Computer Organization and Software Systems

Contact Session 1

Introduction to Computer Systems

Dr. Lucy J. Gudino

Team

Instructors:

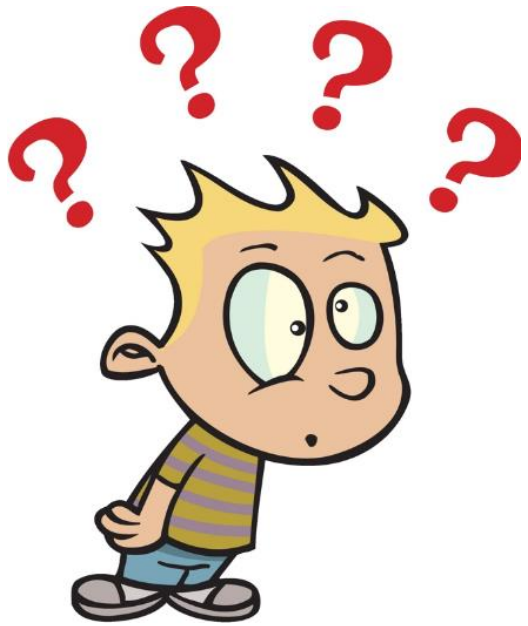
- Dr. Lucy J Gudino (IC)
- Prof. Lakshmikanta
- Prof. Pradeep H K

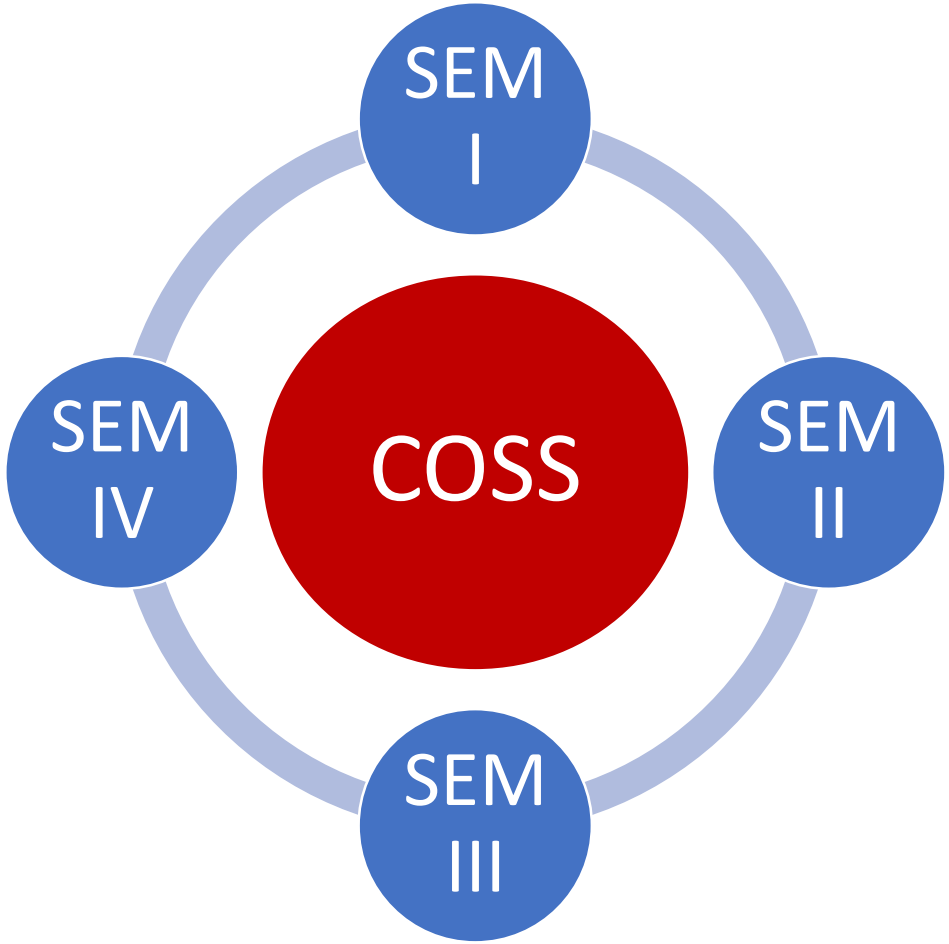
Learning Facilitators:

- Dr. VAIBHAV JAIN (Lead TA)
- Prof. Vivekananda M R
- Dr. Shamanth Nagraj
- Prof. Puneet Kumar

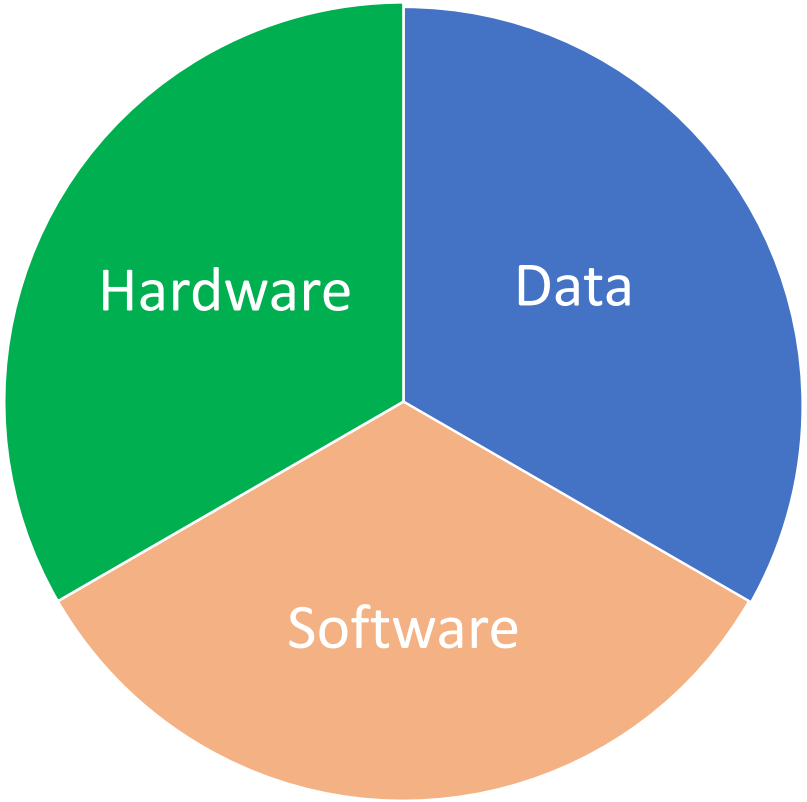
Introduction

- **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**.

Three courses

- Computer Organization and Software Systems (Core Course)
- Systems for Data Analytics (Elective)
- Big Data Systems (Elective)

Benefits

- Understanding System Architecture
- Efficient Code Implementation
- Performance Optimization
- Memory Management
- Parallel Computing
- System-level Troubleshooting
- Integration with Existing Systems
- Scalability Considerations
- Resource Utilization

```
int count;

for (count = 1; count <= 10; count ++)
        printf("Count = %d", count);
```

Text Books and Reference Books

Text Books:

(T1) W. Stallings, Computer Organization & Architecture, PHI, 10th ed., 2010.

(T2) A Silberschatz, Abraham and others, Operating Systems Concepts, Wiley Student Edition, 8th 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, 3rd 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

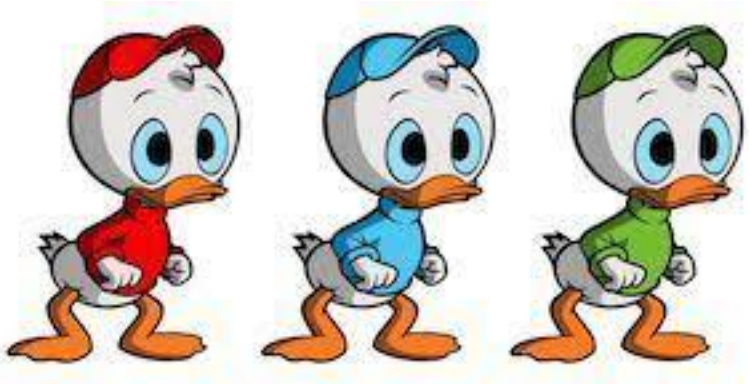
- 5 unit course.

Sl No.	Evaluation Component	Weightage %	Nature of Component
1	Mid Sem Exam	30%	CB
2	Comprehensive Examination	40%	OB
3	Quiz	5% (Two quizzes – Best of two)	OB
4	Assignments	25%	OB

Assignments

- Two assignments:
  - One pre-midsem exam : 10%
  - One post-midsem : 15%
- Lab based
- Simulator to be used : CPU-OS simulator
  - Nuvepro Cloud Lab

Assignment should not be



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 eLearn discussion forum
5. In case of query:
   - Subject related queries (Evaluation, Assignments, Quiz, discrepancy in Marks etc.) ➔ Prof. Vaibhav Jain (vaibhav.jain@wilp.bits-pilani.ac.in)
   - Operation related queries (Admission, registration, Exam related matters, grade sheets etc.) ➔ support@wilp.bits-pilani.ac.in

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
1-2	**Introduction to Computer Systems** • Hardware Organization of a computer • Basic uniprocessor architecture • Instruction Cycle State Diagram • Operating System role in Managing Hardware • Running a Hello Program	T1

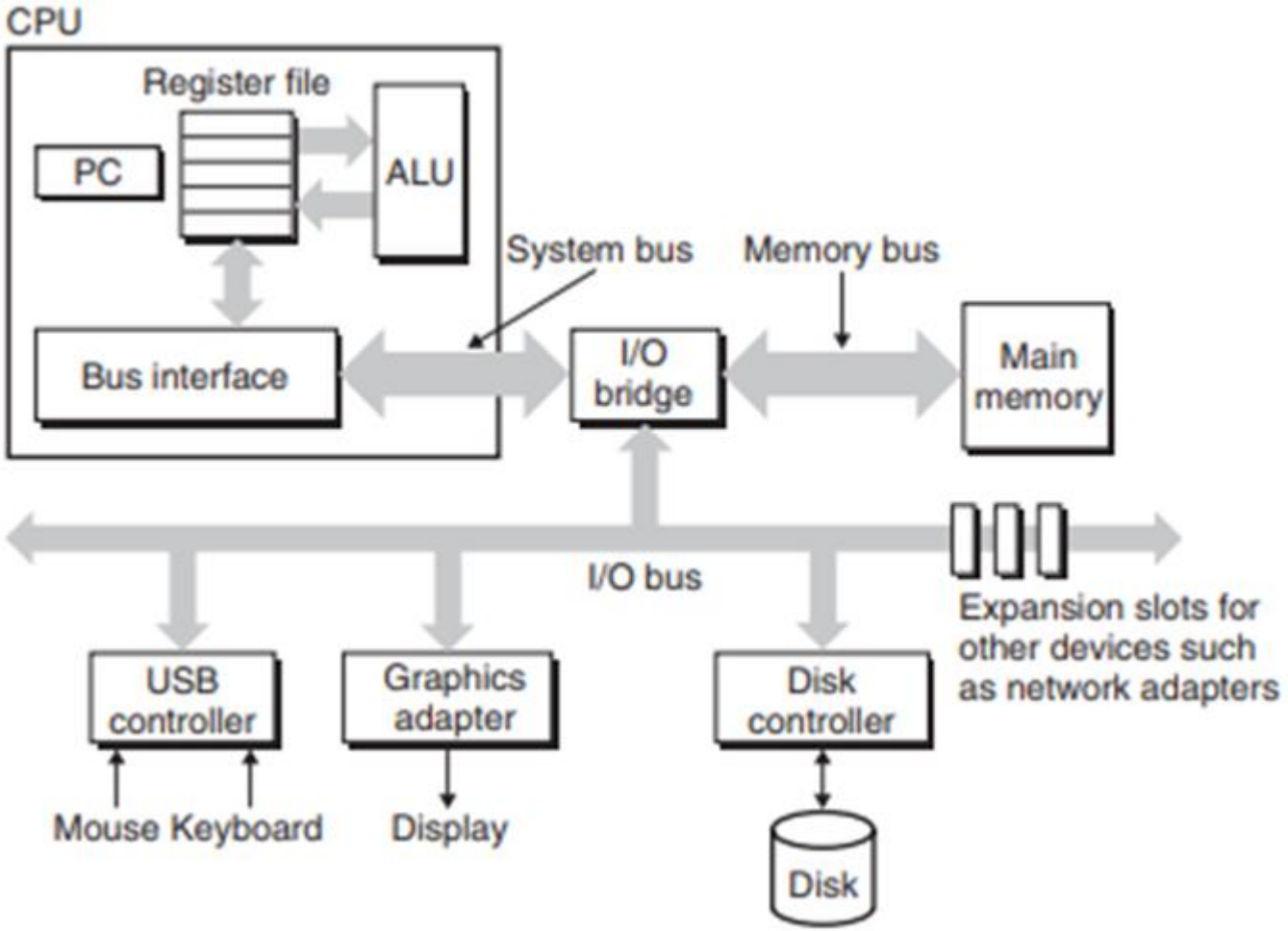
Definition of a Computer

- 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

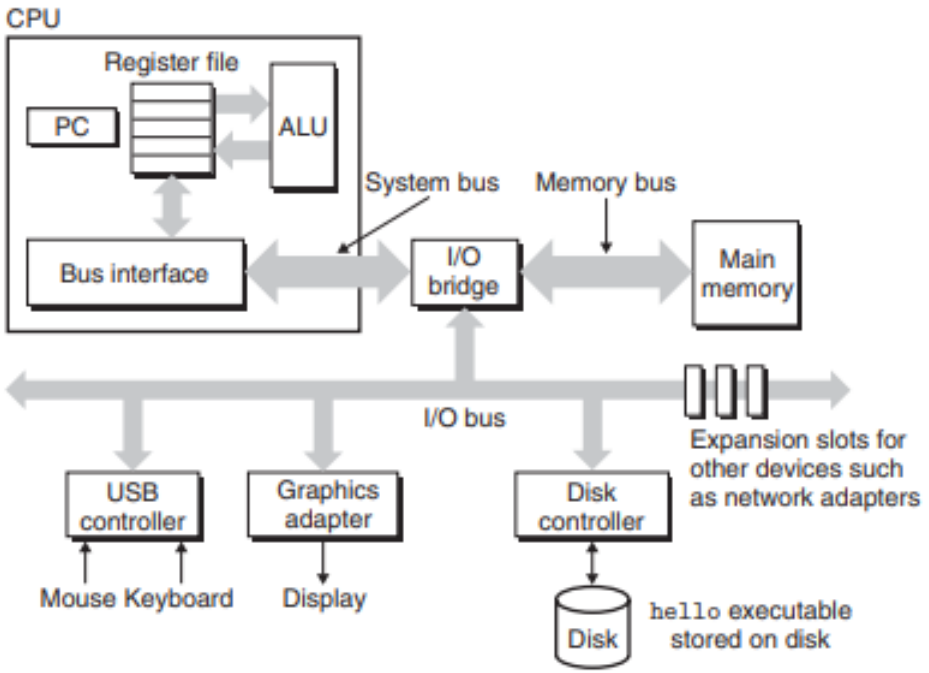
- **Hardware**
  - Central Processing Unit (CPU)
  - Memory
  - I/O devices
- **Software**
  - System Software
    - System Management Software
    - Tools and Utilities for Developing the software
  - Application Software
    - General Purpose Software
    - Specific Purposed Software

Hardware Organization of a computer



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

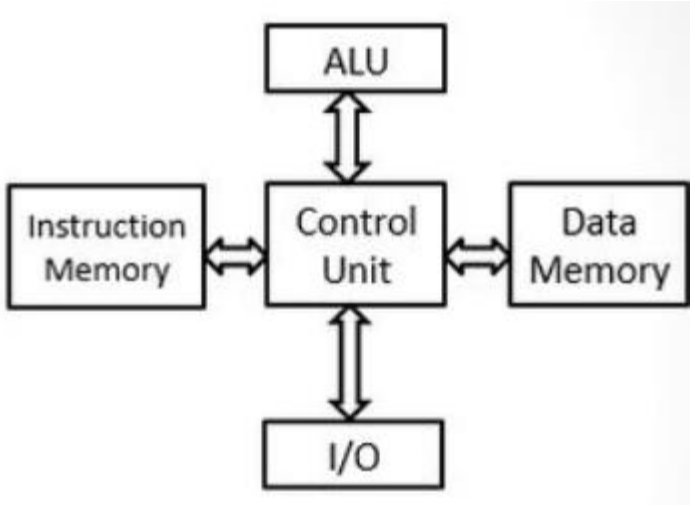


Von Neumann Architecture…

- 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)

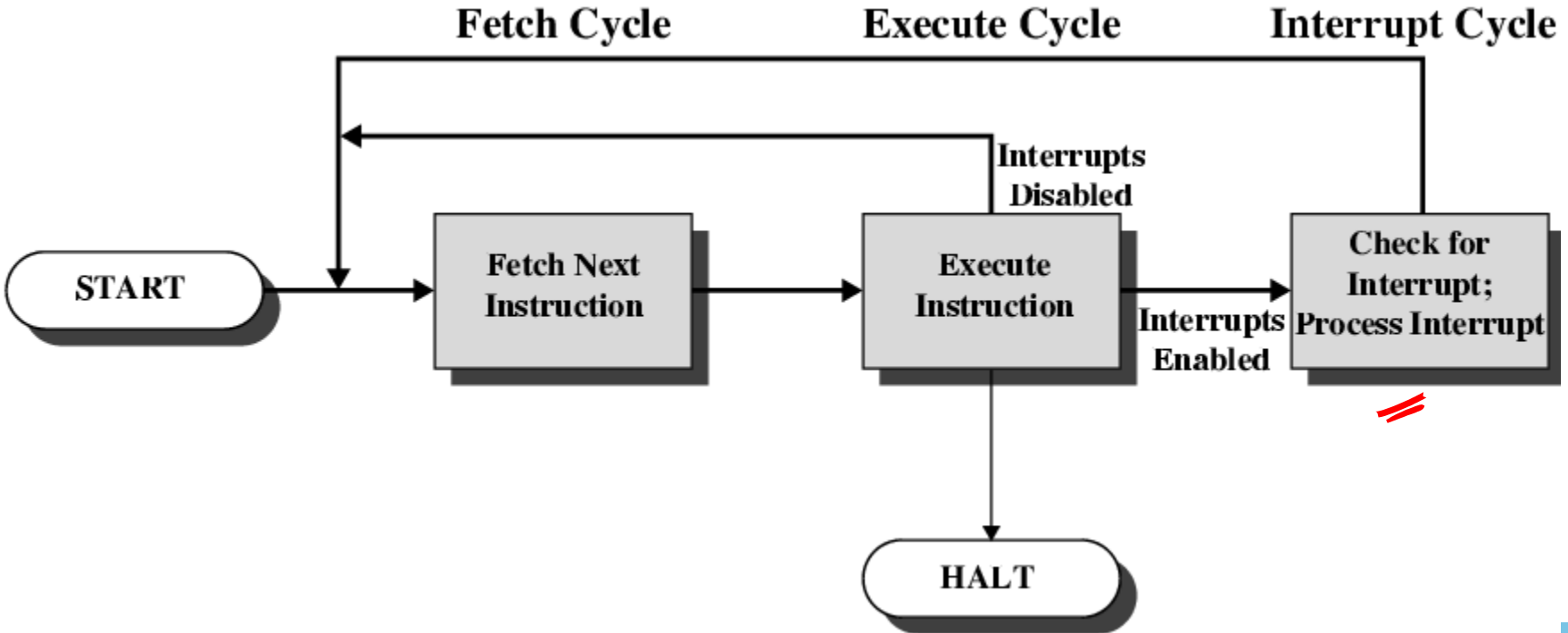
Harvard Architecture

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



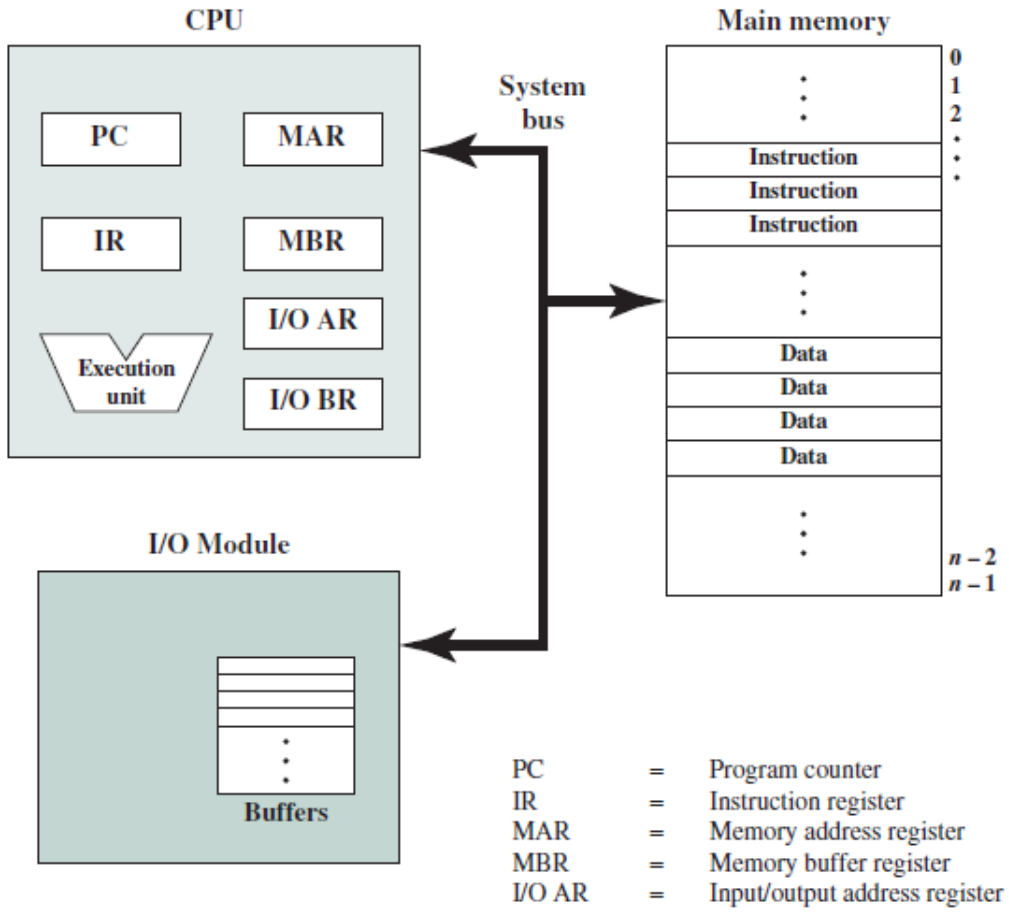
Instruction Cycle Diagram

- 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



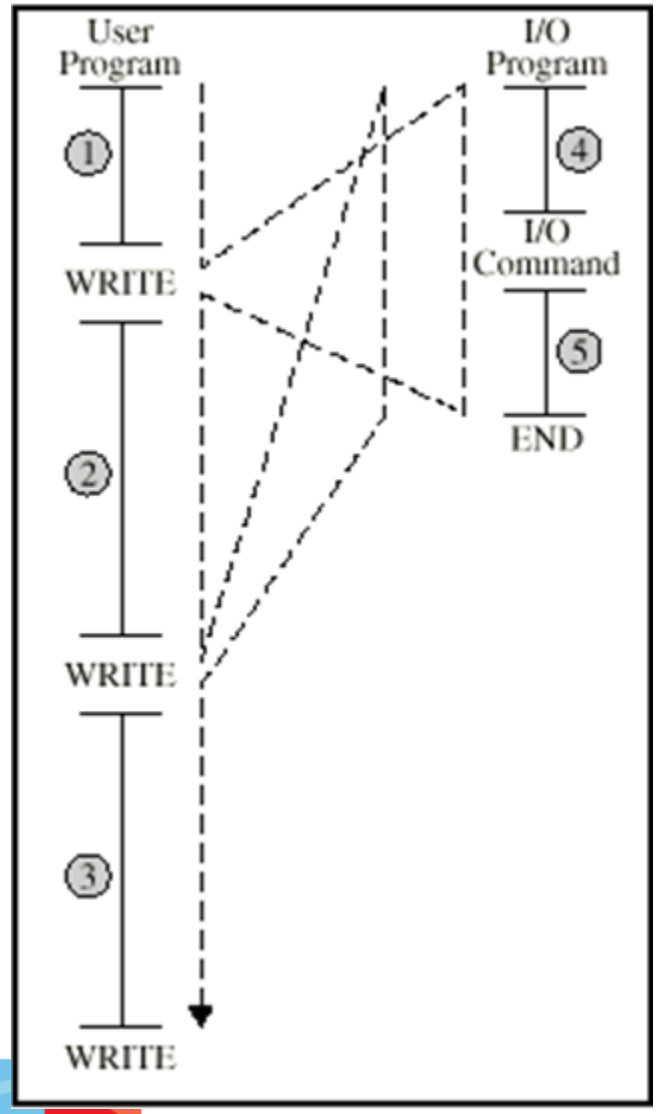
Execute Cycle

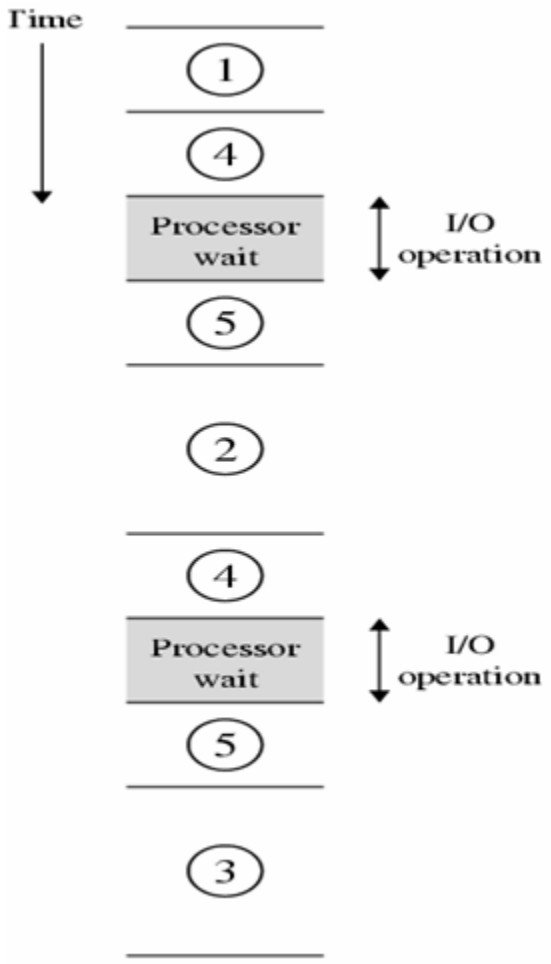
- Processor - memory
  - Data transfer between CPU and main memory
- Processor - I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above

Interrupt Cycle

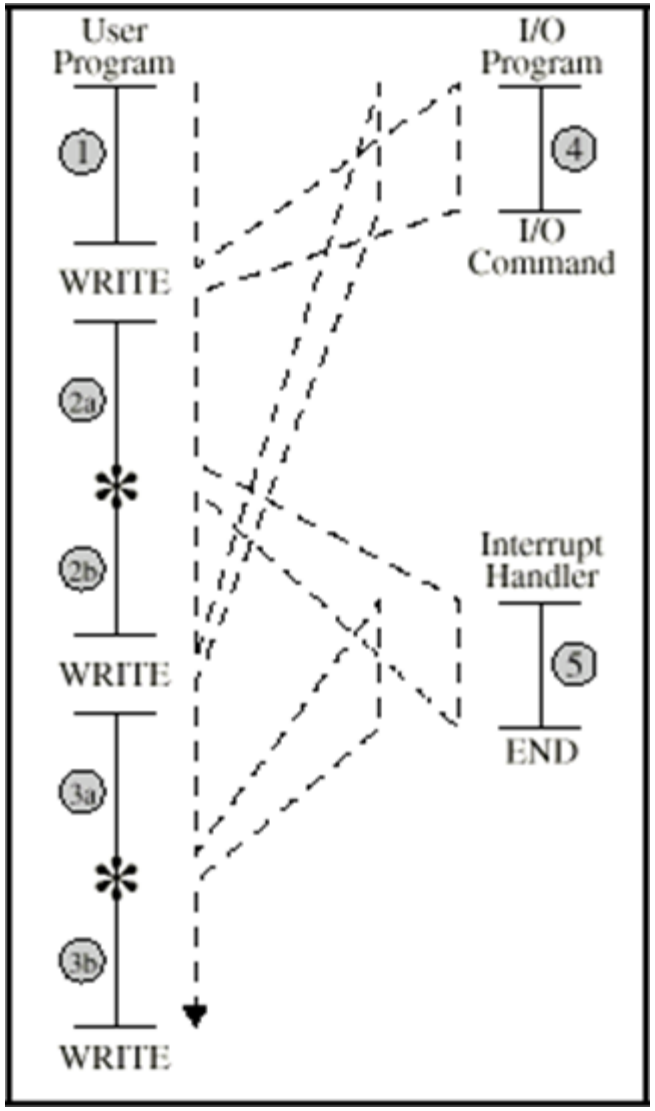
- Interrupts: Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Interrupts enhances processing efficiency

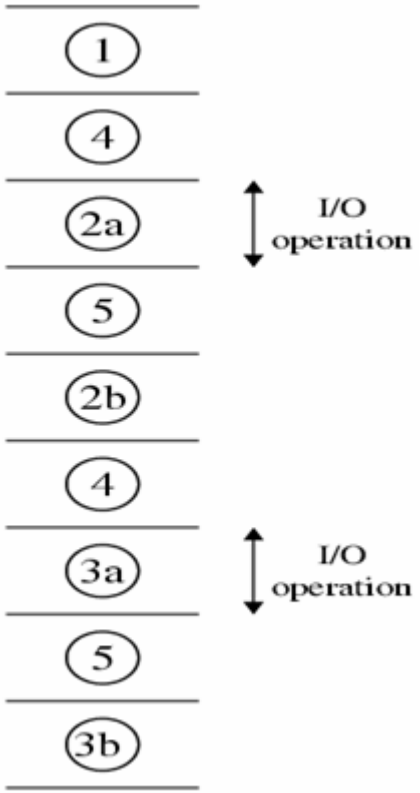
Program Flow Control (No Interrupts)





Program Flow Control (With Interrupts)





Types of Interrupts

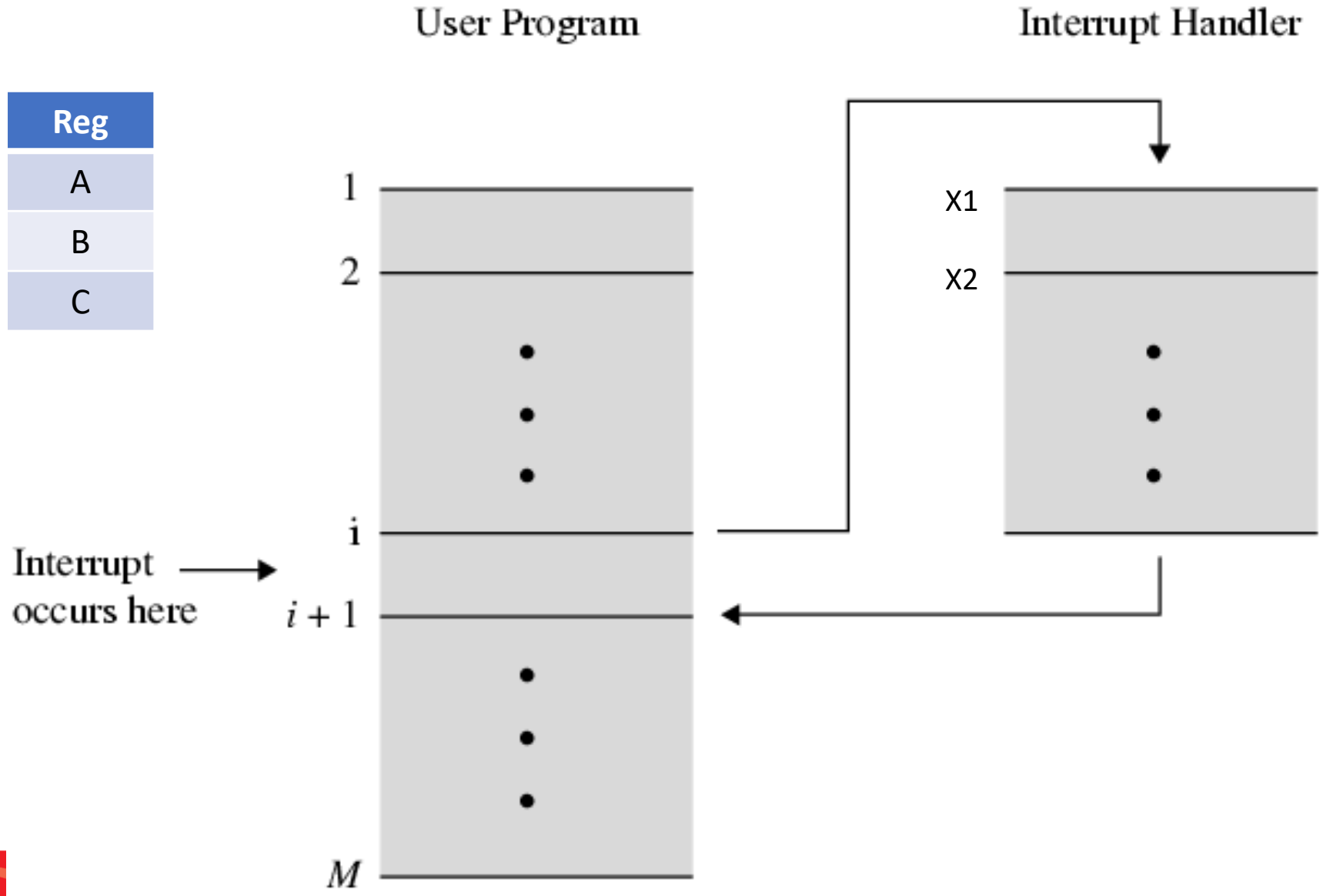
- Types of interrupts:
  - Program
    - e.g. overflow, division by zero
  - Timer
    - Generated by internal processor timer
    - Used in pre-emptive multi-tasking
  - I/O
    - from I/O controller
  - Hardware failure
    - e.g. memory parity error

Interrupt Cycle

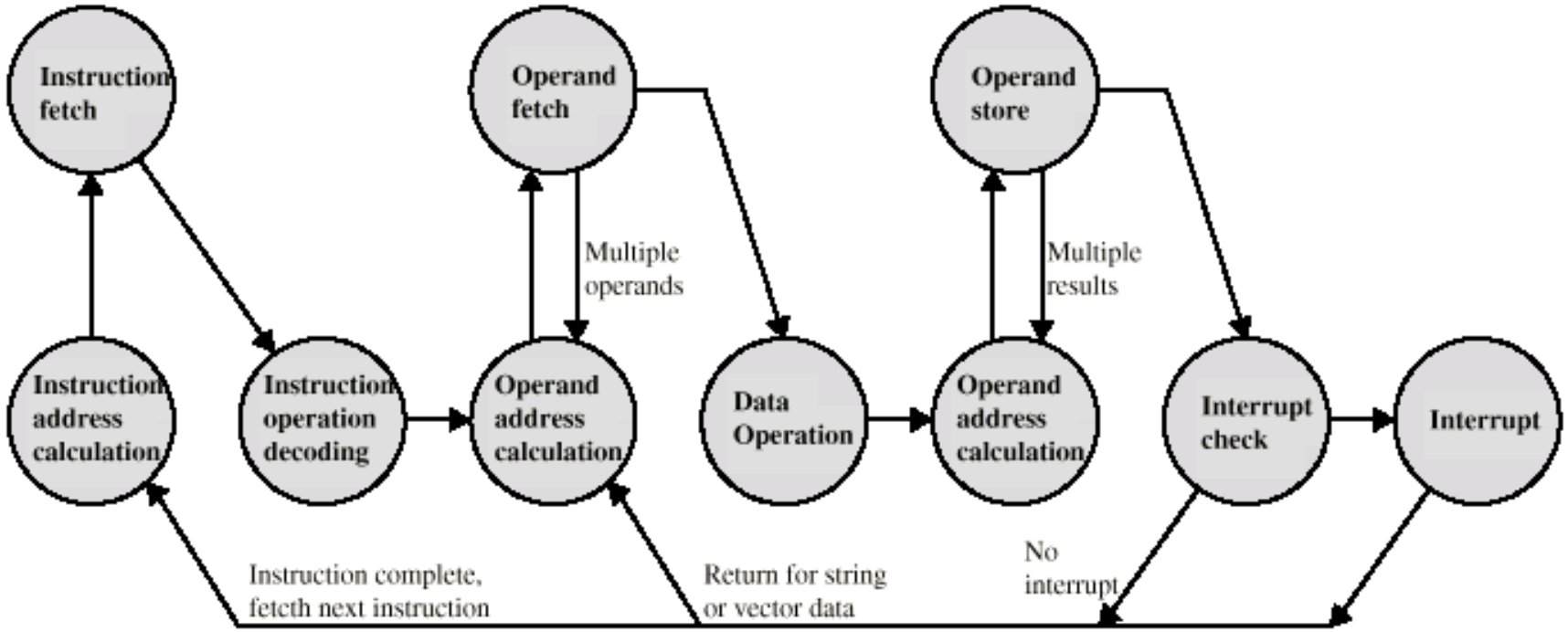
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue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

Transfer of Control via Interrupts

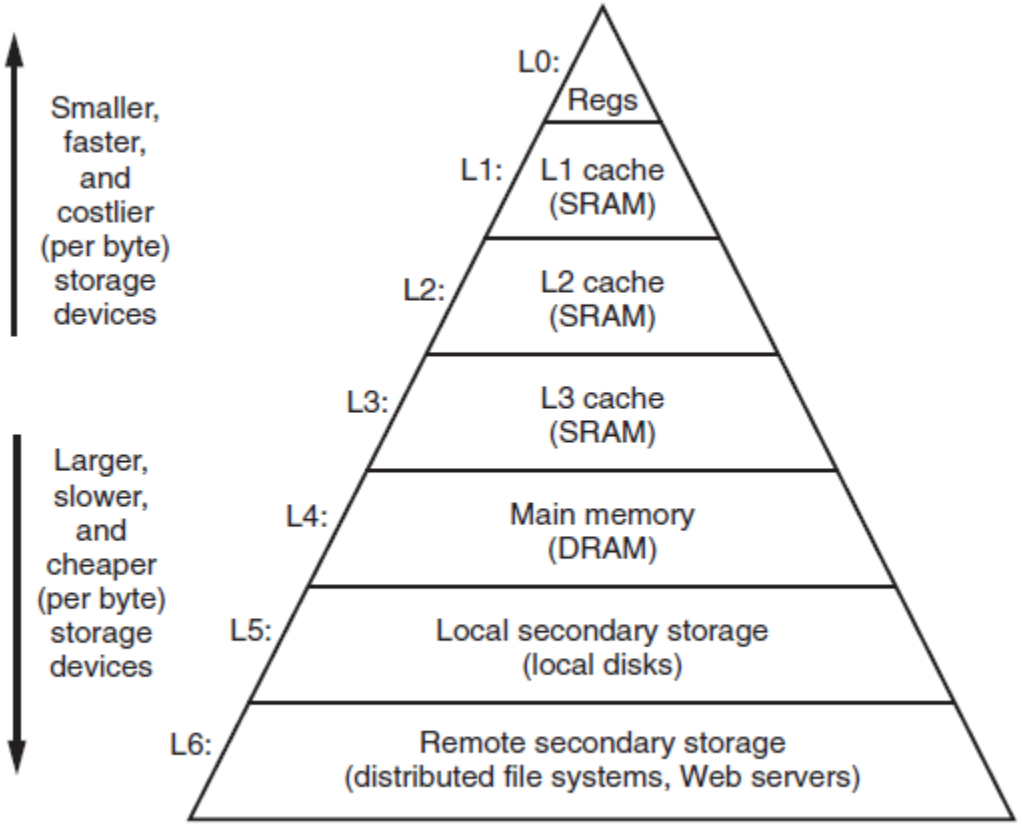
Reg
A
B
C



Instruction Cycle - State Diagram

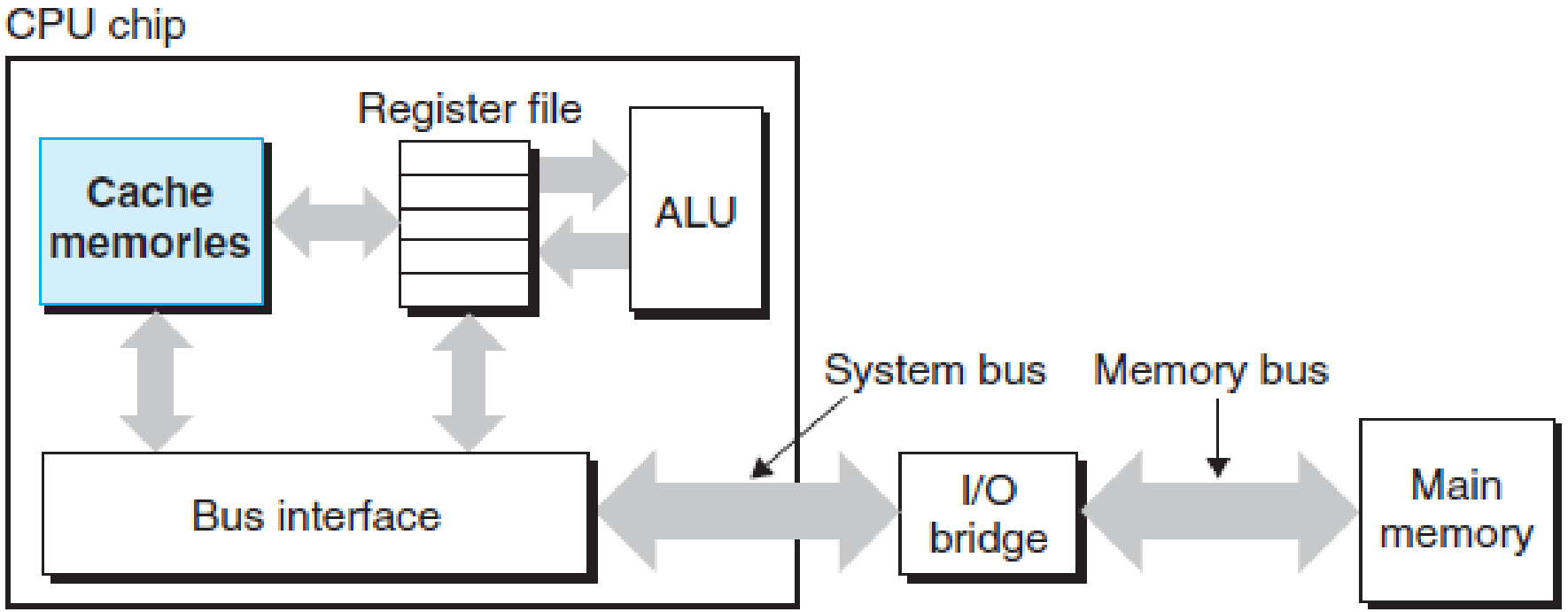


Memory Hierarchy



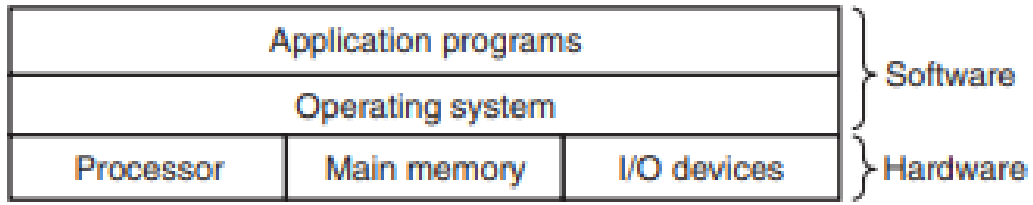
An example of a memory hierarchy.

