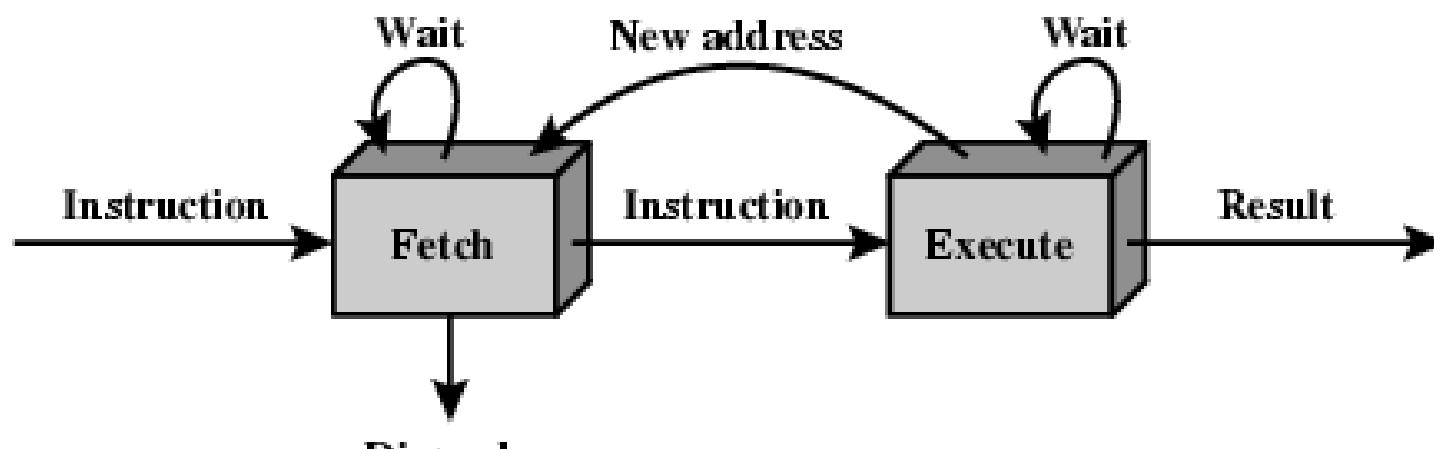
---

- Contains
  - Instruction Fetch (IF) ✓
  - Execute (EX) ✓
- Example: 8086 microprocessor ✓
- Instruction Fetch unit is implemented by means of first in first out buffer (Queue)

# Two Stage Pipeline



(a) Simplified view



(b) Expanded view

# Issues

1. The execution time will generally be longer than the fetch time
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown.



# Four Segment instruction pipeline

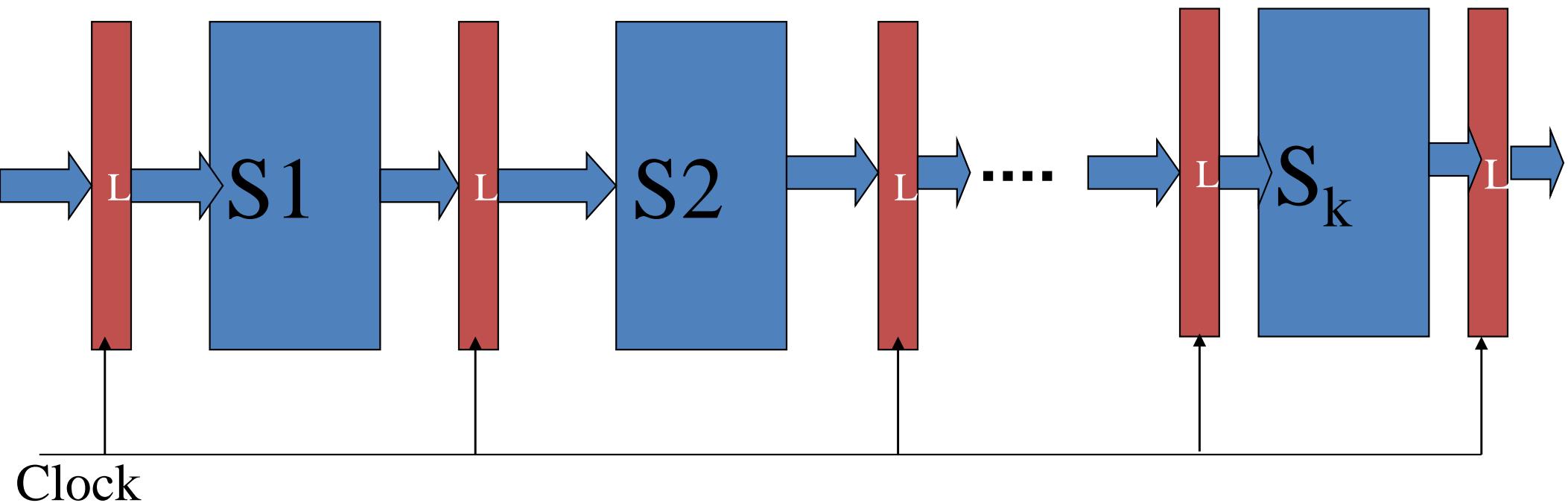
- Contains
  - FI : fetch instruction
  - DA : Decode instruction and calculate effective address
  - FO :Fetch operand
  - EX: Execute instruction



# Six stage pipeline

- Contains
  - FI : fetch instruction
  - DI : Decode instruction
  - CO : calculate effective address
  - FO :Fetch operand
  - EI : Execute instruction
  - WO : Write Operand

# Structure of a pipeline



# Classification

---

- Arithmetic pipelining
- Instruction pipelining
- Processor pipelining
- Unifunction and multifunction pipelining
- Static and Dynamic pipelining
- Scalar and Vector pipelining

# Arithmetic pipelining

- Arithmetic and logic units of a computer can be segmentized for pipeline operations
- Usually found in high speed computers
- Example:
  - Star 100 → 4 stage
  - TI-ASC → 8 stage
  - Cray-1 → 14 stage
  - Cyber 205 → 26 stages
  - Intel Cooper Lake (3rd Gen Intel Xeon) = 14 stages
- Floating point adder pipeline

$$X = A * 2^a$$

$$Y = B * 2^b$$

# Instruction pipelining

---

- The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode.....of subsequent instructions
- Sequence of steps followed in most general purpose computer to process instruction
  1. IF: Fetch the instruction from memory
  2. ID: Decode the instruction
  3. CO: Calculate the effective address
  4. FO: Fetch the operands from memory
  5. EI: Execute the instruction
  6. WO: Store the result in the proper place

# Timing Diagram for Instruction Pipeline Operation

Time →

$I_p + (C - 1)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

# Important Points to be noted

- Do all the instructions need all the stages?
- At t=6, WO, FO and FI accesses the memory. Is there any issue?
- Is there any implication on having different time duration for different stages?
- Any issues with conditional branch?
- Dependency of CO stage on register used in previous stage

