Computer Organization

Prepared By:-

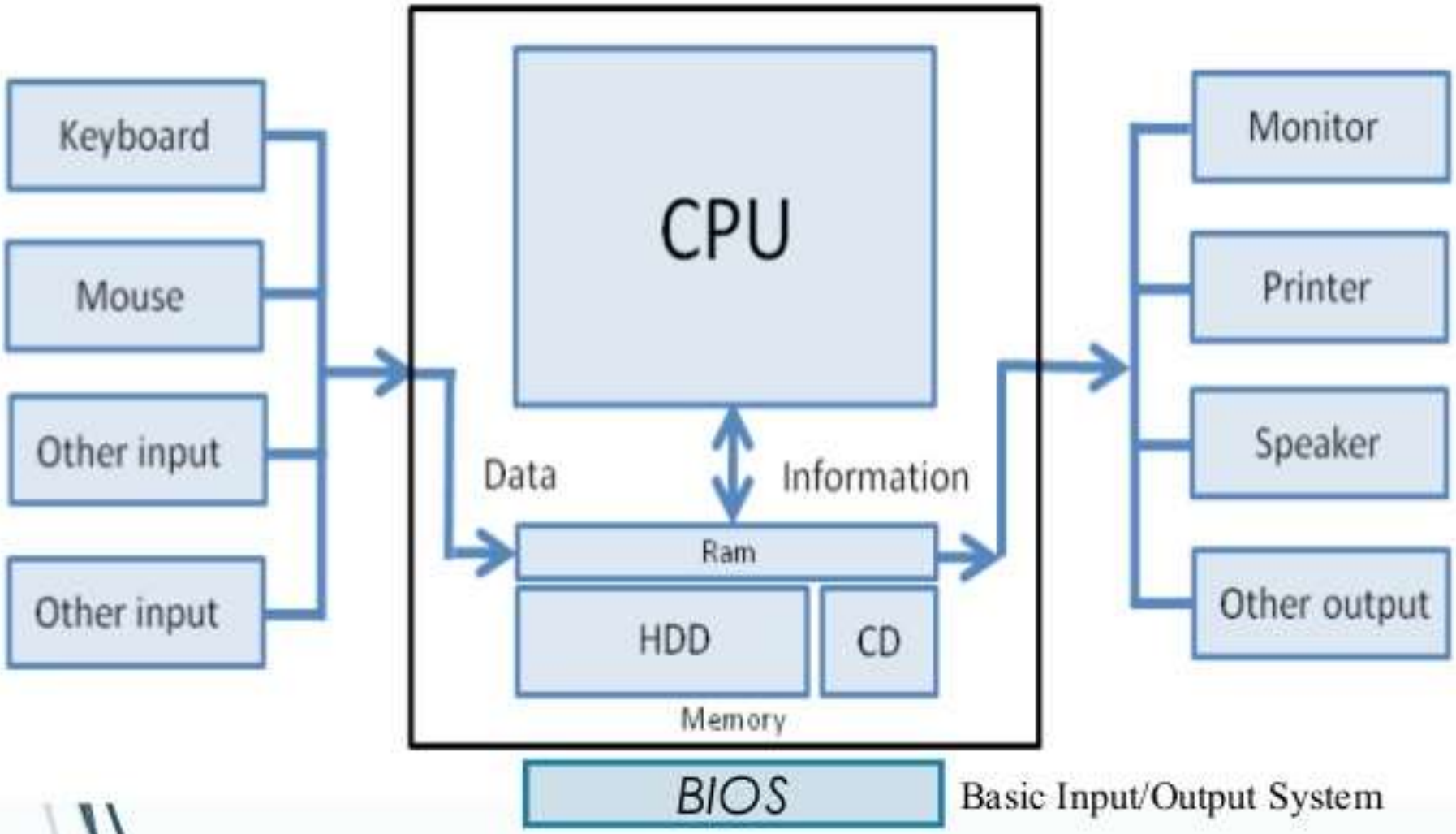
Parminder Mann

Objectives

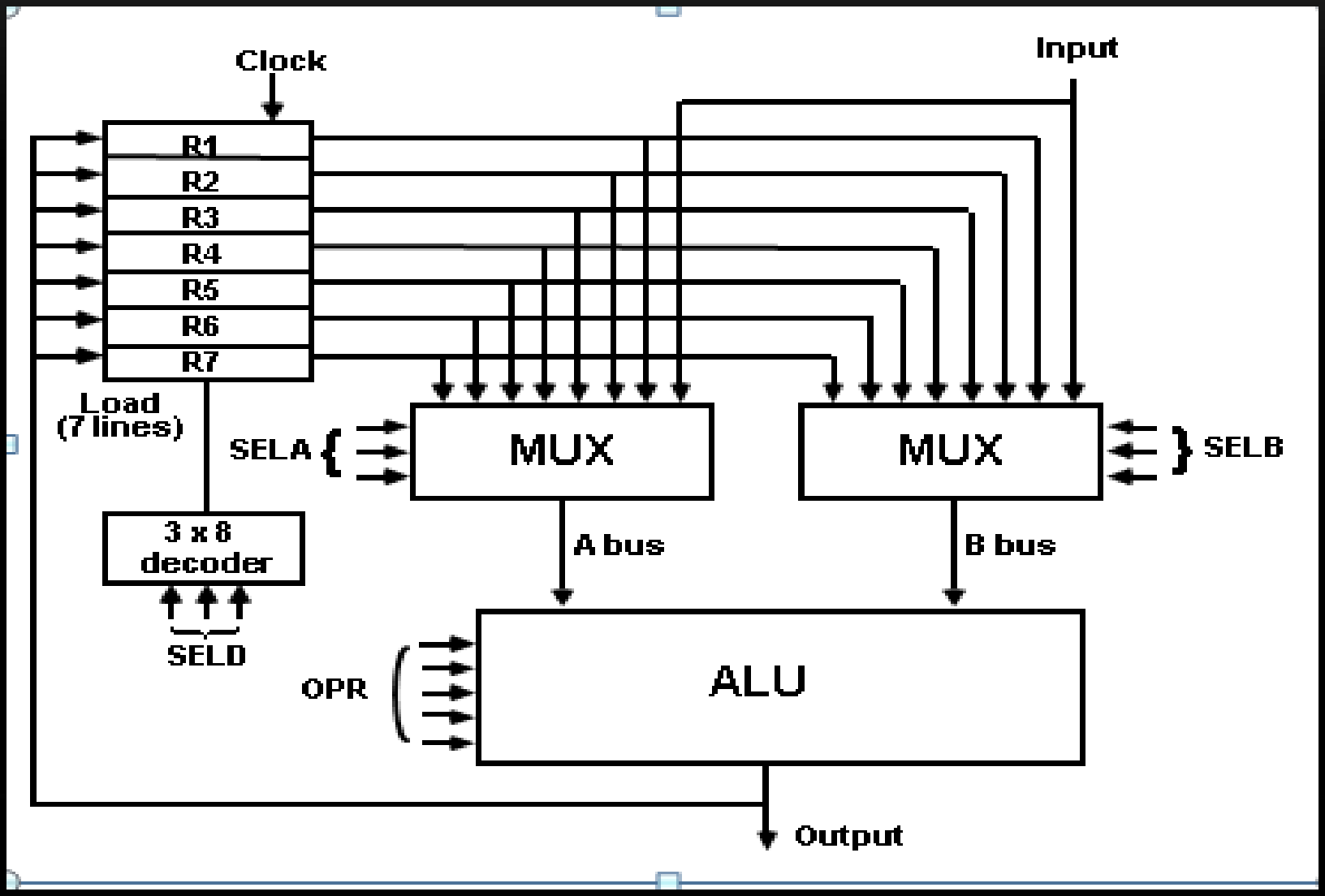
After reading this chapter, the reader should be able to:

- Distinguish between the three components of a computer hardware.
- List the functionality of each component.
- Understand memory addressing and calculate the number of bytes for a specified purpose.
- Distinguish between different types of memories.
- Understand how each input/output device works.