Role of Cache Memory



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

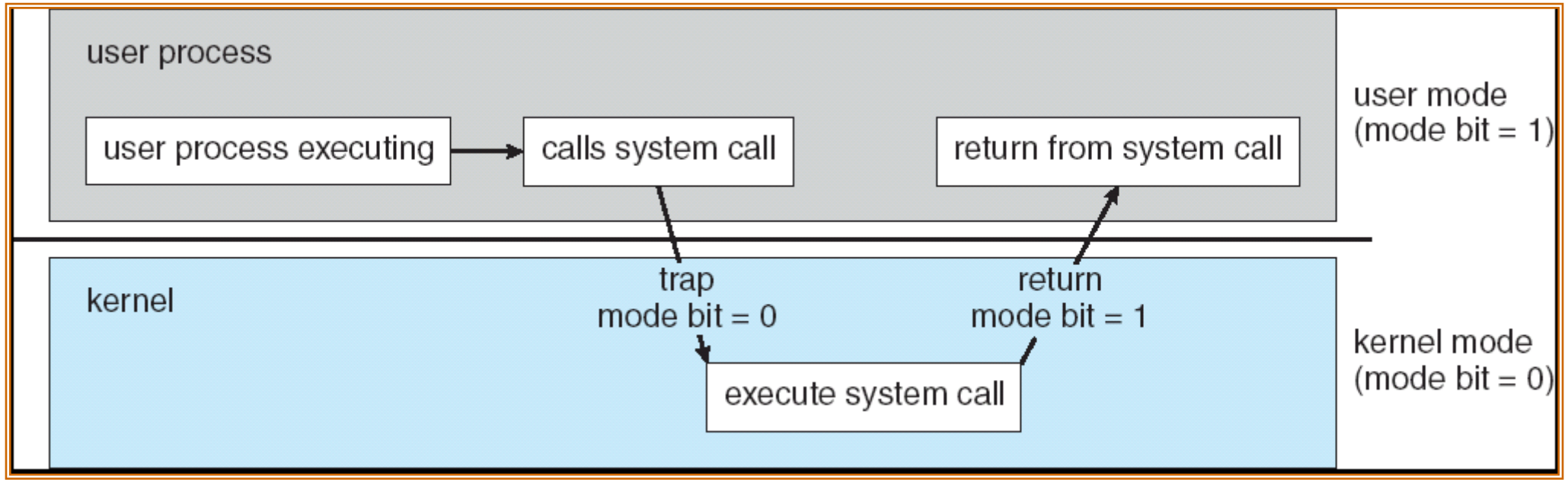
- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

- "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

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

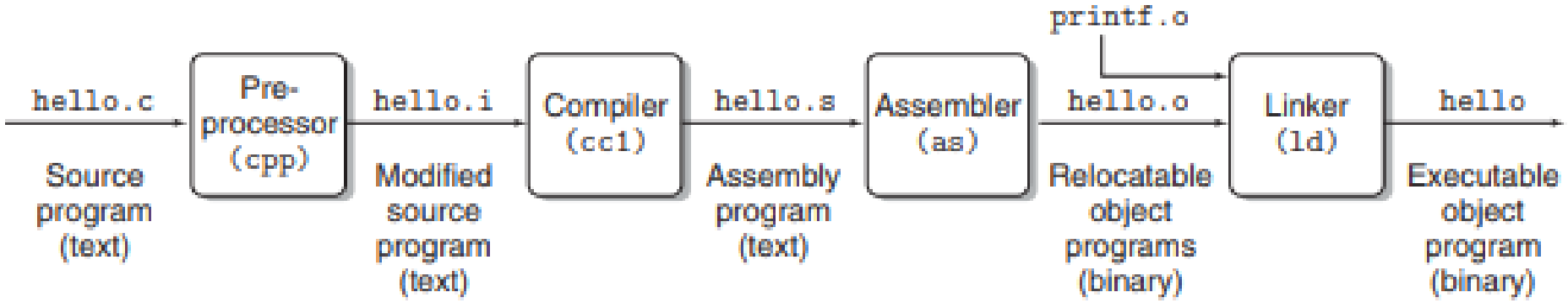
Transition from user to kernel mode



Running a Hello.c Program

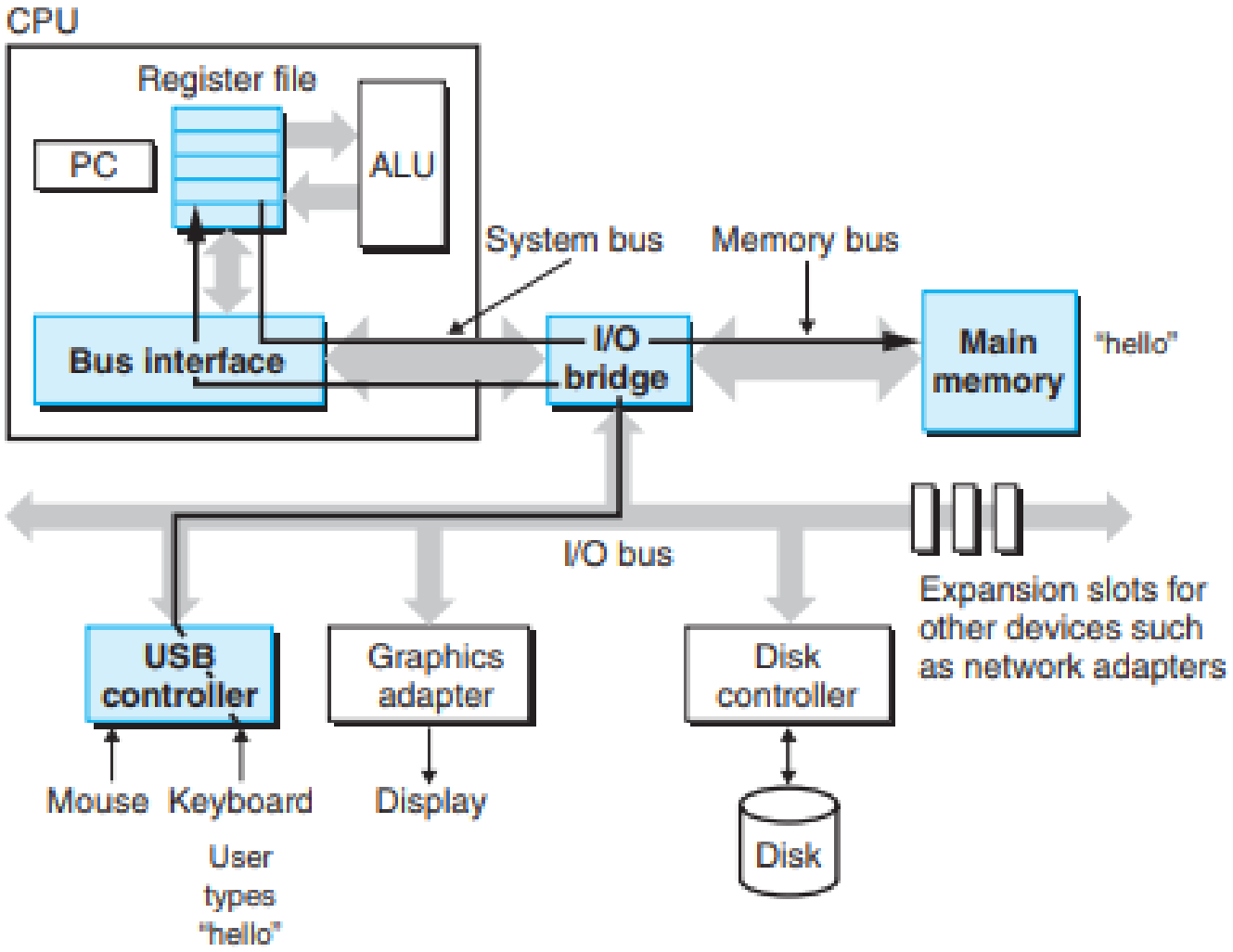
```
#include <stdio.h>

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



The compilation system.

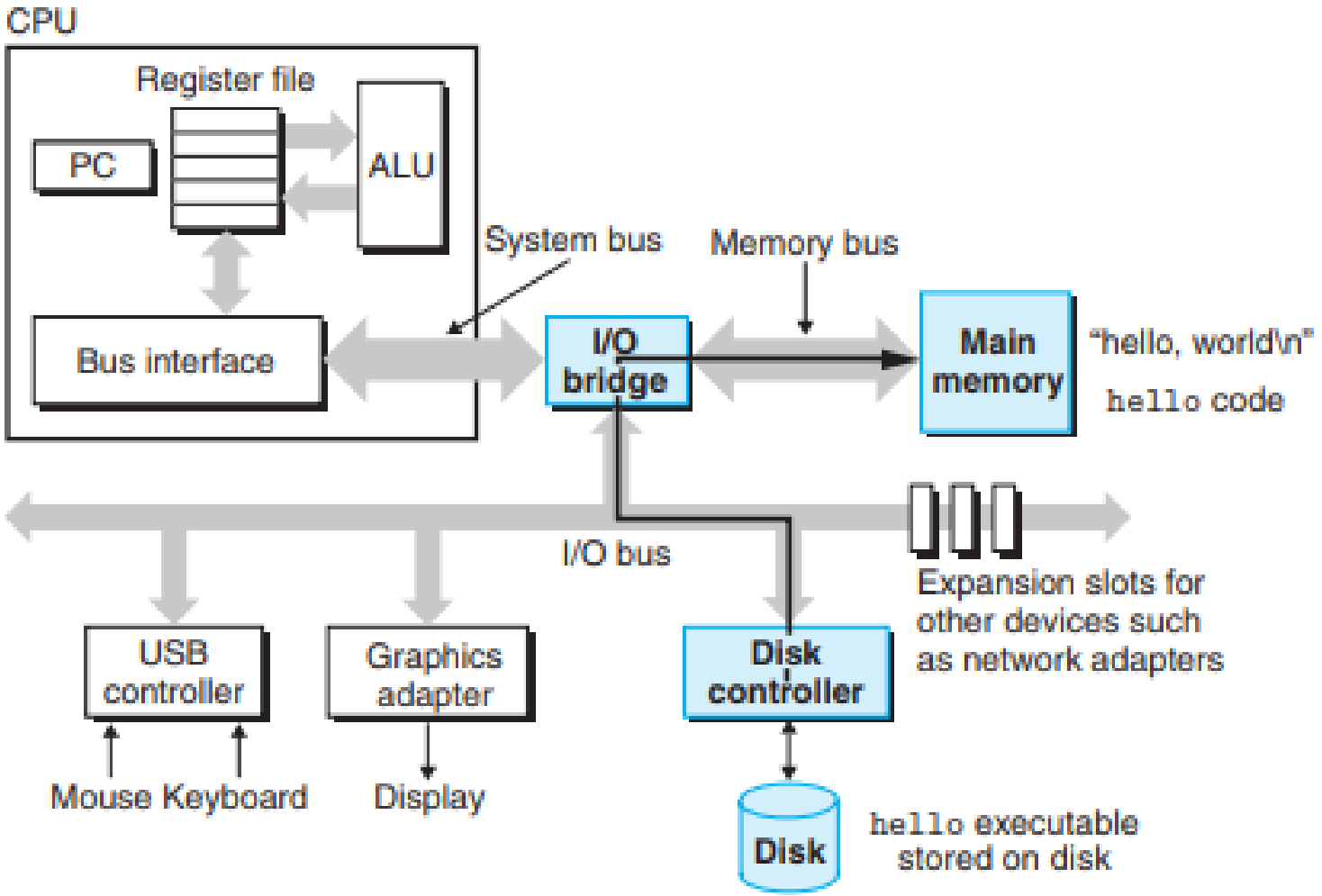
Reading ./hello command from Keyboard



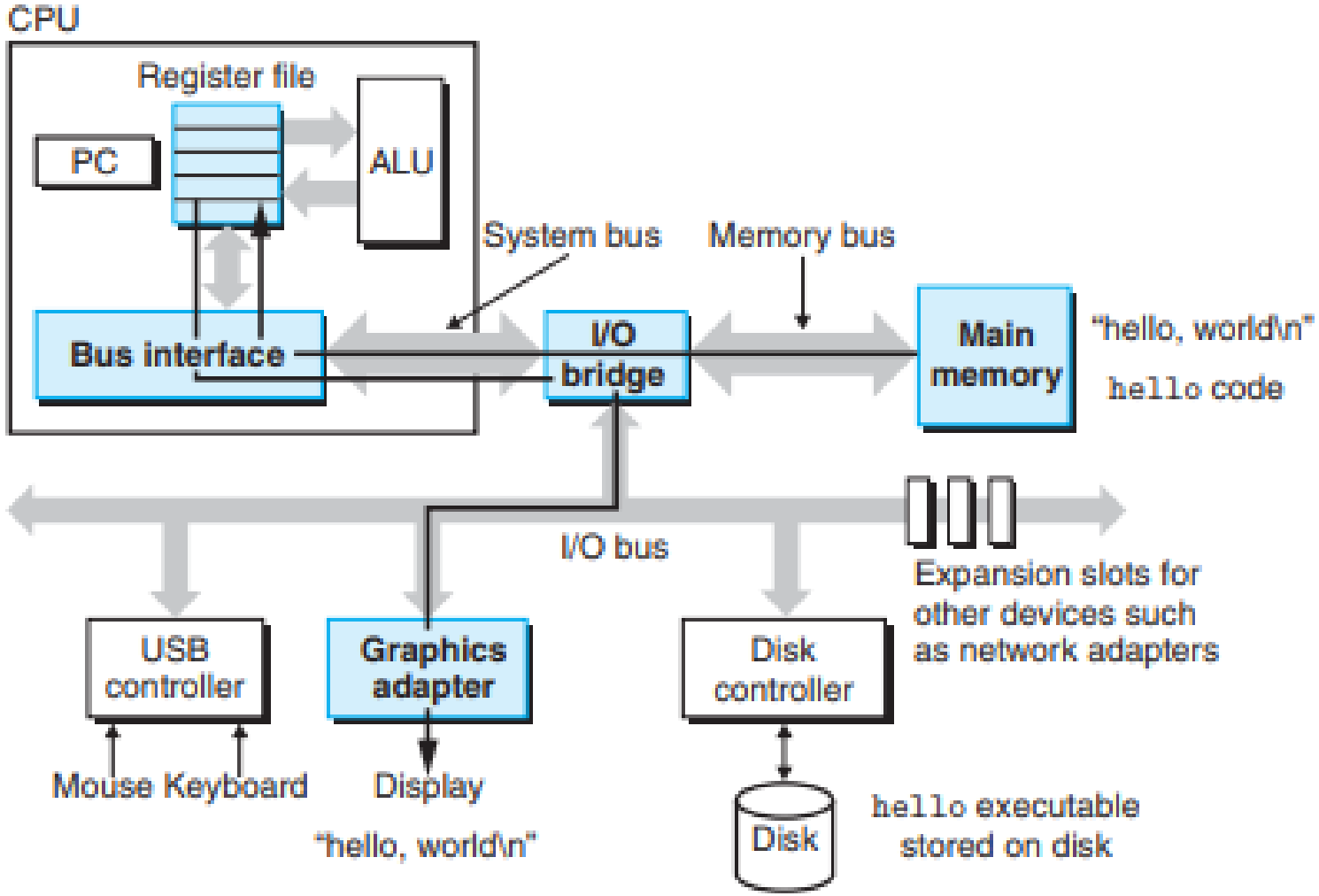

```c
#include <stdio.h>

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

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.

Computer Organization and Software Systems

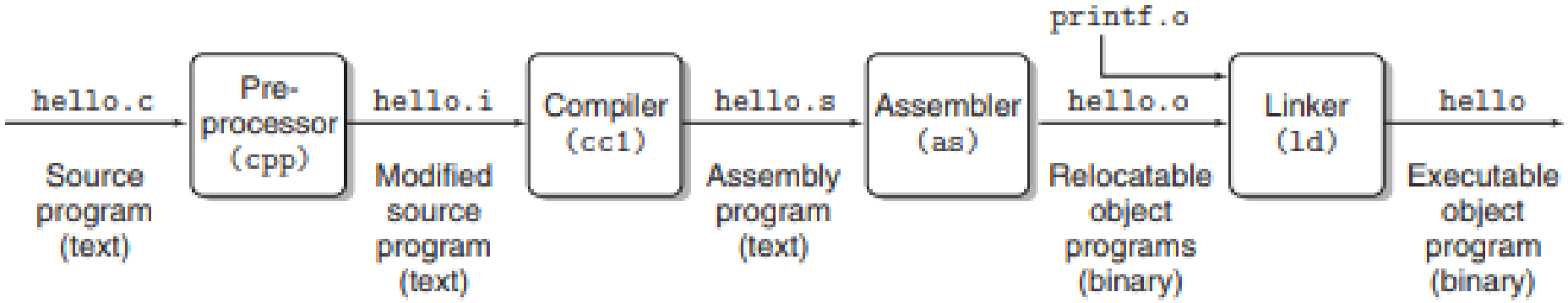
Contact Session 2

Dr. Lucy J. Gudino

Running a Hello.c Program

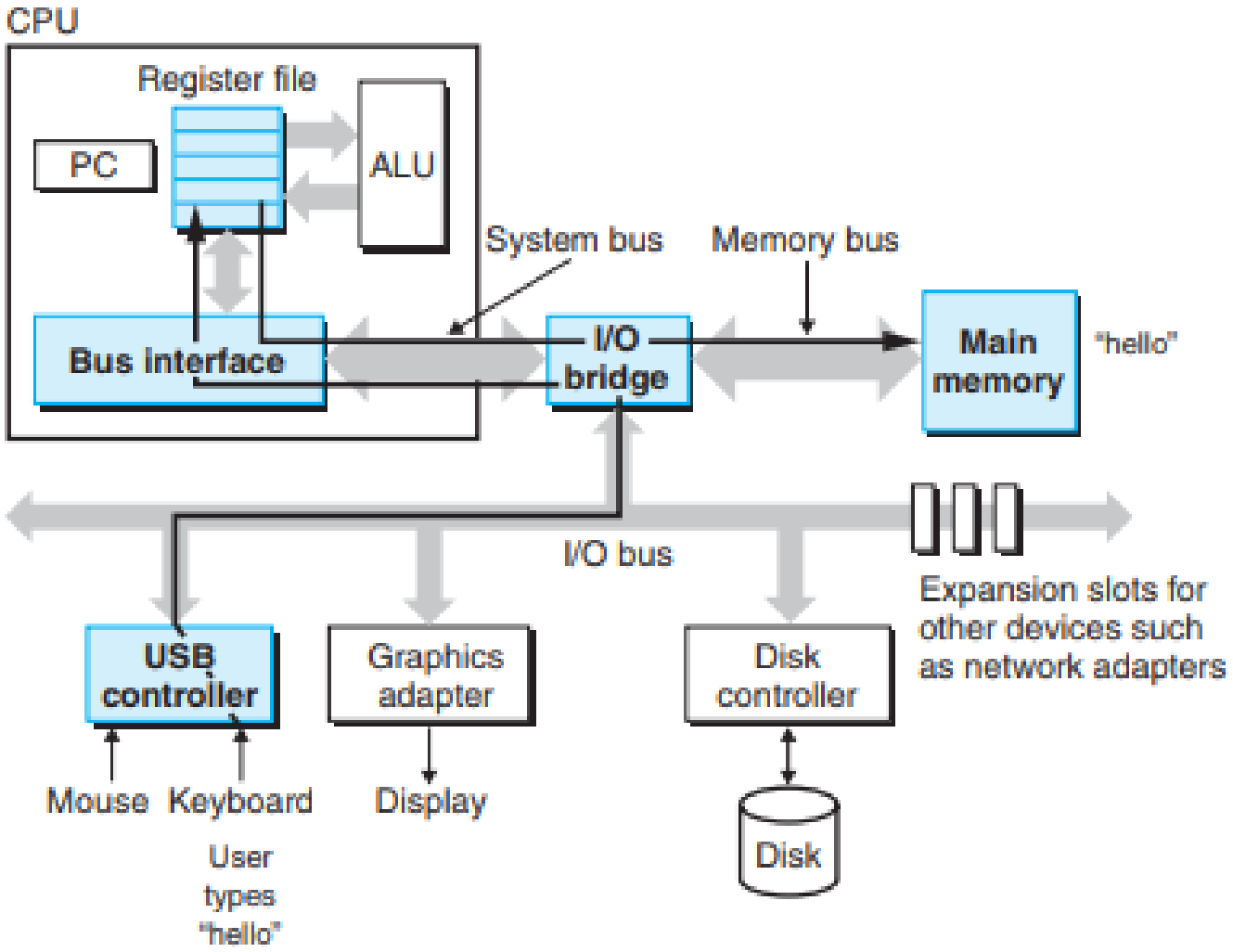
```
#include <stdio.h>

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



The compilation system.

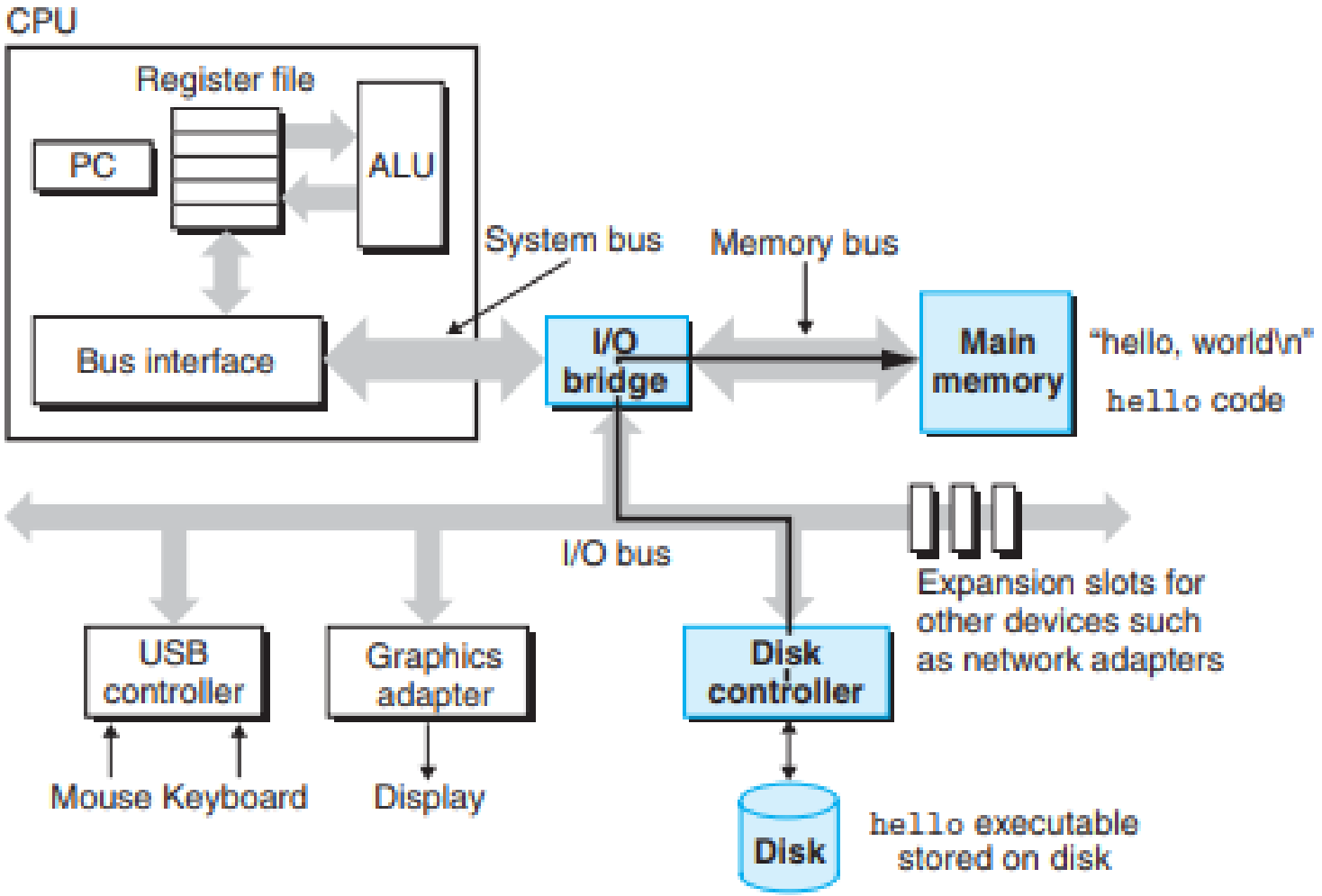
Reading ./hello command from Keyboard



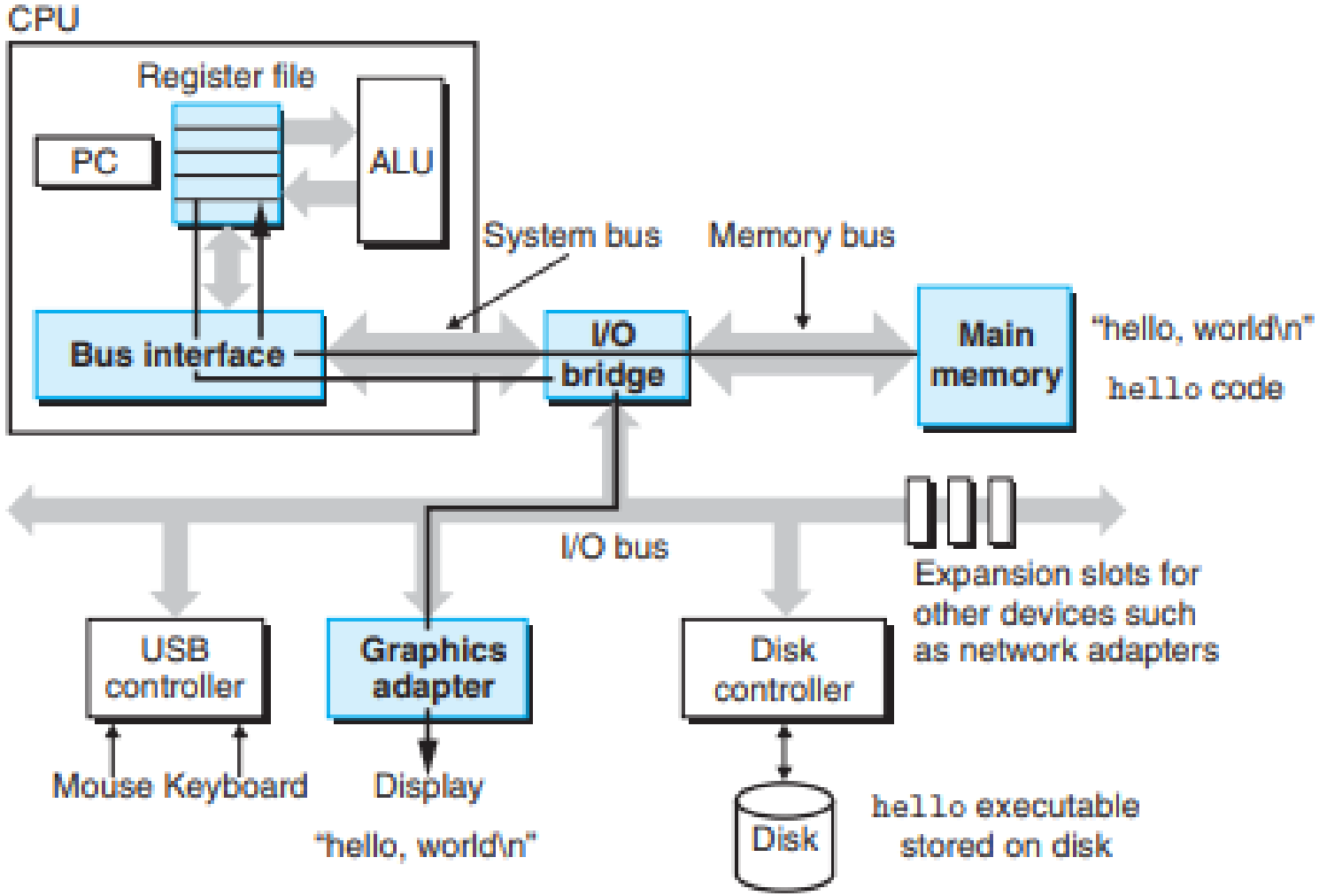

```c
#include <stdio.h>

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

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.

Today's Class

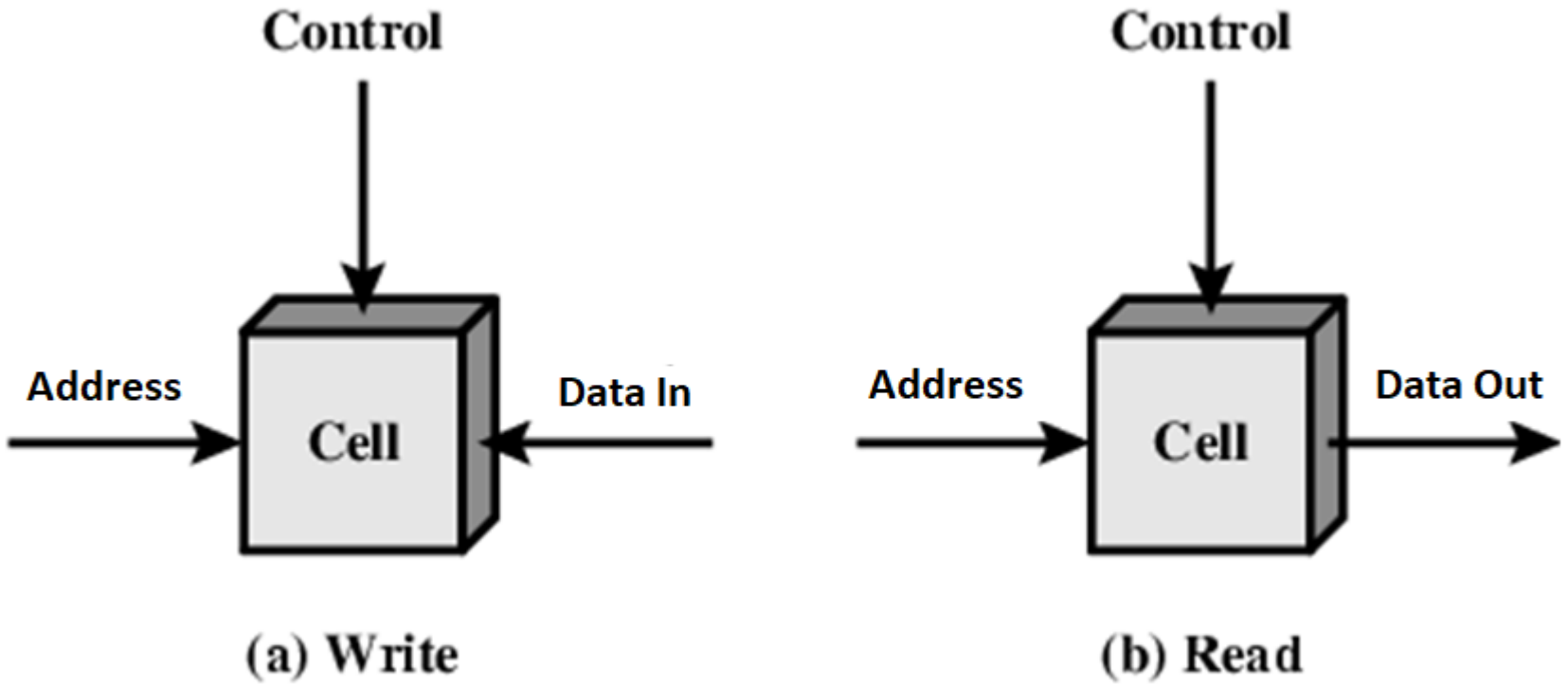
Contact Hour	List of Topic Title	Text/Ref Book/external resource
3-4	Memory Organization - Internal Memory - External Memory	T1, R2

Memory Organization

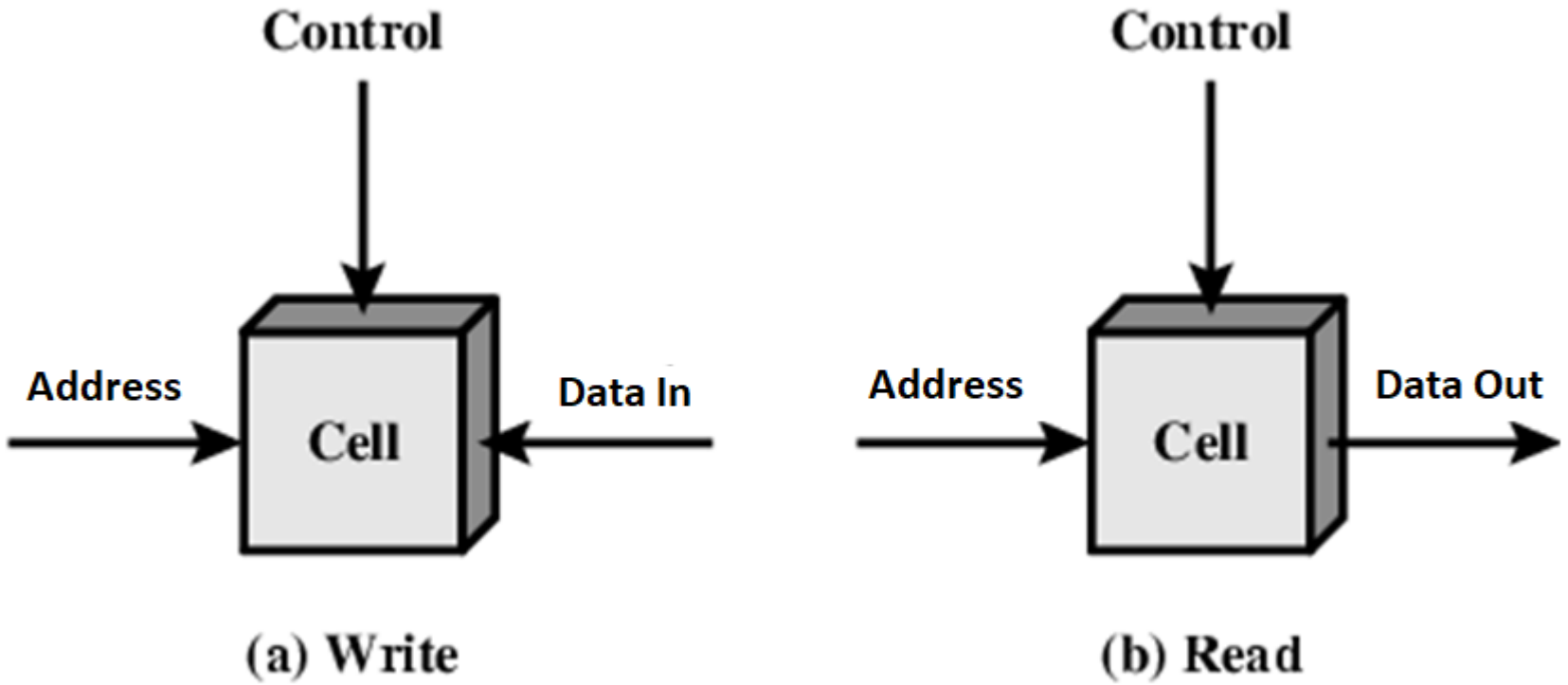
Internal Memory Organization

- Internal Memory
  - Also known as main memory or Primary Memory
  - Small data storage but quick access.
  - Examples : RAM, ROM
- External Memory
  - Also Known as secondary Memory
  - Huge data stored persistently
  - Examples: hard disk, solid state drives, USB flash drives etc.

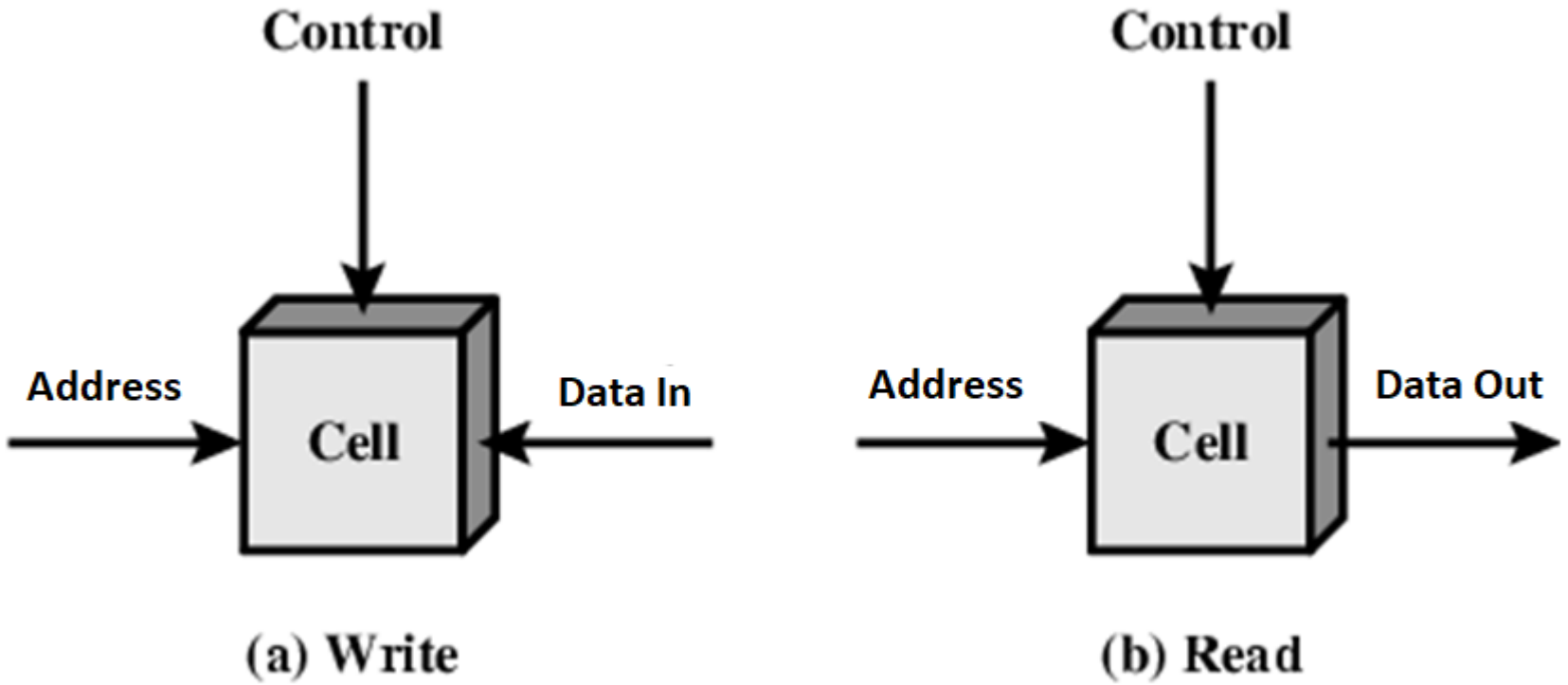
Semiconductor Memory



Semiconductor Memory



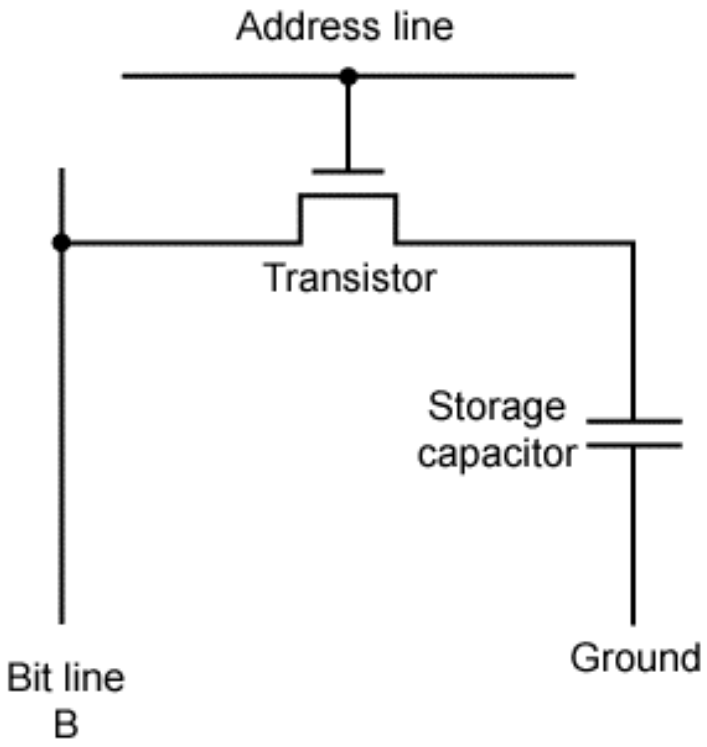
Semiconductor Memory

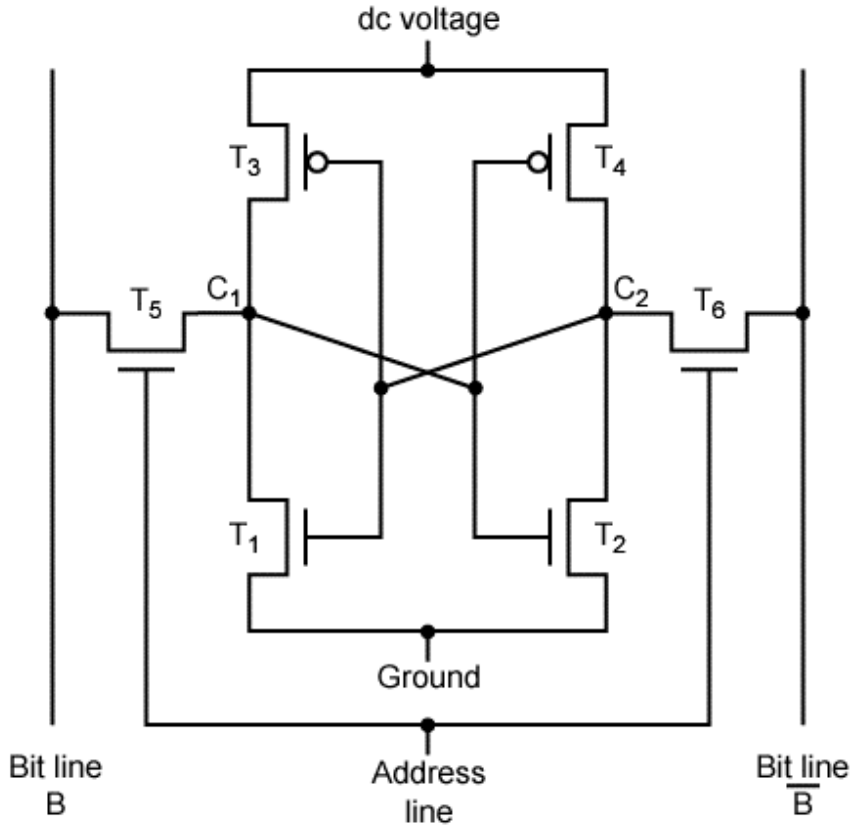


Random-Access Memory (RAM)

- 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
  - Loose information if powered off.