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

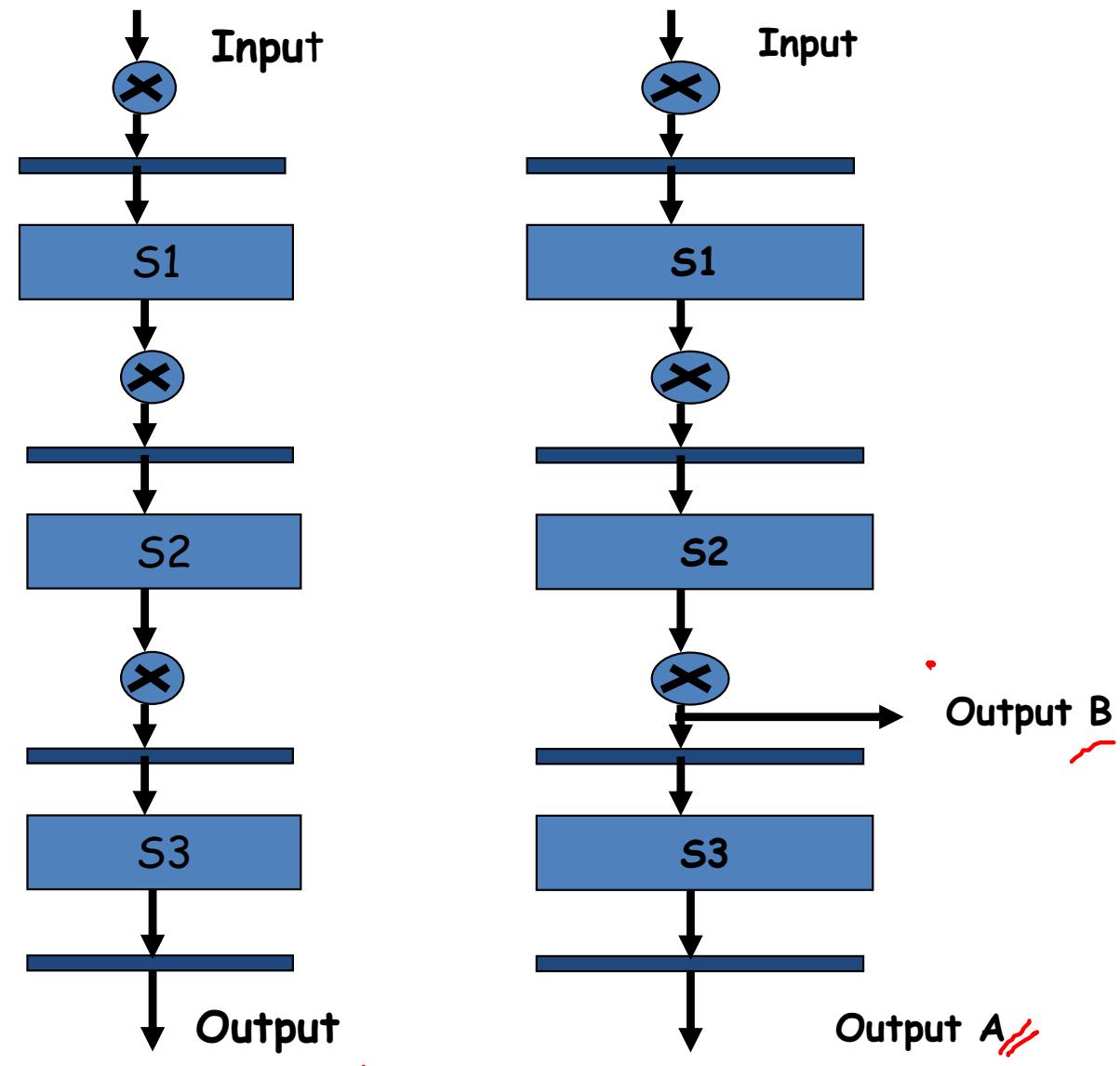
Time →

Branch Penalty ↪

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

# Unifunction and multifunction pipelining

- Unifunction
  - Pipeline with a fixed and dedicated function
  - Ex: Floating point adder
- Multifunction
  - Pipeline may perform different functions

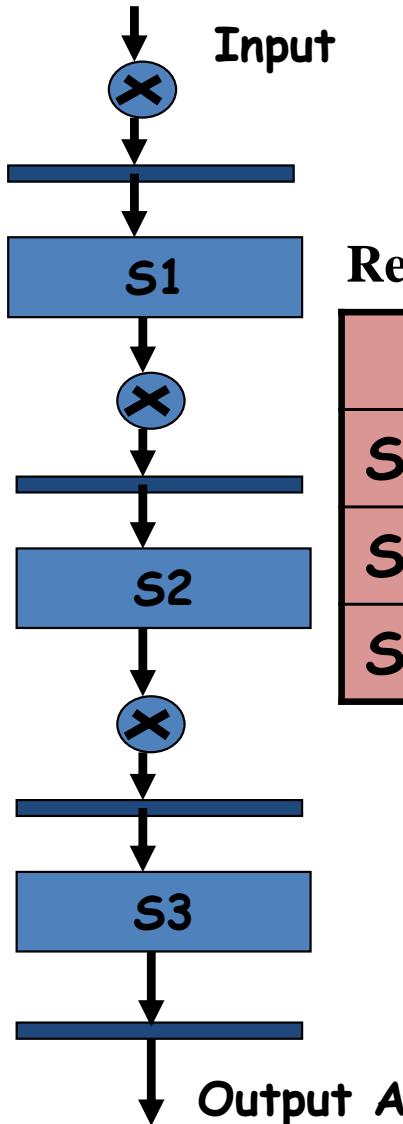




# Reservation Table

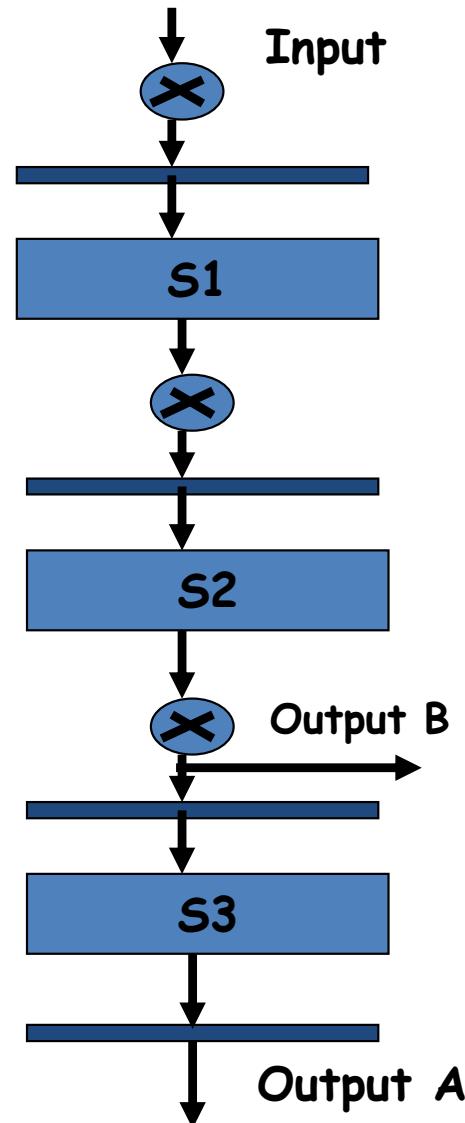
- Is a two dimensional chart
- Used to show how successive pipeline stages are utilized or reserved