Computer System Hardware



GENERAL REGISTER ORGANISATION



ADDRESSING MODES

- Addressing Modes:

- * Specifies a rule for interpreting or modifying the
- address field of the instruction (before the operand
- is actually referenced)
-
- * Variety of addressing modes

- - to give programming flexibility to the user
- - to use the bits in the address field of the
- instruction efficiently
-

TYPES OF ADDRESSING MODES

Implied Mode

Address of the operands are specified implicitly
in the definition of the instruction
- No need to specify address in the instruction
- EA = AC, or EA = Stack[SP], EA: Effective Address.

Immediate Mode

Instead of specifying the address of the operand,
operand itself is specified
- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

Register Mode

Address specified in the instruction is the register address
- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)

TYPES OF ADDRESSING MODES

Register Indirect Mode

Instruction specifies a register which contains the memory address of the operand

- Saving instruction bits since register address is shorter than the memory address
- Slower to acquire an operand than both the register addressing or memory addressing
- EA = [IR(R)] ([x]: Content of x)

Auto-increment or Auto-decrement features:

Same as the Register Indirect, but:

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 (after or before the execution of the instruction)

TYPES OF ADDRESSING MODES

Direct Address Mode

Instruction specifies the memory address which can be used directly to the physical memory

- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory space
- EA = IR(address), (IR(address): address field of IR)

Indirect Addressing Mode

The address field of an instruction specifies the address of a memory location that contains the address of the operand

- When the abbreviated address is used, large physical memory can be addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory access
- EA = M[IR(address)]

PROGRAM INTERRUPT

Types of Interrupts:

External interrupts

External Interrupts initiated from the outside of CPU and Memory
- I/O Device -> Data transfer request or Data transfer complete
- Timing Device -> Timeout
- Power Failure

Internal interrupts (traps)

Internal Interrupts are caused by the currently running program
- Register, Stack Overflow
- Divide by zero
- OP-code Violation
- Protection Violation

Software Interrupts

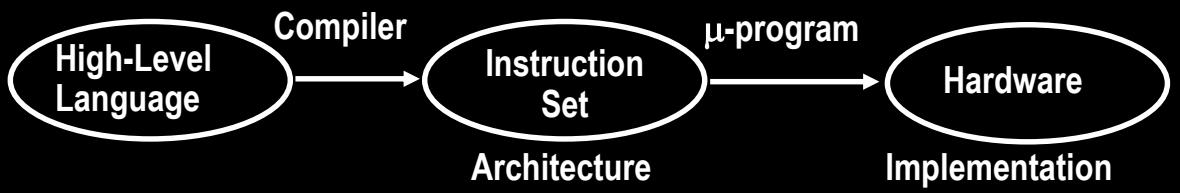
Both External and Internal Interrupts are initiated by the computer Hardware.
Software Interrupts are initiated by texecuting an instruction.
- Supervisor Call -> Switching from a user mode to the supervisor mode
  -> Allows to execute a certain class of operations
  which are not allowed in the user mode

RISC: REDUCED INSTRUCTION SET COMPUTERS

Historical Background

IBM System/360, 1964

- The real beginning of modern computer architecture
- Distinction between *Architecture* and *Implementation*
- Architecture: The abstract structure of a computer seen by an assembly-language programmer



Continuing growth in semiconductor memory and microprogramming
-> A much richer and complicated instruction sets
=> CISC(Complex Instruction Set Computer)

- Arguments advanced at that time
  - Richer instruction sets would simplify compilers
  - Richer instruction sets would alleviate the software crisis
    - move as much functions to the hardware as possible
    - close *Semantic Gap* between machine language and the high-level language
  - Richer instruction sets would improve the *architecture quality*