DRAM vs SRAM





	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

innovate achieve lead

Read Only Memory

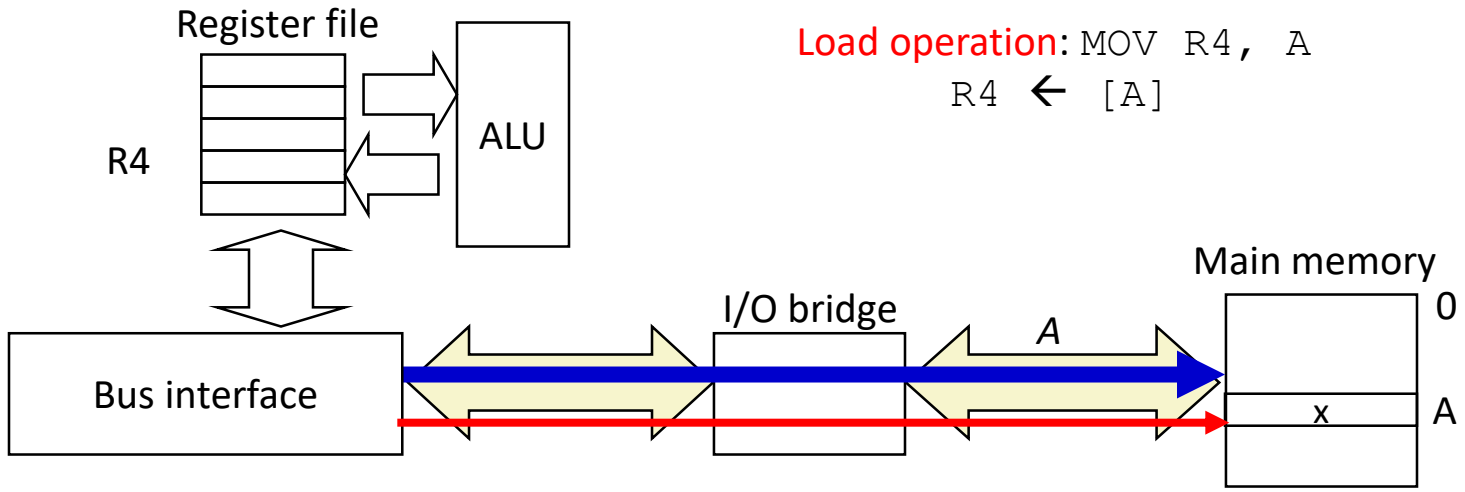
- Permanent Storage and Nonvolatile Memories
- Read Only Memory Variants:
  - Read-only memory (ROM): programmed during production
  - Programmable ROM (PROM): can be programmed once
  - Erasable PROM (EPROM): can be bulk erased (UV, X-Ray)
  - Electrically erasable PROM (EEPROM): electronic erase capability
  - 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)

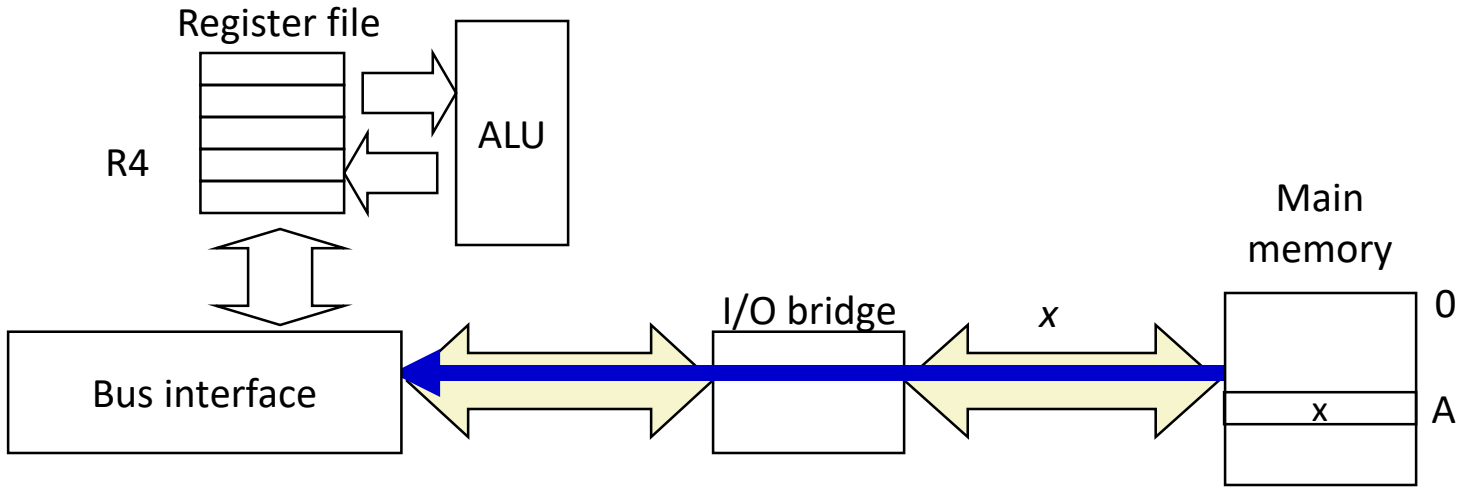
- CPU places address *A* and then read control signal on the memory bus



Memory Read Operation (2)

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

Load operation: `MOV R4, A`
`R4 ← [A]`

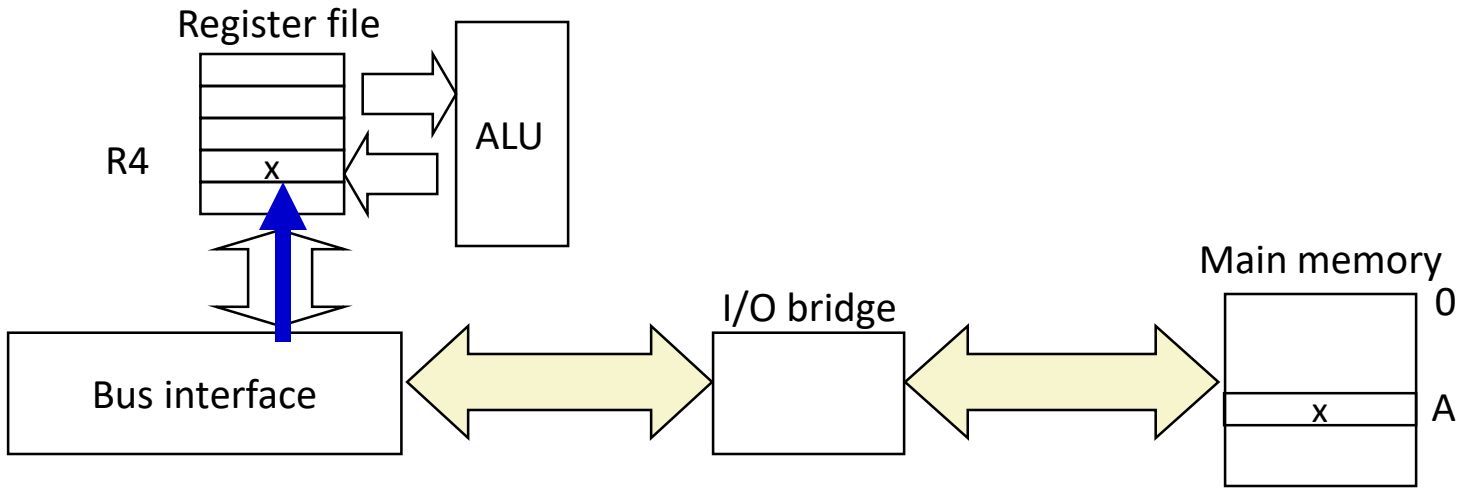


Memory Read Operation (3)

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

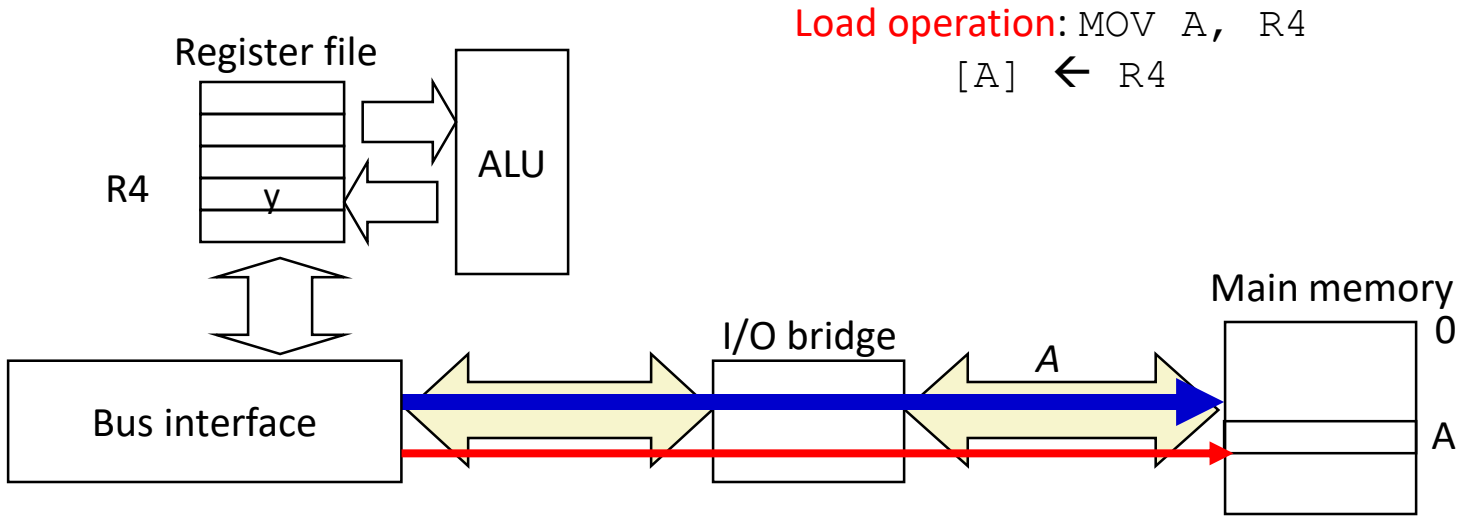
Load operation: `MOV R4, A`

`R4 ←[A]`



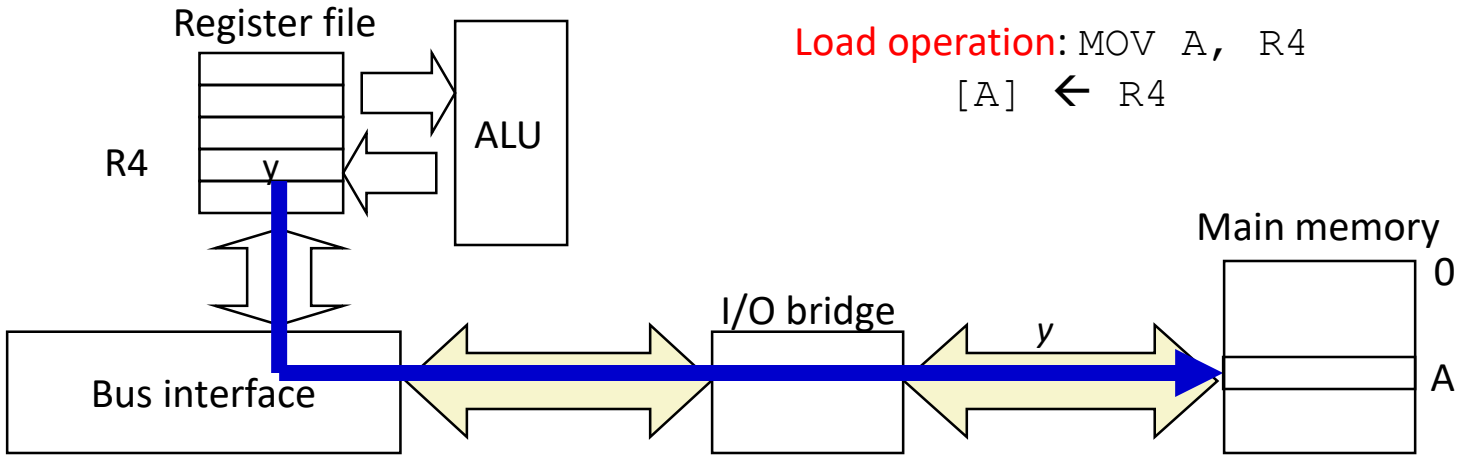
Memory Write Operation (1)

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



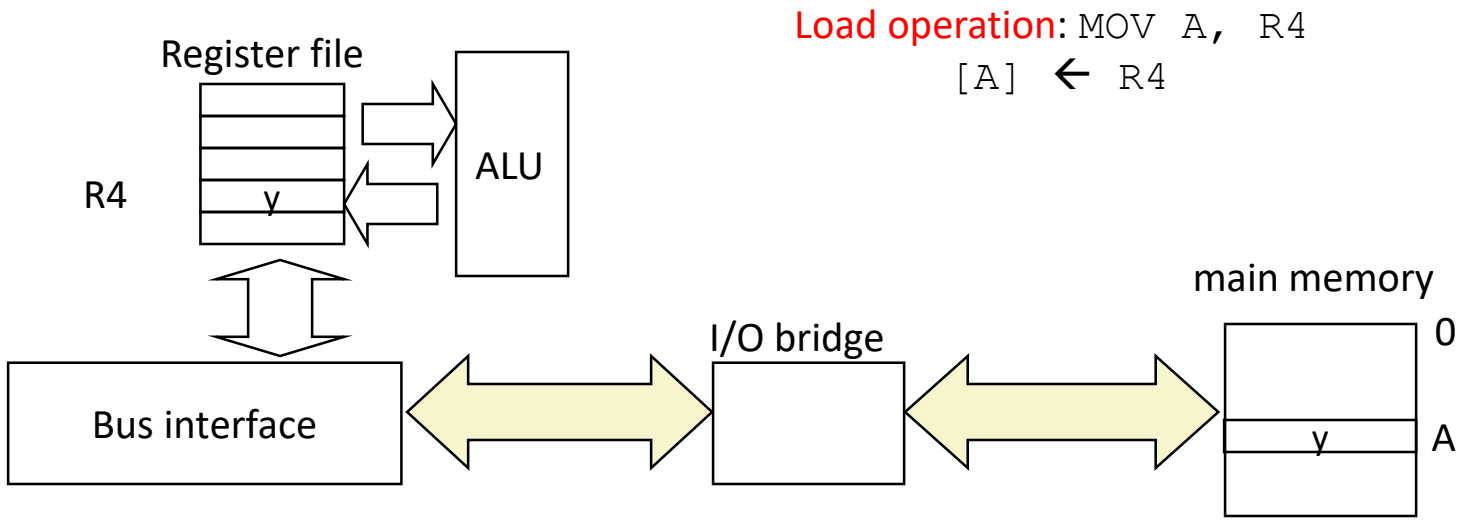
Memory Write Operation (2)

- CPU places data word y on the bus



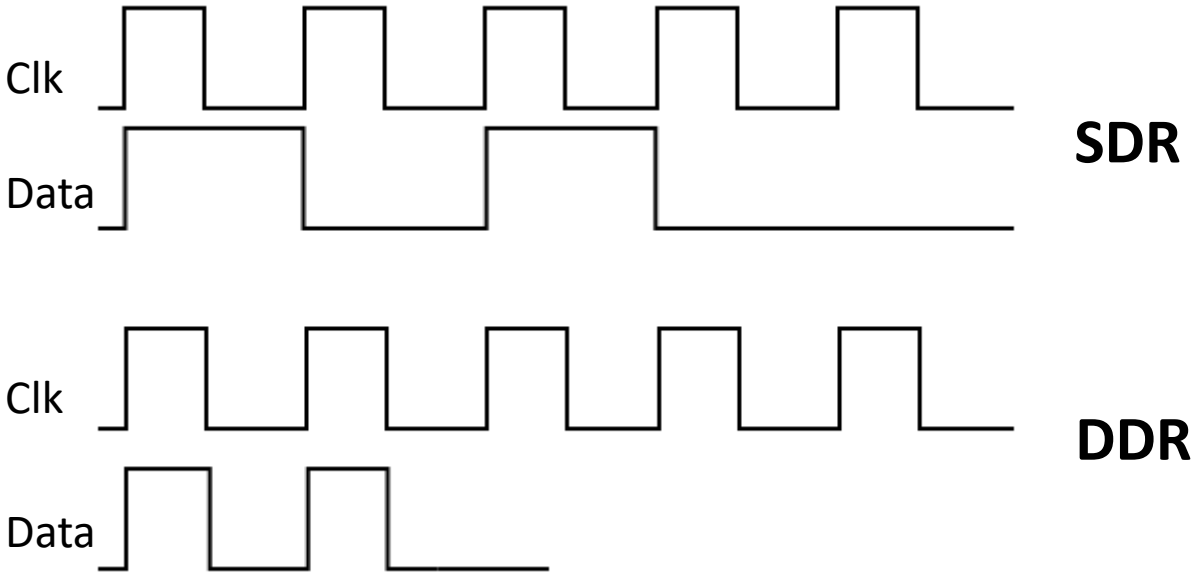
Memory Write Operation (3)

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



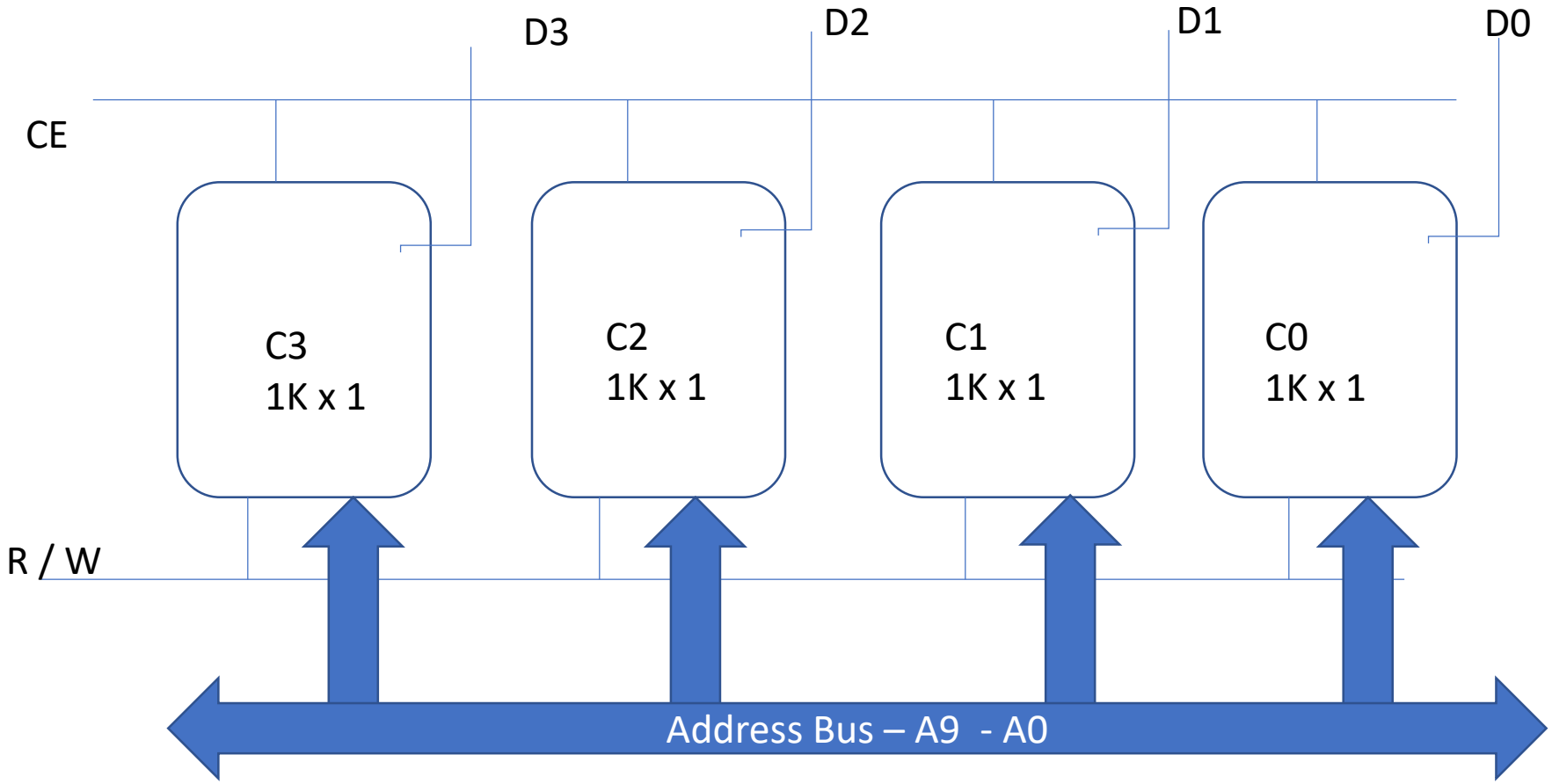
SDR Vs DDR

- SDR ➔ Single Data Rate
- DDR ➔ Double Data Rate



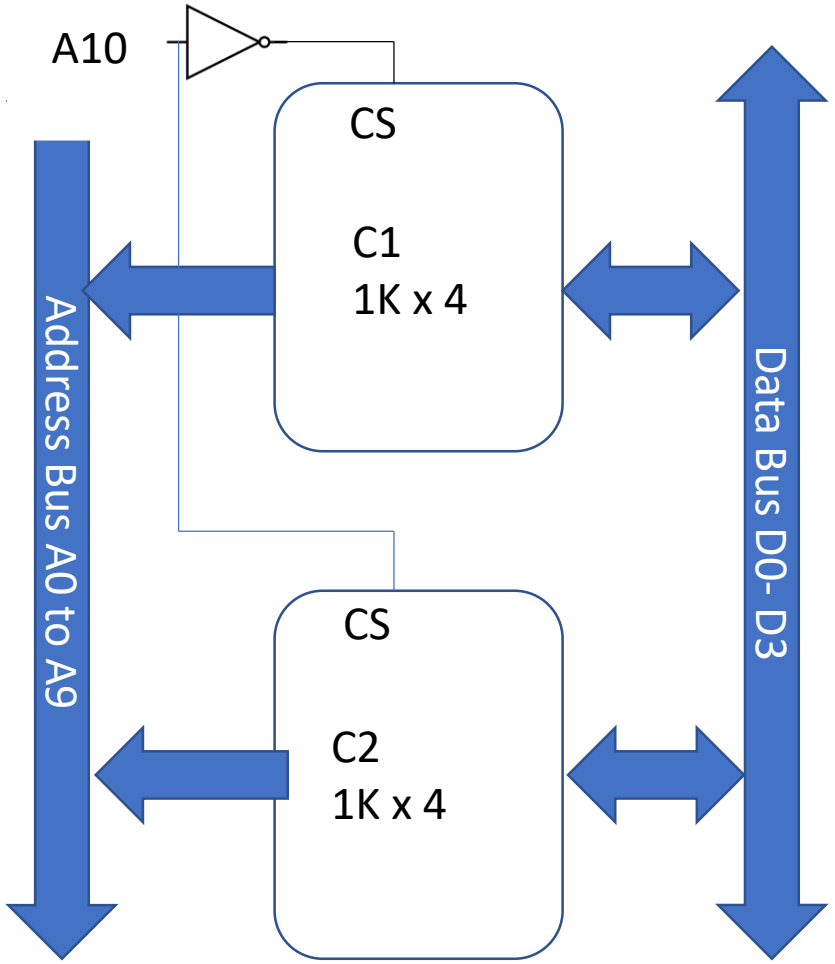
Typical Memory Connection Examples

- Construct 1K X4 bit memory using 1Kx1 bit chip



Typical Memory Connection Examples

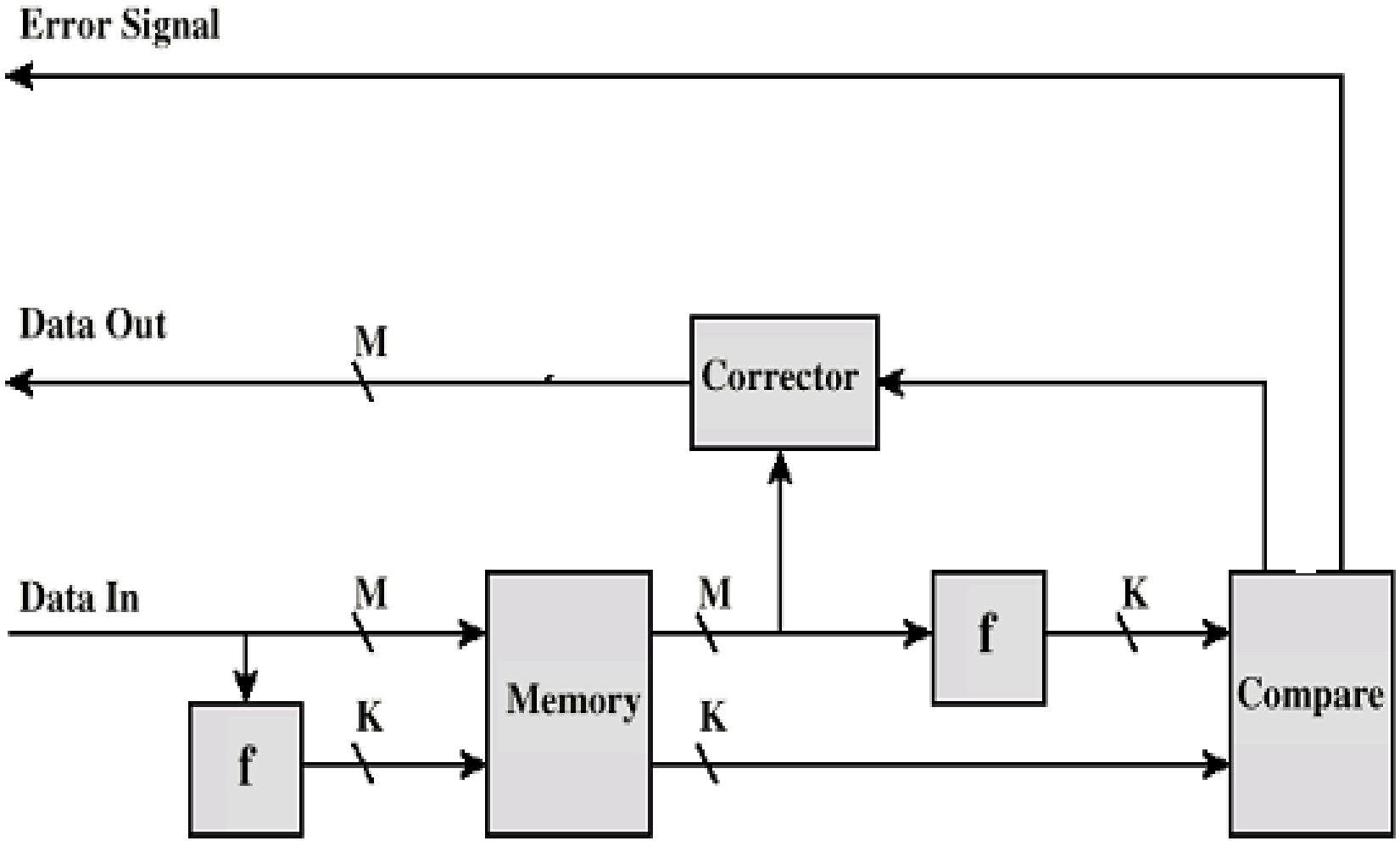
- Construct 2K X4 bit memory using 1Kx4 bit chip



Error Correction

- Hard Failure
  - Caused by harsh environmental abuse or manufacturing defects or wear
  - Memory cell is permanently stuck at 0 or 1
  - Permanent defect
- Soft Error
  - Random, non-destructive
  - alters the contents of one or more memory cells without damaging the memory.
  - No permanent damage to memory
  - Caused by power supply problems
- Detected using Hamming error correcting code

Error Correcting Code Function



Error Correcting Code Function

- The comparison logic receives as input two K-bit values. A bit-by-bit comparison is done by taking the exclusive-OR of the two inputs. The result is called the *syndrome word.*
- The comparison yields one of three results
  - No errors are detected.
  - An error is detected and it is possible to correct the error
  - An error is detected but it is not possible to correct it.

Hamming Code…..

- What should be the length of the code K ?
- Result of comparison is known as syndrome word
- length of the syndrome word is K bits, Length of Data is “M” bits
- Length of K should satisfy
- $2^k-1 >= M+K$

Hamming code….

- Generate 4-bit syndrome for an 8 bit data word with following characteristics
  - If the syndrome contains all 0’s, no error has been detected.
  - If the syndrome contains one and only one bit set to one, then an error has occurred in one of the 4 check bits. No correction is needed
  - If the syndrome contains more than one bit set to 1, then the numerical value of the syndrome indicates the position of the data bit in error. This data bit is inverted to correction

Layout of Data and Check bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

Layout of Data and Check bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

- C1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7
- C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7
- C3 = D2 ⊕ D3 ⊕ D4 ⊕ D8
- C4 = D5 ⊕ D6 ⊕ D7 ⊕ D8

Layout of Data and Check bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

- C1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7
- C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7
- C3 = D2 ⊕ D3 ⊕ D4 ⊕ D8
- C4 = D5 ⊕ D6 ⊕ D7 ⊕ D8

Problem 1

Consider the data + code k is as follows:

110101011101

Find out if there is an error. If so which bit is having error?

Problem 1 - Solution

- C1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7
- C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7
- C3 = D2 ⊕ D3 ⊕ D4 ⊕ D8
- C4 = D5 ⊕ D6 ⊕ D7 ⊕ D8

Consider the data + check bit is as follows:

110101011101

Find out if there is an error. If so which bit is having

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

Problem 2

- C1 = D1 ⊕ D2 ⊕ D4 ⊕ D5 ⊕ D7
- C2 = D1 ⊕ D3 ⊕ D4 ⊕ D6 ⊕ D7
- C3 = D2 ⊕ D3 ⊕ D4 ⊕ D8
- C4 = D5 ⊕ D6 ⊕ D7 ⊕ D8

Data : 10101100 (M = 8)

Compute Check Bits

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C4				C3		C2	C1

Computer Organization and Software Systems

Contact Session 3

Dr. Lucy J. Gudino

Last Class

Contact Hour	List of Topic Title	Text/Ref Book/external resource
3-4	Memory Organization - Internal Memory - External Memory (HDD)	T1, R2

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
5-6	**Memory Organization** - External Memory (RAID, SSD) Cache Memory Organization - Locality - Locality of Reference to Program Data - Locality of instruction fetches	T1, R2

Types of External memory

- Magnetic Disk
  - RAID Memories
  - Removable Disks
- Optical
  - CD-ROM
  - CD-Recordable (CD-R)
  - CD-R/W
  - DVD
- Magnetic Tape

Magnetic Disk Drive

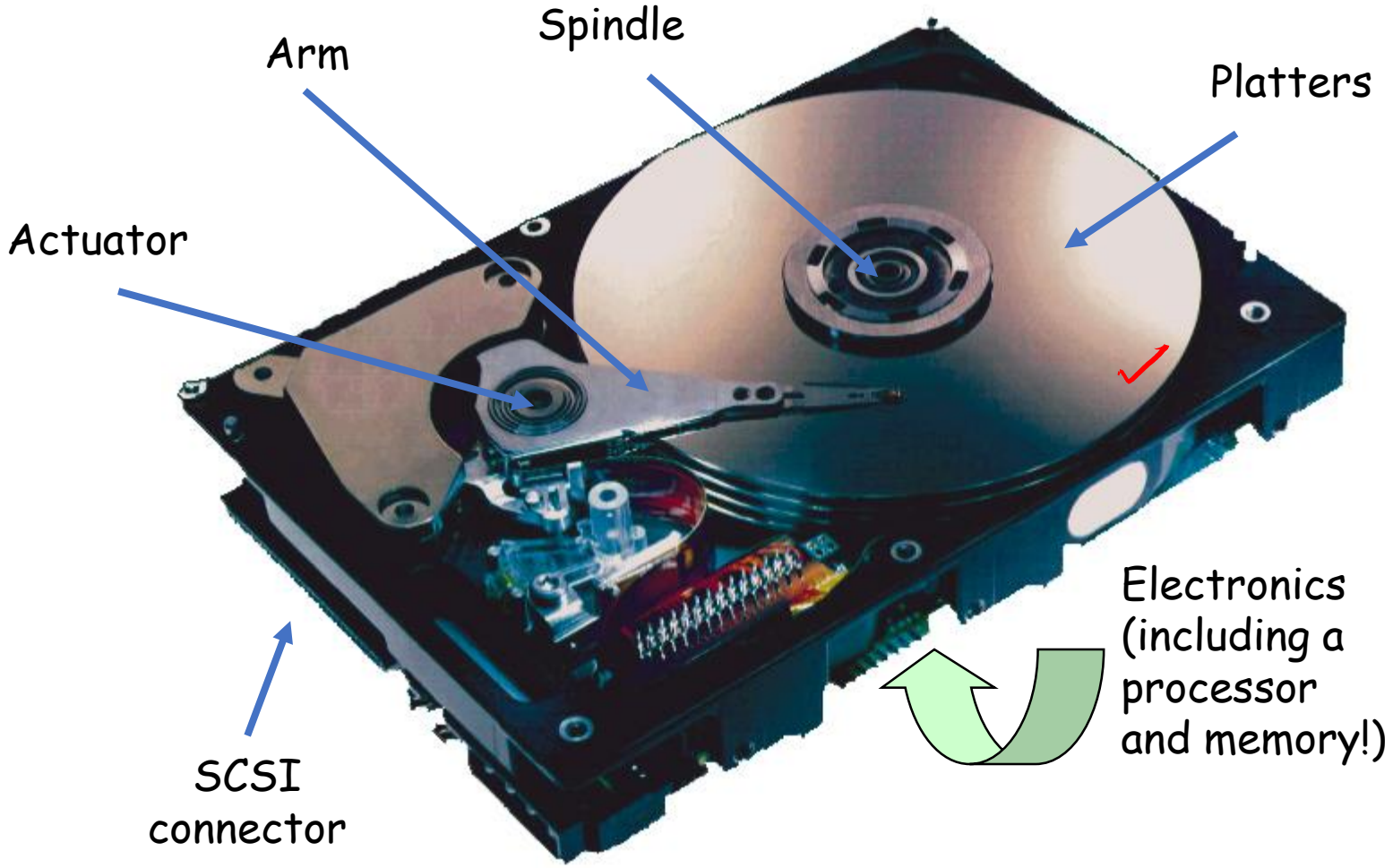
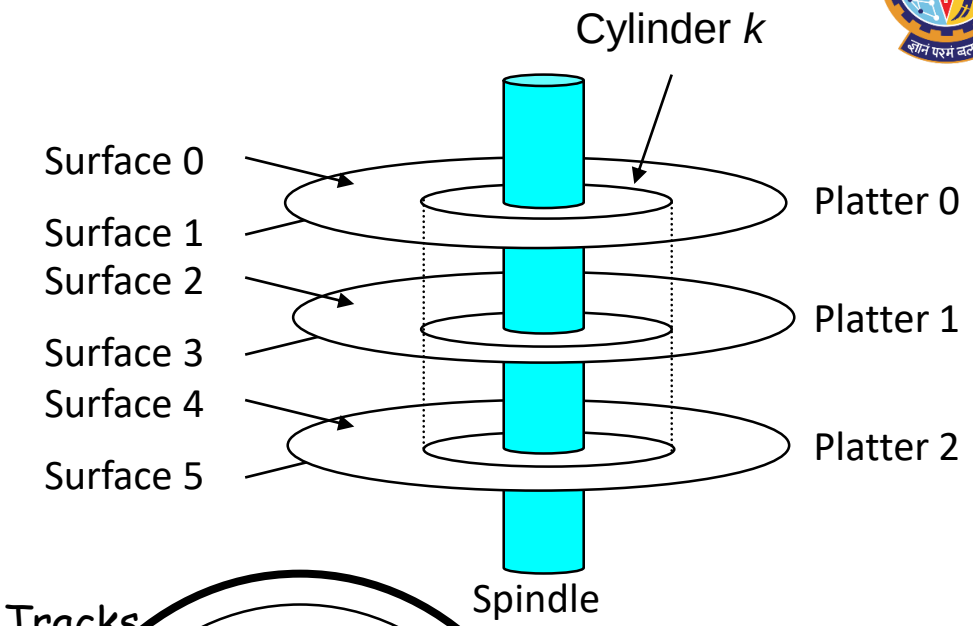
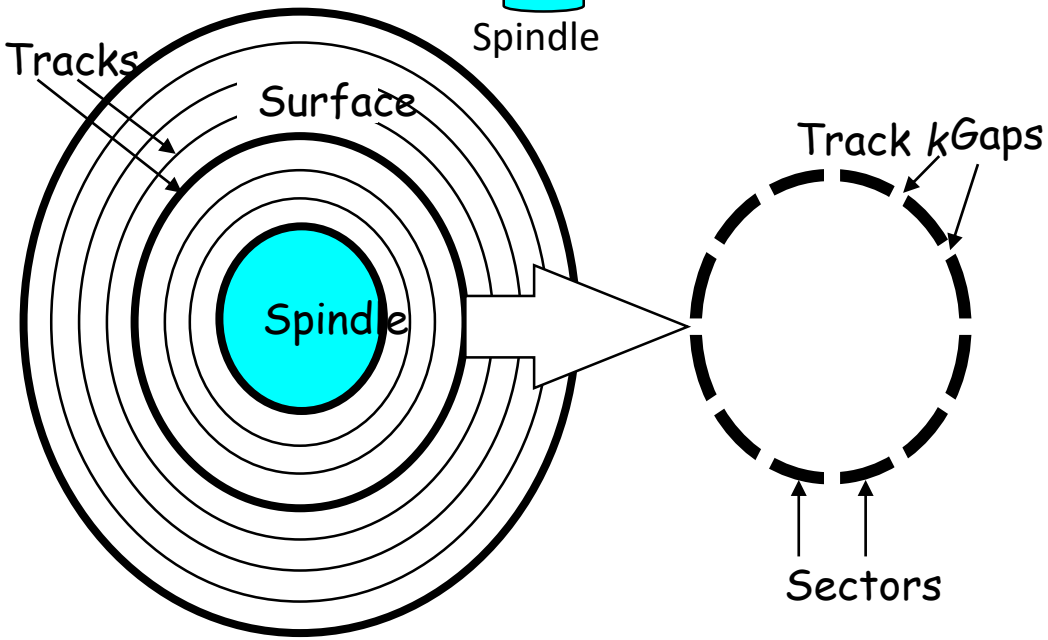


Image courtesy of Seagate Technology

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



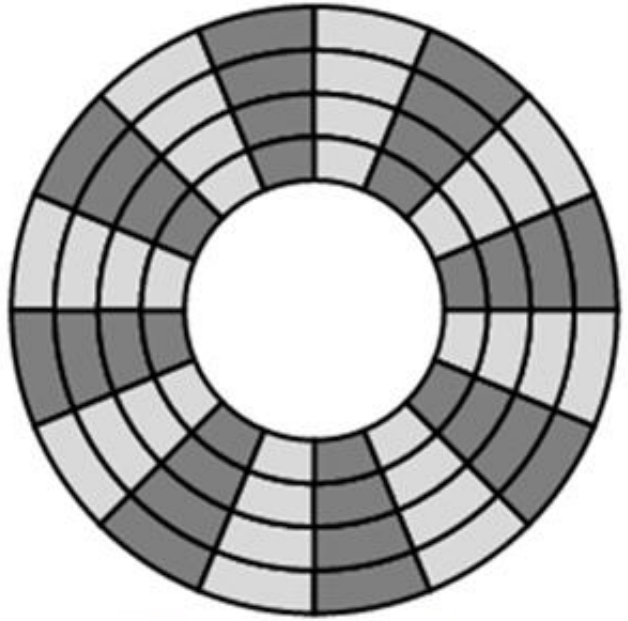


Disk Capacity

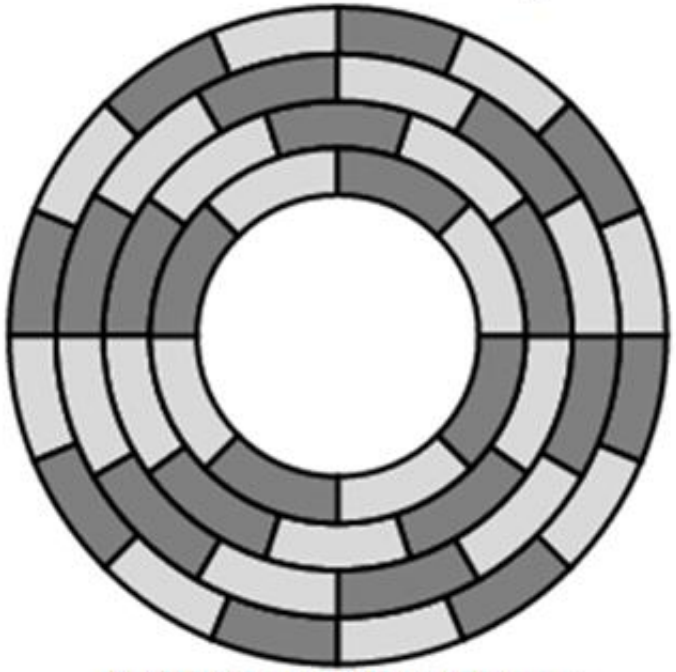
- Capacity: maximum number of bits that can be stored.
  - Vendors express capacity in units of gigabytes (GB /TB), where 1 GB = 2^{30} Bytes, 1 TB = 2^{40} Bytes,
- Capacity is determined by these technology factors:
  - Recording density (bits/in): number of bits that can be squeezed into a 1 inch segment of a track.
  - Track density (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment.
  - Areal density (bits/in2): 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

innovate achieve lead

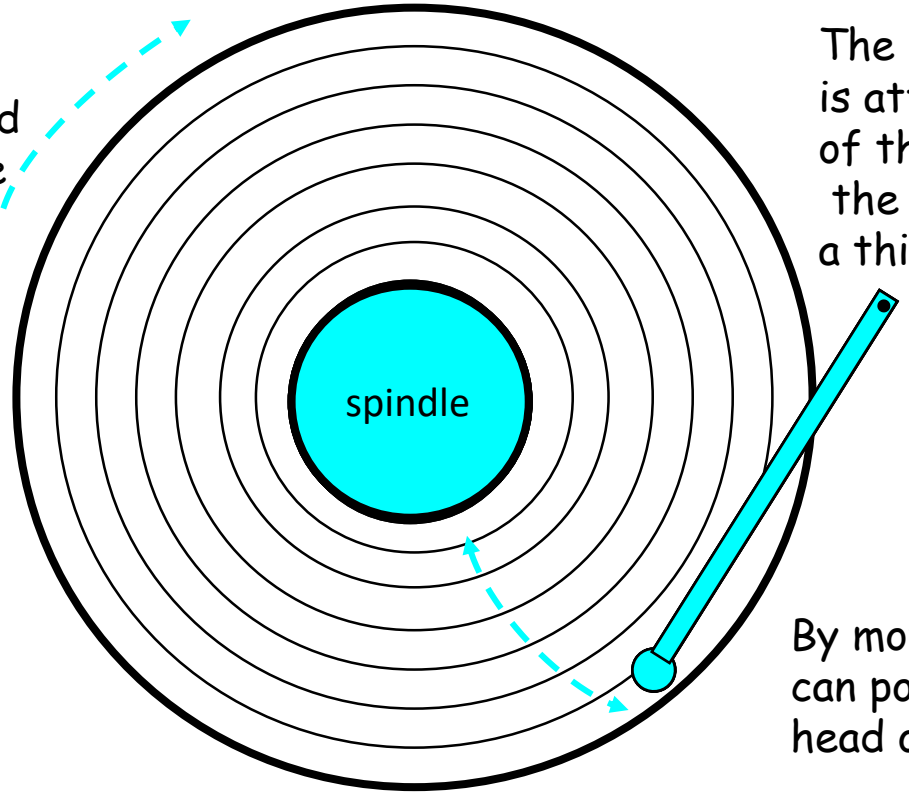
Computing Disk Capacity

- Capacity = (# bytes/sector) x (avg. # sectors/track) x
  (# tracks/surface) x (# surfaces/platter) x
  (# platters/disk)
- Example:
  - 512 bytes/sector
  - 300 sectors/track (on average)
  - 20,000 tracks/surface
  - 2 surfaces/platter
  - 5 platters/disk

- Capacity = 512 x 300 x 20000 x 2 x 5
  = 30,720,000,000
  = 28.61 GB

Disk Operation (Single-Platter View)

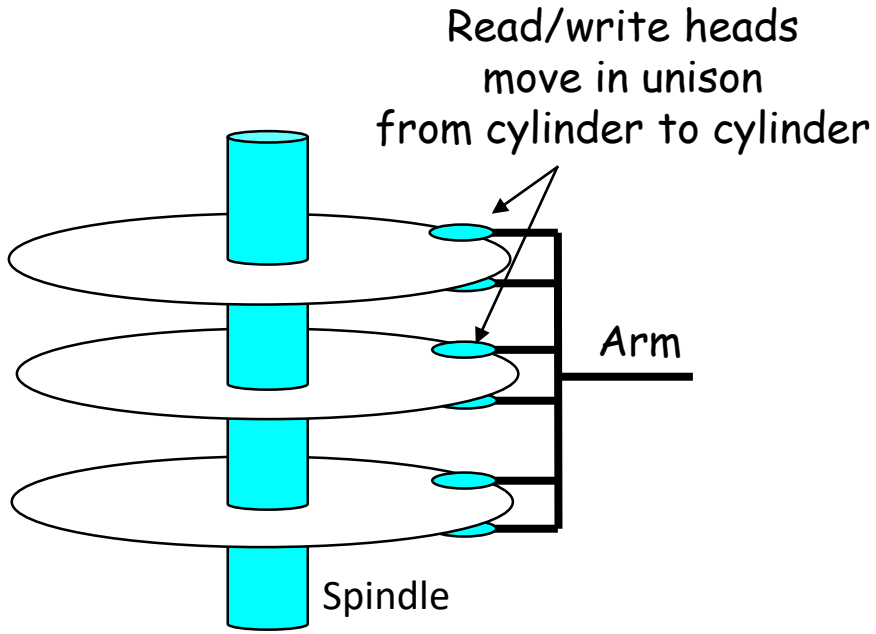
The disk surface spins at a fixed rotational rate



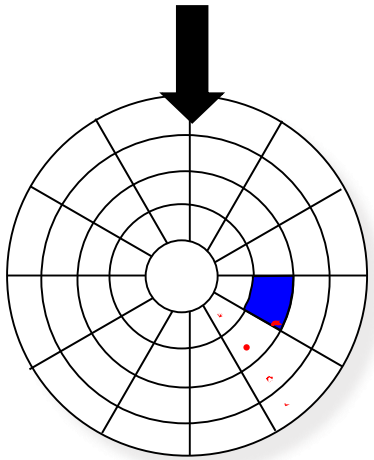
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air.