# Uni-function Vs Multifunction



	T0	T1	T2
S1	A		
S2		A	
S3			A

**Reservation Table A**



**Reservation Table A**

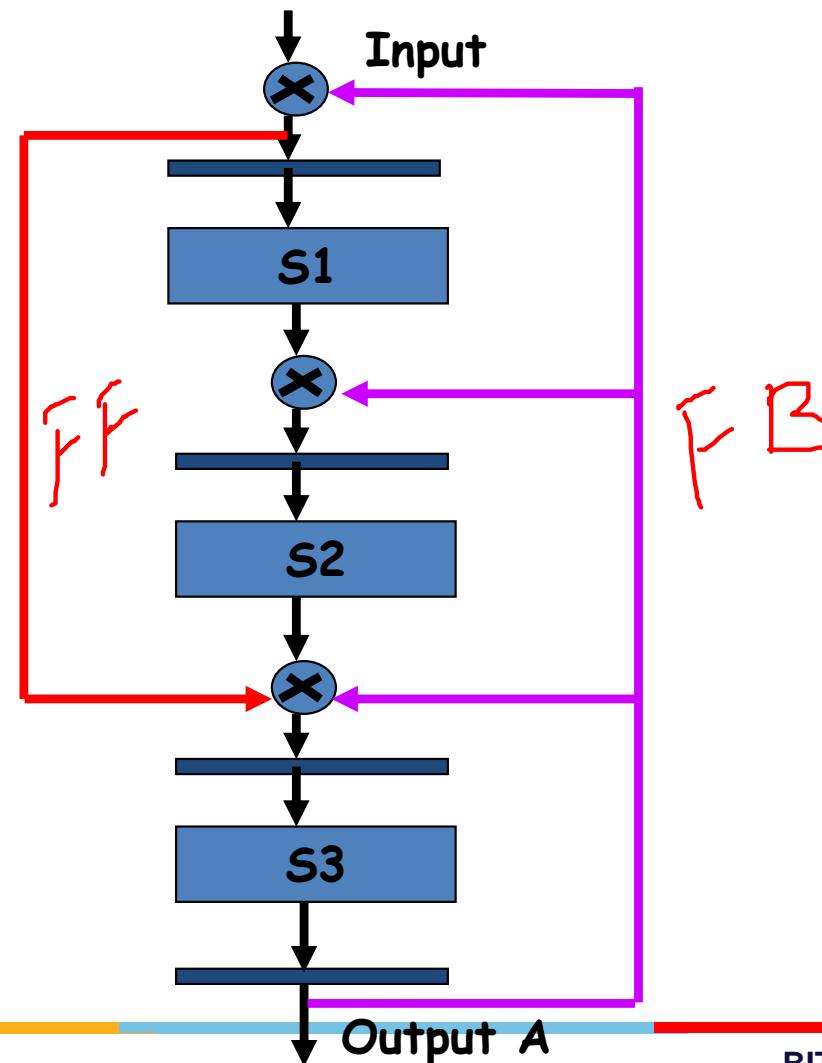
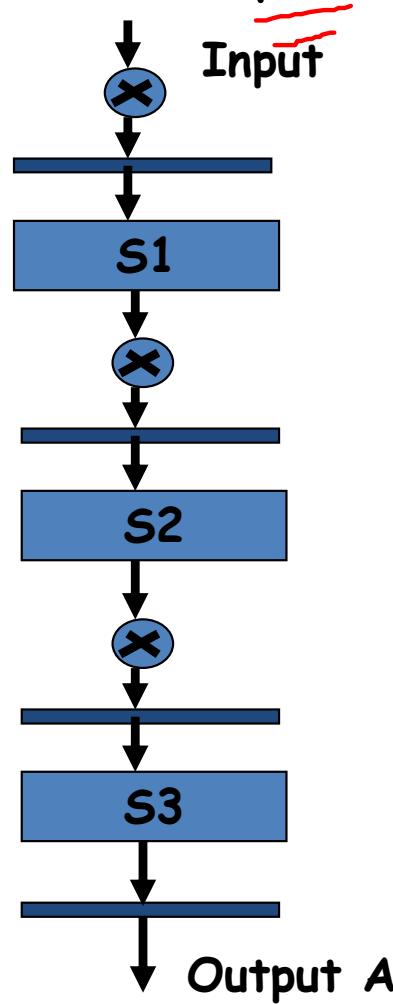
	T0	T1	T2
S1	A		
S2		A	
S3			A

**Reservation Table B**

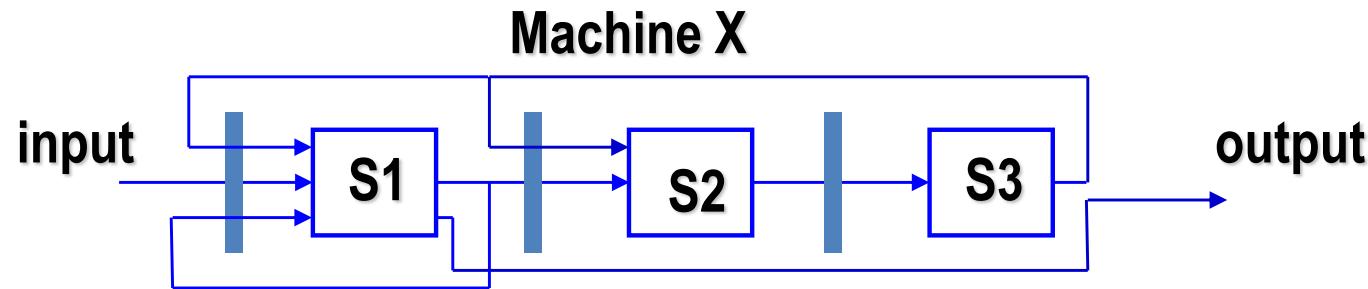
	T0	T1
S1	B	
S2		B

# Linear and Nonlinear Pipelines

- Linear Pipeline: Without feed forward and feed back connection
- Nonlinear Pipeline with feed forward and/or feed back connection



# Reservation Table

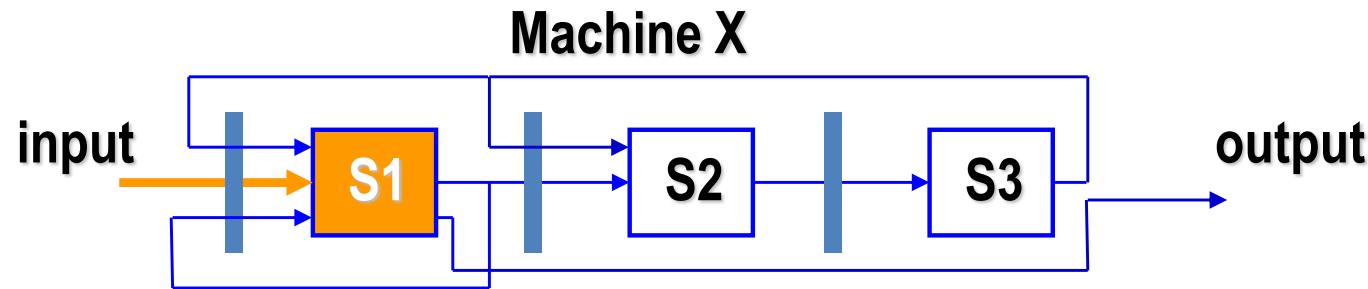


## Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table



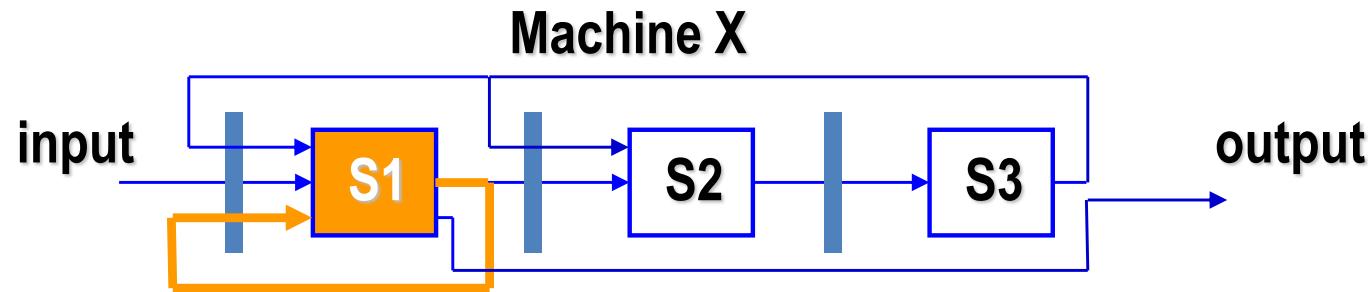
## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Stage →