CHARACTERISTICS OF RISC

Common RISC Characteristics

- Operations are register-to-register, with only LOAD and STORE accessing memory
- The operations and addressing modes are reduced

Instruction formats are simple

More characteristics are as:

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction format
- Hardwired rather than microprogrammed control

COMPLEX INSTRUCTION SET COMPUTERS: CISC

High Performance General Purpose Instructions

Characteristics of CISC:

1. A large number of instructions (from 100-250 usually)
2. Some instructions that performs a certain tasks are not used frequently.
3. Many addressing modes are used (5 to 20)
4. Variable length instruction format.
5. Instructions that manipulate operands in memory.

MAJOR COMPONENTS OF CPU

Storage Components:

- Registers
- Flip-flops

Execution (Processing) Components:

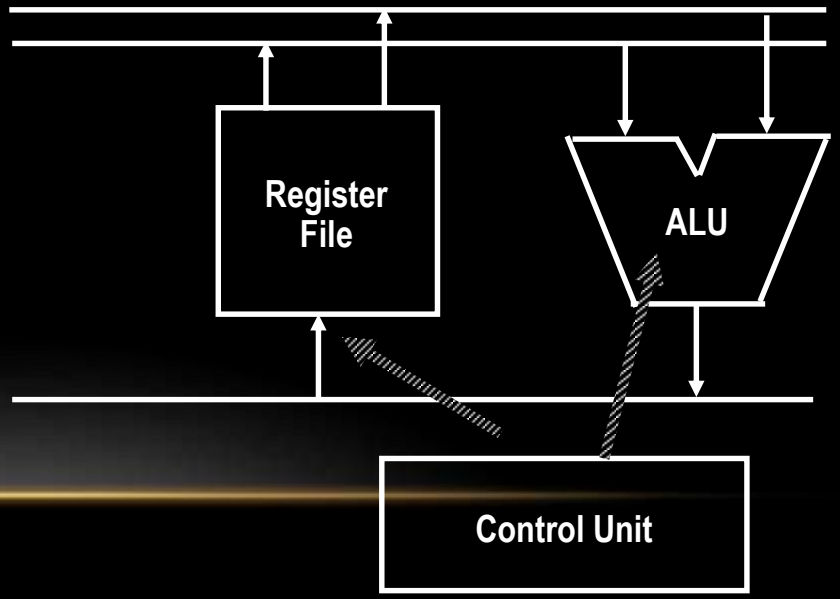
- Arithmetic Logic Unit (ALU):
- Arithmetic calculations, Logical computations, Shifts/Rotates

Transfer Components:

- Bus

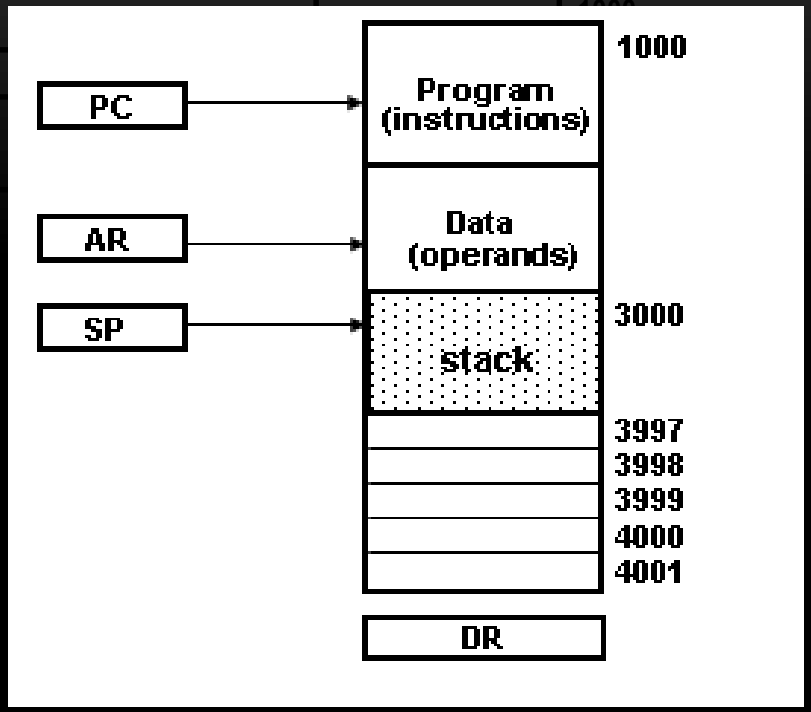
Control Components:

- Control Unit



MEMORY STACK ORGANIZATION

Memory with Program, Data, and Stack Segments



- A portion of memory is used as a stack with a processor register as a stack pointer

- PUSH: $SP \leftarrow SP - 1$
  $M[SP] \leftarrow DR$
- POP: $DR \leftarrow M[SP]$
  $SP \leftarrow SP + 1$

- Most computers do not provide hardware to check stack overflow (full stack) or underflow(empty stack)

INSTRUCTION FORMAT

Instruction Fields

OP-code field - specifies the operation to be performed
Address field - designates memory address(s) or a processor register(s)
Mode field - specifies the way the operand or the effective address is determined

The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

Single accumulator organization:

```
ADD     X               /* AC ← AC + M[X]  */
```

General register organization:

```
ADD     R1, R2, R3          /* R1 ← R2 + R3  */
ADD     R1, R2          /* R1 ← R1 + R2  */
MOV     R1, R2          /* R1 ← R2  */
ADD     R1, X           /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH    X               /* TOS ← M[X]  */
ADD
```

THREE, AND TWO-ADDRESS INSTRUCTIONS

Three-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
ADD     R1, A, B    /*  R1 ← M[A] + M[B]     */
ADD     R2, C, D    /*  R2 ← M[C] + M[D]     */
MUL     X, R1, R2               /*  M[X] ← R1 * R2 */
```

- Results in short programs
- Instruction becomes long (many bits)

Two-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
MOV    R1, A          /* R1 ← M[A]          */
ADD    R1, B          /* R1 ← R1 + M[B]  */
MOV    R2, C          /* R2 ← M[C]          */
ADD    R2, D          /* R2 ← R2 + M[D]  */
MUL    R1, R2         /* R1 ← R1 * R2       */
MOV    X, R1          /* M[X] ← R1          */
```

ONE, AND ZERO-ADDRESS INSTRUCTIONS

One-Address Instructions:

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :

```
LOAD    A       /*  AC ← M[A]          */
ADD     B       /* AC ← AC + M[B]      */
STORE   T       /*  M[T] ← AC          */
LOAD    C       /*  AC ← M[C]          */
ADD     D       /*  AC ← AC + M[D]     */
MUL     T       /*  AC ← AC * M[T]     */
STORE   X       /*  M[X] ← AC          */
```

Zero-Address Instructions:

- Can be found in a stack-organized computer
- Program to evaluate X = (A + B) * (C + D) :

```
PUSH    A       /*  TOS ← A                    */
PUSH    B       /*  TOS ← B                    */
ADD             /*  TOS ← (A + B)              */
PUSH    C       /*  TOS ← C                    */
PUSH    D       /*  TOS ← D                    */
ADD             /*  TOS ← (C + D)              */
MUL             /*  TOS ← (C + D) * (A + B)    */
POP     X       /*  M[X] ← TOS                 */
```

MULTIPROCESSOR

- A multiprocessor is a computer system with two or more central processing units (CPUs), with each one sharing the common main memory as well as the peripherals. This helps in simultaneous processing of programs.
- The key objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

Benefits of using a multiprocessor include:

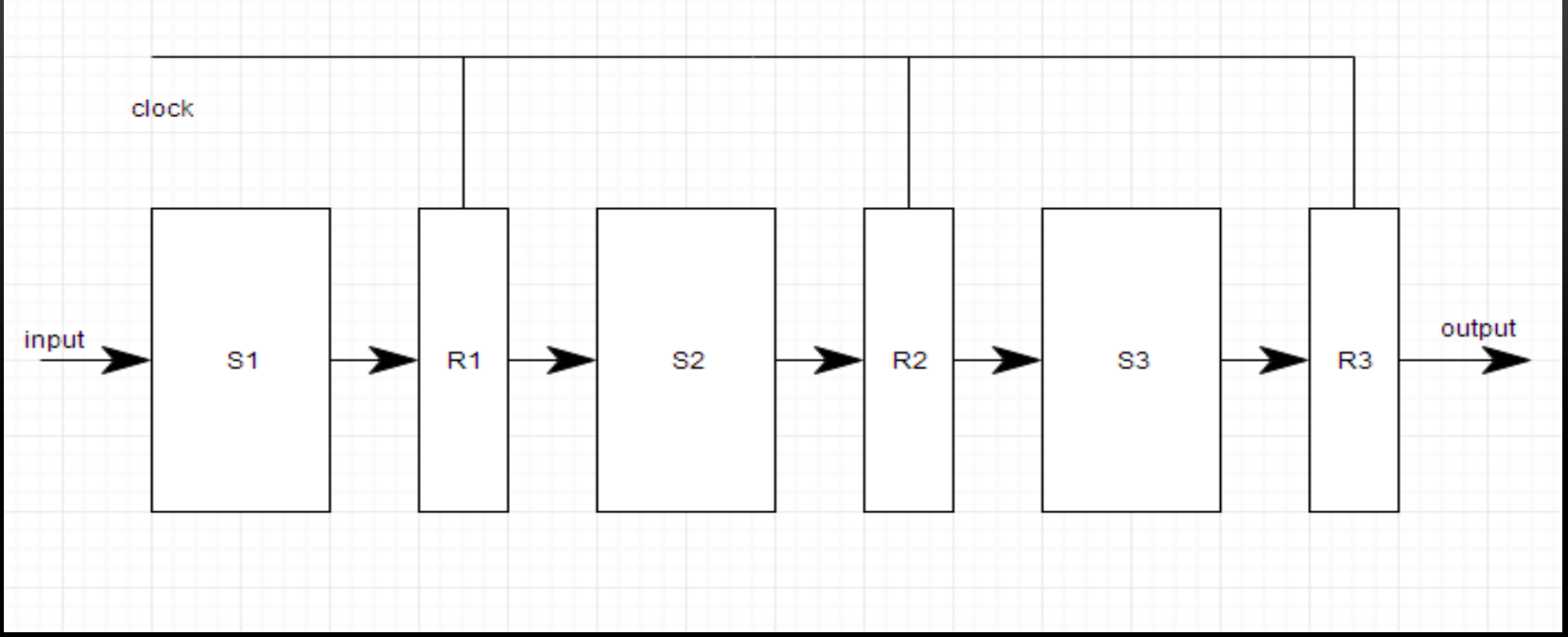
- Enhanced performance
- Multiple applications
- Multiple users
- Multi-tasking inside an application
- High throughput and/or responsiveness
- Hardware sharing among CPUs

DIFFERENT WAYS OF USING A MULTIPROCESSOR

- As a uniprocessor, such as single instruction, single data (SISD)
- Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data (MIMD)
- A single series of instructions in various perspectives, such as single instruction, multiple data (SIMD), which is usually used for vector processing
- Multiple series of instructions in a single perspective, such as multiple instruction, single data (MISD), which is used for redundancy in failsafe systems and, occasionally, for describing hyper-threading or pipelined processors