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

Disk Operation (Multi-Platter View)

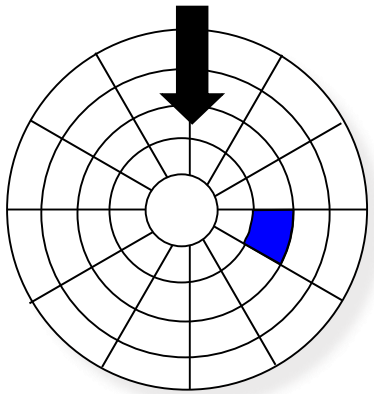


Disk Access



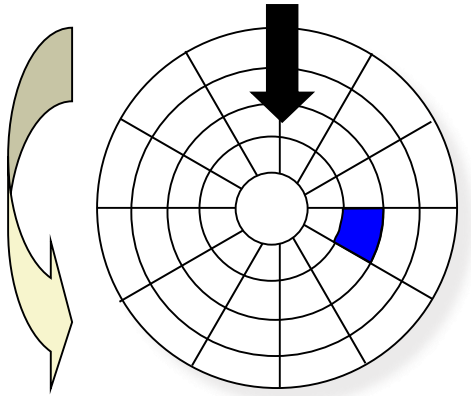
Need to access a sector colored in blue

Disk Access



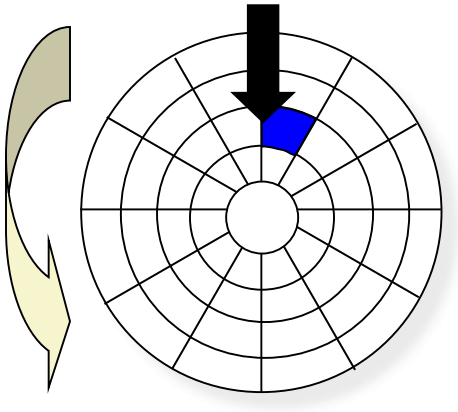
Head in position above a track

Disk Access



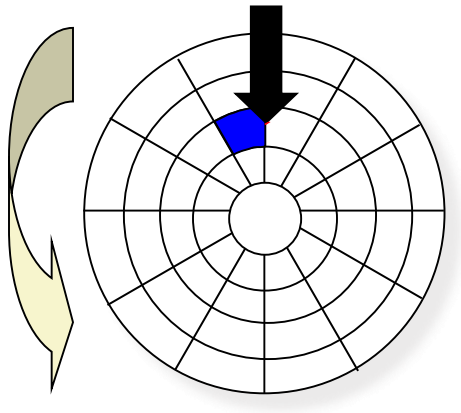
Rotate the platter in counter-clockwise direction

Disk Access - Read



About to read blue sector

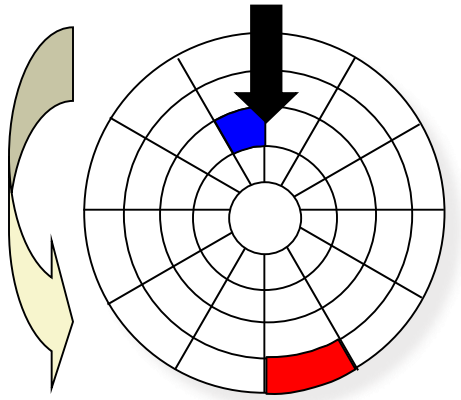
Disk Access - Read



After BLUE read

After reading blue sector

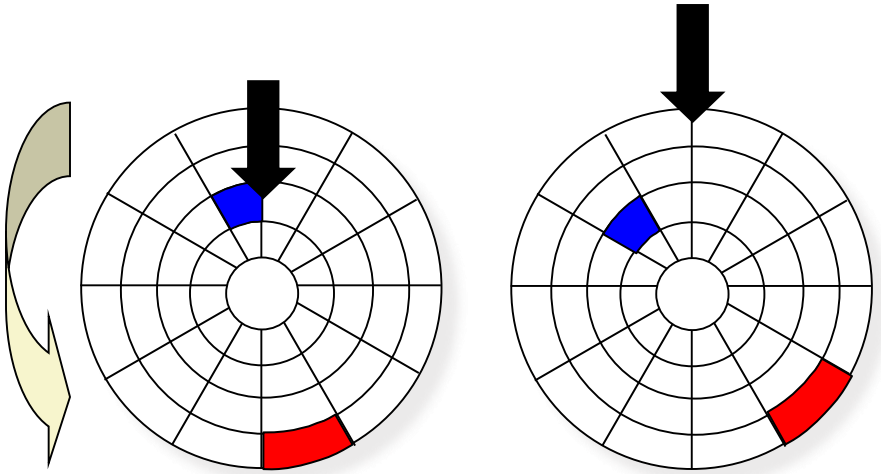
Disk Access - Read



After BLUE read

Red request scheduled next

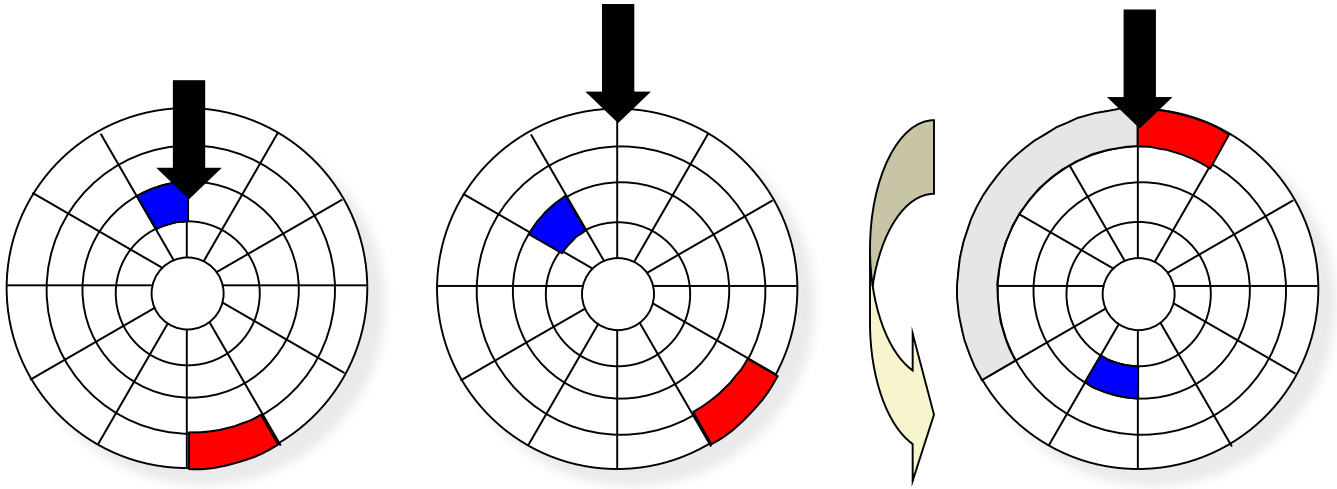
Disk Access - Seek



After BLUE read

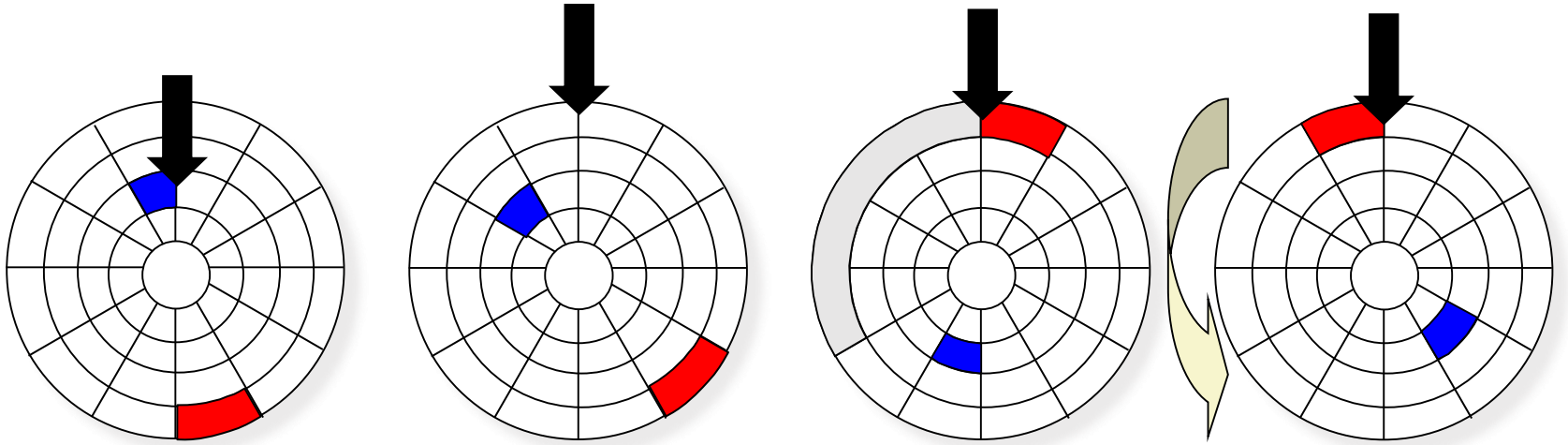
Seek for RED

Disk Access – Rotational Latency



Wait for red sector to rotate around

Disk Access – Read



After BLUE read

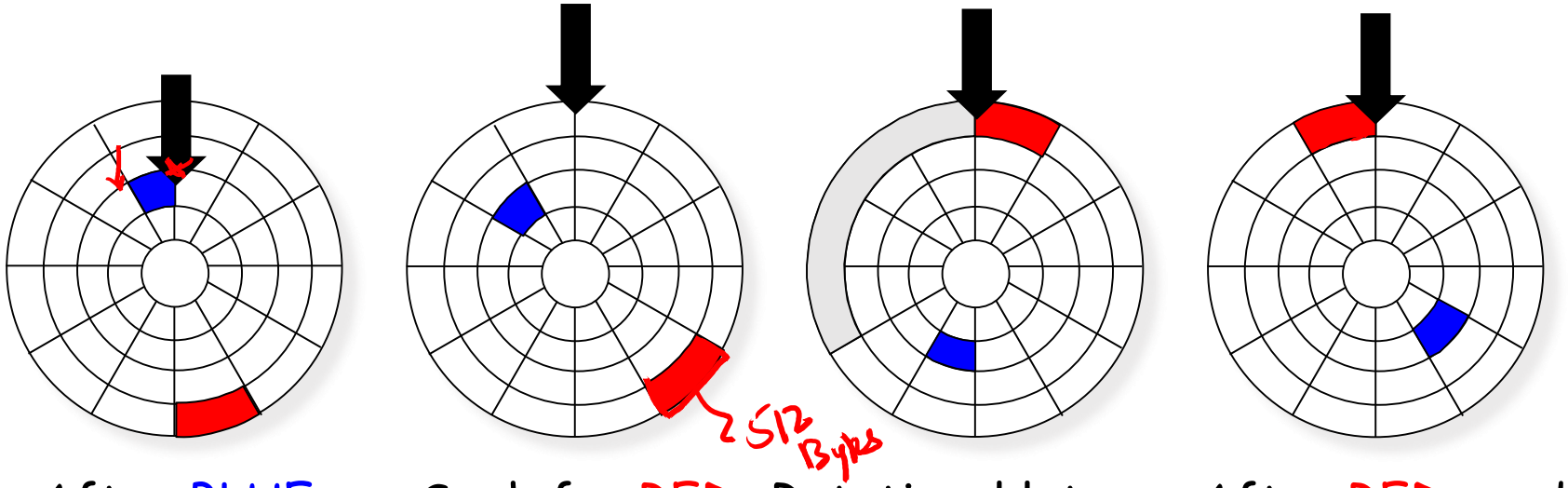
Seek for RED

Rotational latency

After RED read

Complete read of red

Disk Access – Access Time Components



Today's Class

Contact Hour	List of Topic Title	Text/Ref Book/external resource
5-6	**Memory Organization** - External Memory (RAID, SSD) Cache Memory Organization - Locality - Locality of Reference to Program Data - Locality of instruction fetches	T1, R2

RAID

- RAID - Redundant Array of Independent Disks
- Variety of ways in which the data can be organized
- Need for RAID:
  - Parallel I/O
  - Reliability
  - Redundancy through multiple inexpensive disks

Common Characteristics

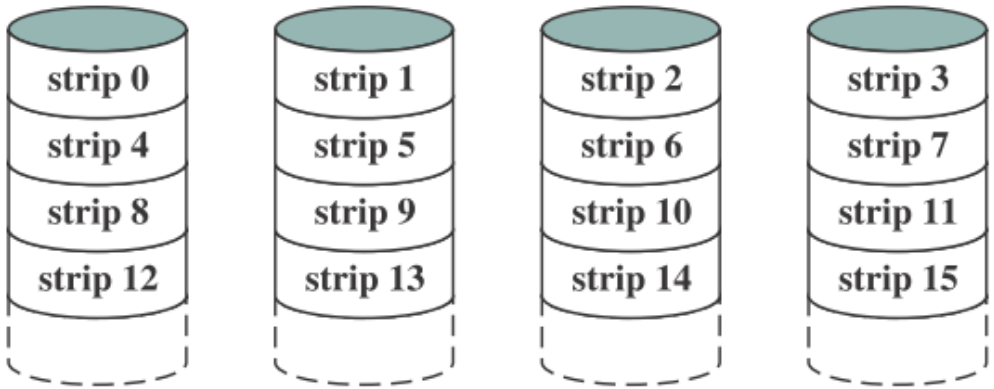
1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Data are distributed across the physical drives of an array in a scheme known as striping.
3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

Categories

- RAID category
  - Striping (Level 0)
  - Mirroring (Level 1)
  - Parallel access (Level 2,3)
  - Independent access (Level 4,5,6)

RAID Level 0

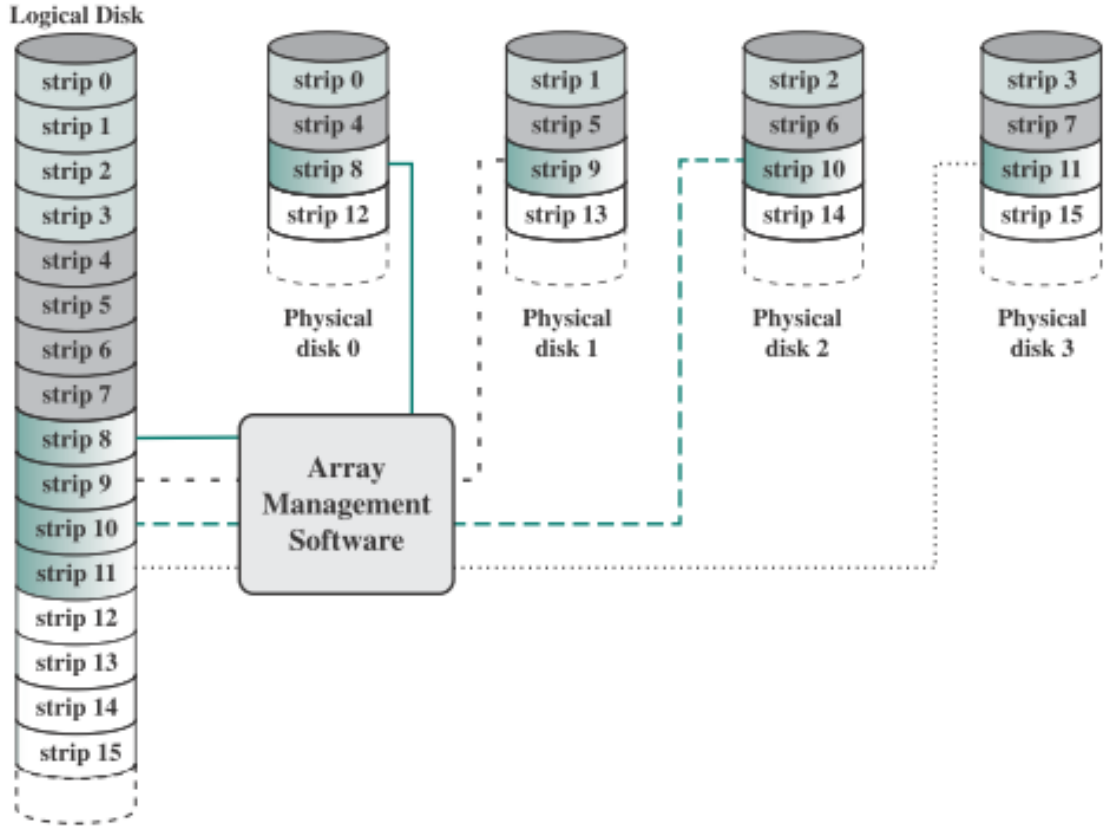
- Is not a true member of RAID family – No Redundancy
- The data are striped across the available disks
- Data are distributed across all of the disks in the array
- IO requests can be issued in Parallel
- Reduce IO queuing time



(a) RAID 0 (Nonredundant)

RAID level 0 configuration

- RAID 0 for
  - high data transfer capacity
  - high I/O request rate

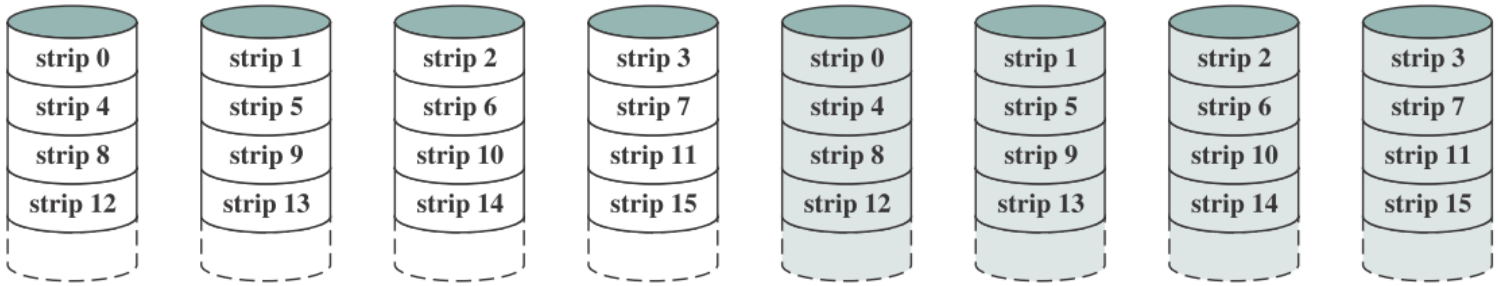


RAID Level 0

Level	Advantages	Disadvantages	Applications
0	• Very simple design • Easy to implement • I/O performance is greatly improved by spreading the I/O load across many drives • No parity calculation overhead is involved	The failure of just one drive will result in all data in an array being lost	• Video production and editing • Image Editing • Any application requiring high bandwidth

RAID Level 1

- Mirroring: redundancy is achieved by duplicating all the data
- Each logical strip is mapped to two separate physical disks
- Every disk in the array has a mirror disk that contains the same data



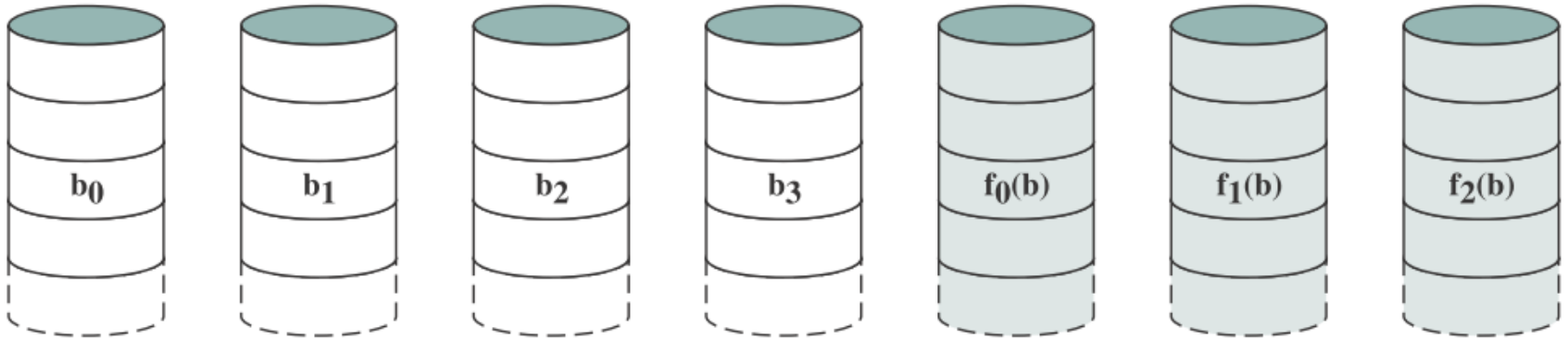
(b) RAID 1 (Mirrored)

RAID Level 1

Level	Advantages	Disadvantages	Applications
1	• Simplest RAID storage subsystem design • 100% redundancy of data • RAID 1 can sustain multiple simultaneous drive failures	• Highest disk overhead of all RAID types • inefficient	• Accounting • Payroll • Financial • Any application requiring very high availability

RAID Level 2

- Use parallel access technique
- In a parallel access array, all member disks participate in the execution of every I/O request
- Spindles of the individual drives are synchronized



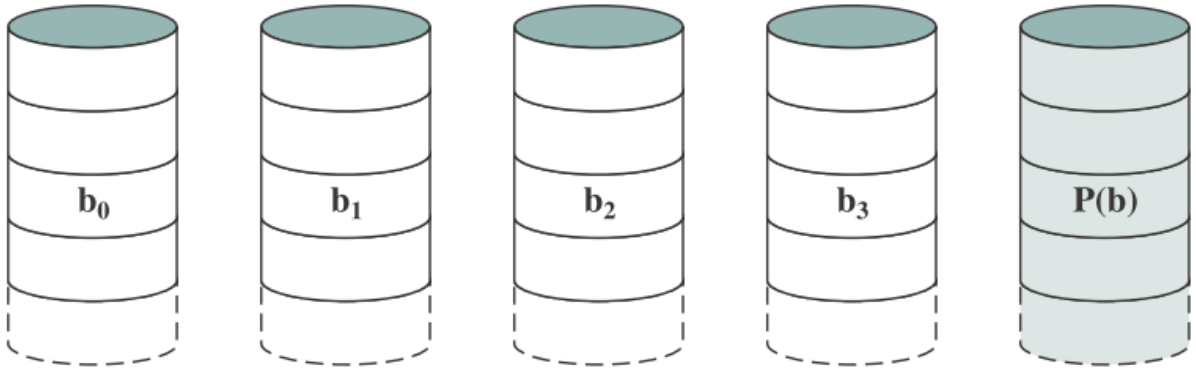
(c) RAID 2 (Redundancy through Hamming code)

RAID Level 2

Level	Advantages	Disadvantages	Applications
2	• Extremely high data transfer rates possible • The higher the data transfer rate required, the better the ratio of data disks to ECC disks • Relatively simple controller design compared to RAID levels 3, 4, & 5	• Very high ratio of ECC disks to data disks with smaller word sizes— inefficient • Entry level cost very high— requires very high transfer rate requirement to justify	• No commercial implementations exist/not commercially viable

RAID Level 3

- RAID 3 is organized in a similar fashion to RAID 2
- RAID 3 requires only a single redundant disk
- RAID 3 employs parallel access, with data distributed in small strips
- Parity bit is computed for the set of individual bits in the same position on all of the data disks



(d) RAID 3 (Bit-interleaved parity)

RAID Level 3

- Redundancy: Event of a drive failure - parity drive is accessed and data is reconstructed from the remaining devices
    - Missing data can be restored on the new drive
- Data reconstruction is simple

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

- Suppose drive X1 fails

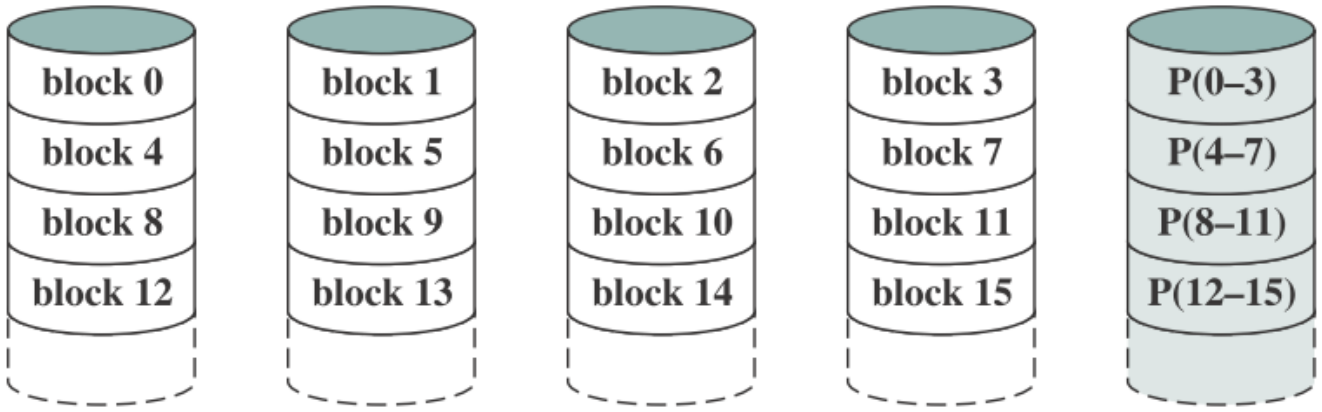
$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

RAID Level 3

Level	Advantages	Disadvantages	Applications
3	• Very high read data transfer rate • Very high write data transfer rate • Disk failure has an insignificant impact on throughput • Low ratio of ECC (parity) disks to data disks means high efficiency	• In case of small size files it performs slowly. • Controller design is fairly Complex	• Video production and live streaming • Image editing • Video editing • Prepress applications • Any application requiring high throughput

RAID Level 4

- RAID levels 4 through 6 make use of an independent access technique
- Each member disk operates independently, so that separate I/O requests run in parallel



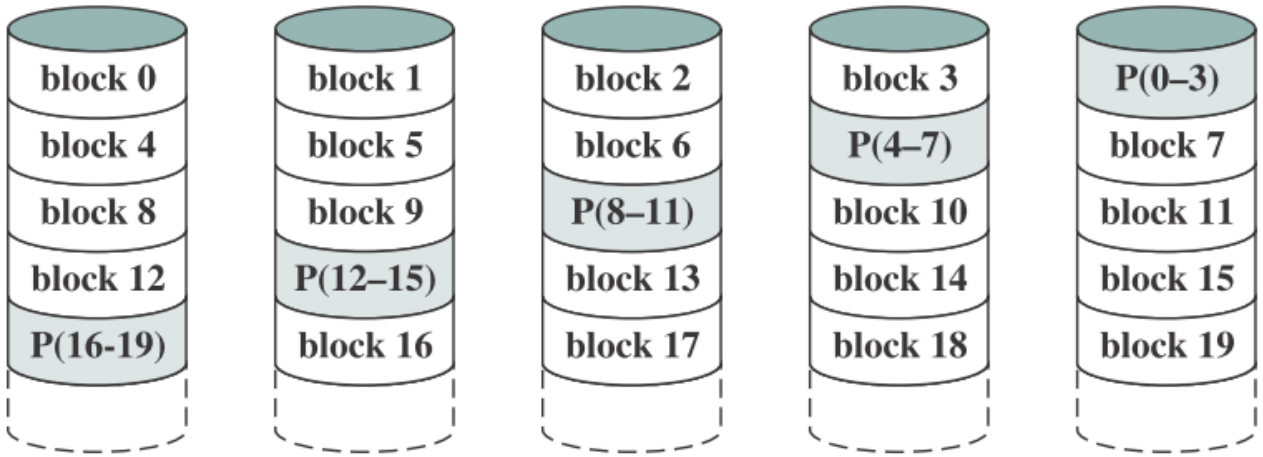
(e) RAID 4 (Block-level parity)

RAID Level 4

Level	Advantages	Disadvantages	Applications
4	• Low ratio of ECC (parity) disks to data disks means high efficiency • Very high Read data transaction rate	• Worst write transaction rate and Write aggregate transfer rate • Quite complex controller design • Difficult and inefficient data rebuild in the event of disk failure	• No commercial implementations exist/not commercially viable

RAID Level 5

- RAID 5 is organized in a similar fashion to RAID 4.
- The difference is that RAID 5 distributes the parity strips across all disks



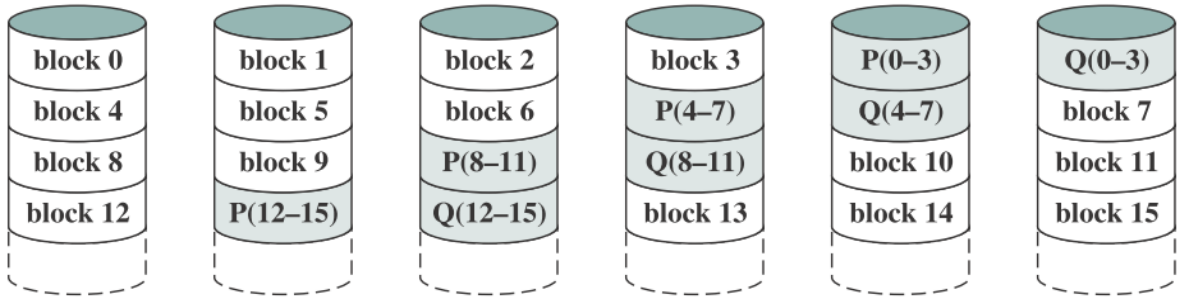
(f) RAID 5 (Block-level distributed parity)

RAID Level 5

Level	Advantages	Disadvantages	Applications
5	• Highest Read data transaction rate • Low ratio of ECC (parity) disks to data disks means high efficiency • Better reliability	• Most complex controller design • Difficult to rebuild in the event of a disk failure (as compared to RAID level 1)	• File and application servers • Database servers • Web, e- mail, and news servers • Intranet servers

RAID Level 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- RAID 6 array whose user data require N disks consists of N + 2 disks
- One of the two is the exclusive- OR calculation used in RAID 4 and 5
- Other is an independent data check algorithm
- Possible to regenerate data even if two disks containing user data fail



(g) RAID 6 (Dual redundancy)

RAID Level 6

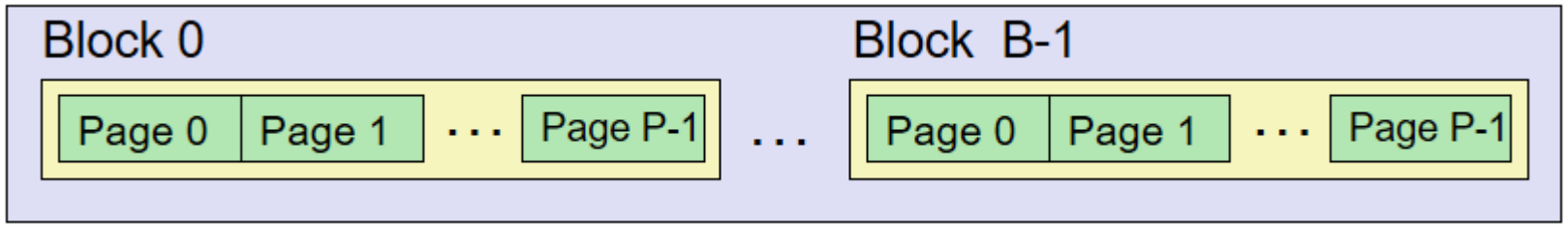
Level	Advantages	Disadvantages	Applications
6	• Provides for an extremely high data fault tolerance and can sustain multiple simultaneous drive failures	• More complex controller design • Controller overhead to compute parity is extremely high	• Perfect solution for mission critical applications

Solid state drives

- SSDs have the following advantages over HDDs:
  - High- performance input/output operations per second (IOPS): Significantly increases performance I/O subsystems.
  - Lower access times and latency rates: Over 10 times faster than the spinning disks in an HDD.
  - Lower power consumption: SSDs use considerably less power than comparable- size HDDs.
  - Durability: Less susceptible to physical shock and vibration.
  - Longer lifespan: SSDs are not susceptible to mechanical wear.
  - Quieter and cooler running capabilities: Less space required, lower energy costs, and a greener enterprise.

Solid State Disks (SSDs)

- SSDs contain Blocks and Pages
- Pages: 512KB 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 Organization

- SSD contains the following components:
  - Controller: Provides SSD device level interfacing and firmware execution.
  - Addressing: Logic that performs the selection function across the flash memory components.
  - Data buffer/cache: High speed RAM memory components used for speed matching and to increased data throughput.
  - Error correction: Logic for error detection and correction.
  - Flash memory components: Individual NAND flash chips.

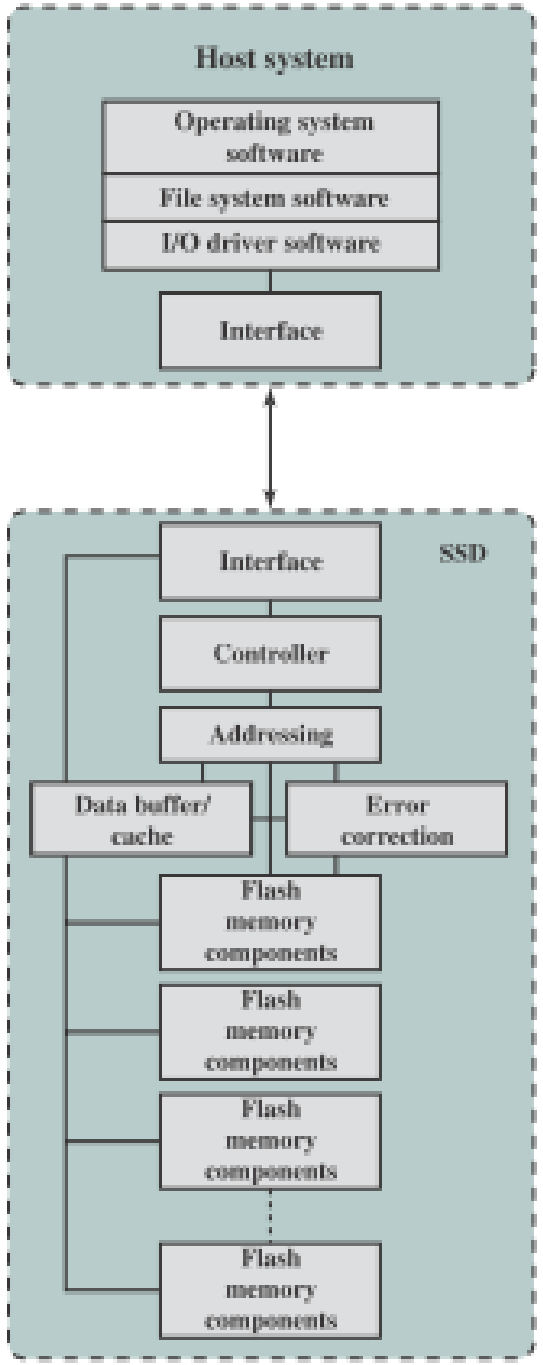


Figure 6.8 Solid State Drive Architecture

Issues with SSD

- Two practical issues:
  - SSD performance has a tendency to slow down as the device is used
  - Memory becomes unusable after a certain number of writes

First Issue

- Files are stored on disk as a set of pages, typically 4 KB in length
- Pages are not necessarily stored as a contiguous set of pages on the disk
- Flash memory is accessed in blocks, block size of 512 KB, (128 pages per block)
- Write a page onto a flash memory:

  1. The entire block must be read from the flash memory and placed in a RAM buffer. Then the appropriate page in the RAM buffer is updated.

  2. Before the block can be written back to flash memory, the entire block of flash memory must be erased— it is not possible to erase just one page of the flash memory.

  3. The entire block from the buffer is now written back to the flash memory.

Second Issue

- Flash memory becomes unusable after a certain number of writes.
- Typical limit is 100,000 writes

Comparison of Solid State Drives and Disk Drives

	NAND Flash Drives	Seagate Laptop Internal HDD
File copy/write speed	200–550 Mbps	50–120 Mbps
Power draw/battery life	Less power draw, averages 2–3 watts, resulting in 30+ minute battery boost	More power draw, averages 6–7 watts and therefore uses more battery
Storage capacity	Typically not larger than 512 GB for notebook size drives; 1 TB max for Desktops	Typically around 500 GB and 2 TB max for notebook size drives; 4 TB max for desktops
Cost	Approx. $0.50 per GB for a 1-TB drive	Approx. $0.15 per GB for a 4-TB drive

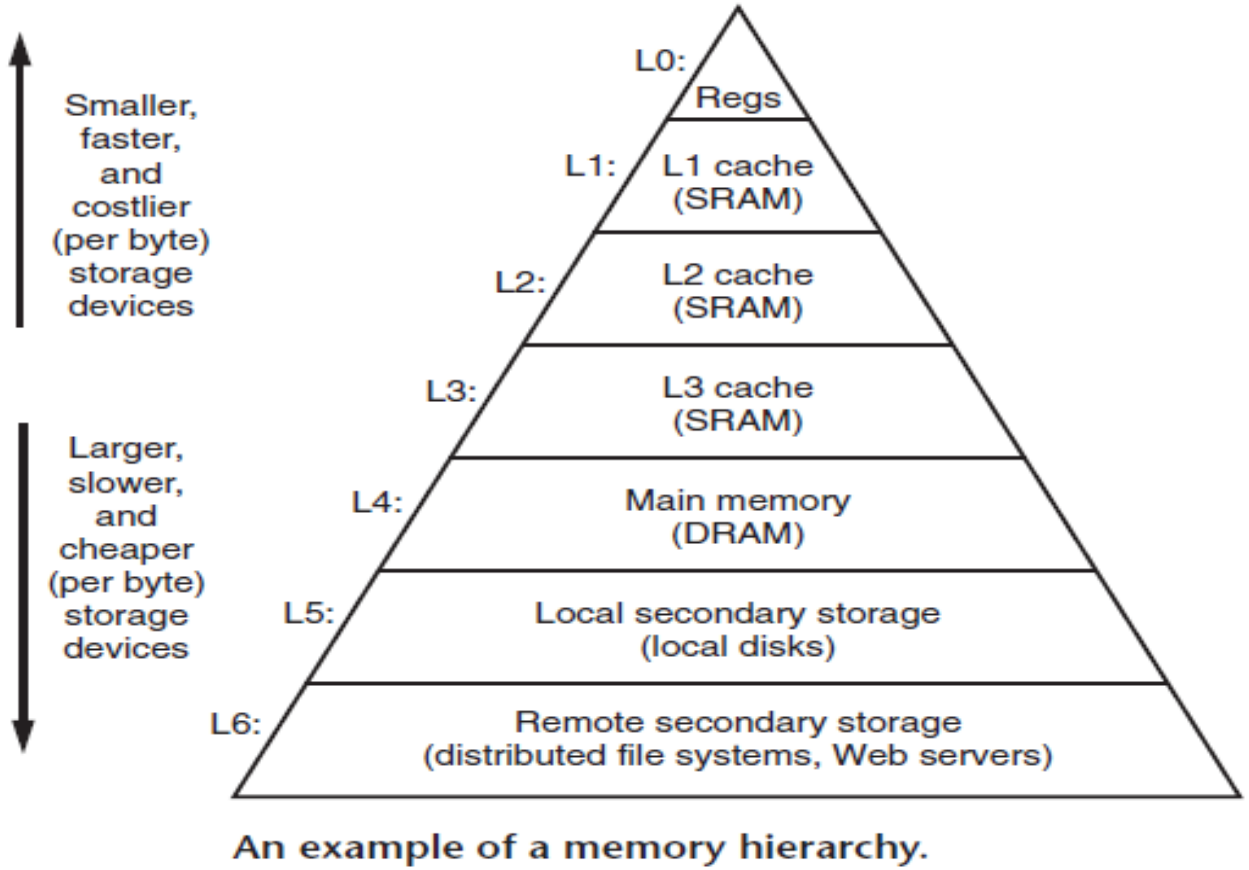
Computer Organization and Software Systems

Contact Session 4

Dr. Lucy J. Gudino

The Bottom Line

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



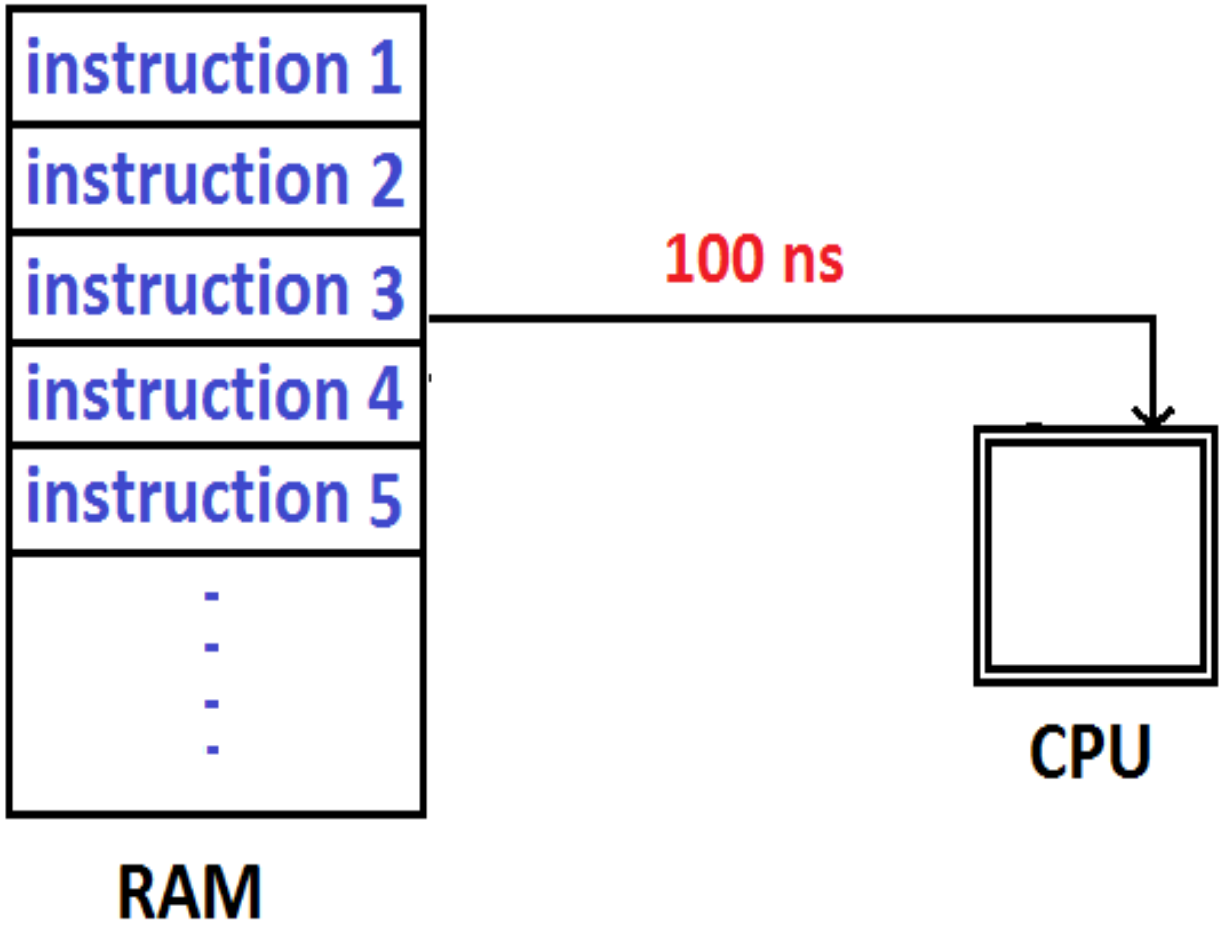
An example of a memory hierarchy.

- 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

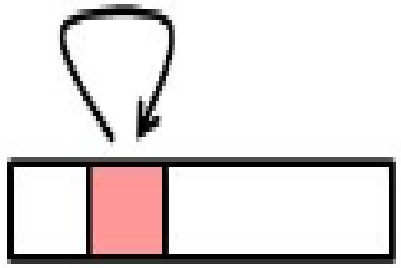


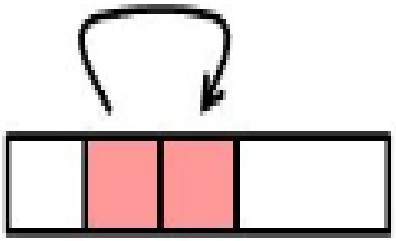
Performance enhancement -Motivation

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.