# Reservation Table

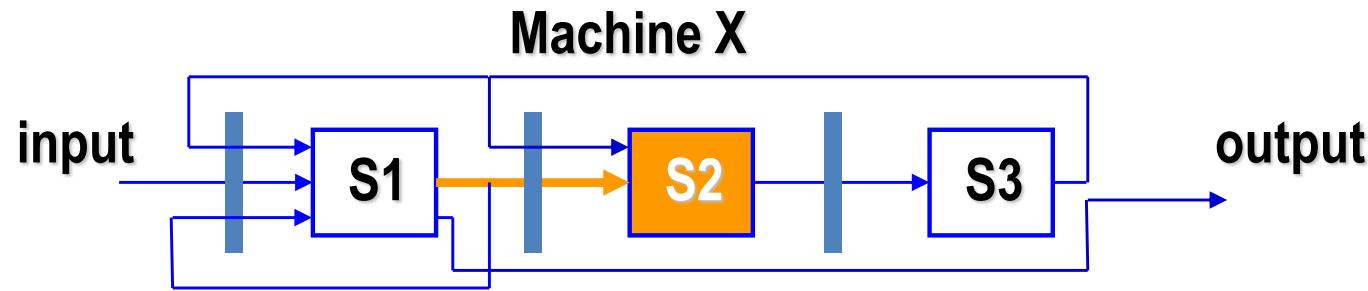


## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table



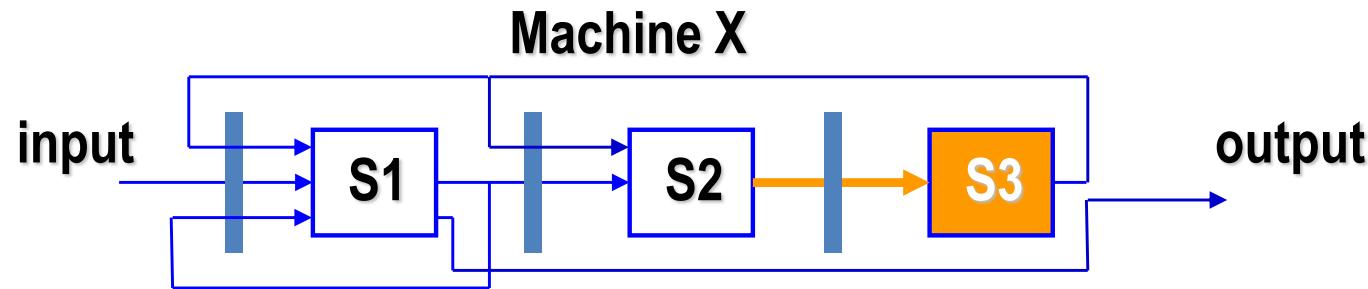
## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