PIPELINING

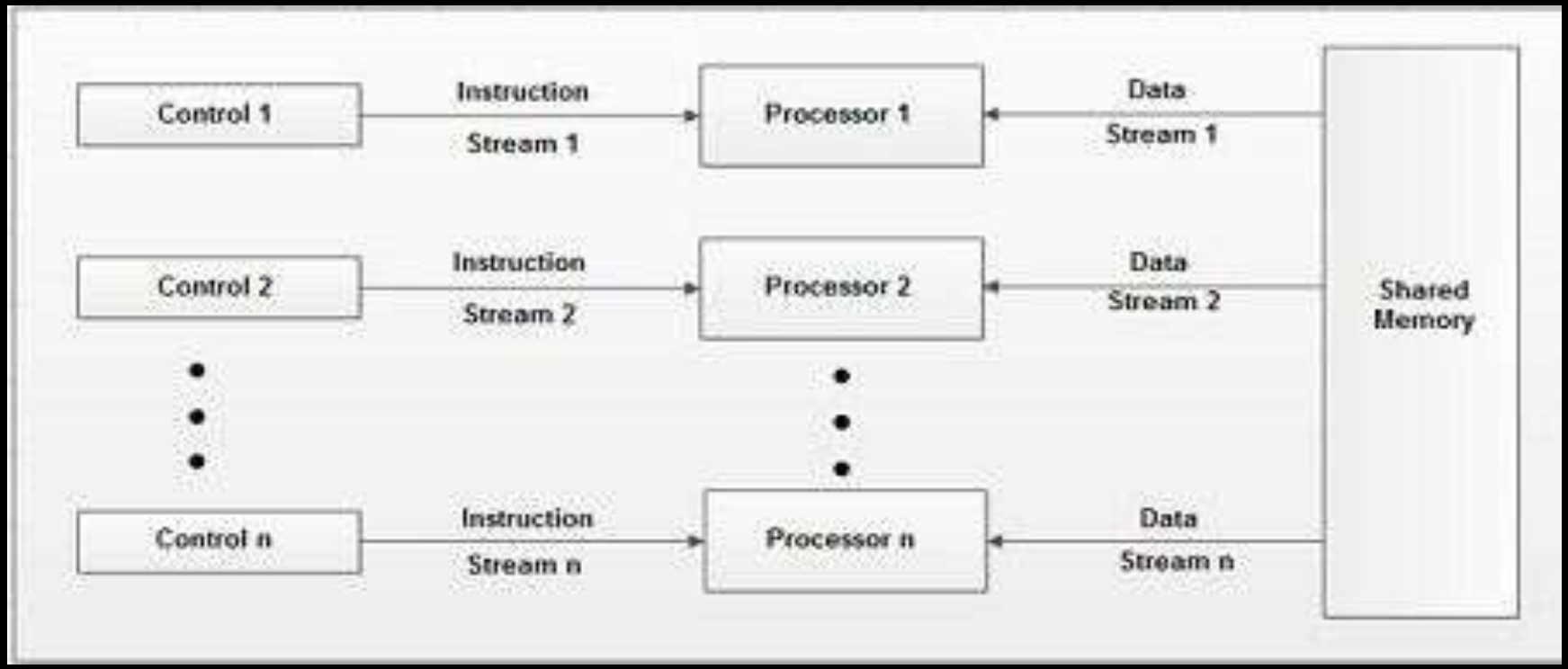
- Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as **pipeline processing**.
- Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.
- Pipelining increases the overall instruction throughput.
- In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.



Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

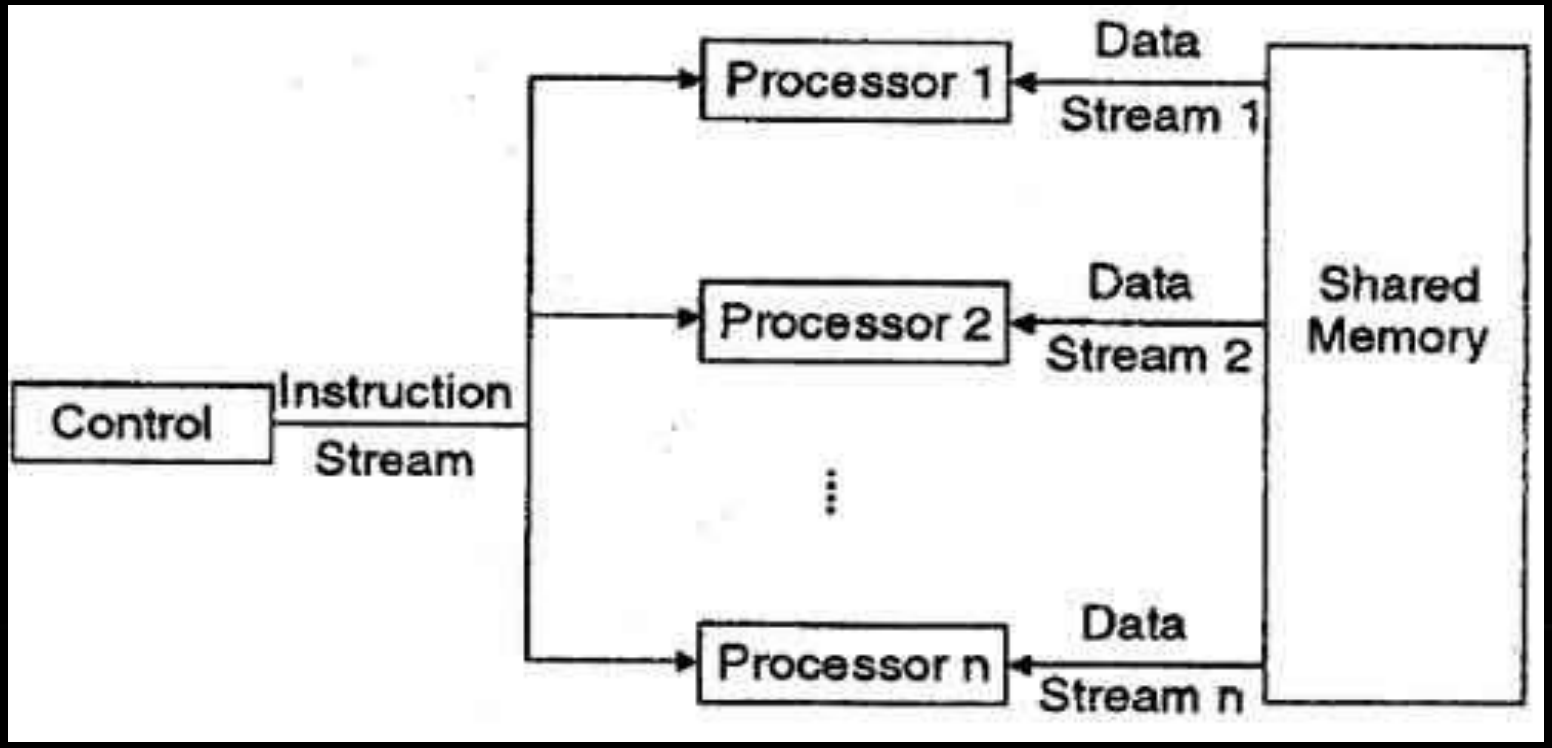
MIMD COMPUTERS

MIMD computers are consisting of 'n' processing units; each with its own stream of instruction and each processing unit operate on unit operates on a different piece of data. MIMD is the most powerful computer system that covers the range of multiprocessor systems. The block diagram of MIMD computer is shown.



SIMD COMPUTERS

- SIMD computers are consisting of ‘n' processing units receiving a single stream of instruction from a central control unit and each processing unit operates on a different piece of data. Most SIMD computers operate synchronously using a single global dock. The block diagram of SIMD computer is shown below:



THANK YOU