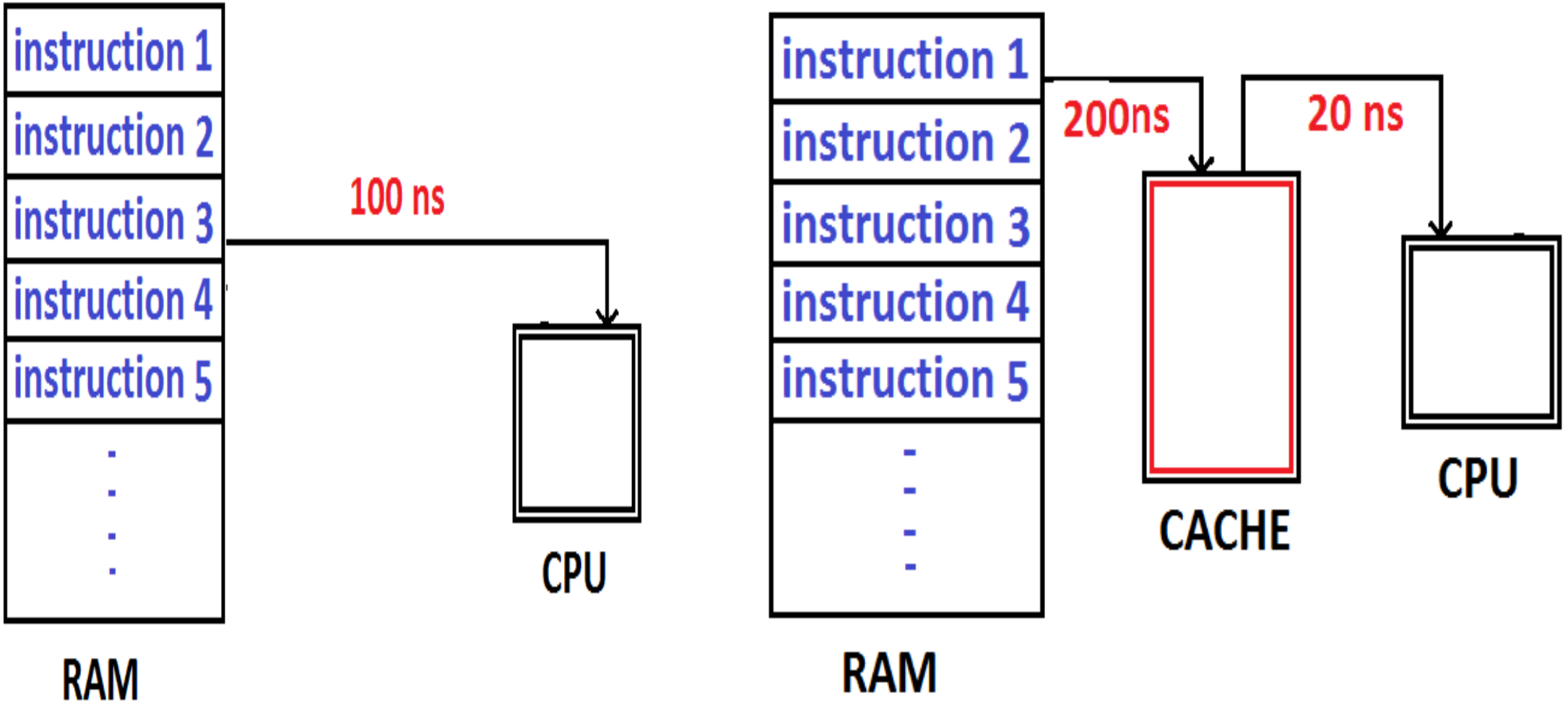


Example

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

- Data :
  - Access array elements in succession – spatial locality
  - Reference to "product" "i" and "n" in each iteration – Temporal locality
- Instructions :
  - Reference instructions in sequence : Spatial locality
  - Looping through : Temporal locality

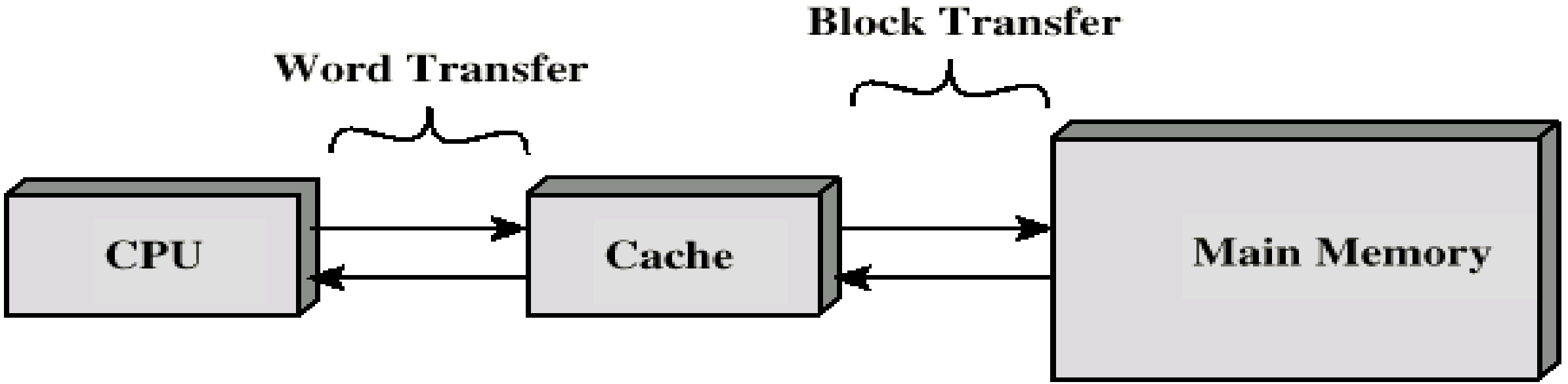
Performance enhancement -Motivation



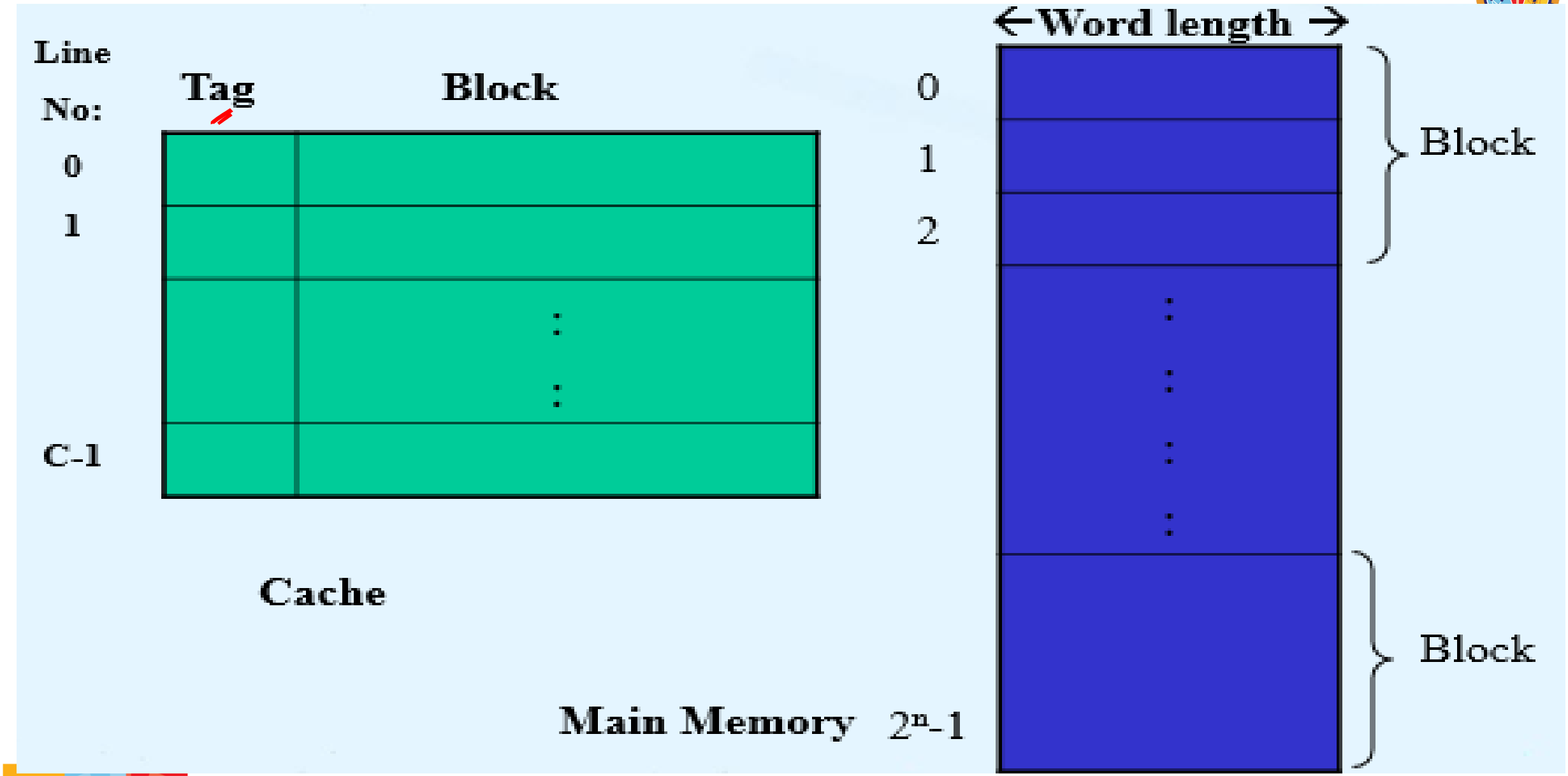
Cache

Cache

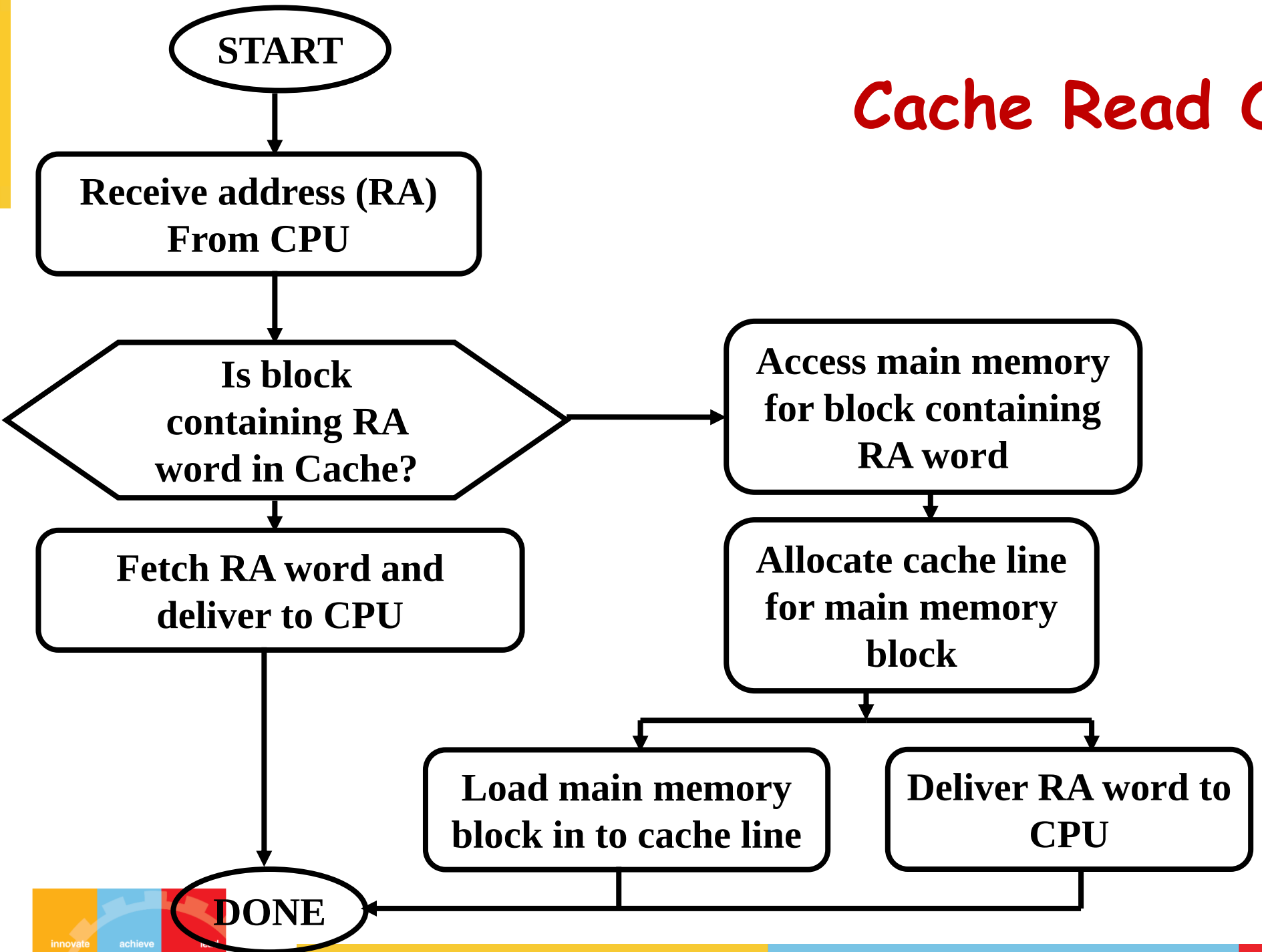
- Small, fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or separate module



Cache and Main Memory Structure



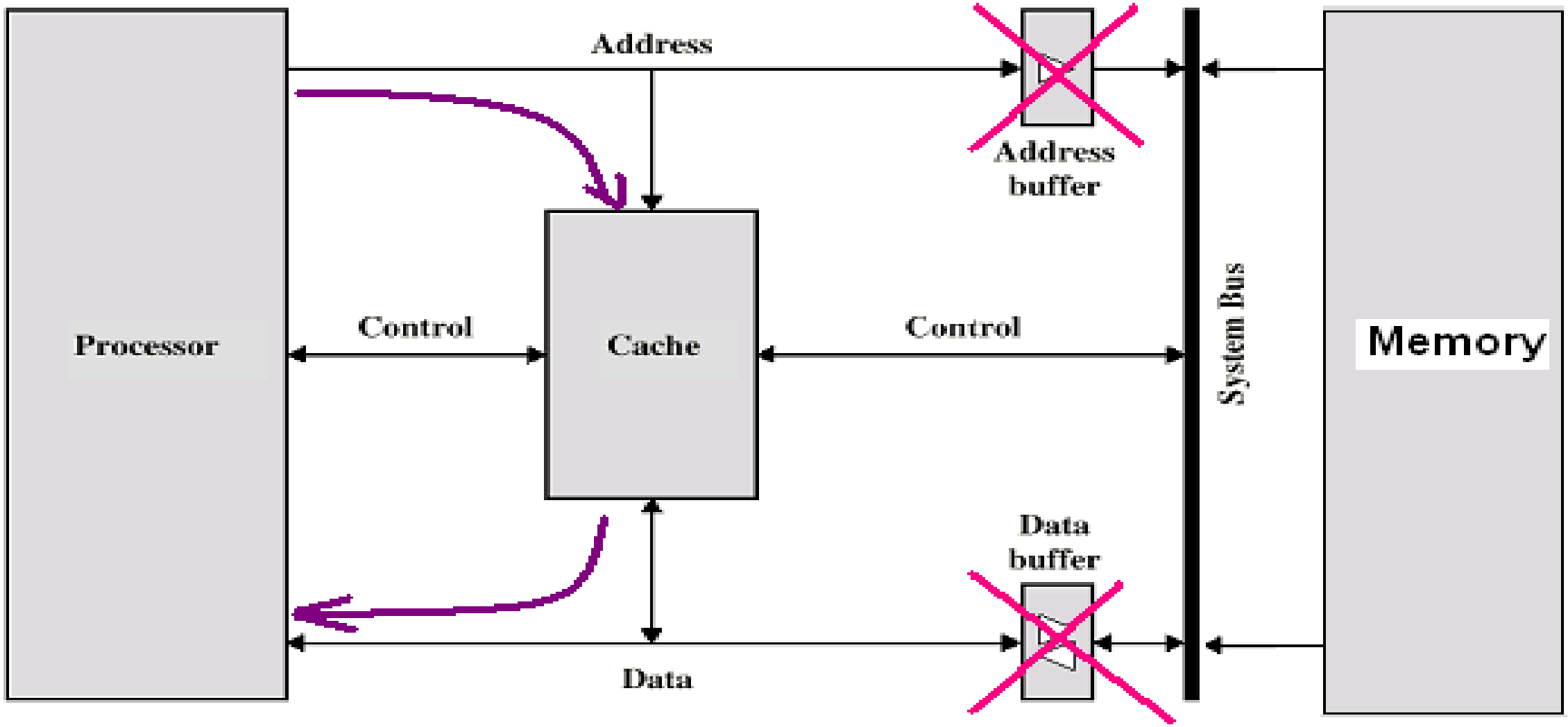
Cache Read Operation



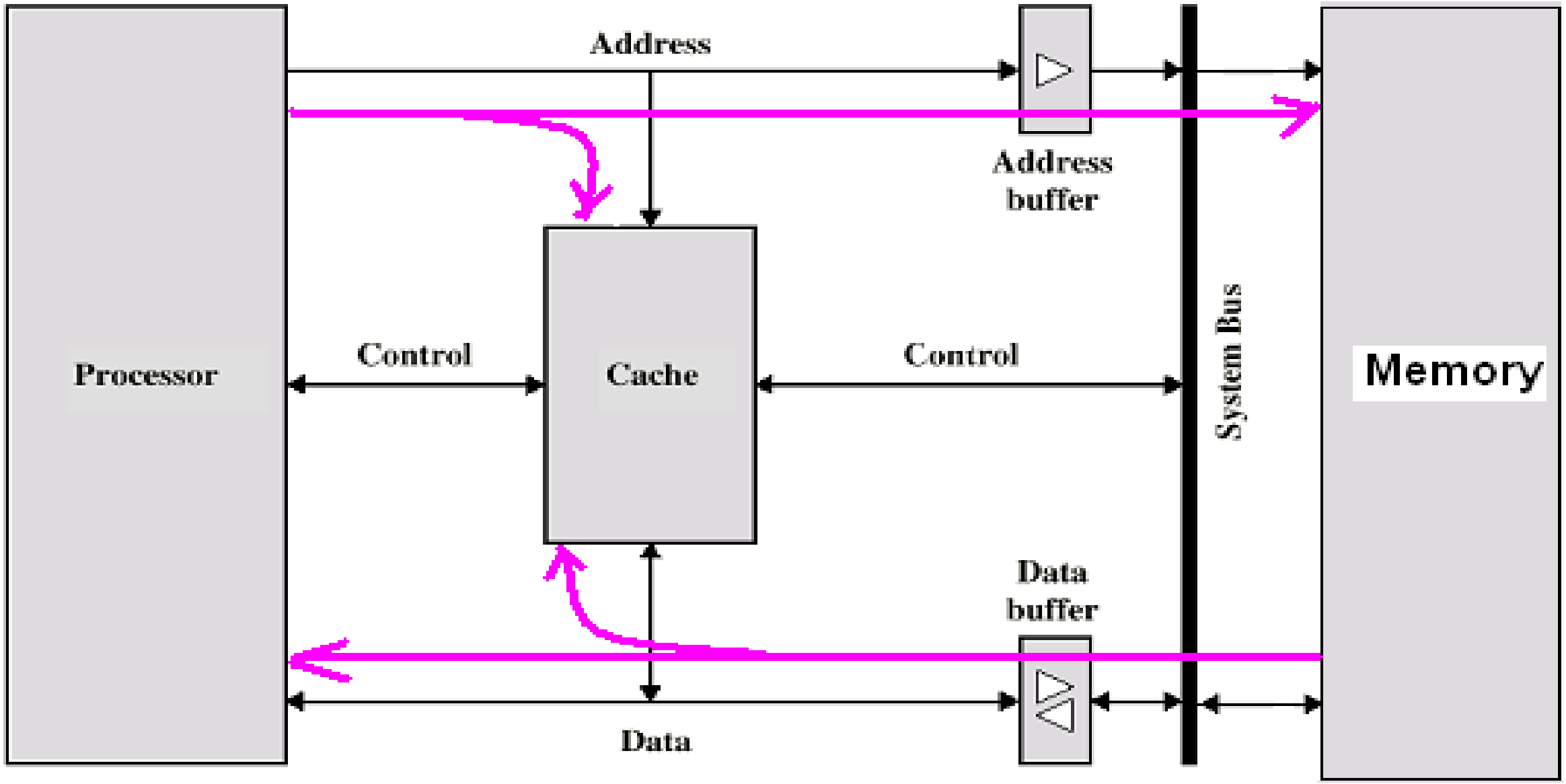
Performance of cache

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

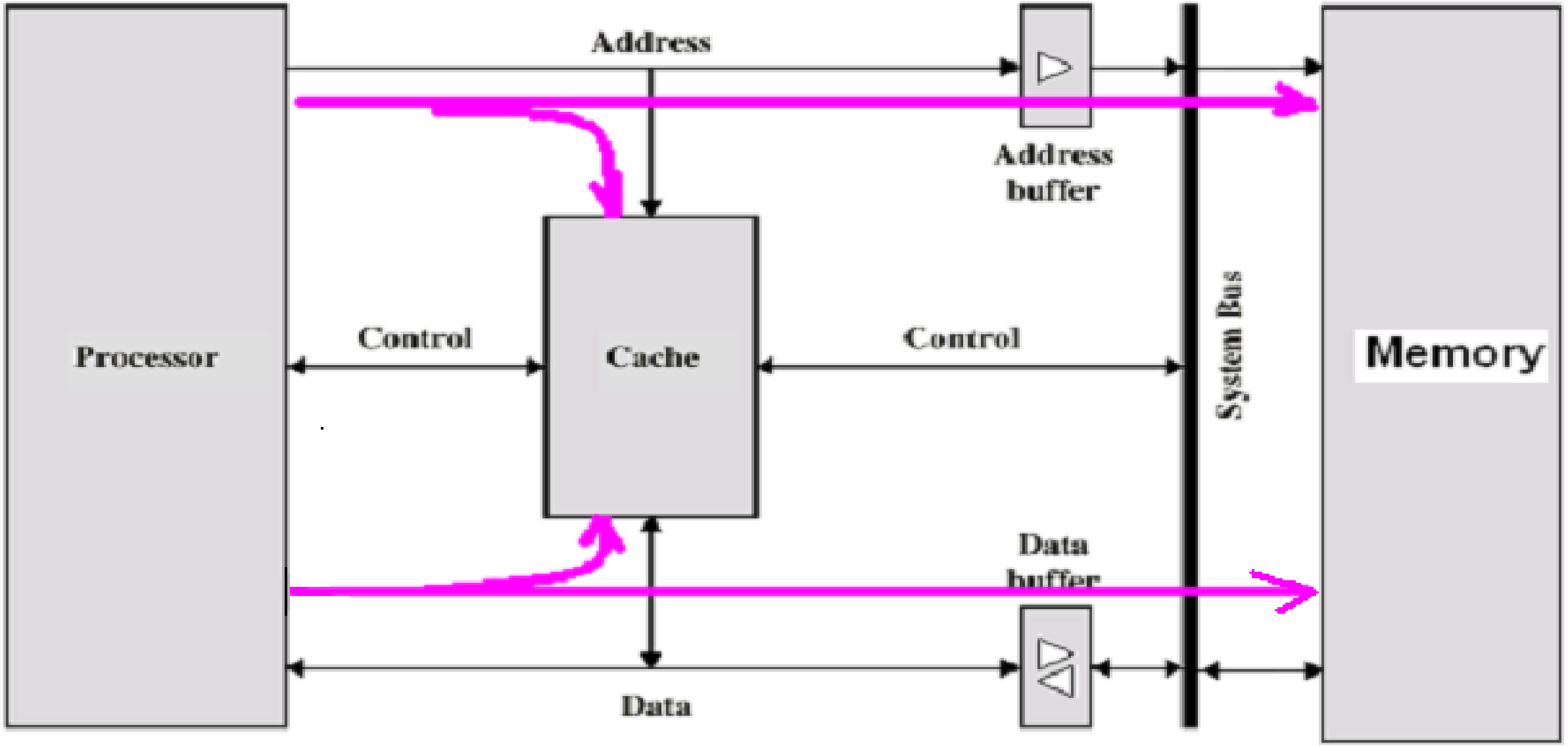
Read Hit


Address
Address buffer
Processor
Control
Cache
Control
System Bus
Memory
Data buffer
Data

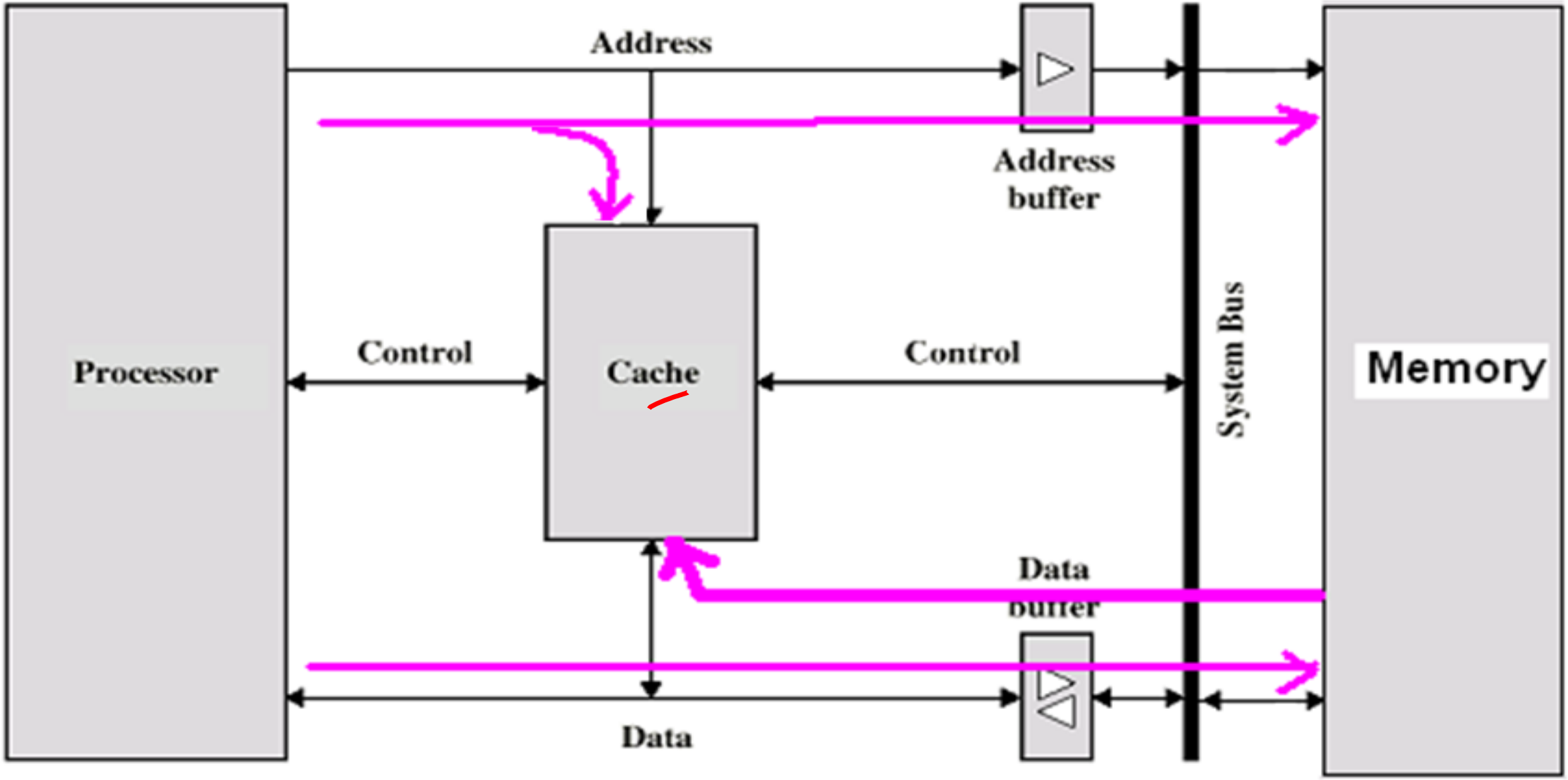
Read Miss


Address
Address buffer
Processor
Control
Cache
Control
System Bus
Memory
Data buffer
Data

Write hit


Address
Address buffer
Processor
Control
Cache
Control
System Bus
Memory
Data buffer
Data

Write miss

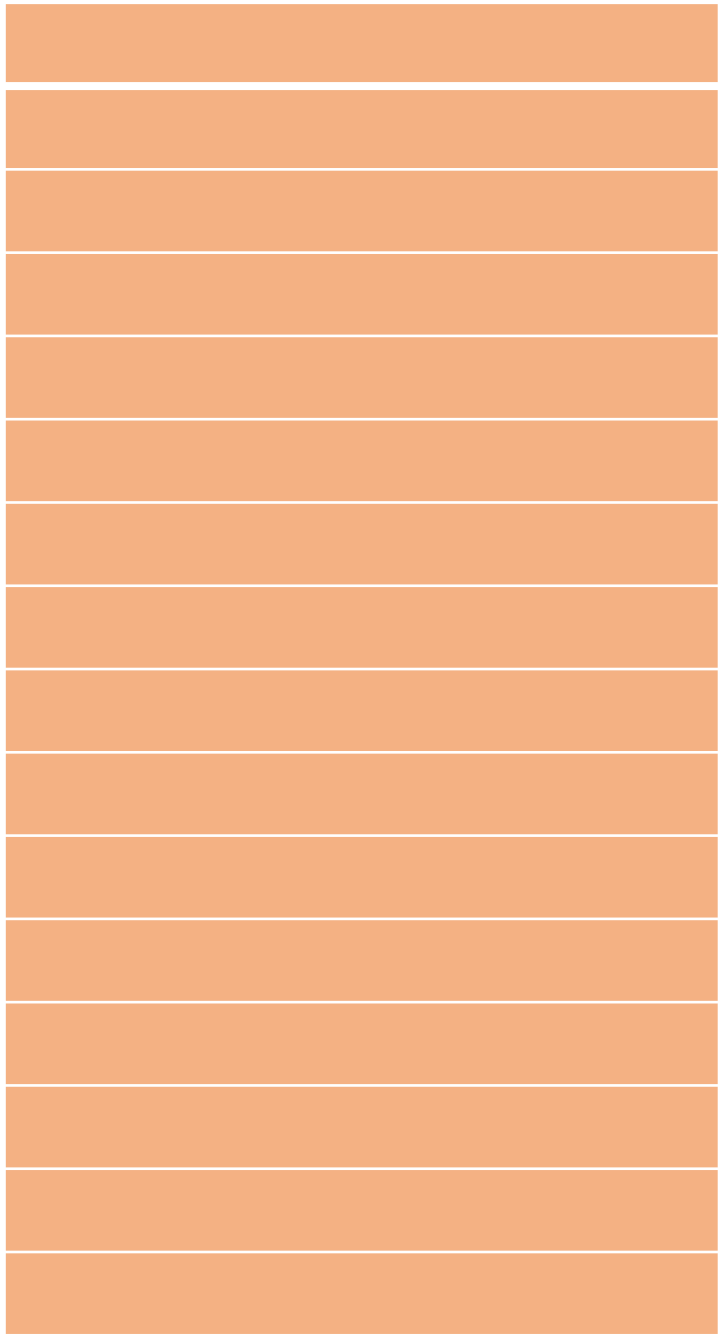

Address
Address buffer
Processor
Control
Cache
Control
System Bus
Memory
Data buffer
Data

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 4 bytes
- Cache of 8 Byte
  - How many cache lines?
  - cache contains 2 lines (4 bytes per Line)



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 modulo m**

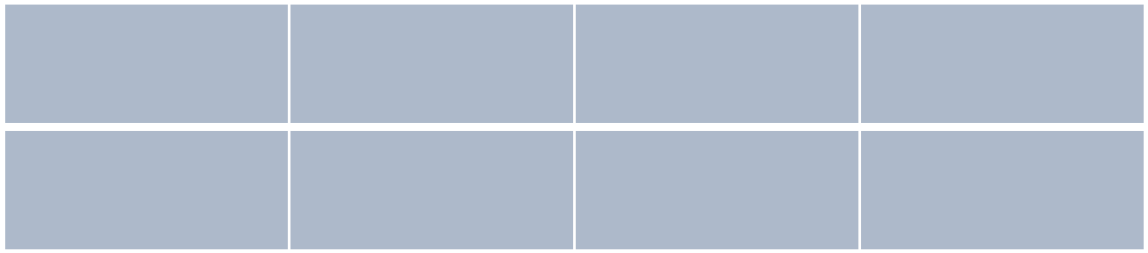
where i = cache line number

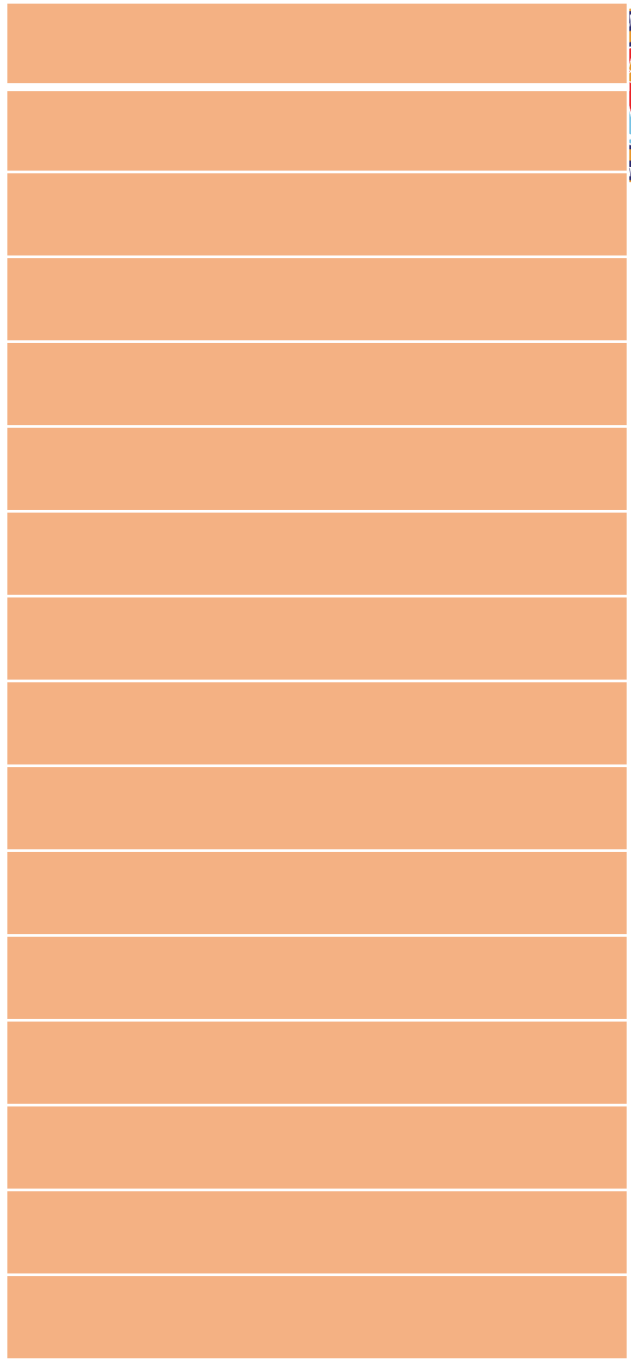
j = main memory block no.

m = no.of lines in the cache

Direct Mapped Cache

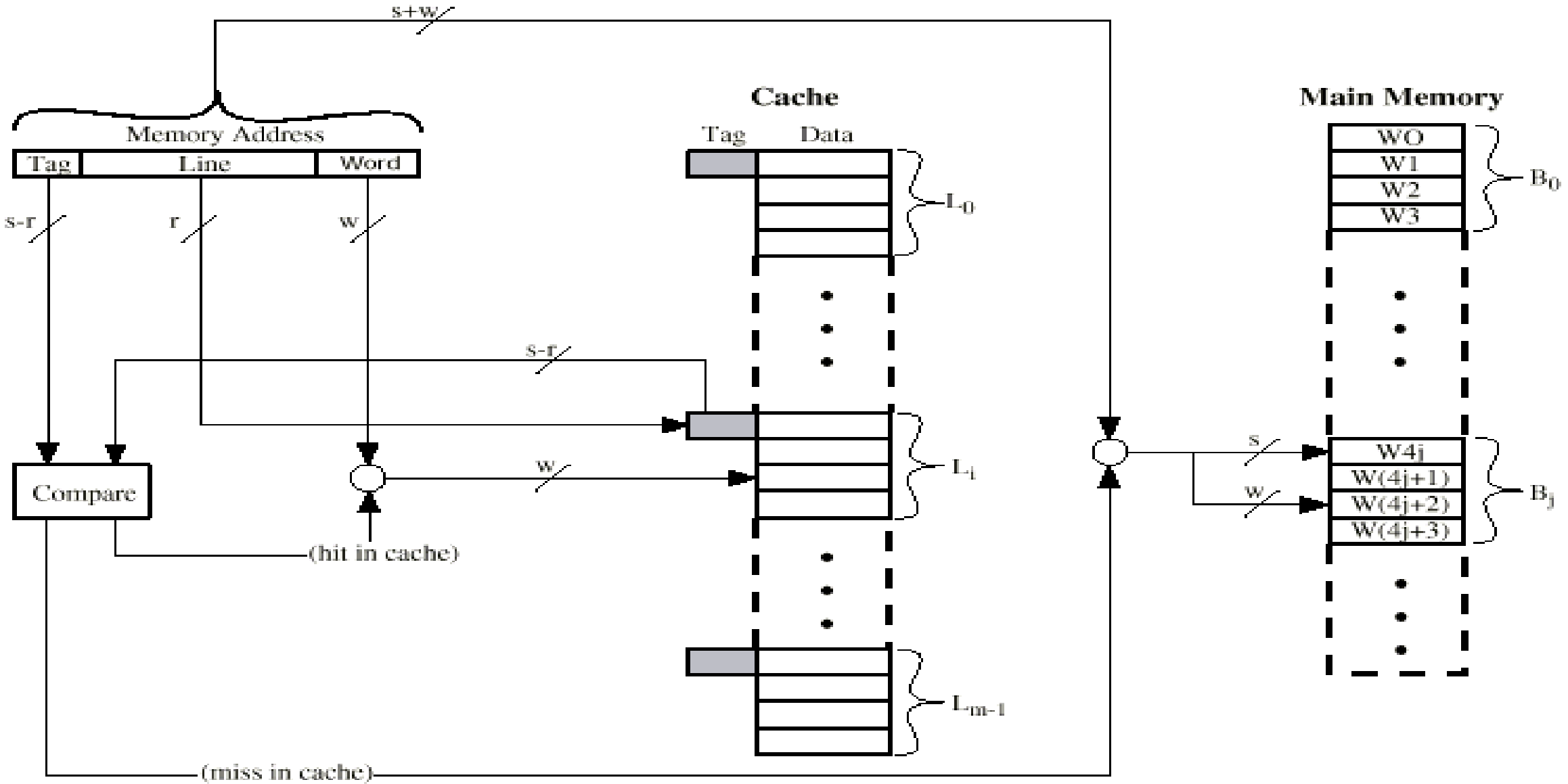
i = j modulo m





- 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

Problem 1 : Direct Mapped Cache

•Given :

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

•Find out

a) Number of bits required to address the main memory

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) Number of bits to identify a block

f) Tag, Line, Word

Solution 1

2^n = capacity

$2^{10} \Rightarrow K$
$2^{20} \Rightarrow M$
$2^{30} - G$
$2^{40} - T$

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024 = 1K$

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

MM Block Size

- Find out

a) Number of bits required to address the main memory

$2^n = 16M$
$= 2^4 \cdot 2^{20} \Rightarrow 2^{24}$
$\therefore n = 24$ bits

- b) Number of blocks in main memory

$$= \frac{\text{Capacity}}{\text{MM Block Size}} = \frac{16MB}{4B} = \frac{2^{24}}{2^2} = 2^{24-2} = 2^{22} = 2^2 \cdot 2^{20}$$
$= 4M$ Blocks

- c) Number of cache lines

$$\frac{\text{Cache Size}}{\text{Line Size}} = \frac{64KB}{4B} = \frac{2^6 \cdot 2^{10} B}{2^2 B} = 2^{16-2} \Rightarrow 2^{14}$$
$= 16K$ lines

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?

$2^n = 4 = 2^2$ ⇒ $n = 2$ bits

- e) Number of bits required to identify a block

$2^n = 4M \Rightarrow 2^{22} \Rightarrow n = 22$ bits

- f) Tag, Line, Word

Cache lines = 16K lines

$2^n = 16K$

$= 2^{14}$

$n = 14$

Tag = # address bits − line − word

$= 24 - 14 - 2 \Rightarrow 8$

22 bit

2 bits

Tag s-r	Line r	Word w
8	14	2

24

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?

Solution 2

64 32 16 8 4 2 1 ←
0 0 1 1 0
0 0 0 1 1
1 0 1 0 1

Tag = 8 bits	line 5 bits	word 3-bits

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?

Blocksize = 8 bytes

$2^w = 8 \Rightarrow 2^3$ word = 3 bits

$2^r = 32 \Rightarrow 2^5$ line = 5 bits

Tag = # address bit − r − w = 16 − 5 − 3 = 8 bits

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

	Tag	line	word				
0001	0001	0001	1011	⇒	00011	⇒ l_3 ⇒	line 3
1100	0011	0011	0100	⇒	00110	⇒ l_6 ⇒	line 6
1101	0000	0001	1101	⇒	00011	⇒ l_3 ⇒	line 3
1010	1010	1010	1010	⇒	10101	⇒ l_{21} ⇒	line 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?

Tag line W

000

0001 1010 0001 1**000**
0001 1010 0001 1**001**
0001 1010 0001 1**010** : given in the problem
0001 1010 0001 1**011**
0001 1010 0001 1**100**
0001 1010 0001 1**101**
0001 1010 0001 1**110**
0001 1010 0001 1**111**

111

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?

lines × block size = 32 × 8 bytes ⇒ 256 bytes

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

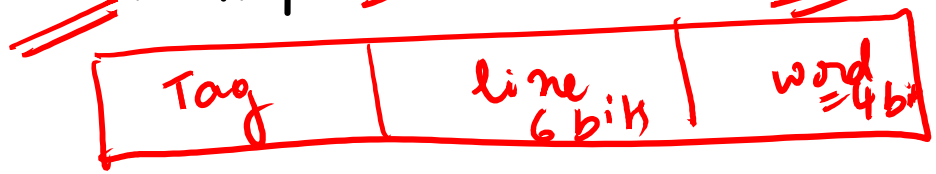
Problem 3

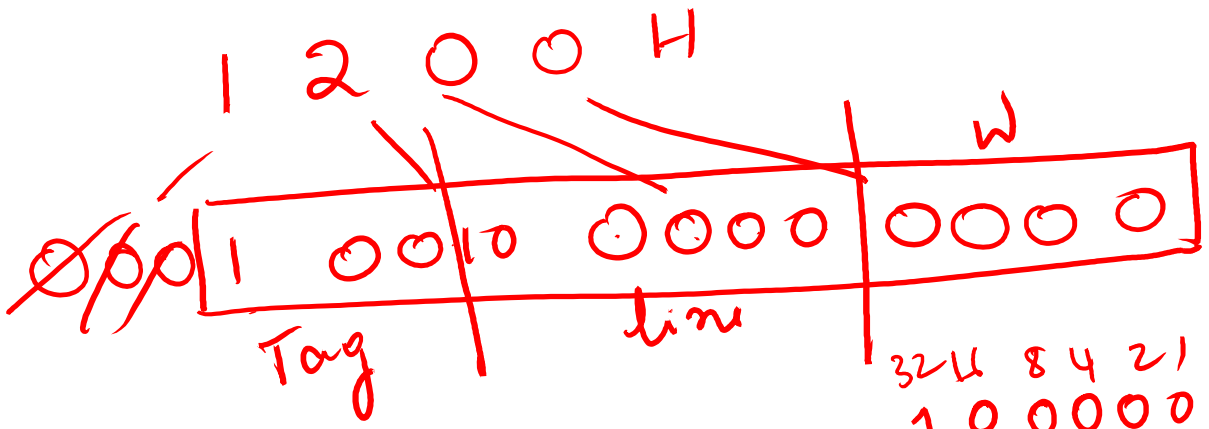
MM size = 8K

$2^n = 8K \Rightarrow 2^3 \cdot 2^{10} = 2^{13}$

$n = 13$ bits

- 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? → line # 32





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

Block size = 16 bytes

$2^w = 16 \Rightarrow 2^4 \Rightarrow w = 4$ bits

Cache lines = 64

$2^r = 64 \quad r = 6$

Problem 4

Tag s-r	Line r	Word w
17	11	4

- The system uses a L1 cache with direct mapping and 32-bit address format is as follows:
- bits 0 - 3 = offset (word) = 4 bits ⇒ 3-0+1 = 4
- bits 4 - 14 = index bits (Line) 14-4+1 = 11
- bits15 - 31 = tag 31-15+1 = 17

a) What is the size of cache line? ⇒ word field = 16 bytes

b) How many Cache lines are there? ⇒ Line field ⇒ 2^{11} ⇒ $2^{1} \cdot 2^{10}$ = 2K ⇒ 2048 lines

c) How much space is required to store the tags in the L1 cache? ⇒ # cache lines × tag size
2048 × 17 = ____ bits

d) What is the total Capacity of cache including tag storage?

= 2048 × 4 B + 2048 × 17 bits

= 2048 × 4 × 8 + 2048 × 17 bits

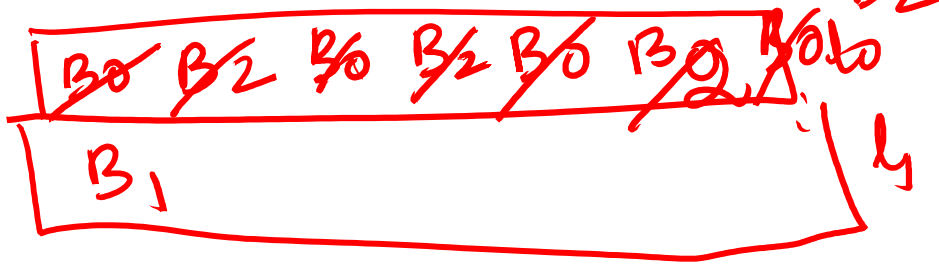
= ____ / 8 = ____ Bytes.

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 :

0 2 0 2 2 0 0 2 0 0 0 2 1

- Find out hit ratio.



j mod m

B0 ⇒ 0 mod 2 = lo

B1 ⇒ 1 mod 2 = l1

B2 ⇒ 2 mod 2 = lo

0 2 0 2 2 0 0 2 0 0 0 2 1

M M M M H M H M M H H M M

4/13

Problem 5 – Direct Mapped Cache

- 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 :

  0 2 0 2 2 0 0 2 0 0 0 2 1

Find out hit ratio.

0 2 0 2 2 0 0 2 0 0 0 2 1

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?