Stage →

# Reservation Table

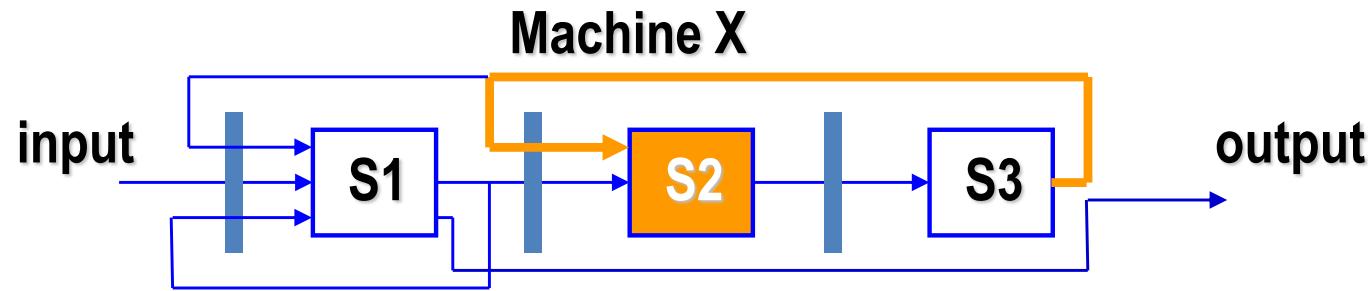


## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table

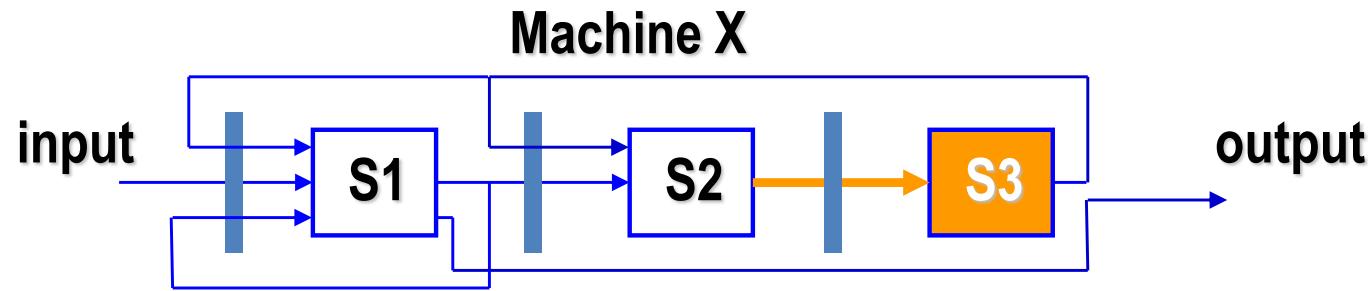


## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table

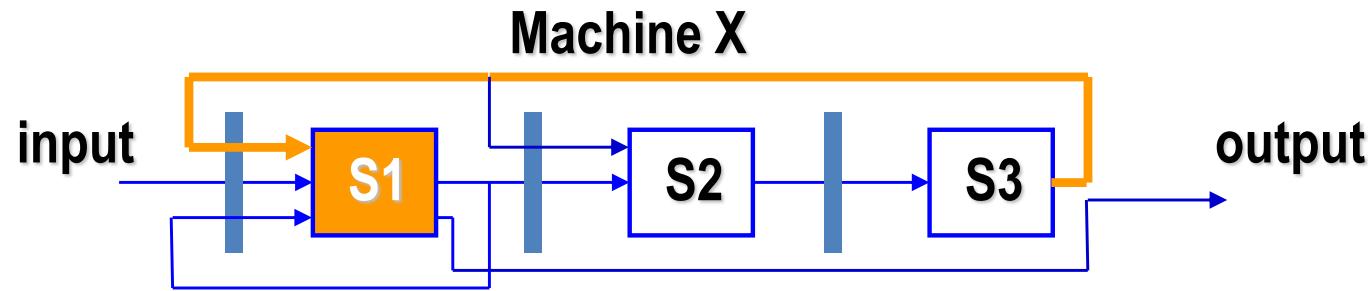


## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table

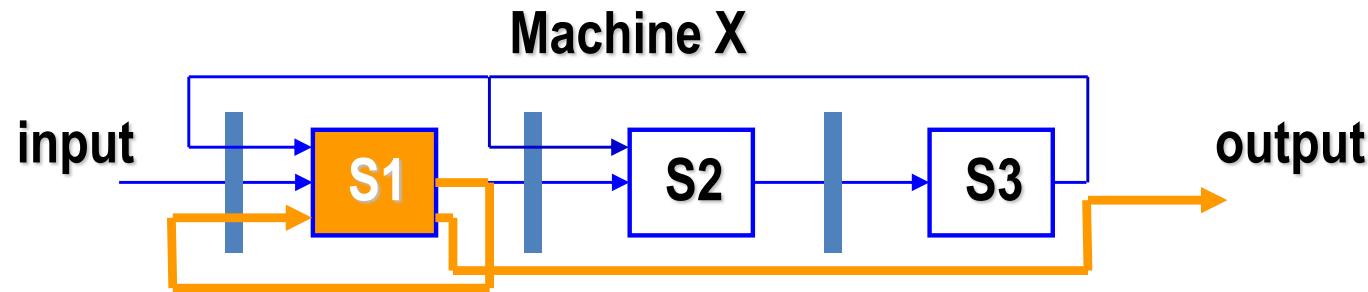


## Reservation Table

Time →

Stage →	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table

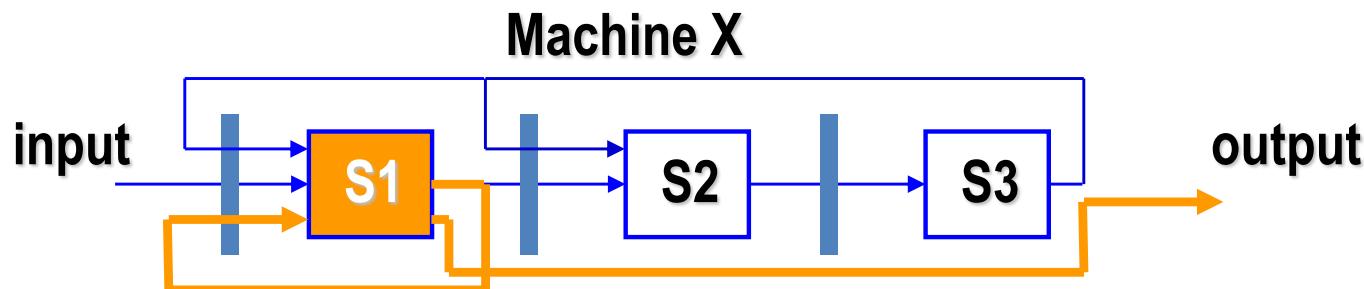


## Reservation Table

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Reservation Table



**Reservation Table**

Time →

	0	1	2	3	4	5	6	7
S1	X	X					X	X
S2			X		X			
S3				X		X		

# Static and Dynamic pipelining

- Dynamic pipeline allows more frequent changes in its configuration
- Require more elaborate sequencing and control mechanisms

=

# Scalar and Vector pipelining

---

- Based on the operand types or instruction type
- Scalar pipeline processes scalar operands
- Vector pipeline operate on vector data and instructions.



# COMPUTER ORGANIZATION AND SOFTWARE SYSTEMS SESSION 8



**BITS** Pilani  
Pilani Campus

Dr. Lucy J. Gudino  
WILP & Department of CS & IS



# Pipeline Continued

**BITS Pilani**  
Pilani Campus

# Important Terms

Clock period:  $\tau$

$\tau_i$ : time delay of  $S_i$  stage

$\tau_L$ : time delay of latch



$$\tau = \max\{\tau_i\} + \tau_L,$$

Pipeline processor frequency  $f = 1/\tau$

$$S_1 (F0) \Rightarrow 5 \text{ ns}$$

$$S_2 (I) \Rightarrow 3 \text{ ns}$$

$$S_3 (F0) \Rightarrow 5 \text{ ns}$$

$$S_4 (Ex) \Rightarrow 10 \text{ ns}$$

$$S_5 (H0) \Rightarrow 3 \text{ ns}$$

$$\max \{5, 3, 10, 3\} + 1 \text{ ns}$$

$$10 + 1 = 11 \text{ ns}$$

# Important Terms

Time taken to complete n tasks by k stage pipeline is

$$T_k = [k + (n-1)]\tau$$

Time taken by the nonpipelined processor

?

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

# Important Terms

Time taken to complete  $n$  tasks by  $k$  stage pipeline is

$$T_p = [k + (n-1)]\tau$$

Time taken by the nonpipelined processor

$$T_1 = k * n * \tau$$

=

$$T_n\tau$$

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

# Important Terms

Speedup: speedup of a k-stage linear-pipeline over an equivalent non pipelined processor

$$S_k = \frac{T_1}{T_k}$$

$$= \frac{n^*k^*\cancel{\tau}}{[k+(n-1)]\cancel{\tau}}$$

$$= \frac{n^*k}{[k+(n-1)]//}$$

# Important Terms.....

The maximum speedup is  $S_k \rightarrow k$  when  $n \rightarrow \text{INF}$

Maximum speedup is very difficult to achieve because  
of data dependencies between successive tasks,  
program branches, interrupts etc.

# Important Terms

Efficiency: the ratio of actual speedup to ideal speedup k

$$\eta = \frac{\frac{n.k}{k.[k + (n - 1)]}}{n}$$
$$= \frac{[k + (n - 1)]}{//}$$

# Contd...

- Maximum efficiency  
 $\boxed{n} \rightarrow 1 \text{ as } n \rightarrow \infty //$
- Implies that the larger the number of tasks flowing through the pipeline, the better is its efficiency
- In steady state of a pipeline, we have  $\underline{n} \gg \underline{k}$ , then efficiency should approach 1
- However, this ideal case may not hold all the time because of program branches and interrupts and data dependencies

# Important Terms

Throughput : The number of tasks that can be completed by a pipeline per unit time

$$H_k = \frac{n}{[k + (n-1)]\tau} = \frac{nf}{[k + (n-1)]} = nf$$

# Problem 1

$$(6 + [8 - 1]) \times 1 = 13$$

- 4) Draw a space-time diagram for a six segment pipeline showing the time it takes to process eight tasks
- 5) Determine the number of clock cycles that it takes to process 200 tasks in a six segment pipeline

	1	2	3	4	5	6	7	8	9	10	11	12	13
T1	S1	S2	S3	S4	S5	S6							
T2		S1	S2	S3	S4	S5	S6						
T3			S1	S2	S3	S4	S5	S6					
T4				S1	S2	S3	S4	S5	S6				
T5					S1	S2	S3	S4	S5	S6			
T6						S1	S2	S3	S4	S5	S6		
T7							S1	S2	S3	S4	S5	S6	
T8								S1	S2	S3	S4	S5	S6

# Problem 1

---

Draw a space-time diagram for a six segment pipeline showing the time it takes to process eight tasks

Determine the number of clock cycles that it takes to process 200 tasks in a six segment pipeline

# Problem 2

Assume each task is subdivided in to 6 subtasks and clock cycle is 10 microseconds.  $\rightarrow T$

- Determine the number of clock cycles that is taken to process 50 tasks in six stage pipeline  $(6+4 \times 10) \times 50 = 550$  us
- Determine the number of clock cycles that is taken to process 50 tasks in non-pipeline processor  $50 \times 6 \times 10 \Rightarrow 3000$  us
- Compute speed up, efficiency and throughput

$$\text{Speed up} \rightarrow \frac{3000}{550} \approx 5.45$$

$$\text{eff} = 0.91 \quad \text{tho}z 0.09\checkmark$$

# Pipeline Hazards / Conflicts

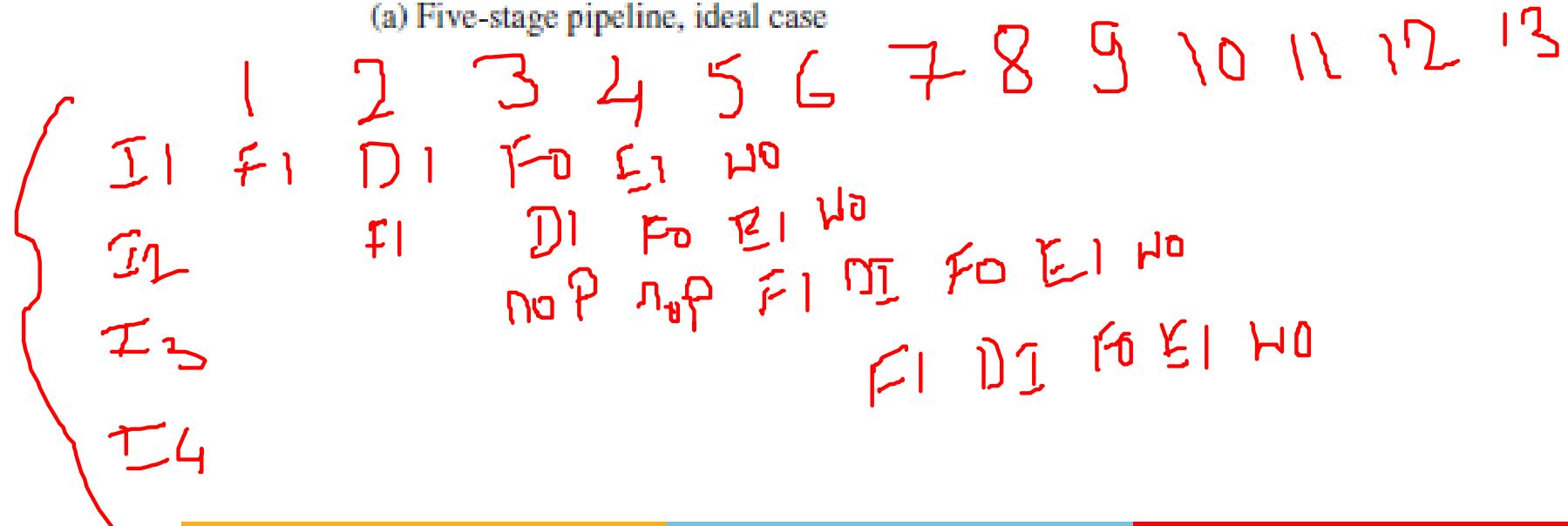
- **Resource Hazard**: access to same resource by two segments at the same time
- **Data Hazard** : an instruction depends on the result of a previous instruction, but this result is not yet available
- **Control Hazard**: arise from branch and other instructions that change the value of PC