$2^n = 1MB \Rightarrow 2^{20}$ ∴ $n = 20$ bits

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

hit miss

Avg access time = hit ratio * cache access + (1- hit ratio) * (cache access + memory access)

$= 0.95 * 0.1\,\mu s + (0.05) * (0.1\,\mu s + 1\,\mu sec)$

$=$

Computer Organization and Software Systems

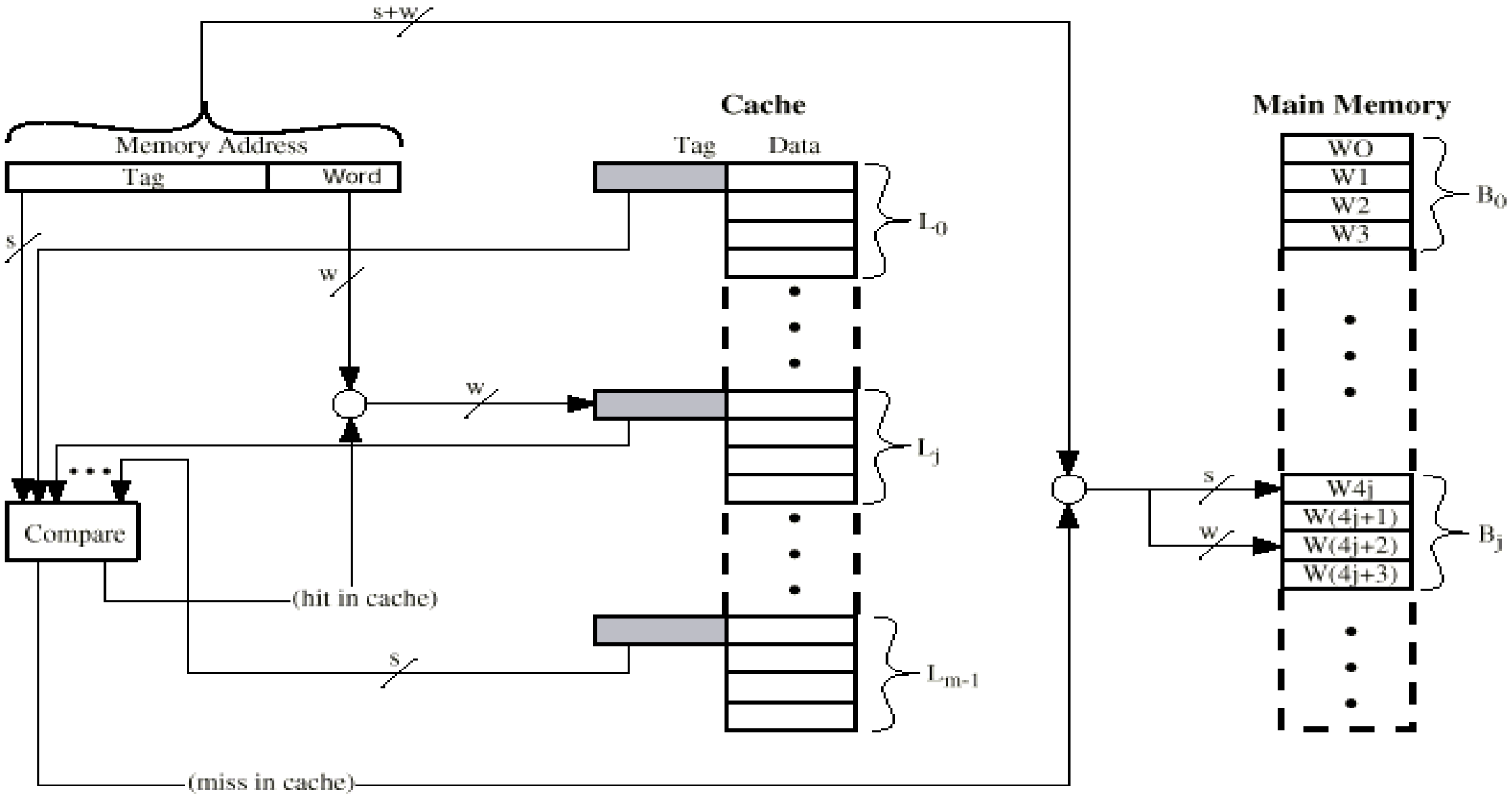
Contact Session 5

Dr. Lucy J. Gudino

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 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

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

(a) $2^n = 32\,MB = 2^5\ 2^{20}\ B = 2^{25}$

$n = 25$ bits

(b) # blocks = $\frac{\text{MM capacity}}{\text{Block size}} = \frac{2^{25}\ B}{2^3\ B} = 2^{25-3} = 2^{20}\ 2^2 = 2^{22}$ blocks

$= 4\,M$ Blocks

(c) # cache lines = $\frac{\text{Cache capacity}}{\text{Block size}} = \frac{128\,KB}{8\,B} = \frac{2^7\ 2^{10}\ B}{2^3\ B} = 2^{14} = 2^4\ 2^{10}$

$= 16K$ lines

(d) $2^w = 2^3\ B \Rightarrow w = 3$

Tag = 22	w = 3

25 bits

innovate achieve lead

Problem 8

•Cache of 64KByte, Cache block of 4 bytes , 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 9

blocks = 4 Blocks ⇒ B_0, B_1, B_2, B_3

- 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.

Cache

l_0	0
l_1	2

0 2 0 2 2 0 0 2 0 0 0 2 1

M M H H H H H H H H H M

Hit ratio = 10/13

DM ⇒ 4/13

Set Associative Mapping

- Cache is divided into a number of sets (v sets each with k lines)
- $m = v * k$
- $i = j \text{ modulo } v$

  where i = cache set number

  j = main memory block number

  v = number of sets 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
- ~~m~~ k-way set associative cache
  - 2 way set associative mapping ➔ 2 lines per set
  - A given block can be in one of 2 lines in only one set

v = 10 sets

k = 2

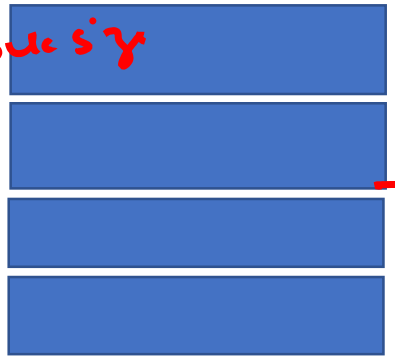
m = v × k

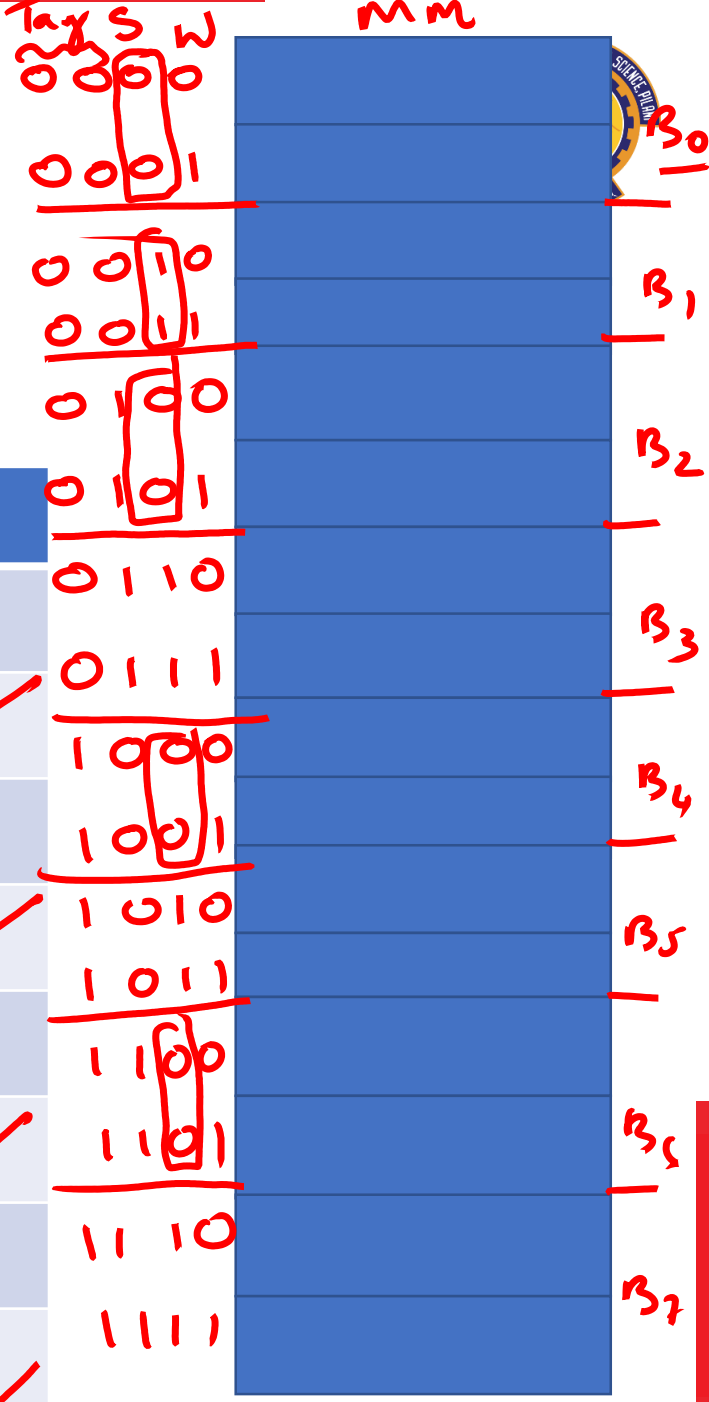
= 10 × 2

= 20

Example

- 16 Bytes main memory, Block Size is 2 Bytes,
- Cache of 8 Bytes, 2 way set associative cache
  - # address bits ⇒ 4 bits
  - Cache line size ⇒ 2 B
  - # main memory blocks
  - # Number of cache lines
  - # lines per set
  - # of sets



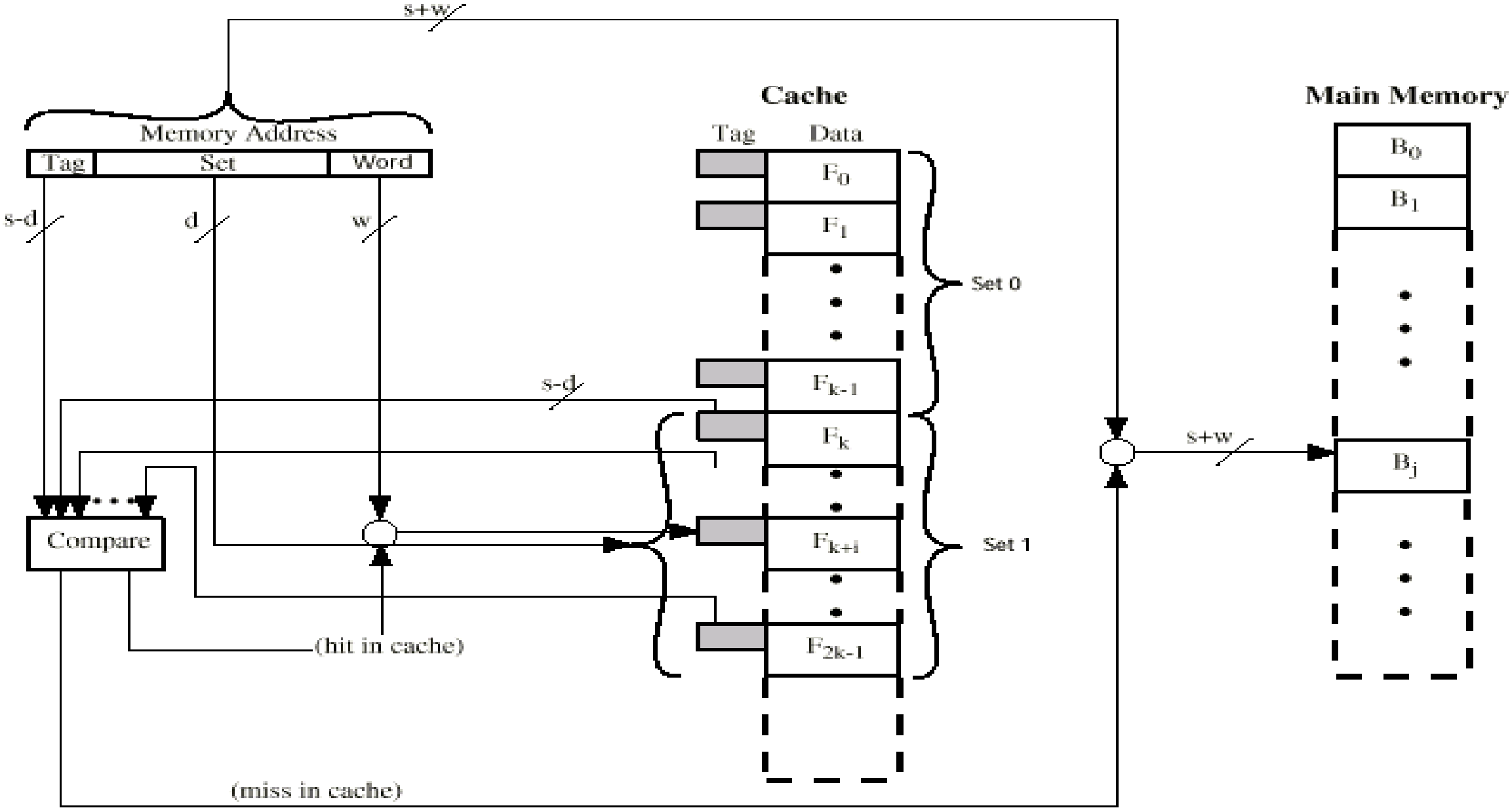


$B_0, B_2, B_4, B_6 \Rightarrow S_0$

$B_1, B_3, B_5, B_7 \Rightarrow S_1$

i = j 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

Two-Way 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

- A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 bytes each. Find out
  - a) Total main memory capacity
  - b) Total cache memory capacity
  - c) Total number of sets in the cache
  - d) Number of bits for TAG, SET and word

Given data

# of lines = 64 ⇒ m

K = 4 ⇓

4 way set associative cache

MM capacity : 4 K Blocks

Block size : 128 B

$2^W = 128B = 2^7 B$

$2^S = 16 \Rightarrow 2^4$

= # of address bits = W - S

= 19 - 7 - 4

= 8 bits

a) capacity of MM : 4K x 128 B $= 2^2 \cdot 2^{10} \cdot 2^7$ B

$= 2^{19}$ B

= 512 KB

(b) Cache Memory capacity = # lines x Block size

$= 64 \times 128B$

$= 2^6 \cdot 2^7 B = 2^{13} B$

= 8 KB

(c) # Sets = $\frac{64}{4} = \frac{m}{K}$ = 16 Sets

(d)

Tag = 8	Set = 4	word = 7

Problem 2: Home work

- A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses **word level addressing**. What is the address format? If we change the cache to a 4- way set associative cache, what is the new address format?

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 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 3

- 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 for a cache with 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_0	0_0	0_2	0_2	0_2	0_2	0_6	0_6	0_6	0_6	0_{10}	0_{10}
L1		4_1	4_1	4_1	4_1	8_5	8_5	8_5	8_5	3_9	3_9	3_9
L2				2_3	2_3	2_3	2_3	2_3	2_8	2_8	2_8	2_8
L3					1_4	1_4	1_4	1_7	1_7	1_7	1_7	4_{11}
H/M	M	M	H	M	M	M	H	H	H	M	H	M

Hit ratio $\Rightarrow 5/12$

Problem 2 - LFU

Ref	0	4	0	2	1	8	0	1	2	3	0	4
L0	O_1	O_1	O_2	O_2	O_2	O_2	O_3	O_3	O_3	O_3	O_4	O_4
L1		4_1	4_1	4_1	4_1	8_1	8_1	8_1	8_1	3_1	3_1	4_1
L2				2_1	2_1	2_1	2_1	2_1	2_2	2_2	2_2	2_2
L3					1_1	1_1	1_1	1_2	1_2	1_2	1_2	1_2
H/M	M	M	H	M	M	M	H	H	H	M	H	M

Hit ratio: 5/12

$L_0 \Rightarrow B_0$

$L_1 \Rightarrow B_4$

$L_2 \Rightarrow B_2$

$L_3 \Rightarrow B_1$

Problem 2 - FIFO

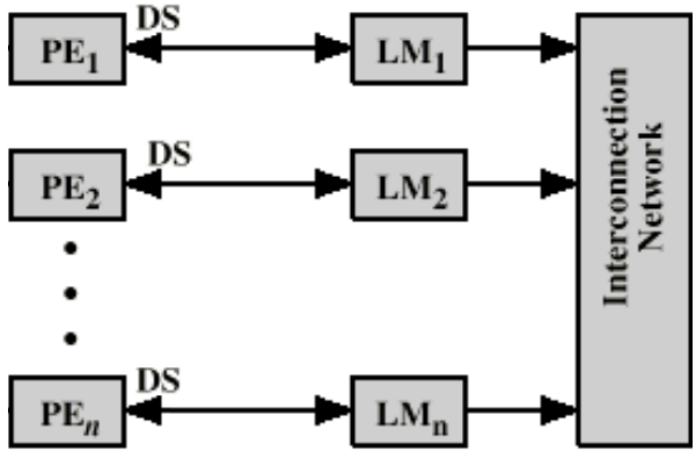
Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0	0_0	0_0	0_0	0_0	0_0	8_5	8_5	8_5	8_5	8_5	8_5	8_5
L1		4_1	4_1	4_1	4_1	4_1	0_6	0_6	0_6	0_6	0_6	0_6
L2				2_3	2_3	2_3	2_3	2_3	2_3	3_9	3_9	3_9
L3					1_4	1_4	1_4	1_4	1_4	1_4	1_4	4_{11}
H/M	M	M	H	M	M	M	M	H	H	M	H	M

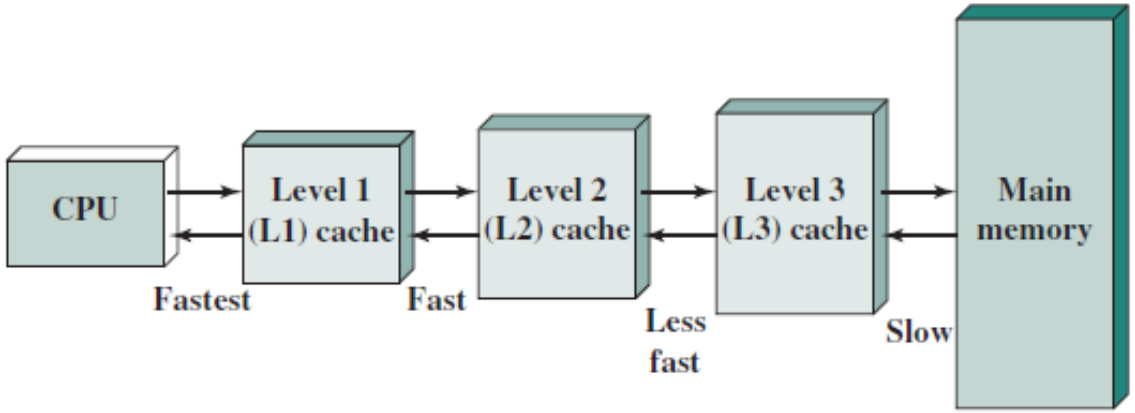
Hit ratio = 4/12

L0 ⇒ B8
L1 ⇒ B0
L2 ⇒ B3
L3 ⇒ B4

Issues with Writes

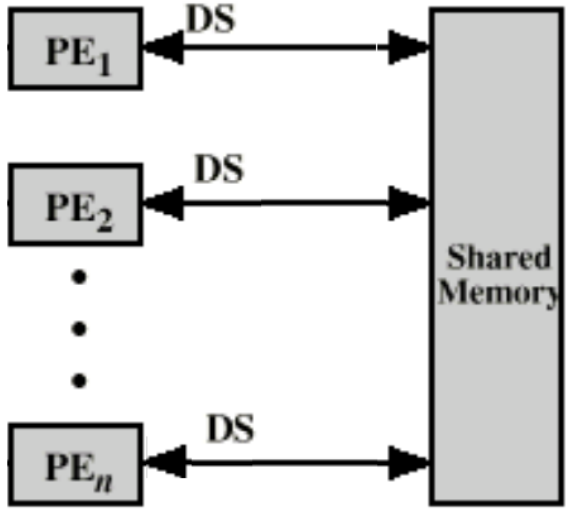
- Multiple copies of data exist:
  - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from memory or not)



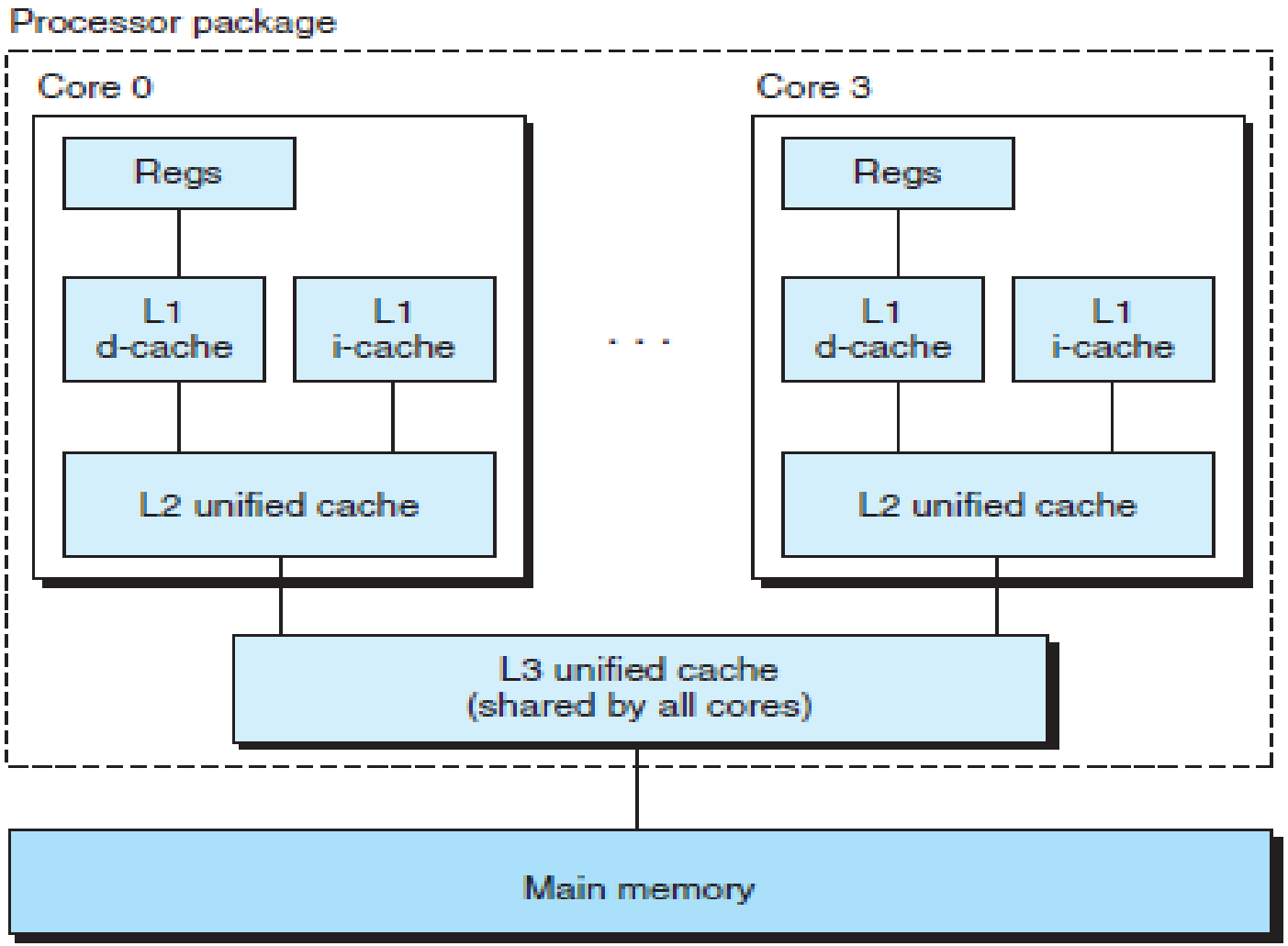


(b) Three-level cache organization

Cache and Main Memory



Intel Core i7 Cache Hierarchy



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.

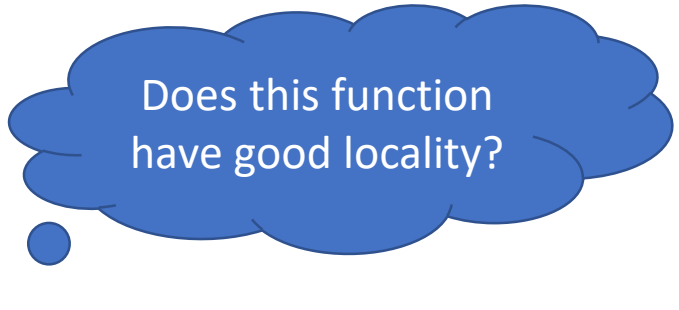
Computer Organization and Software Systems

Contact Session 6

Dr. Lucy J. Gudino

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    }
```



N=8								
Address	0	1	2	3	4	5	6	7
Contents	V[0]	V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]
Access Order	1	2	3	4	5	6	7	8

Stride k – reference pattern

Byte Addressable memory and word length is 1 byte

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

Address difference

Stride=----------

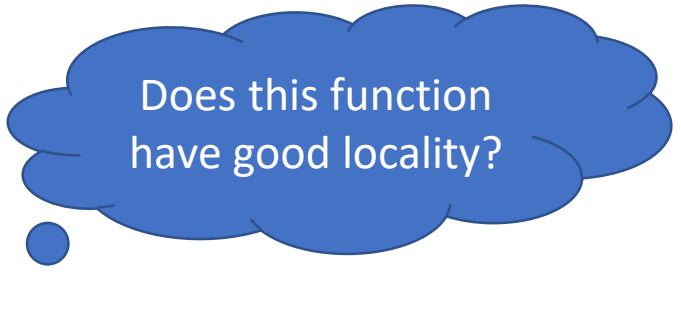
Word Length

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 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       return sum;
9   }
```

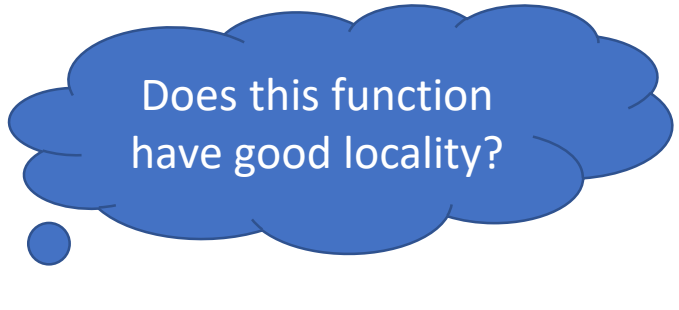


MxN	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6

M = 2, N=3						
Address(Index)	0	1	2	3	4	5
Contents	A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]
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        return sum;
9    }
```



M = 2, N=3						
Address(Index)	0	1	2	3	4	5
Contents	A[0][0]	A[1][0]	A[2][0]	A[1][0]	A[1][1]	A[2][1]
Access Order	1	3	5	2	4	6

NxM	[0]	[1]
[0]	1	2
[1]	3	4
[2]	5	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:

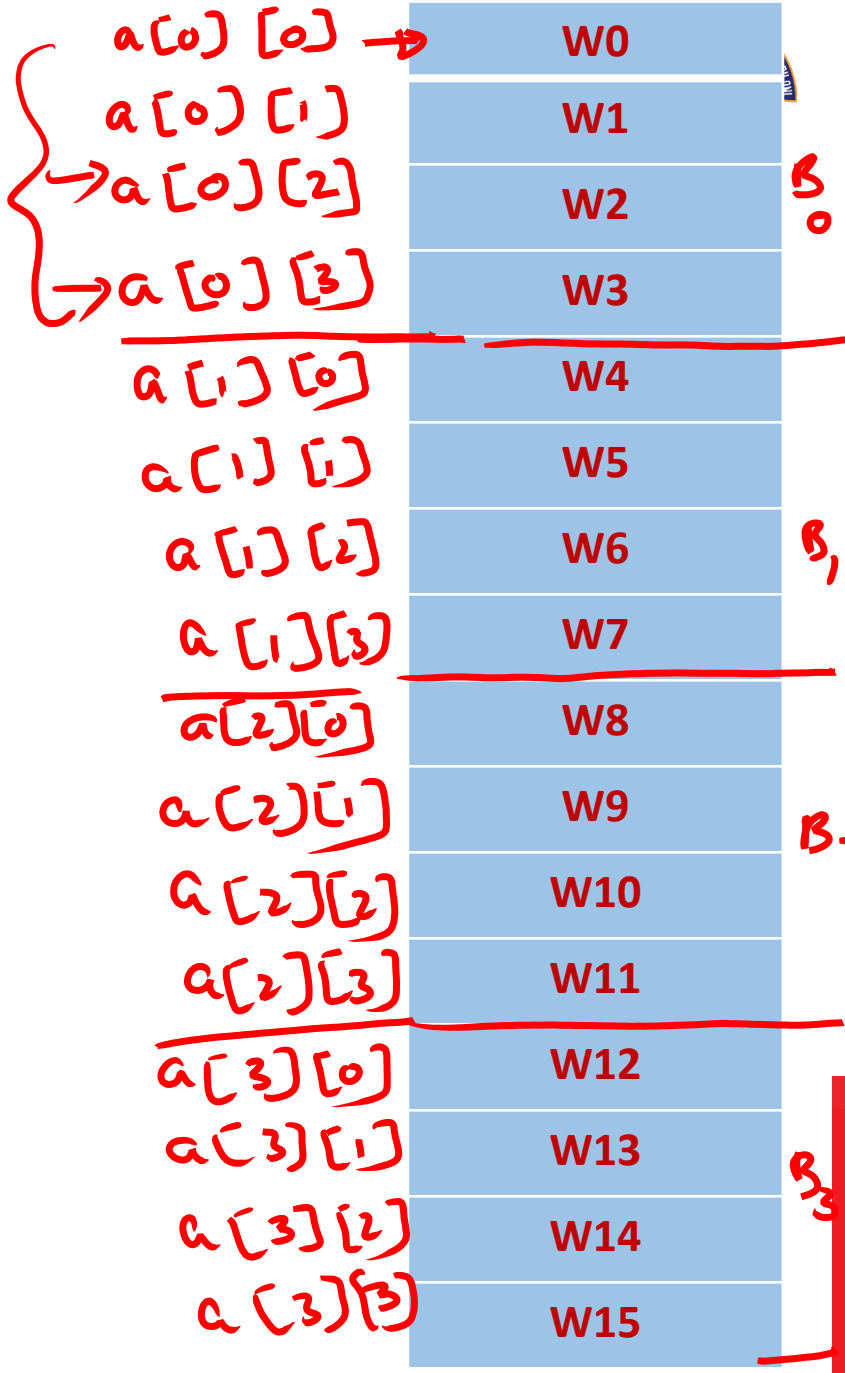
- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

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;
}
```

12/16

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0	W0 M	W1 H	W2 H	W3 H
i = 1	W4 M	W5 H	W6 H	W7 H
i = 2	W8 M	W9 H	W10 H	W11 H
I = 3	W12 M	W13 H	W14 H	M15 H



innovate achieve lead

Example 2

```
int sumarraycols(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 2(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	W0 M	W1 M	W2 M	W3 M
i = 1	W4 M	W5 M	W6 M	W7 M
i = 2	W8 M	W9 M	W10 M	W11 M
I = 3	W12 M	W13 M	W14 M	W15 M

L0	B0 B2 B0 B2
L1	B1 B3 B1

0/16

W0
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15

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];
}
```

Today's Session

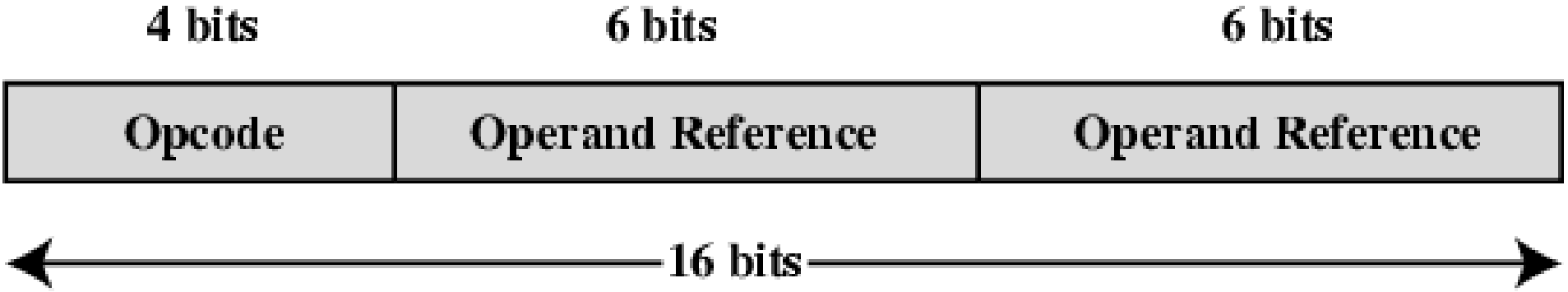
Contact Hour	List of Topic Title	Text/Ref Book/external resource

11-12	• **Instruction Set Architecture - CISC Vs RISC**
	• CISC Instruction Set (Intel x86 as an example)
	  • Machine Instruction Characteristics
	  • Types of Operands
	  • Types of Operations
	  • Addressing Modes
	• Instruction Formats
		T1

Introduction

- 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



- 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 instruction

Number of Addresses (1/2)

- 3 addresses
  - Result, Operand 1, Operand 2
  - c = a + b; add c, a, b
  - 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
  - Reduces length of instruction
  - The original value of a is lost.

Number of Addresses (2/2)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines
- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. c = a + b

```
push a
push b
add
pop c
```

Example

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

Instruction		Comment
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction		Comment
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) One-address instructions

How Many Addresses

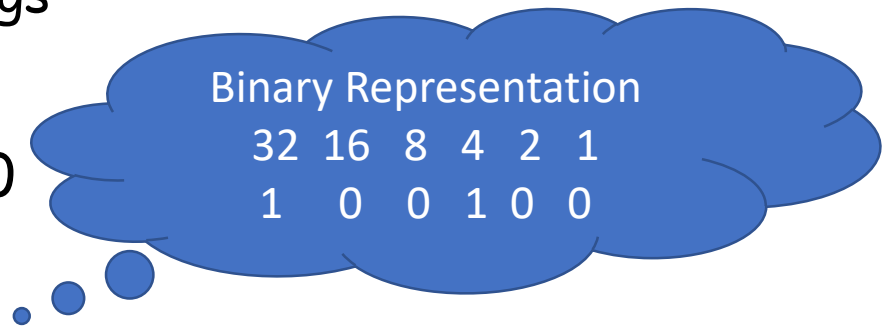
- Fewer addresses
  - More Primitive instructions, shorter length instructions
  - Less complex instructions, hence requires less complex hardware
  - More instructions per program
    - Longer programs
    - More complex programs
    - Longer execution time
- Multiple address instructions
  - Lengthy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

Instruction set Design Decisions

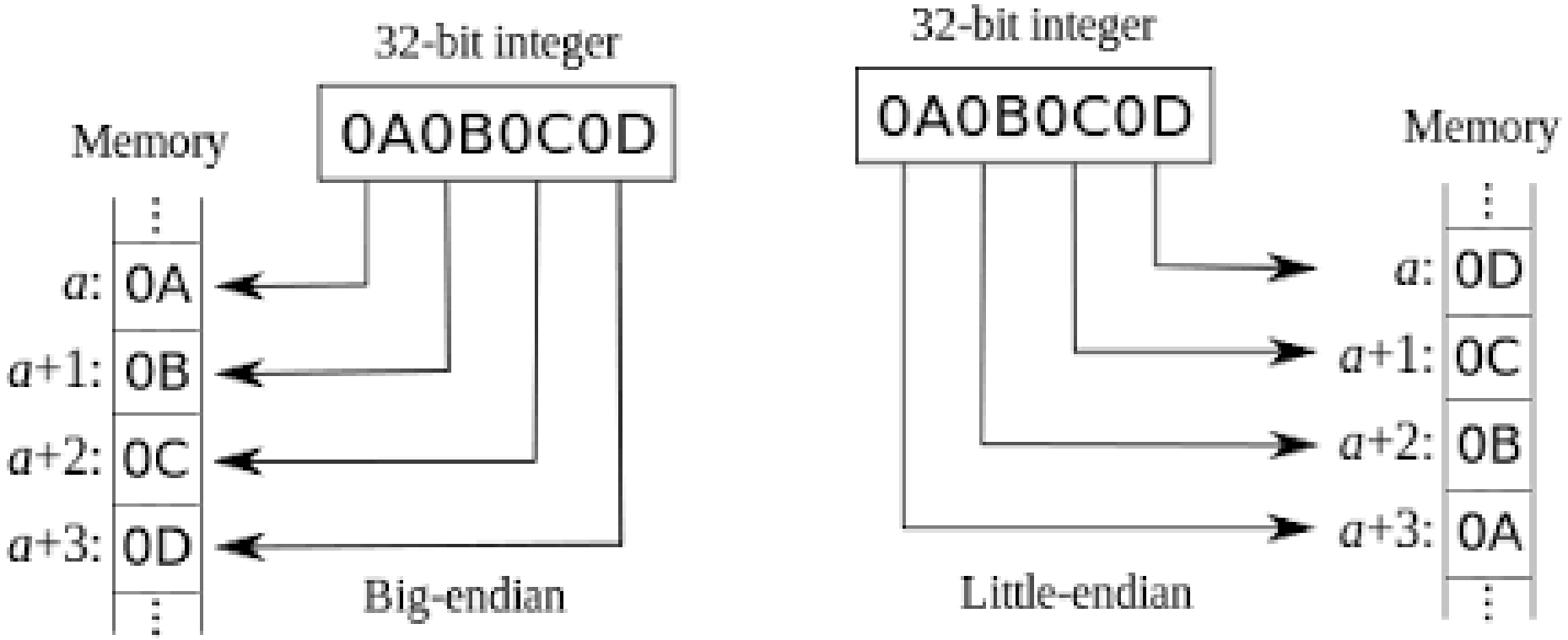
- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses
- Registers
  - Number of CPU registers available
  - Which operations can be performed on which registers?
- Addressing modes

Types of Operand

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



Byte Ordering

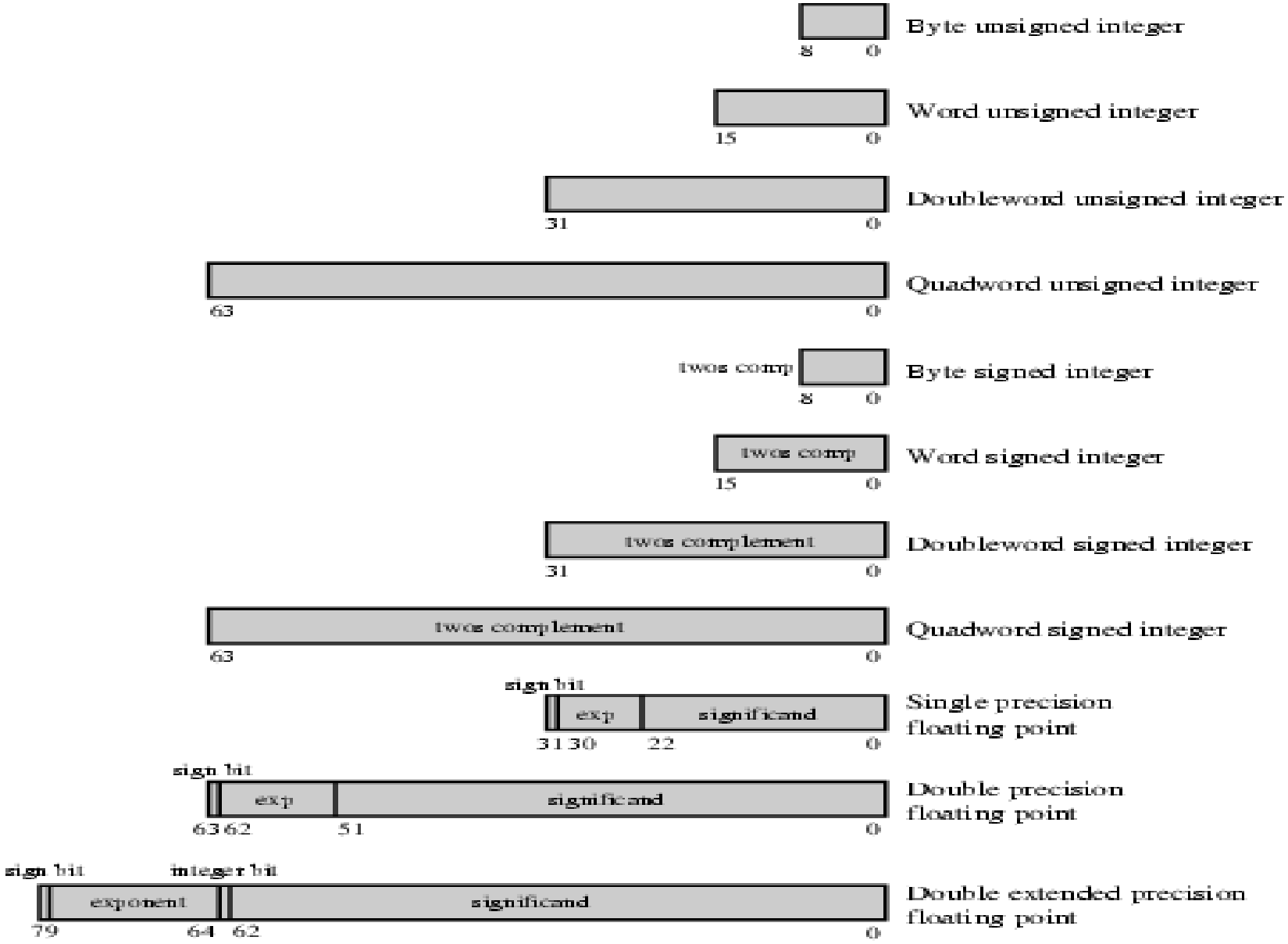


x86 Data Types

- 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 – 16/32 bit offset within segment
- Far pointer - 16/32 bit offset outside segment
- 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 twos 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:

1. Calculate the memory address, based on the address mode
2. If the address refers to virtual memory, translate from virtual to real memory address.

3. Determine whether the addressed item is in cache.

4. If not, issue a command to the memory

Arithmetic

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

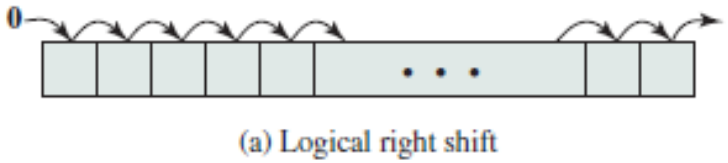
Logical

- Bitwise operations
- AND, OR, NOT

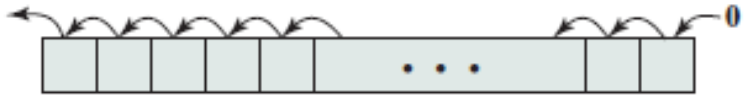
Basic Logical Operations

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

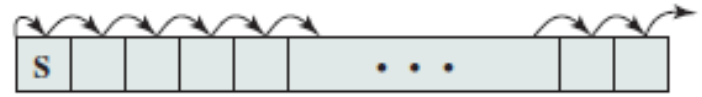
Shift and Rotate Operations



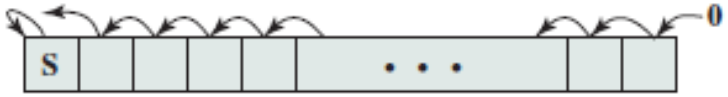
(a) Logical right shift



(b) Logical left shift



(c) Arithmetic right shift



(d) Arithmetic left shift

(e) Right rotate

(f) Left rotate

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

Conversion

- E.g. Binary to Decimal

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

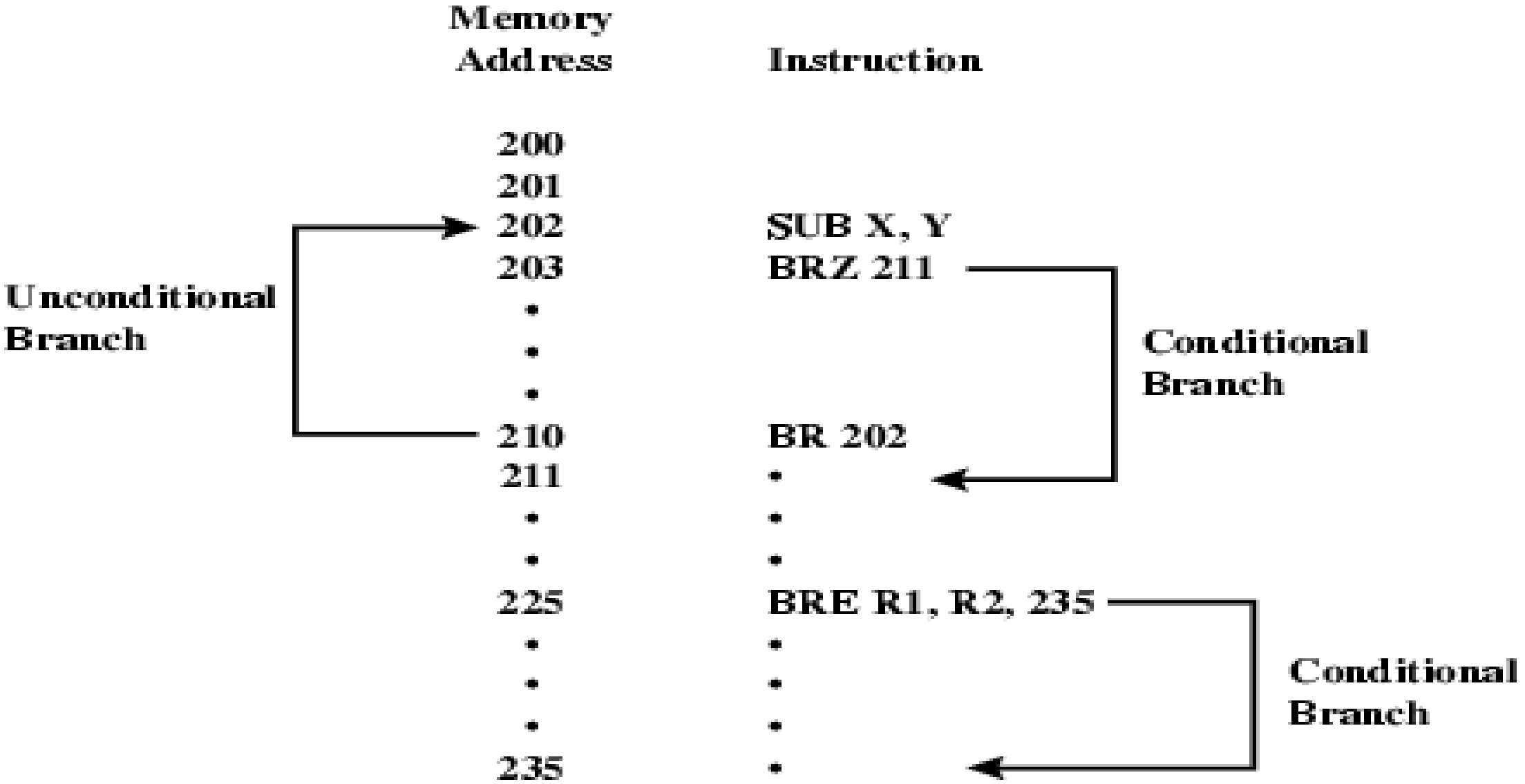
- Privileged instructions
- CPU needs to be in specific state
  - User Mode
  - Kernel mode
- For operating systems use

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 instruction
  - e.g. increment and skip if zero

```
ISZ Register1
Branch xxxx
 ADD A
```

Branch / Jump Instruction



Transfer of Control

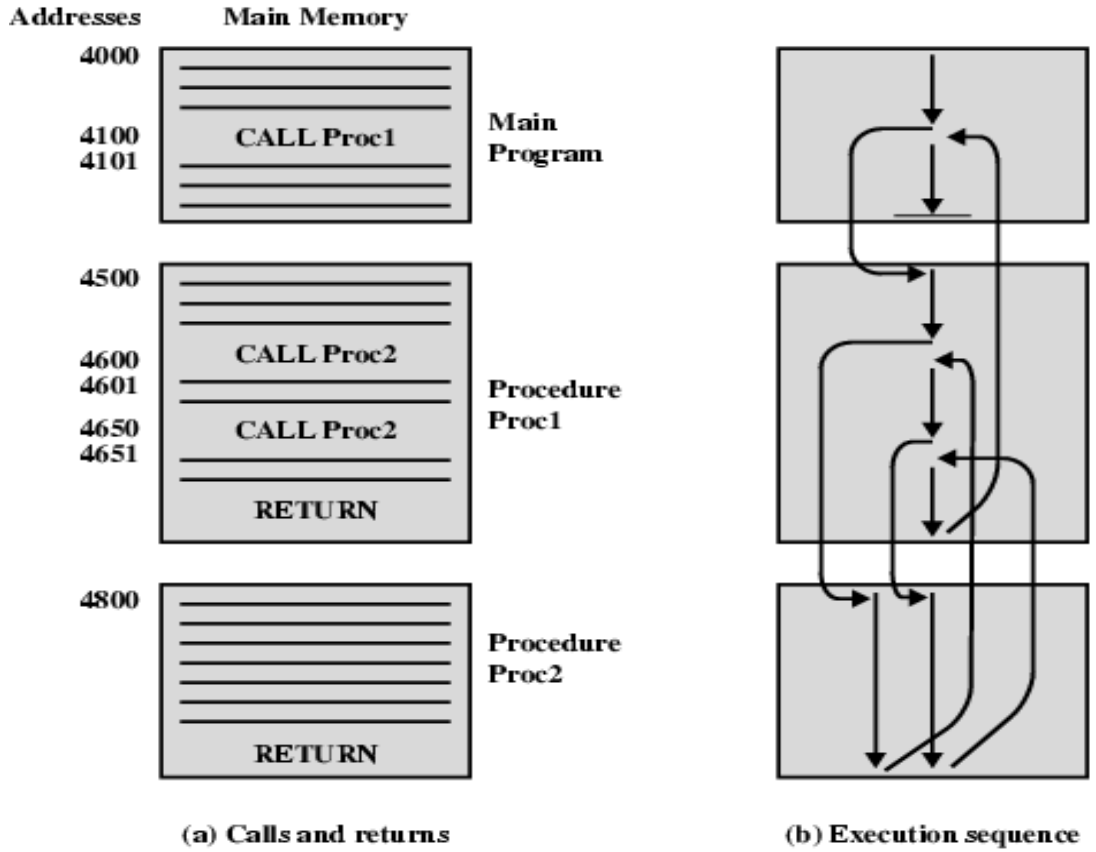
- Jump / Branch (Unconditional / Conditional)
  - e.g. jump to x if result is zero
- Skip (Unconditional / Conditional)
  - skip (unconditional) : Increment to skip next instruction
  - e.g. increment and skip if zero

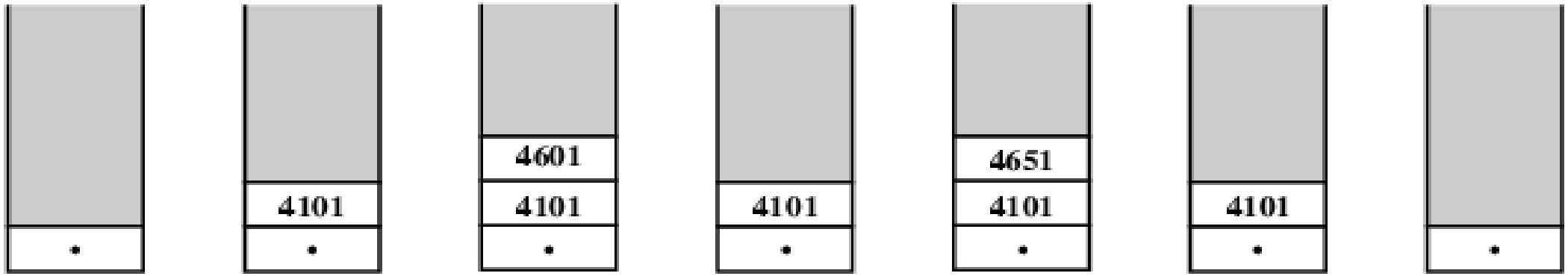
    ISZ Register1

    Branch xxxx

    ADD A
- Subroutine call
- interrupt call

Use of Stack


Addresses
Main Memory
4000
4100
4101
CALL Proc1
Main Program
4500
4600
4601
CALL Proc2
4650
4651
CALL Proc2
RETURN
Procedure Proc1
4800
RETURN
Procedure Proc2
(a) Calls and returns
(b) Execution sequence


4101
4601
4101
4101
4651
4101
4101
(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

Computer Organization and Software Systems

Contact Session 7

Dr. Lucy J. Gudino

Addressing Modes

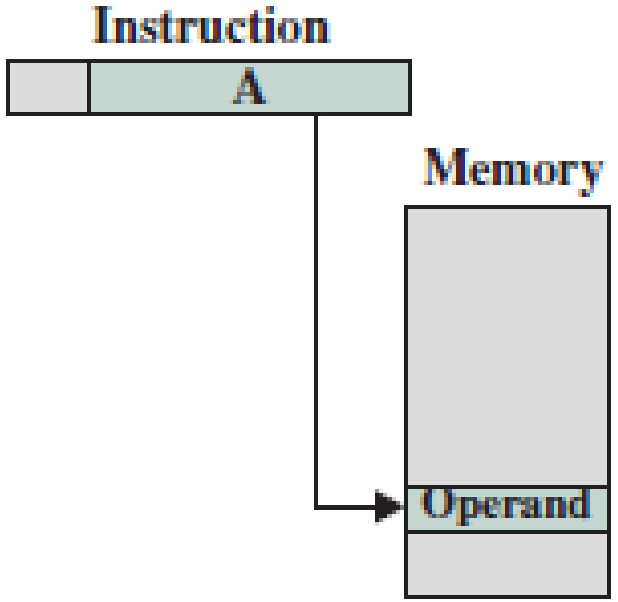
- Addressing modes refers to the way in which the operand of an instruction is specified
- Types:
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register Indirect
  - Displacement (Indexed)
  - Stack

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

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



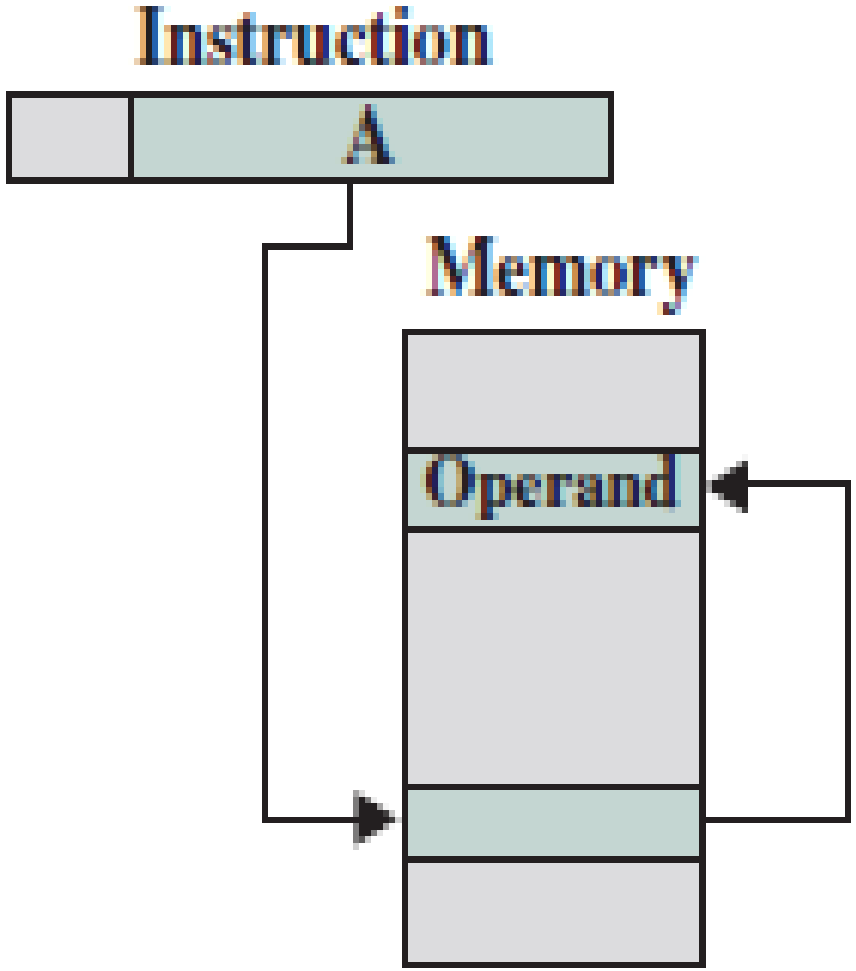
Indirect Addressing

- 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

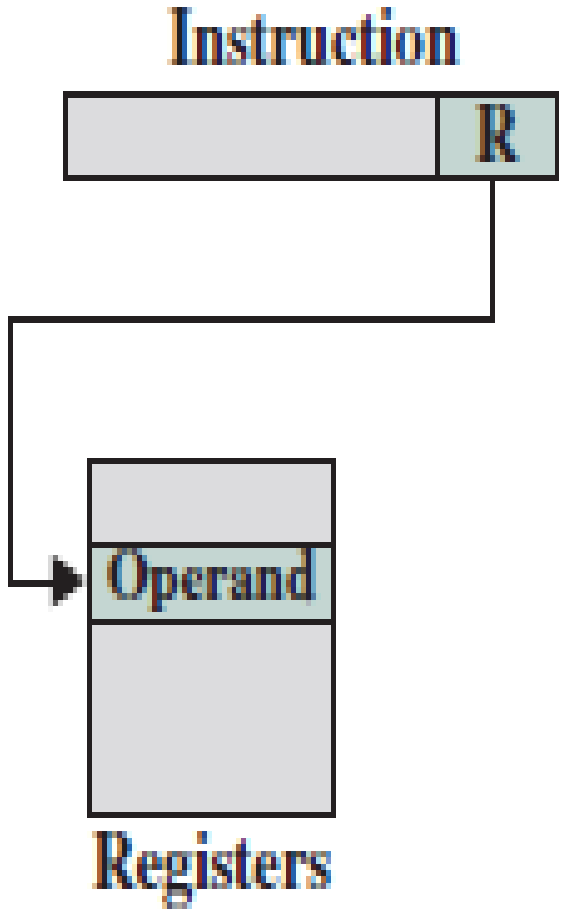


Indirect Addressing…

- 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

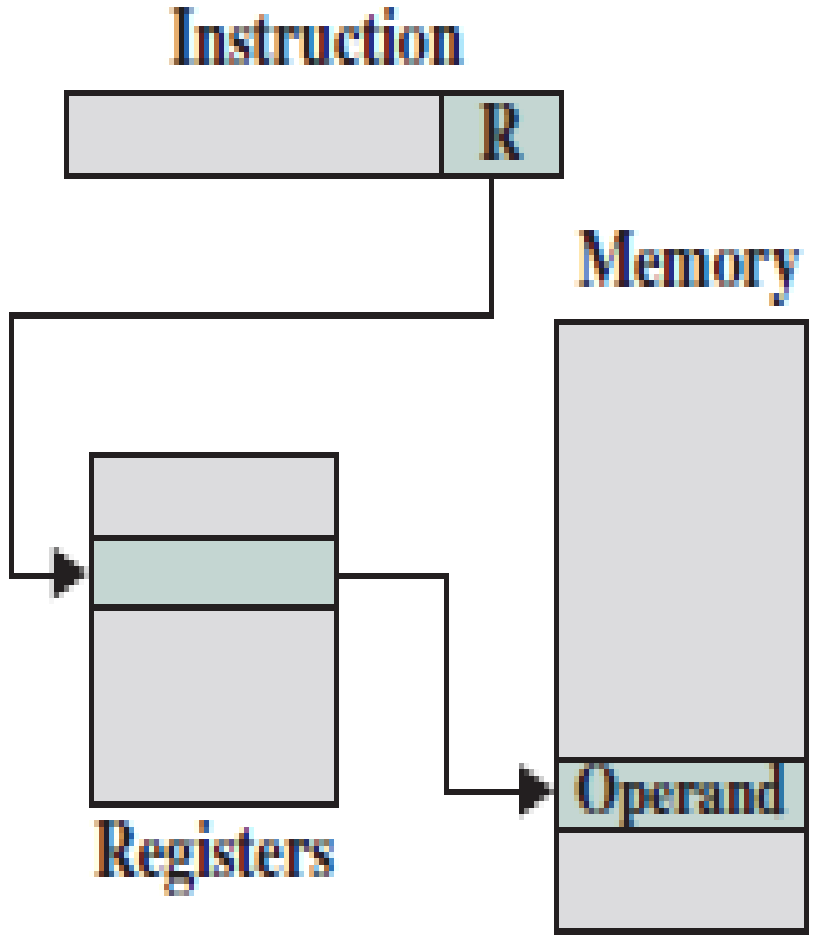
Register Addressing

- Operand is held in register named in address filed
- 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;



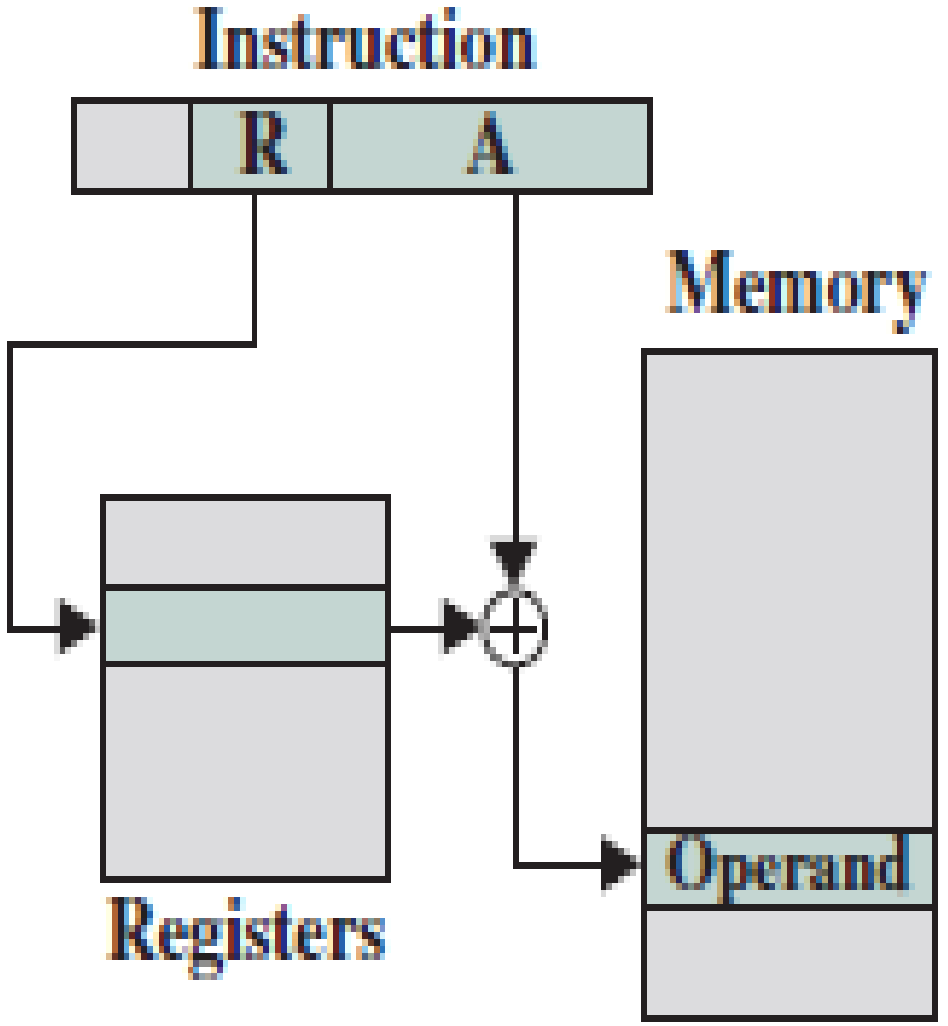
Register Indirect Addressing

- 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 = A + (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

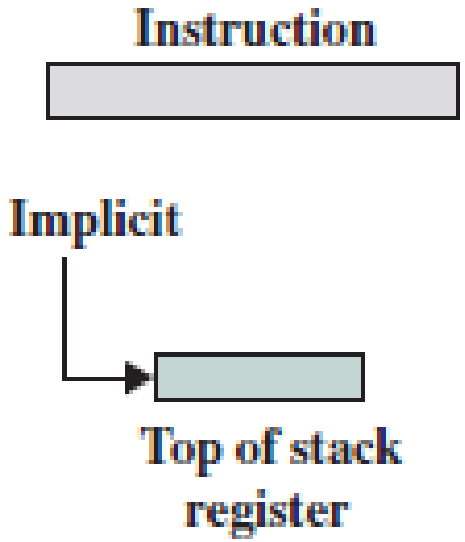
- The referenced register "R" contains a main memory address
- address field contains a displacement A
- R may be explicit or implicit
- e.g. segment registers in 80x86

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++

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



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
- Register versus memory
- Number of register sets
- Address range
- Address granularity

CISC Vs RISC

#	RISC	CISC
1.	Reduced Instruction Set Computer.	Complex Instruction Set Computer.
2.	Fixed length instructions	Variable length instructions
3.	Instructions are executed in one clock cycle.	Takes one or more clock cycle
4.	Relatively simple to design.	Complex to design.
5.	Fewer, Simple addressing modes	Many addressing modes
6.	Hardwired Control Unit	Microprogrammed Control Unit
7.	Harvard Architecture	Von-Neumann Architecture
8.	Examples: SPARC, POWER PC.	Examples: Intel architecture(x86 and x64) AMD.

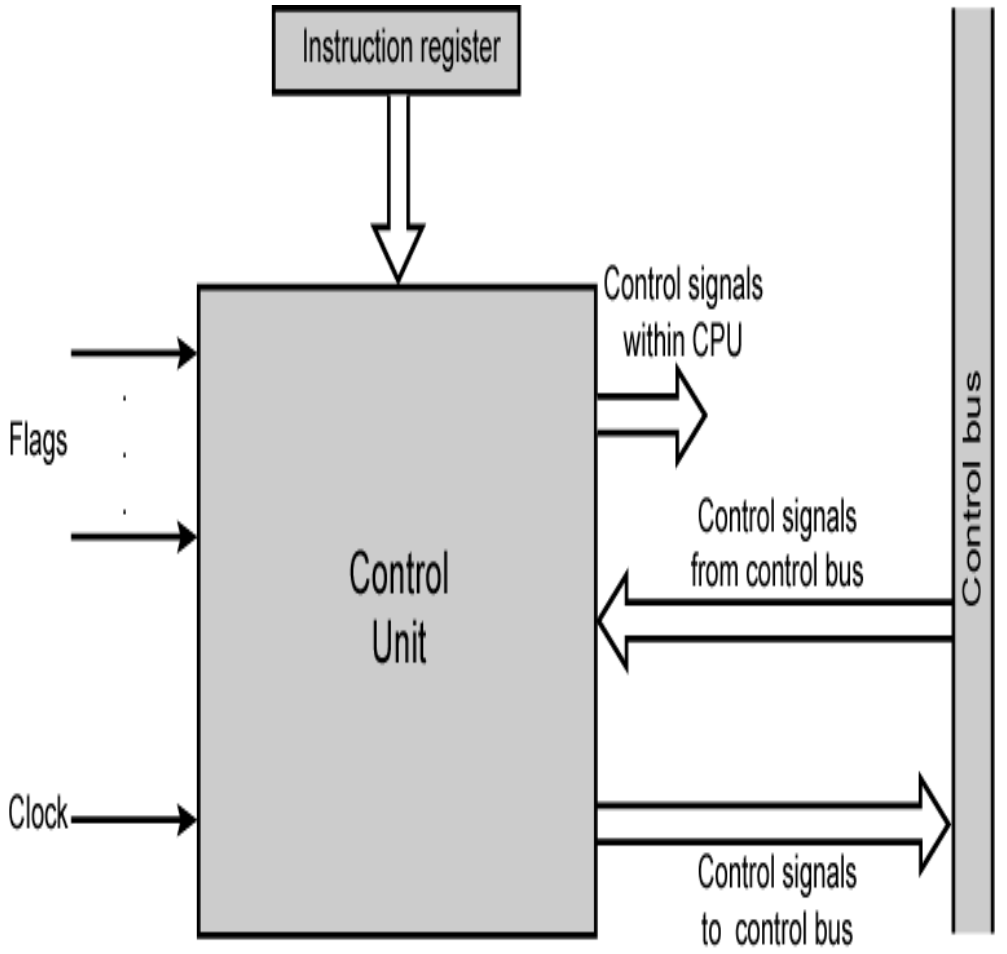
Control Unit implementation

Hardwired control unit (RISC)

Microprogrammed control unit(CISC)

Hardwired Implementation (1)

- 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
- Time efficient



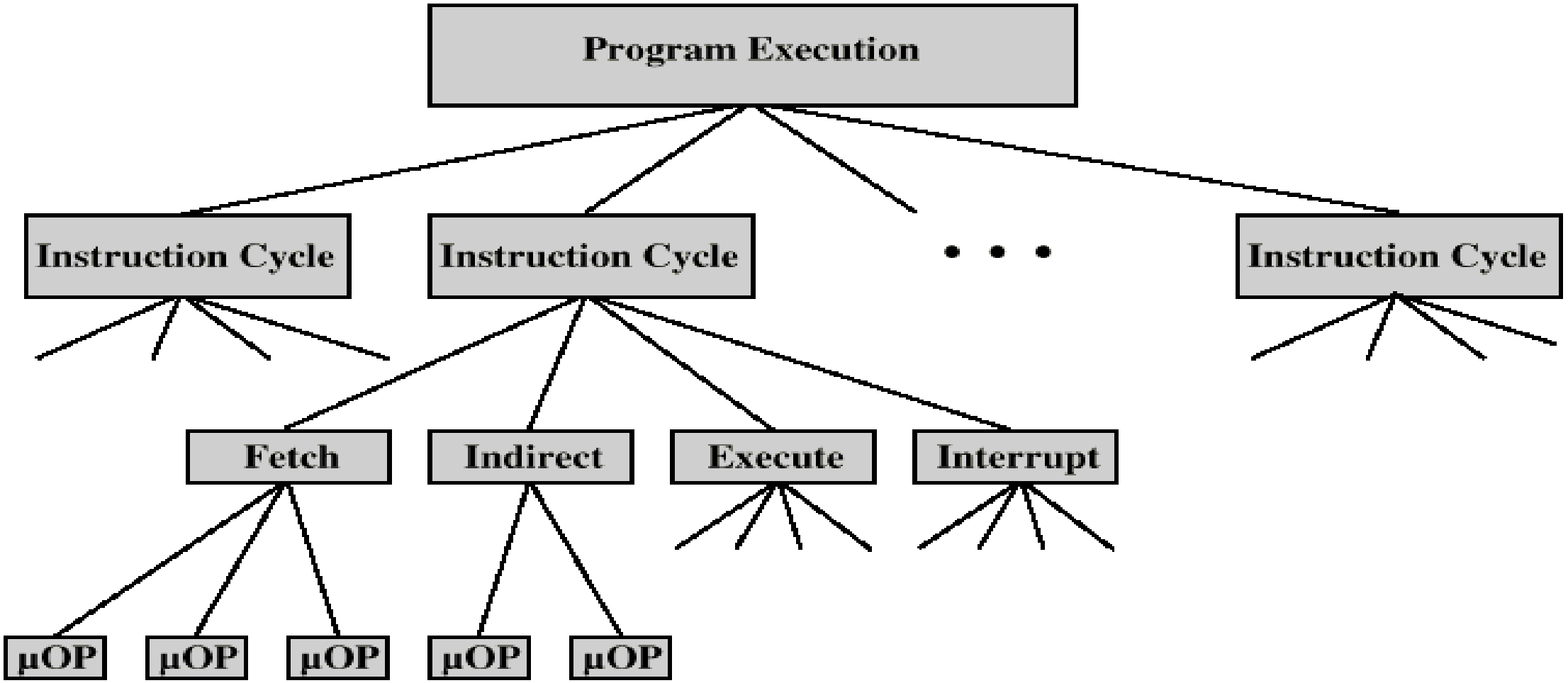
Problems With Hardwired Designs

- 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
- Each cycle has a number of steps
  - Called micro-operations
  - Each step does very little
  - Atomic operation of CPU

Constituent Elements of Program Execution



Example: Fetch Sequence (ADD R1,X)

t1: MAR ← (PC)
t2: MBR ← (memory)
PC ← (PC) +1
t3: IR ← (MBR)

OR

t1: MAR <- (PC)
t2: MBR <- (memory)
t3: PC <- (PC) +1
IR <- (MBR)

Example: Execute Cycle (ADD R1, X)

- Different for each instruction
- e.g. ADD R1,X - add the contents of location X to Register 1 , result in R1

t1: MAR ← ($IR_{address}$)

t2: MBR ← (memory)

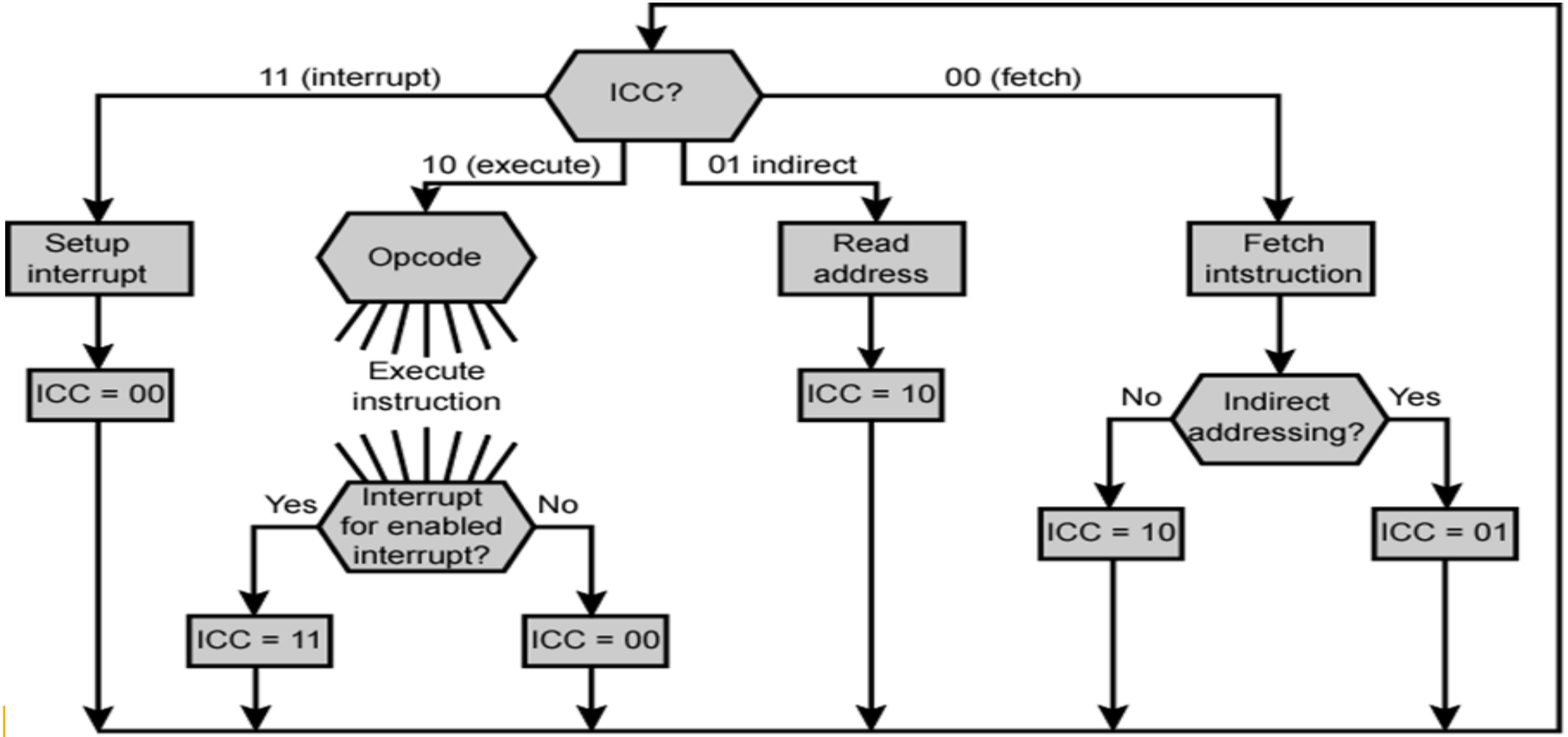
t3: R1 ← R1 + (MBR)

- Note no overlap of micro-operations

Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- 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
- This is done using Control Signals

Example: ADD R1, X

t1: MAR ← (PC)

t2: MBR ← (memory)

PC ← (PC) +1

t3: IR ← (MBR)

t4: MAR ← ($IR_{address}$)

t5: MBR ← (memory)

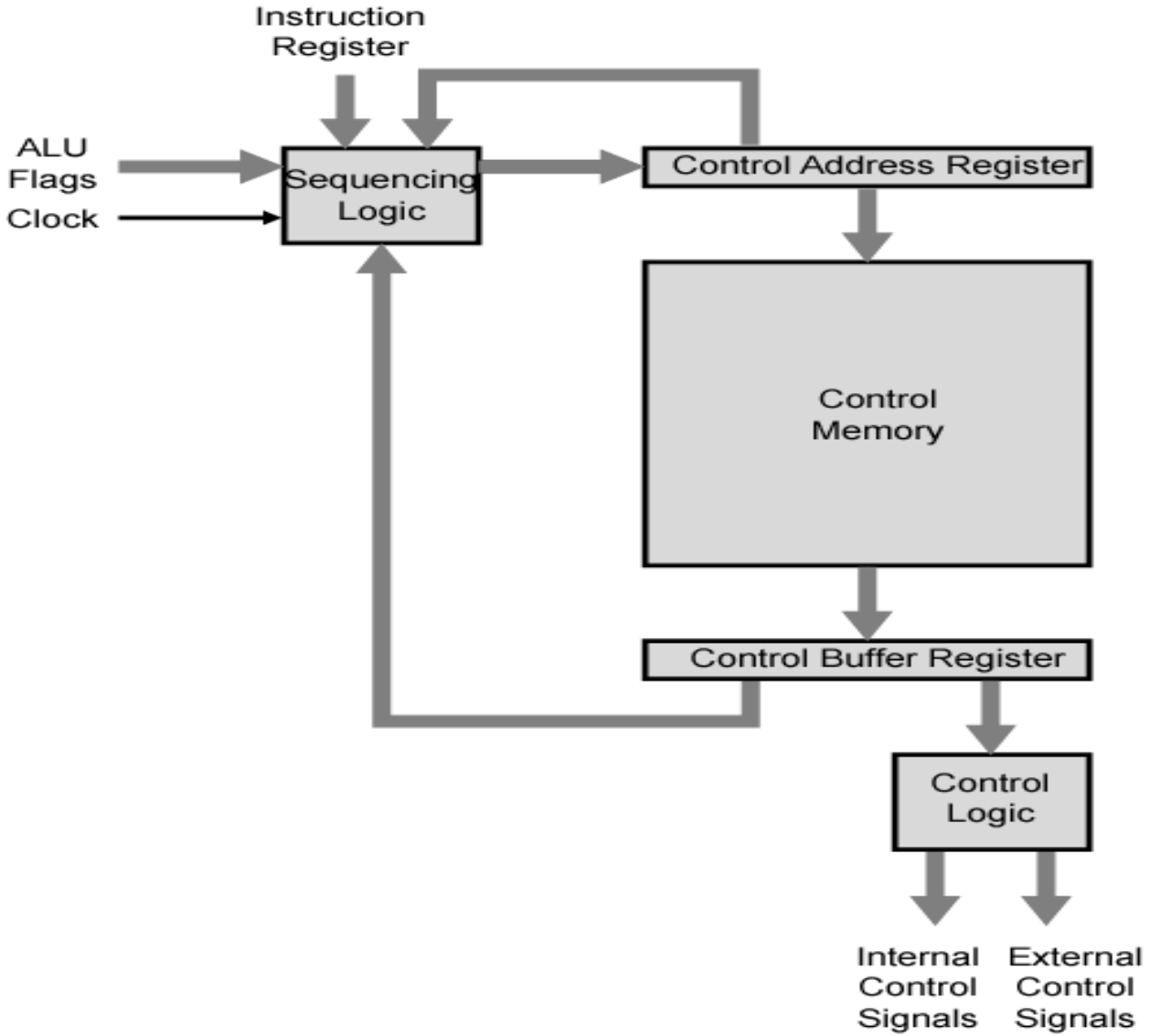
t6: R1 ← R1 + (MBR)

Control Signals

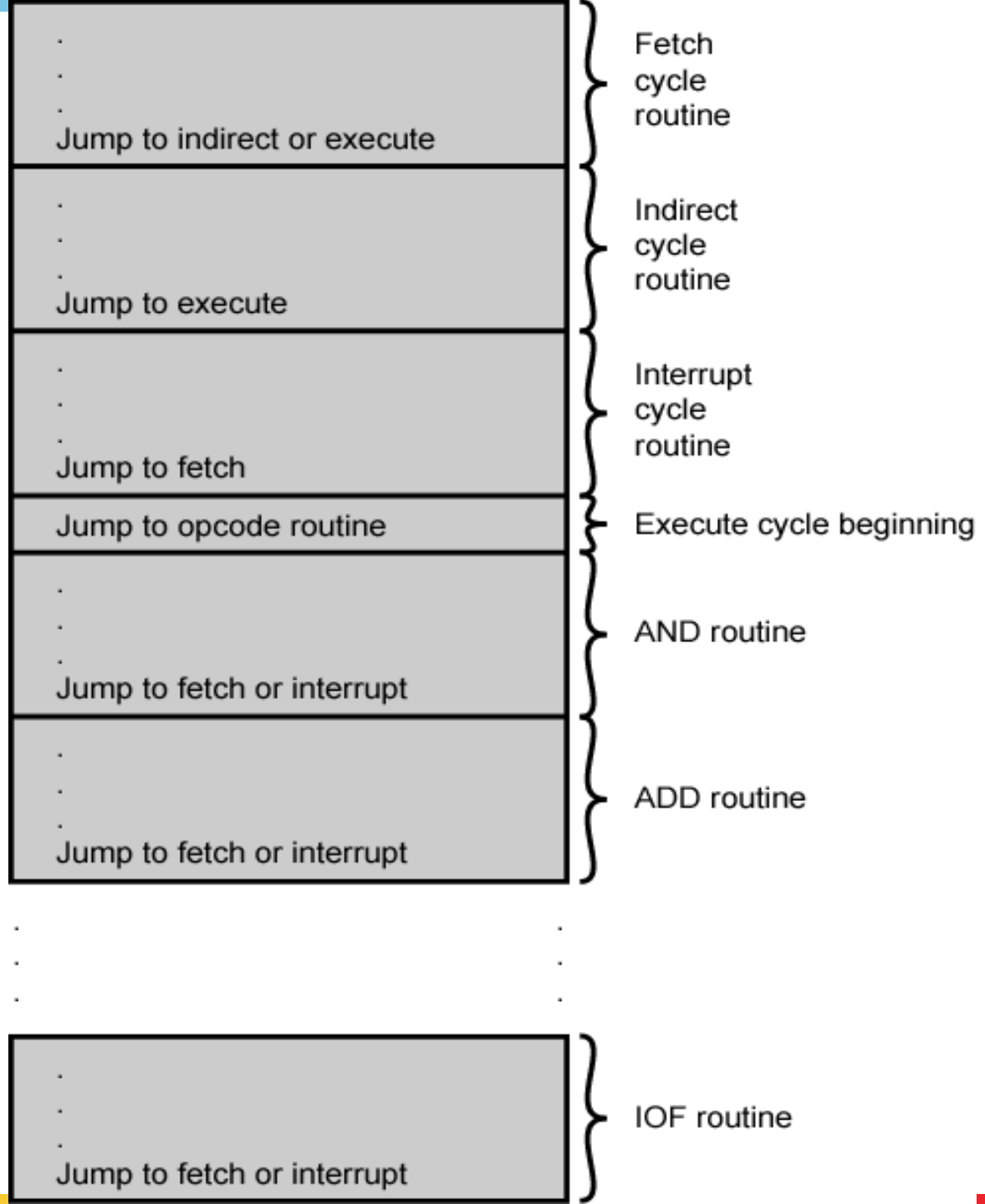
- Inputs to the control unit
  - Clock
    - One micro-instruction (or set of parallel micro-instructions) per clock cycle
  - Instruction register
    - Op-code for current instruction
    - Determines which micro-instructions are performed
  - Flags
    - State of CPU
    - Results of previous operations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

Microprogrammed Control Unit

- Use sequences of micro-instructions to control complex operations called micro-programming or firmware
- Firmware is midway between hardware and software



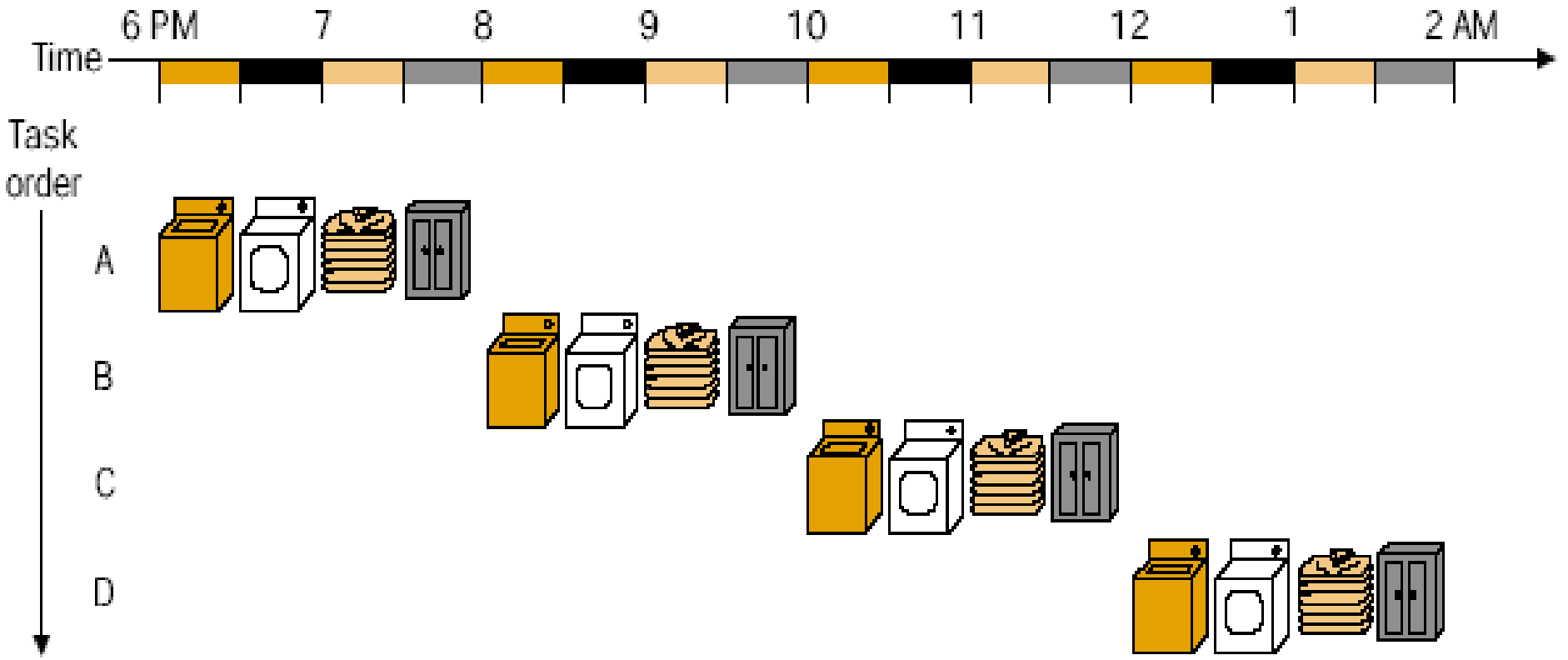
Organization of Control Memory



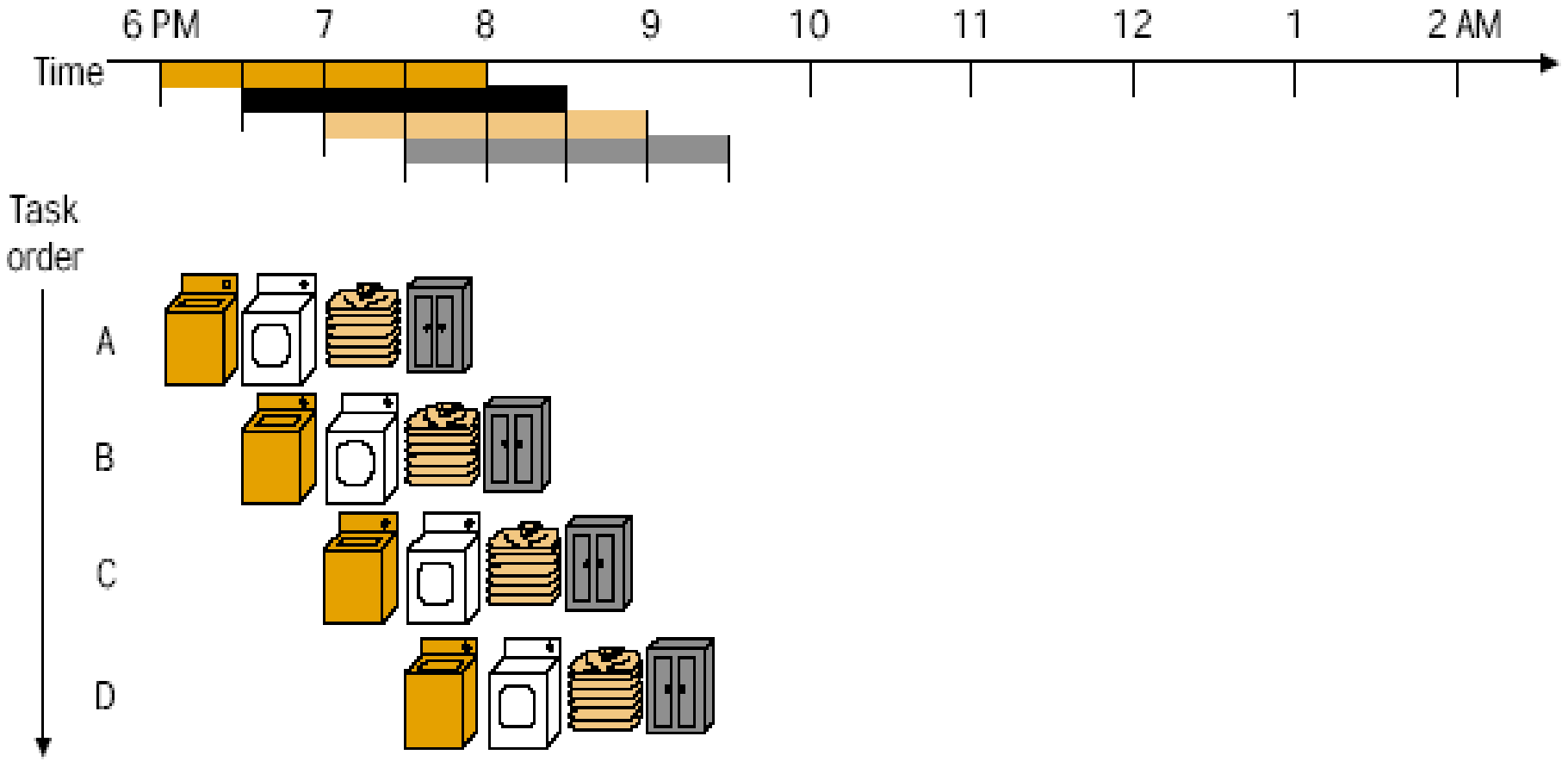
innovate achieve lead

Pipeline

Laundry System



Laundry System……



Pipelining

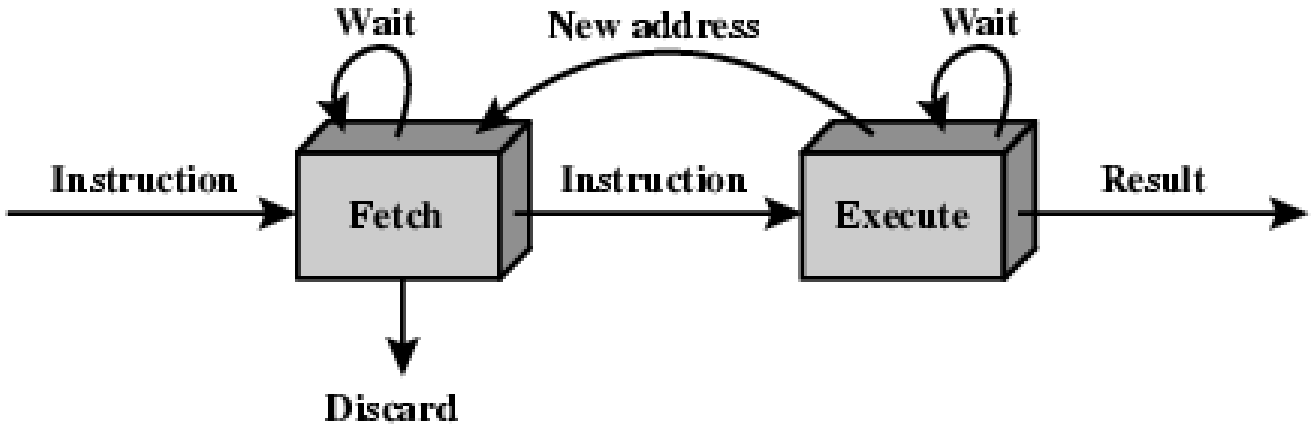
- An overlapped parallelism: overlapped execution of multiple operations
- Pipelining
  - Subdivide the input task into a sequence of subtasks
  - Specialized hardware stages
  - Concurrent operation of all the stages

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.