# Resource Hazard

Clock cycle

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			FI	DI	FO	EI	WO		
I4				FI	DI	FO	EI	WO	

(a) Five-stage pipeline, ideal case



# Problem

Consider the following code. Assume that initial contents of all the registers is zero.

```
START:    MOV R3, #1
          MOV R1, #2
          MOV R2, #3
          ADD R3, R1, R2
          SUB R4, R3, R2
          HLT
```

- Write timing of instruction pipeline with FIVE stages
- What is the content of various registers after the execution of program?

FI DI FO CO EI WO

```

MOV R3, #1
MOV R1, #2
MOV R2, #3
ADD R3, R1, R2
SUB R4, R3, R2
HLT
    
```

$$\begin{aligned} R_1 &= 2 \\ R_2 &= 3 \end{aligned}$$

$$\begin{aligned} R_3 &= \cancel{5} \\ R_4 &= 3 \end{aligned}$$

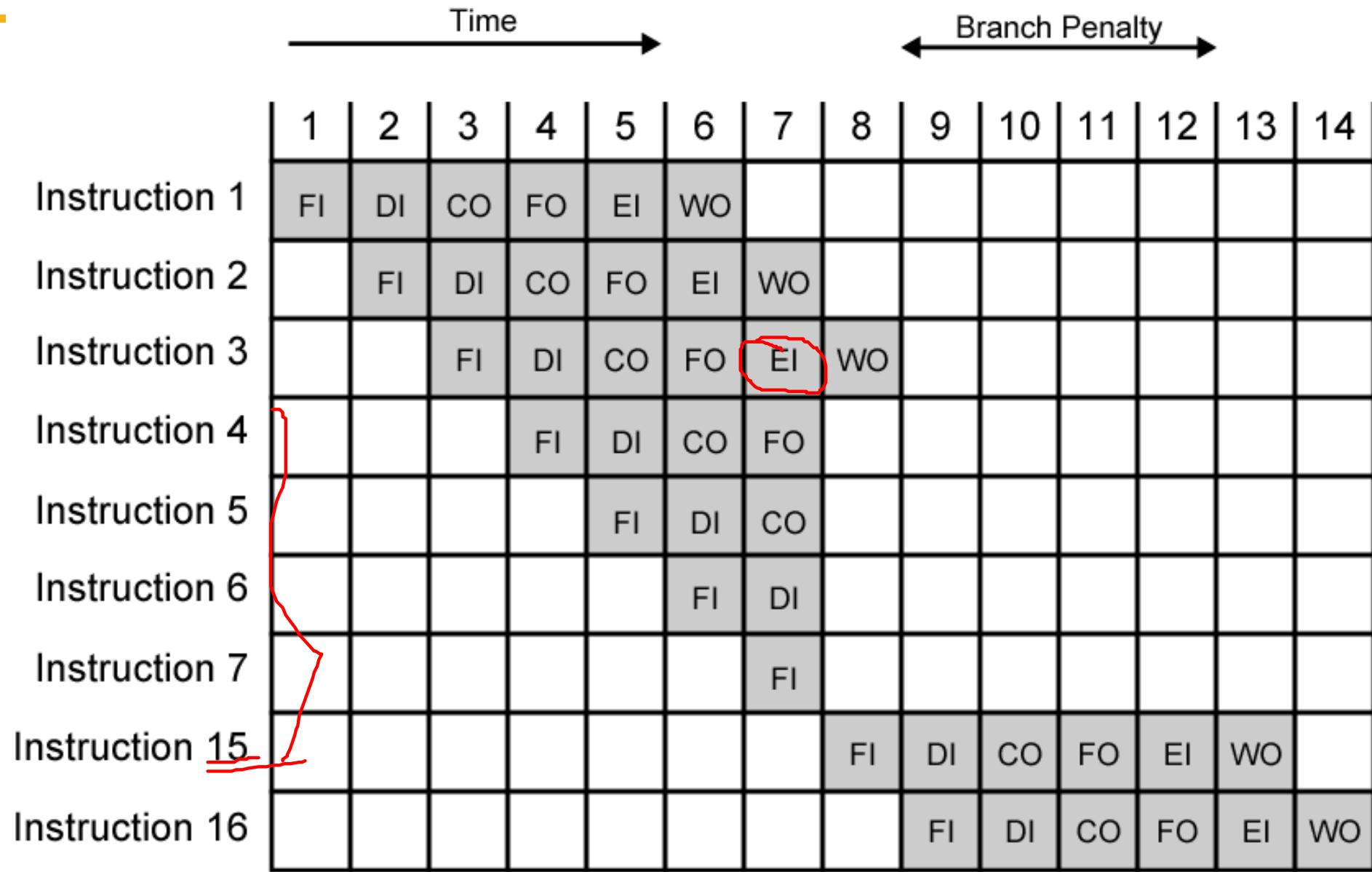
time	0	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO					
I2		FI	DI	FO	EI	WO				
I3			FI	DI	FO	EI	WO			$0 + 0 = 0$
I4				FI	DI	FO	EI	WO		$1 - 0 = 1$
I5					FI	DI	FO	EI	WO	
I6						FI	DI	FO	EI	WO

# Example

```
START: ADD R3, R1, R2  
       SUB R4, R1, R2  
       JNZ NEXT,  
       MUL R5, R1, R2  
       ADD R6, R3, R4  
  
NEXT:  ST R3, LOCN1  
       HLT
```

**Write timing of instruction pipeline**

# The Effect of a Conditional Branch on Instruction Pipeline Operation



# Dealing with Branches

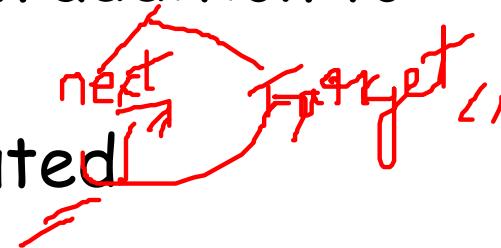
- ✓ Multiple Streams
- ✓ Prefetch Branch Target
- ✓ Loop buffer
- ✓ Branch prediction
- ✓ Delayed branching

# Multiple Streams

- Have two pipelines
- Pre-fetch each branch into a separate pipeline
- Use appropriate pipeline
- Used by IBM 370/168 and the IBM 3033
- Issues :
  - Leads to bus & register contention
  - Multiple branches lead to further pipelines being needed