Computer Organization and Software Systems

Contact Session 8

Dr. Lucy J. Gudino

Pipeline

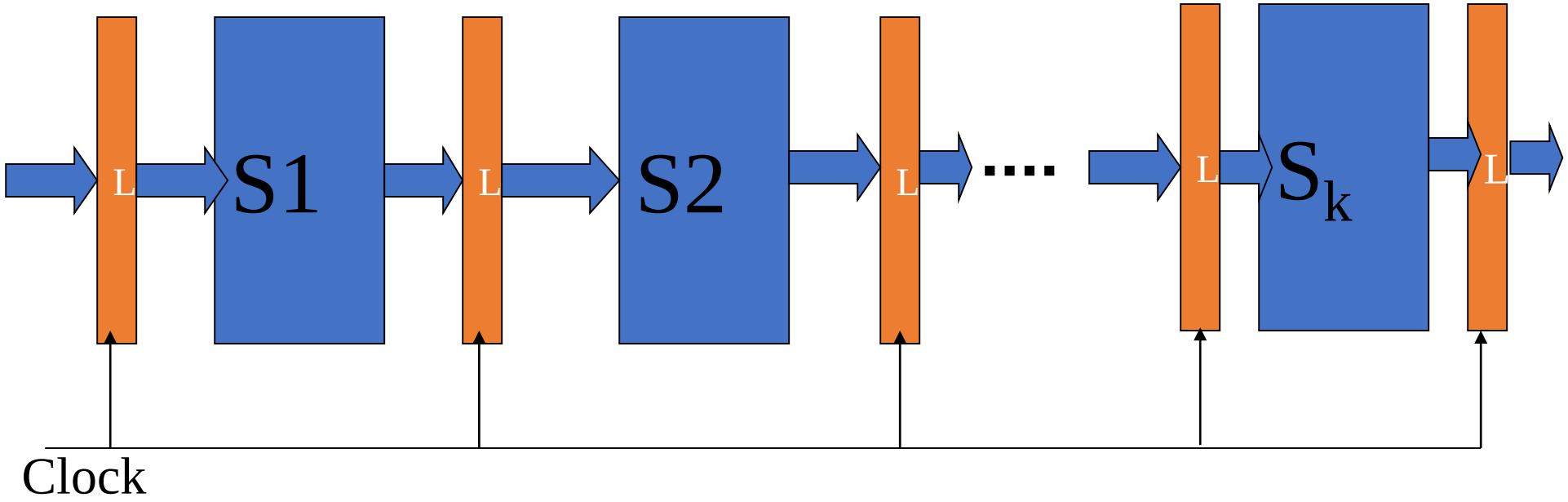
Four Segment instruction pipeline

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

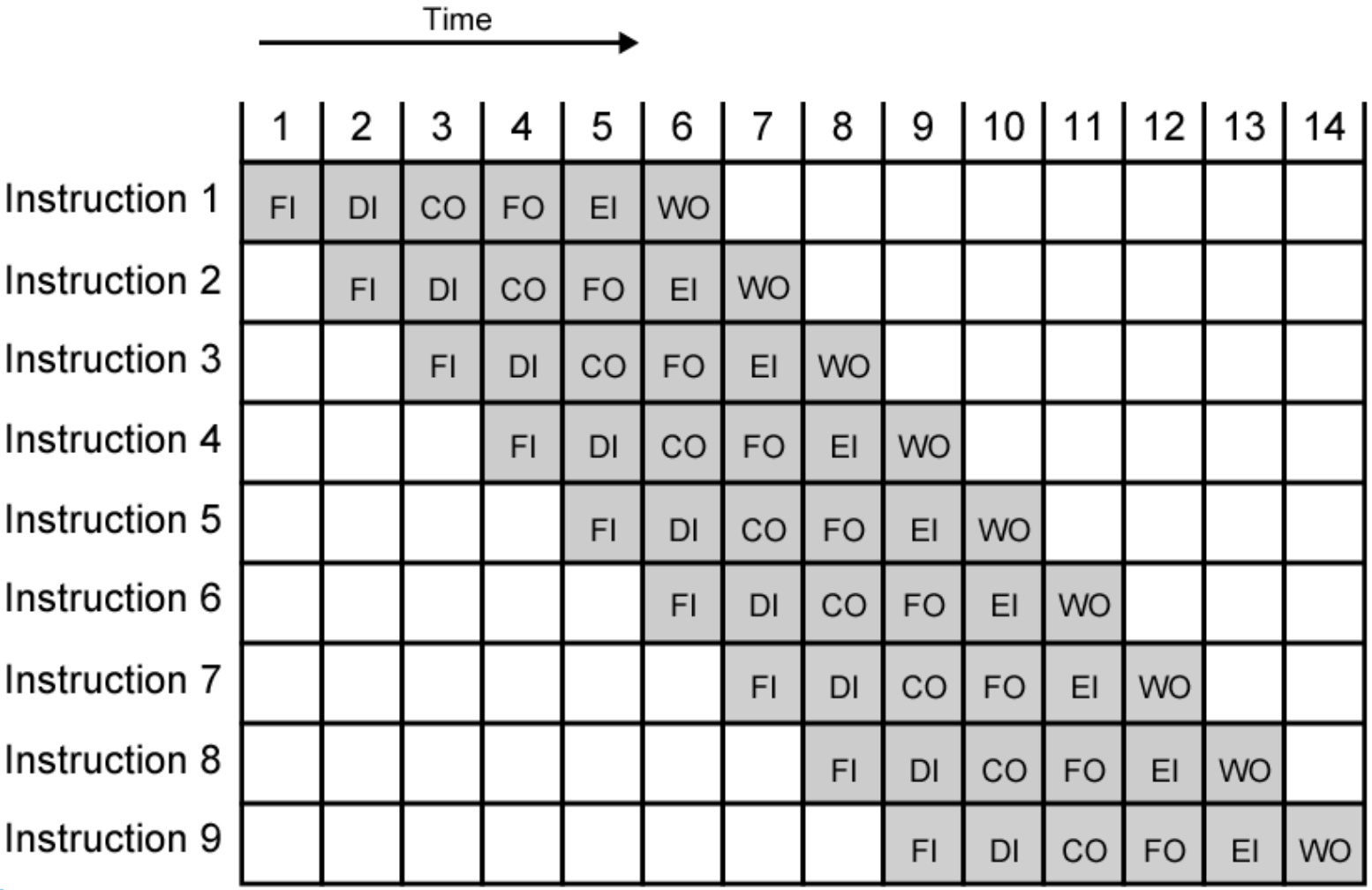
Six stage pipeline

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

Structure of a Pipeline



Timing Diagram for Instruction Pipeline Operation



Classification

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

Arithmetic pipelining

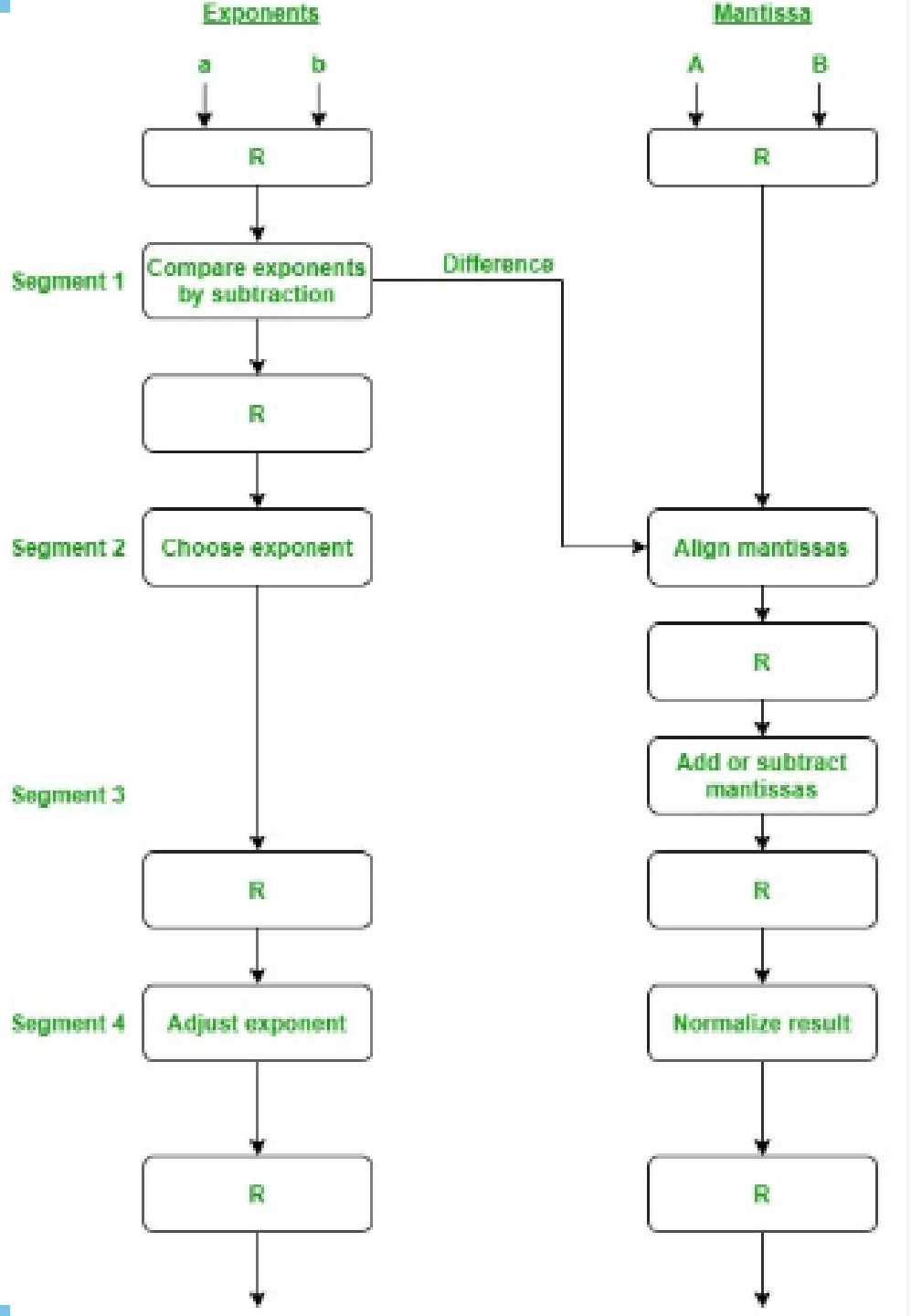
- Arithmetic and logic units of a computer can be segmented for pipeline operations
- Usually found in high speed computers
- Example:
  - Star 100 → 4 stage
  - TI-ASC → 8 stage
  - Cray-1 → 14 stage
  - Cyber 205 → 26 stages
  - Intel Cooper Lake (3rd Gen Intel Xeon) = 14 stages
- Floating point adder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

$$X = A*2^a$$

$$Y = B*2^b$$

Floating point addition

1. Compare the exponents
2. Align the mantissas
3. Add or subtract the mantissas
4. Normalize the result



Numerical Example:

$X = 0.9504*10^3$

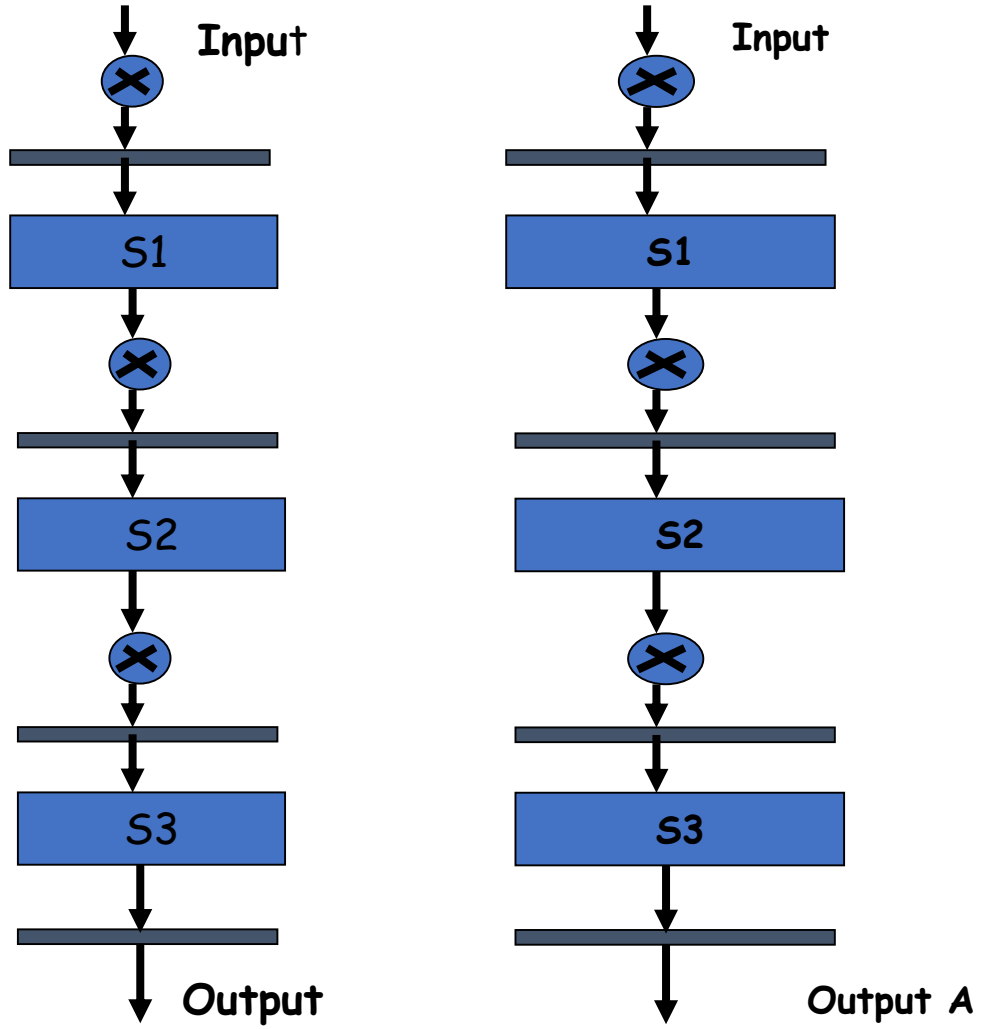
$Y = 0.8200*10^2$

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

Unifunction and multifunction pipelining

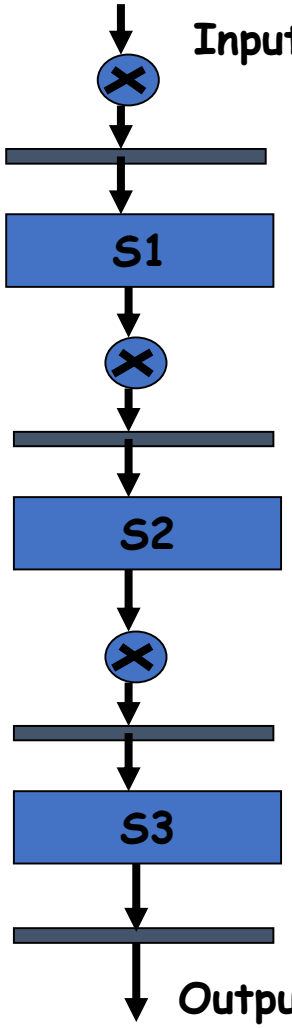
- Unifunction
  - Pipeline with a fixed and dedicated function
  - Ex: Floating point adder
- Multifunction
  - Pipeline may perform different functions



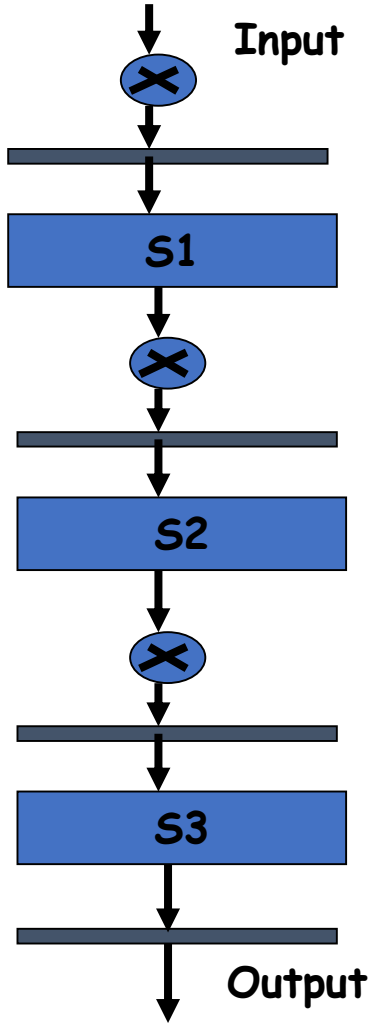
Reservation Table

- 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



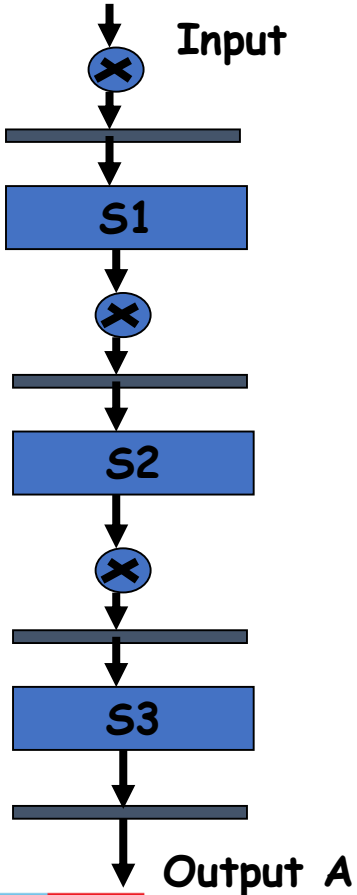
	T0	T1	T2
S1	A		
S2		A	
S3			A

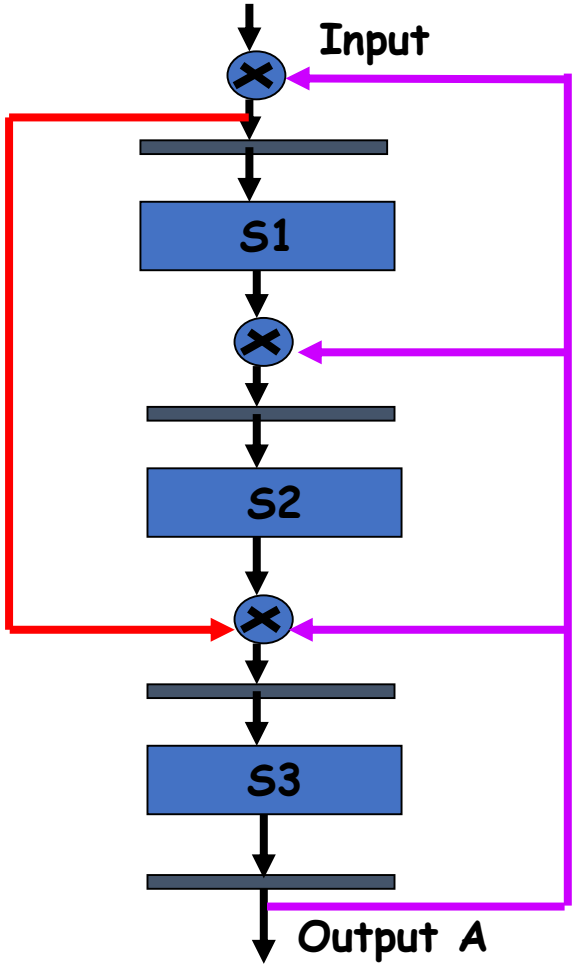
	T0	T1
S1	B	
S2		B

innovate achieve lead

Linear and Nonlinear Pipelines

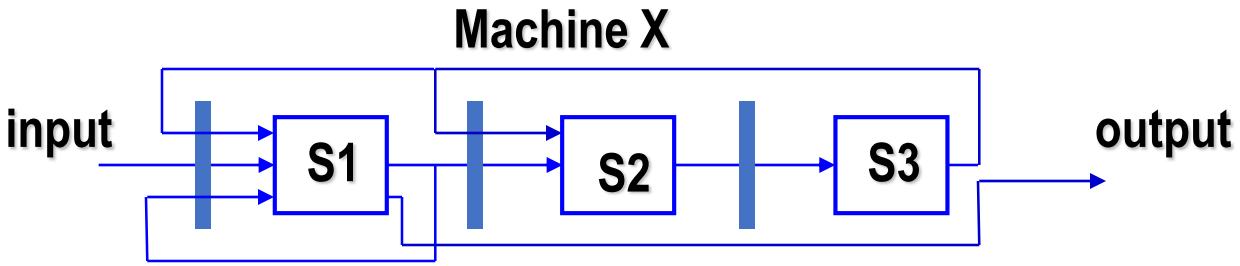
- Linear Pipeline: Without feed forward and feed back connection
- Nonlinear Pipeline with feed forward and/or feed back connection





innovate achieve lead

Reservation Table



Reservation Table

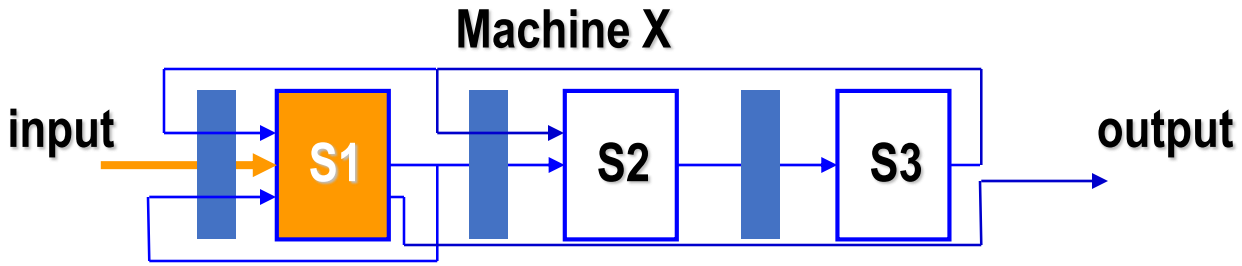
Time →

Stage ↓

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

innovate achieve lead

Reservation Table



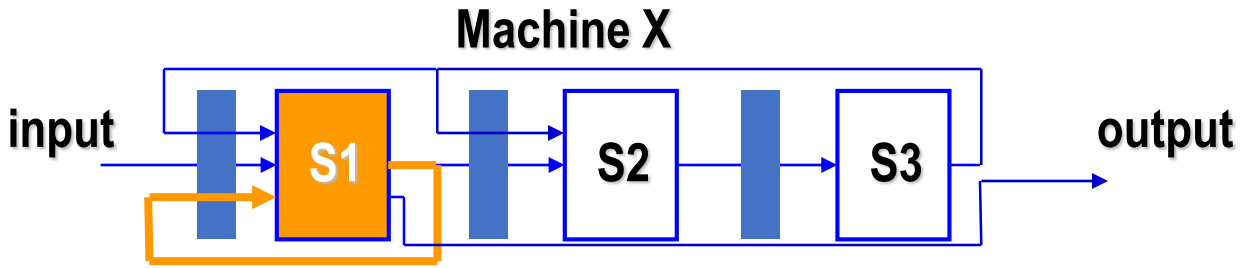
Reservation Table

Time →

Stage ↓

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

Reservation Table



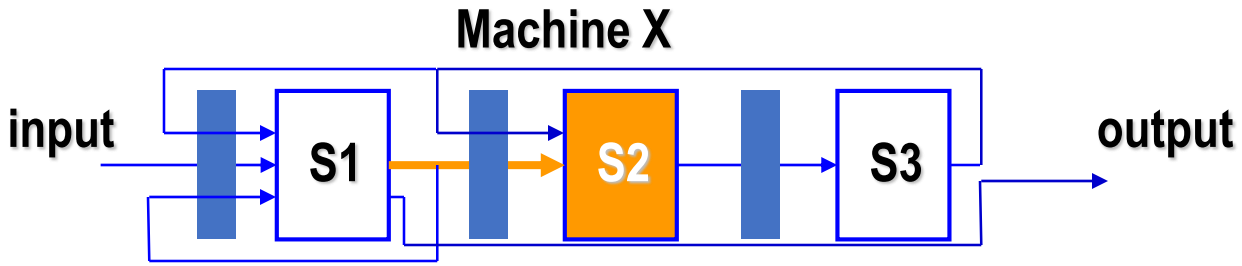
Reservation Table

Time →

Stage ↓

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

Reservation Table



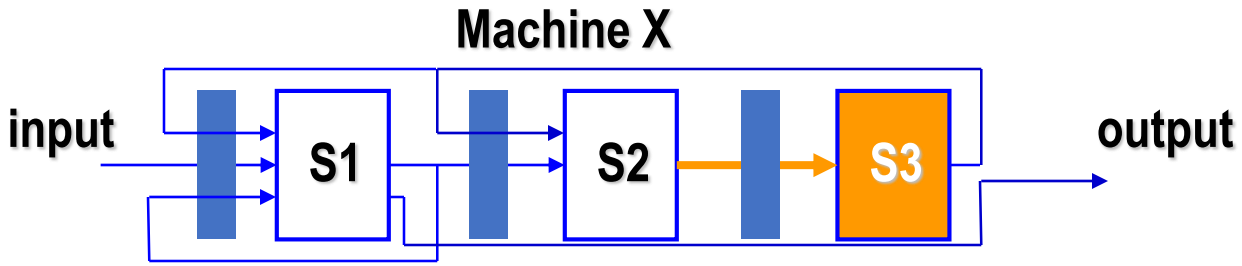
Reservation Table

Time →

Stage ↓

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

Reservation Table



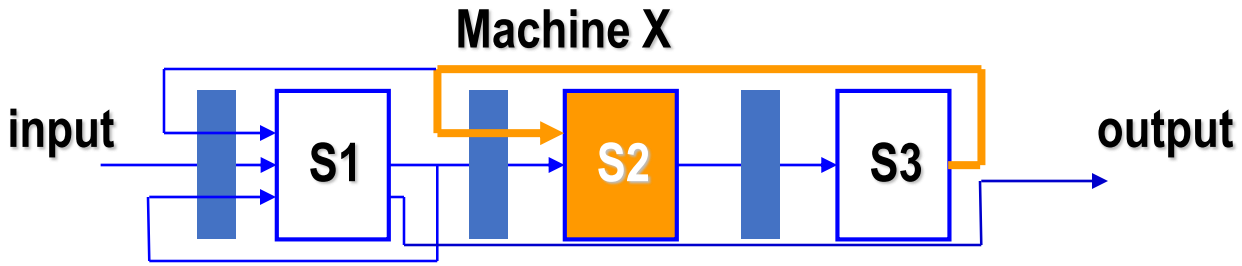
Reservation Table

Time →

Stage ↓

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

Reservation Table



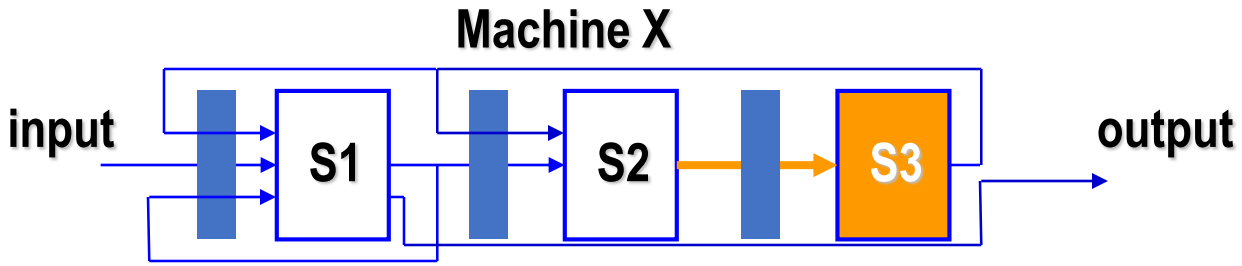
Reservation Table

Time →

Stage ↓

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

Reservation Table



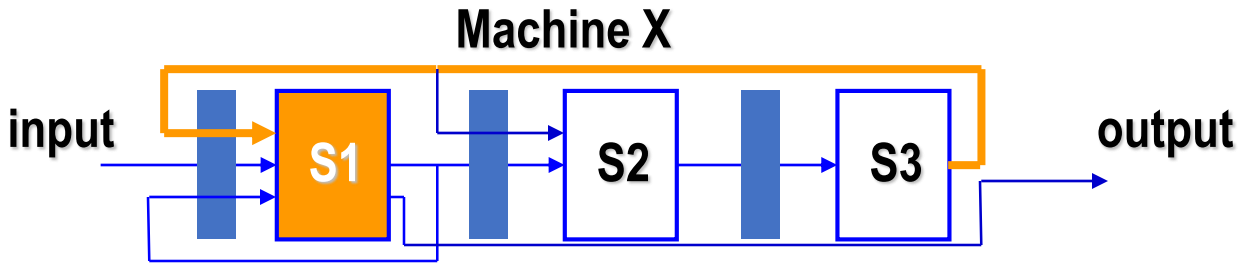
Reservation Table

Time →

Stage ↓

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

Reservation Table



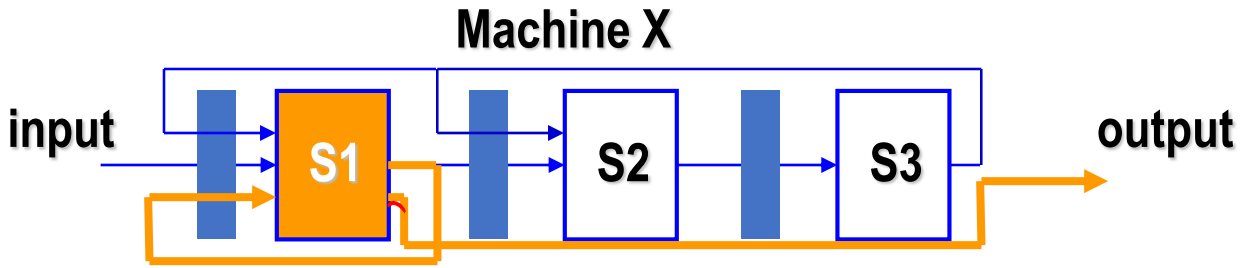
Reservation Table

Time →

Stage ↓

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

Reservation Table



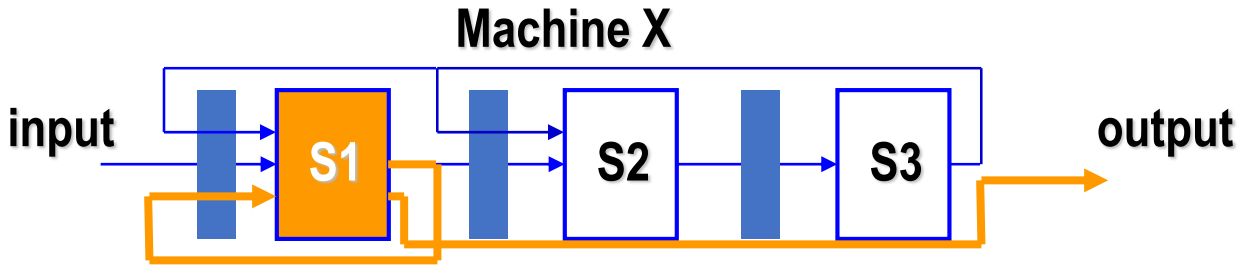
Reservation Table

Time →

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 →

Stage ↓

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

19 cycles

24 cycles

Time span diagram A, B, C

Stage ↓	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
S1	A	A		B	B		A	A		B	B	C	C					C	C	
S2			A		A	B		B						C		C				
S3				A		A	B		B						C		C			

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.

Important Terms

Clock period: τ

τ_i: time delay of S_i stage

τ_L: time delay of latch

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

Pipeline processor frequency $f = 1/\tau$

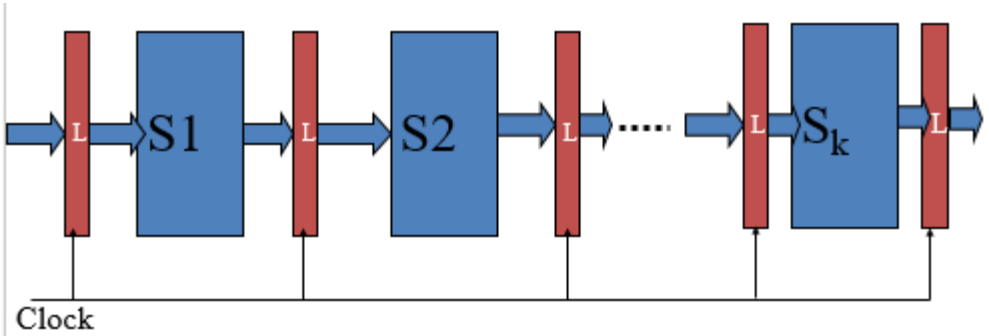


Fig: Structure of a pipeline

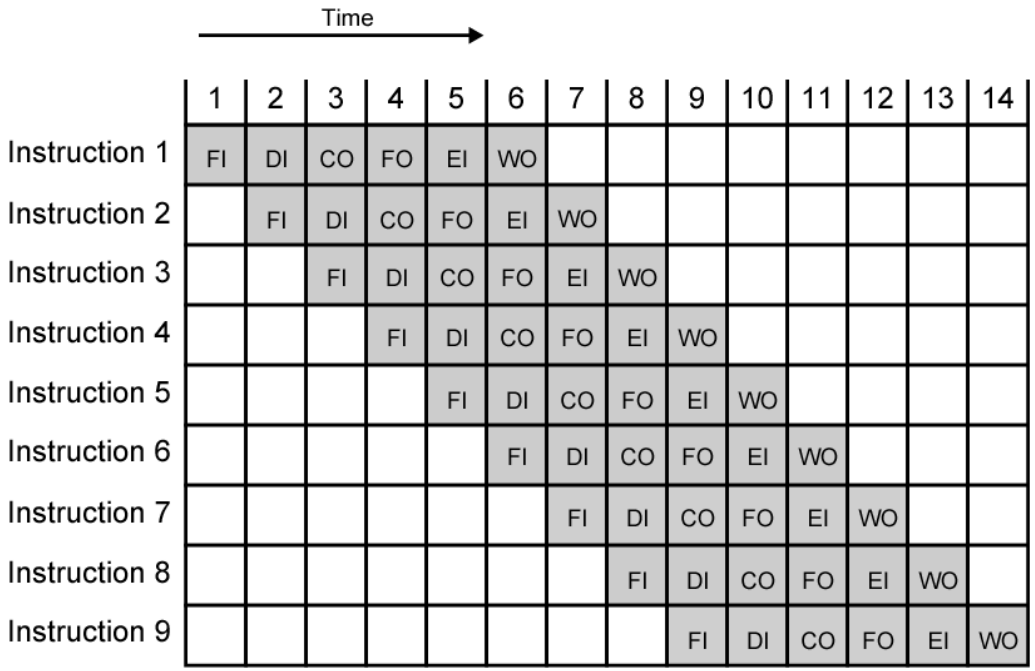
Important Terms

Time taken to complete n tasks by k stage pipeline is

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

Time taken by the nonpipelined processor ?

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



Important Terms

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

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

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

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

- The maximum speedup is $S_k \rightarrow k$ when $n \rightarrow INF$
- 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

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

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

- Maximum efficiency

$$\eta \to 1 \text{ as } n \to \infty$$

- 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 n >> 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

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

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

	1	2	3	4	5	6	7	8	9	10	11	12	13	
S1	T1	T2	T3	T4	T5	T6	T7	T8						
S2		T1	T2	T3	T4	T5	T6	T7	T8					
S3			T1	T2	T3	T4	T5	T6	T7	T8				
S4				T1	T2	T3	T4	T5	T6	T7	T8			
S5					T1	T2	T3	T4	T5	T6	T7	T8		
S6						T1	T2	T3	T4	T5	T6	T7	18	

Problem 2

0.0909 Mega/sec

Assume each task is subdivided in to 6 subtasks and clock cycle is 10 microseconds.

- Determine the number of clock cycles that is taken to process 50 tasks.

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

$= (6 + 49) \times 10\,\mu s \Rightarrow 550\,\mu s$

- Determine the number of clock cycles that is taken to process 50 tasks in non- pipeline processor

$T_1 \quad k \times n * \tau = 6 \times 50 \times 10\,\mu s$

$3000\,\mu s \Rightarrow 3\,ms$

- Compute speed up, efficiency and throughput

$\text{Speedup} \frac{T_1}{T_k} = \frac{3000}{550} = \frac{\quad}{\quad}\mu s \Rightarrow 5.454$

$\text{efficiency} = \frac{\text{speedup}}{k} = \frac{5.454}{6} = 0.909 \Rightarrow 90.9\%$

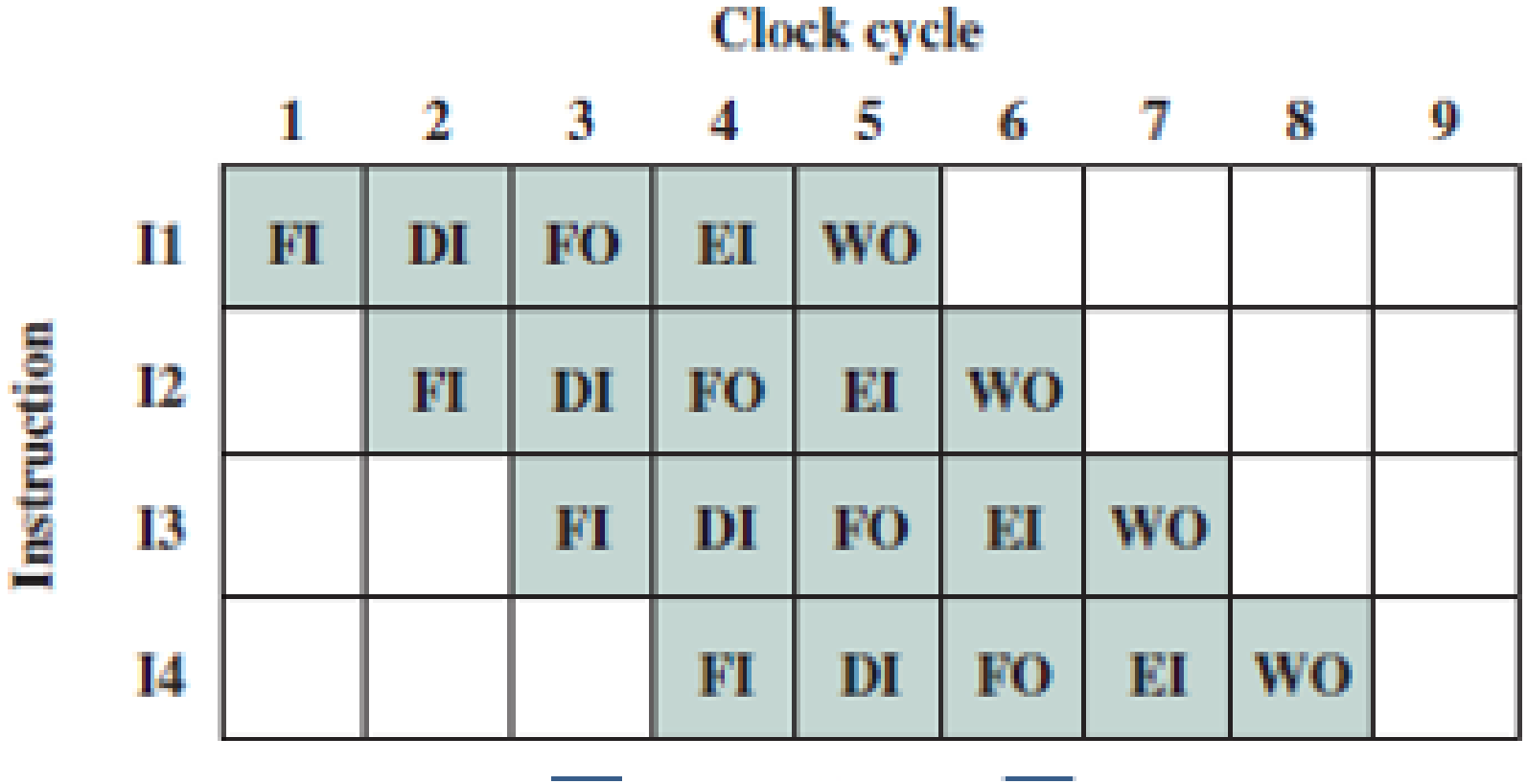
$\text{Throughput } H_k = n \times f$

$= \frac{0.909n}{10\,ns = t}$

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



Problem

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

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

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

Data Dependency Conflict

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

time	1	2	3	4	5	6	7	8	9	10
I1	FI	DI	FO	EI	WO					
I2		FI	DI	FO	EI	WO				
I3			FI	DI	FO	EI	WO			
I4				FI	DI	FO	EI	WO		
I5					FI	DI	FO	EI	WO	
I6						FI	DI	FO	EI	WO

Problem

```
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 time – space diagram for instruction pipeline**

The Effect of a Conditional Branch on Instruction Pipeline Operation

	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

innovate achieve lead