# Prefetch Branch Target

- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91



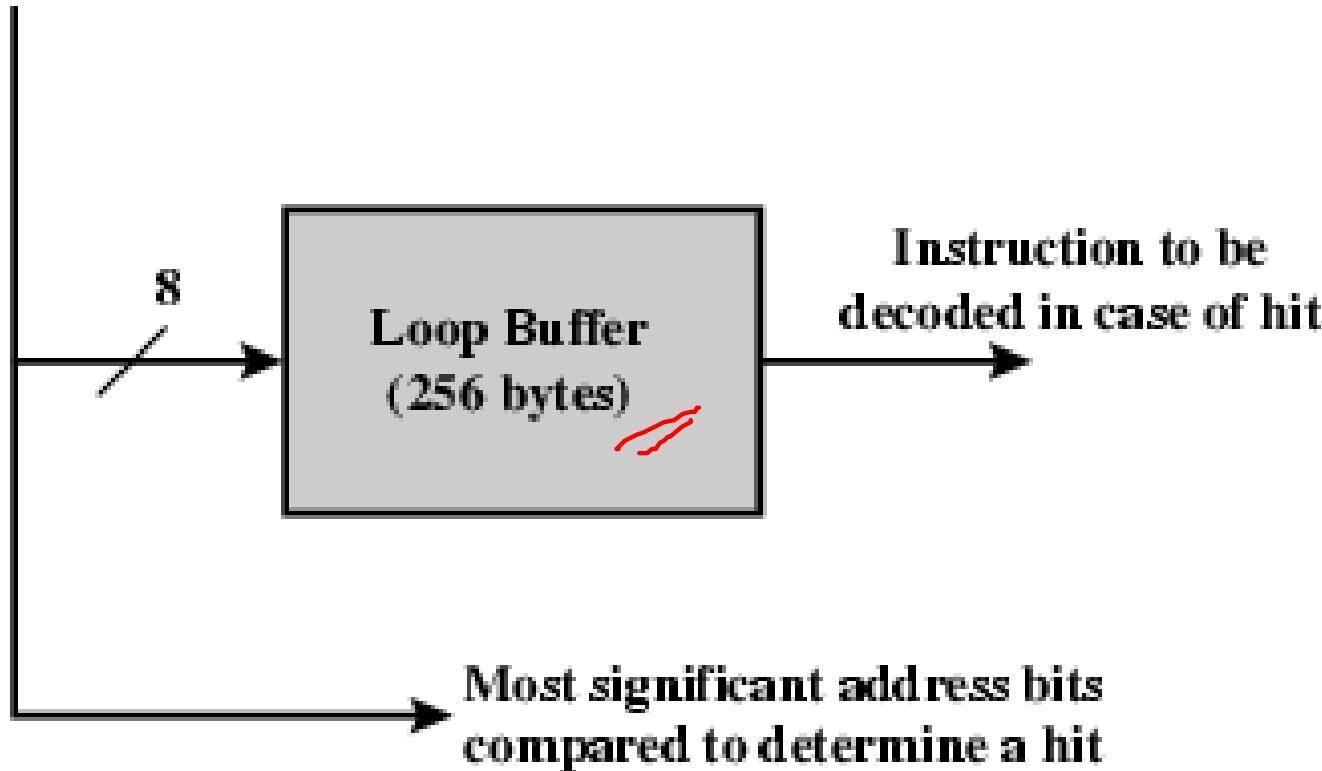
# Loop Buffer

---

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- Working principle is similar to cache
- Used by CRAY-1

# Loop Buffer Diagram

Branch address



# Branch Prediction (1)

- Predict never taken
  - Predict always taken
  - Predict by opcode
  - Taken/not taken switch
  - Branch history table
- } static  
} dynamic

# Branch Prediction (2)

- Predict never taken
  - Assume that jump will not happen →
  - Always fetch next instruction
- Predict always taken
  - Assume that jump will happen
  - Always fetch target instruction
- Predict by Opcode
  - Some instructions are more likely to result in a jump than others
  - Can get up to 75% success

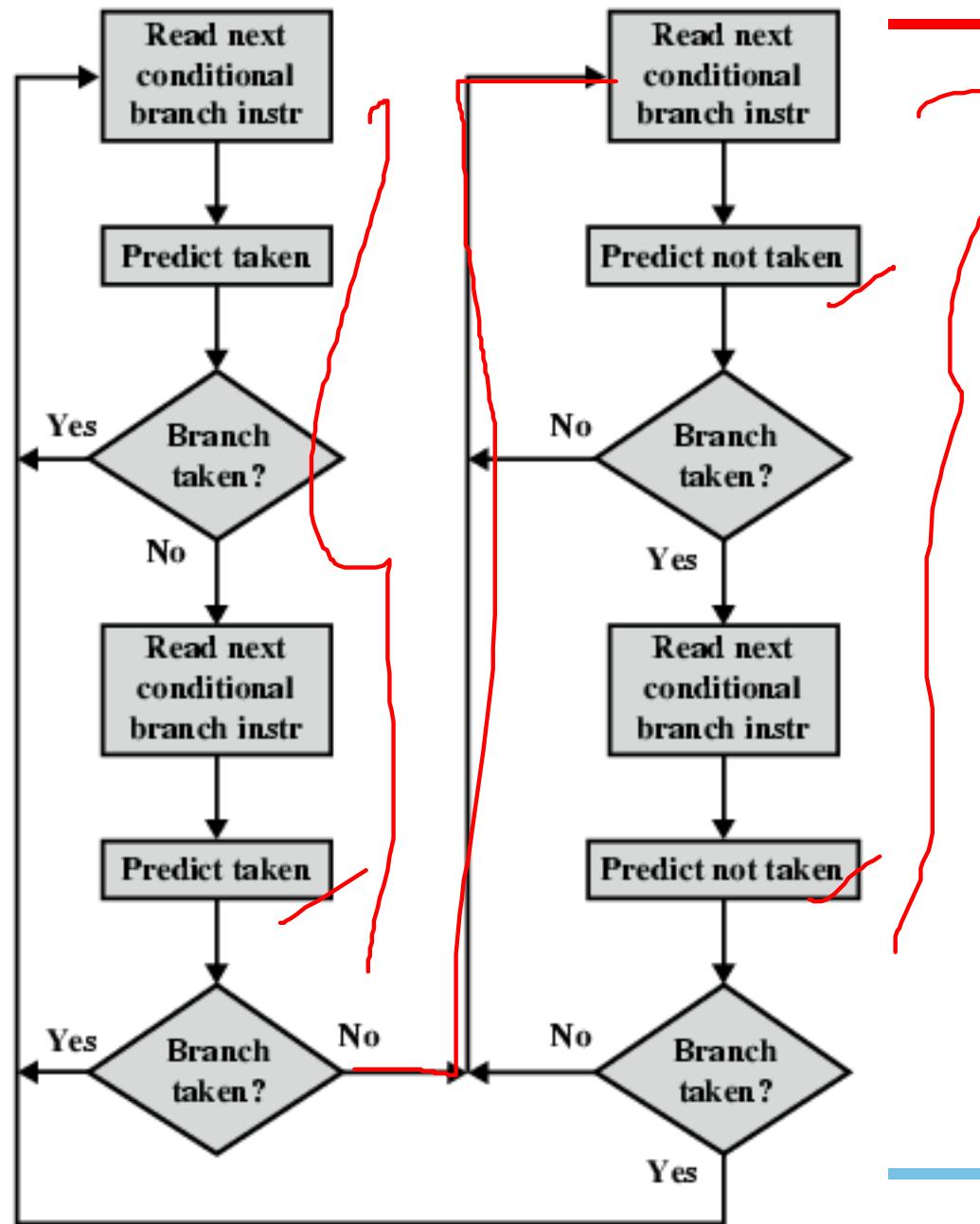


# Branch Prediction (3)

- Taken/Not taken switch
  - Based on previous history
  - Good for loops

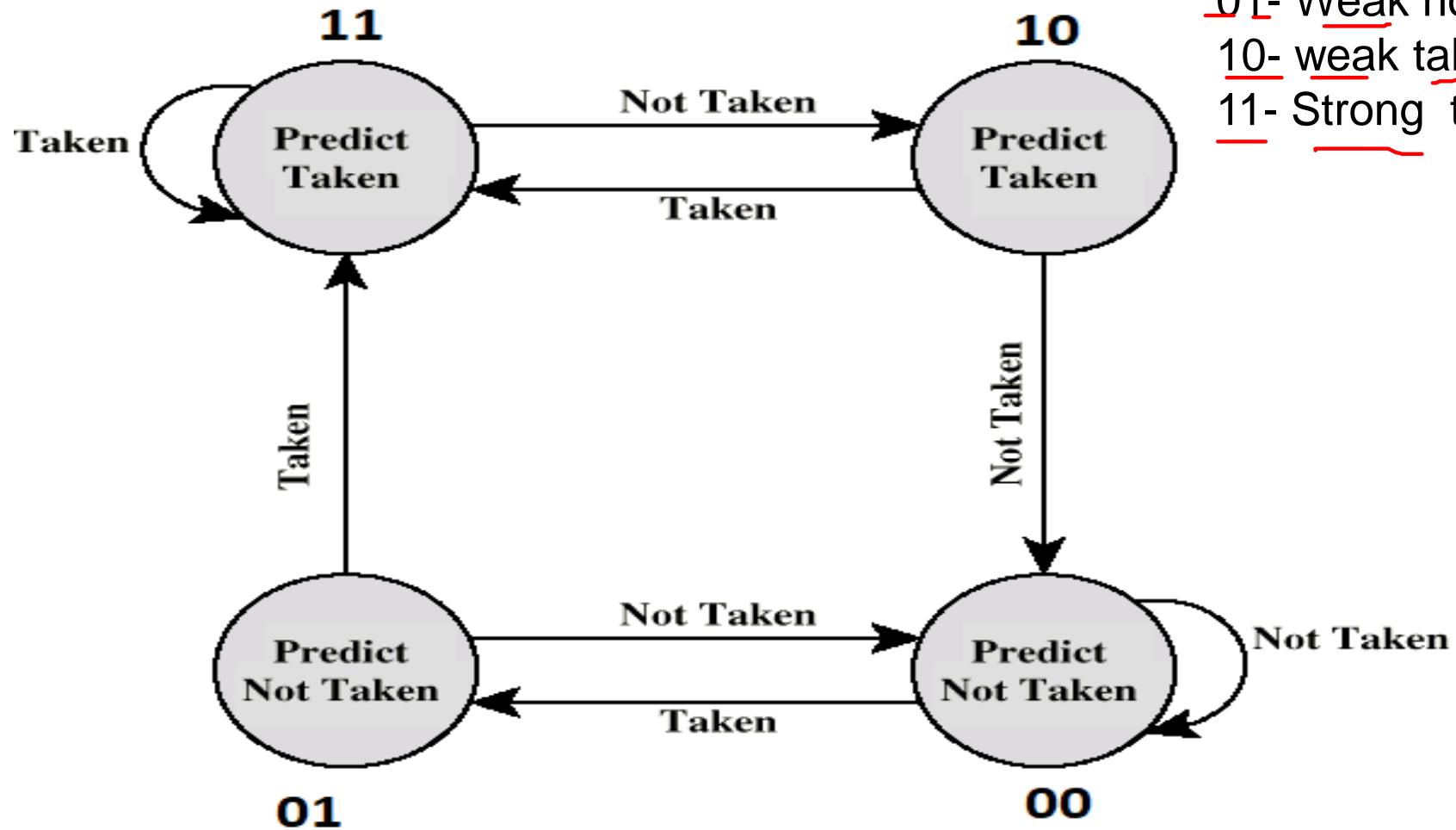
//

# Branch Prediction Flowchart



# Branch Prediction State Diagram

First bit is prediction bit and second bit is conviction bit



00 – strong not taken  
01 - Weak not taken  
10- weak taken  
11- Strong taken

# Branch Prediction (4)

- Delayed Branch
  - Do not take jump until you have to
  - Rearrange instructions

# Problem 3

Consider a computer system that requires four stages to execute an instruction. The details of the stages and their time requirements are listed below:

Instruction Fetch (IF) stage: 30 ns,

Instruction Decode (ID) stage: 9 ns

Execute (EX) stage: 20 ns

Store Results (WO stage): 10 ns.

Assume that inter-stage delay is 1 ns and every instruction in the program must proceed through the stages in sequence.

- What is the frequency of the Processor?
  - What is the minimum time for 10 instructions to complete in non-pipelined manner?
  - What is the minimum time for 10 instructions to complete in pipelined manner?
- (Handwritten notes for part c):*
- $$\{k + \{n-1\}\} * t = \{k + 9\} * 3 \text{ ns} \rightarrow 403 \text{ ns}$$

# Problem 4

---

A computer program is developed to run the DBSCAN algorithm on a processor with L1 cache and main memory organization with hit ratio of 67%. The processor supports 64-bit address bus, an 8-bit data bus and has a 256 KByte data cache memory with 4-way set associativity. The main memory is logically divided into a block size of 256 Bytes. Each cache line entry contains, in addition to address tag, 1 dirty bit.

- i. What is the number of bits in the tag field of an address?
  - ii. If the associativity in the above problem, is changed to 8-way, do you see any performance gain and/ or drawback? Comment.
-

Cont.

i. What is the number of bits in the tag field of an address?

Number of lines in the cache =  $256 \text{ KB} / \underline{256 \text{ B}} = \underline{2^{10}} = 1 \text{ K lines}$

4 way = 4 lines per set

Therefore Number of sets =  $1\text{K} / 4 = \underline{256 \text{ sets}} = 2^8$

Number of bits needed to address sets = 8 bits

Number bits needed to address a word in the block :

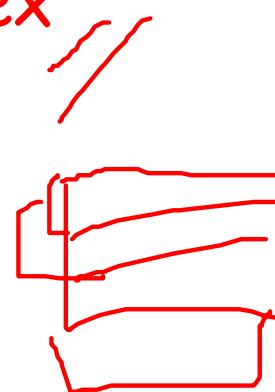
$256 = 2^8 \rightarrow 8 \text{ bits}$

Tag bit =  $64 - \underline{\underline{8}} - \underline{\underline{8}} = \underline{\underline{48 \text{ bits}}}$

Cont.

- 
- ii. If the associativity in the above problem, is changed to 8-way, do you see any performance gain and/ or drawback?

There is a performance gain in terms number of HITS  
Tag comparison circuit becomes complex



# Problem 5

R2000

A RISC based CPU is to be designed to have 32 opcodes, source and destination operands referring to 64 registers, and a displacement of value such as 2ABCH, Specify the instruction format mentioning the various fields and bits required by them.

<u>Opcode [5 bits]</u>	<u>Source operand [6bits]</u>	<u>Destination operand [6 bits]</u>	<u>Displacement [16 Bits]</u>

# Problem 6

---

You are asked to design a Data Analytics Server? What would be the challenges as a hardware designer you would need to address? Also mention the solutions you would propose ?

# Challenges

- Structural Hazards should be avoided. The processor should have adequate data processing and execution Capabilities
- Stalling in the pipelines should be avoided by handling all possible dependencies
- Registers and Instruction Set should be compatible to data analytics requirement
- Cache memory constraints should be taken care. Miss penalty rate should be minimal or zero.
- Speculative capabilities should be handled with minimal penalty rate
- Cache optimization should be used to handle memory constraints.



# Solutions

- Assess the features required and design the processors
- Performance of the server is to be addressed.
- Registers and Instructions should be kept optimal.
- Use Compiler based optimization as far as possible.

//