# Central Processing Unit and Pipeline Processing

Marwadi University

Diploma Studies
Computer Engineering

Unit – 4

Title - Central Processing Unit and Pipeline Processing

Computer Organization (09CE2401)

Prof. Smit Thacker

# Central Processing Unit

# Central Processing Unit

The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.

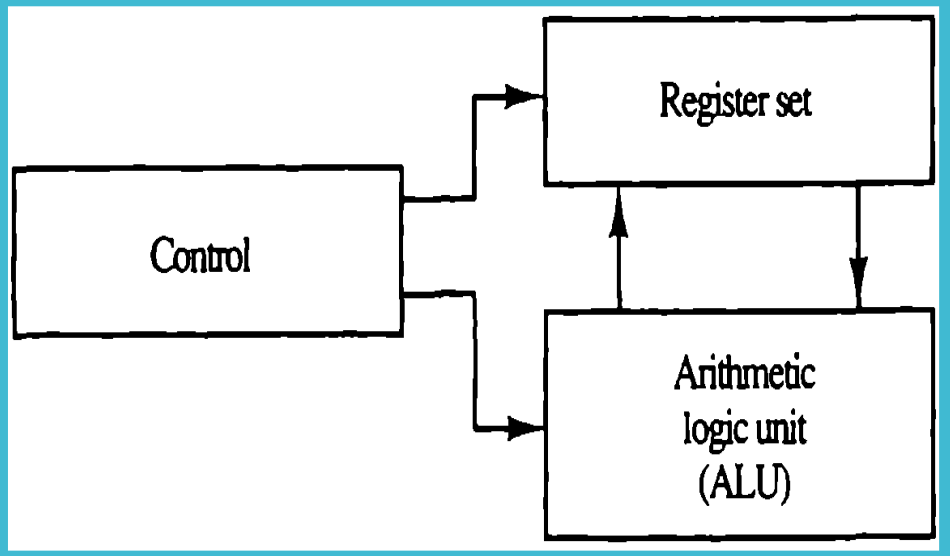
# Central Processing Unit

**The CPU is made up of three major parts.**

**(1) The register set stores intermediate data used during the execution of the instructions.**

**(2) The arithmetic logic unit (ALU) performs the required micro operations for executing the instructions.**

**(3) The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.**



# Central Processing Unit

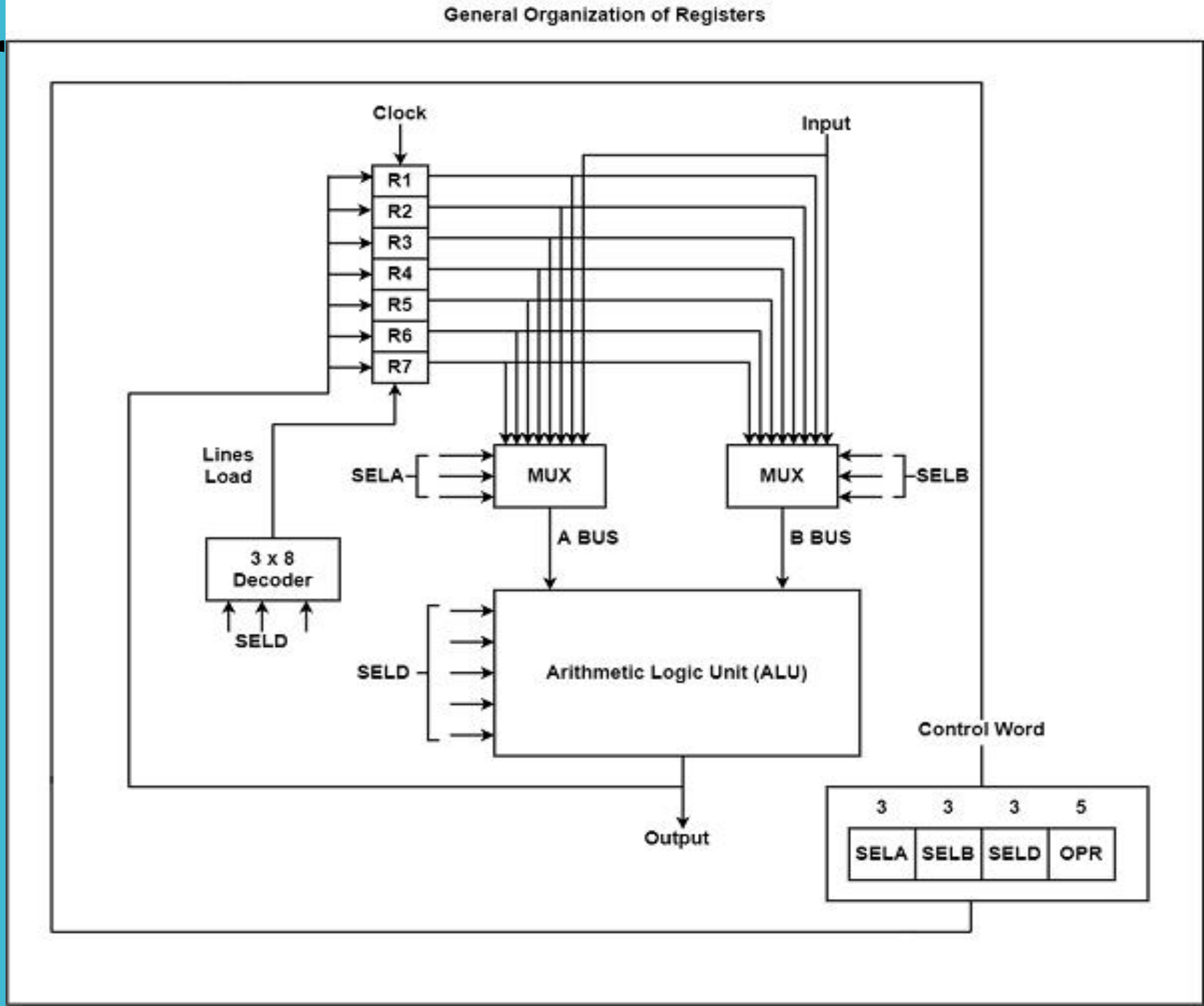
- The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
- Computer architecture is sometimes defined as the computer structure and behavior as seen by the programmer that uses machine language instructions.
- This includes the instruction formats, addressing modes, the instruction set, and the general organization of the CPU registers.
- The design of a CPU is a task that in large part involves choosing the hardware for implementing the machine instructions.
- The user who programs the computer in machine or assembly language must be aware of the register set, the memory structure, the type of data supported by the instructions, and the function that each instruction performs.

# General Register Organization

# General Register Organization

- We saw that memory locations are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication. But memory access is the most time-consuming operation in computer.
- It is more convenient and efficient to store these intermediate values in processor registers. Large number of registers are included in the CPU, and connected through a common bus system.
- The registers communicate with each while performing various micro-operations.
- Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift micro-operations in the processor.

# General Register Organization



# General Register Organization

- A bus organization for seven CPU registers is shown in Fig.
- The output of each register is connected to two multiplexers (MUX) to form the two buses A and B .
- The selection lines in each multiplexer select one register or the input data for the particular bus.
- The A and B buses form the inputs to a common arithmetic logic unit (ALU).
- The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed.
- The result of the micro operation is available for output data and also goes into the inputs of all the registers.

# General Register Organization

- A decoder selects the register that receives the information from the output bus.
- It activates register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.
- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.
- For example, to perform the operation

# General Register Organization

Example: R1 ← R2 + R3

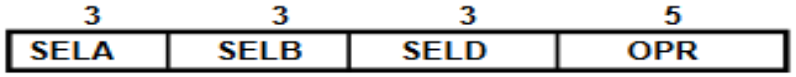
[1] MUX A selector (SELA): BUS A ← R2

[2] MUX B selector (SELB): BUS B ← R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 ← Out Bus

Control Word




- There are 14 binary selection inputs in the unit, and their combined value specifies a control word.
- In above Fig. It consists of four fields. Three fields contain three bits each, and one field has five bits.
- SELA select a source register for the A input of the ALU.
- SELB select a register for the B input of the ALU.
- SELD select a destination register using the decoder and its seven load outputs.
- The five bits of OPR select one of the operations in the ALU.

# Register Stack

# Register Stack

- Stack can be useful while evaluating the expression and also while using subroutine where current content of register is stored on to the stack. below is the organization of register stack.
- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

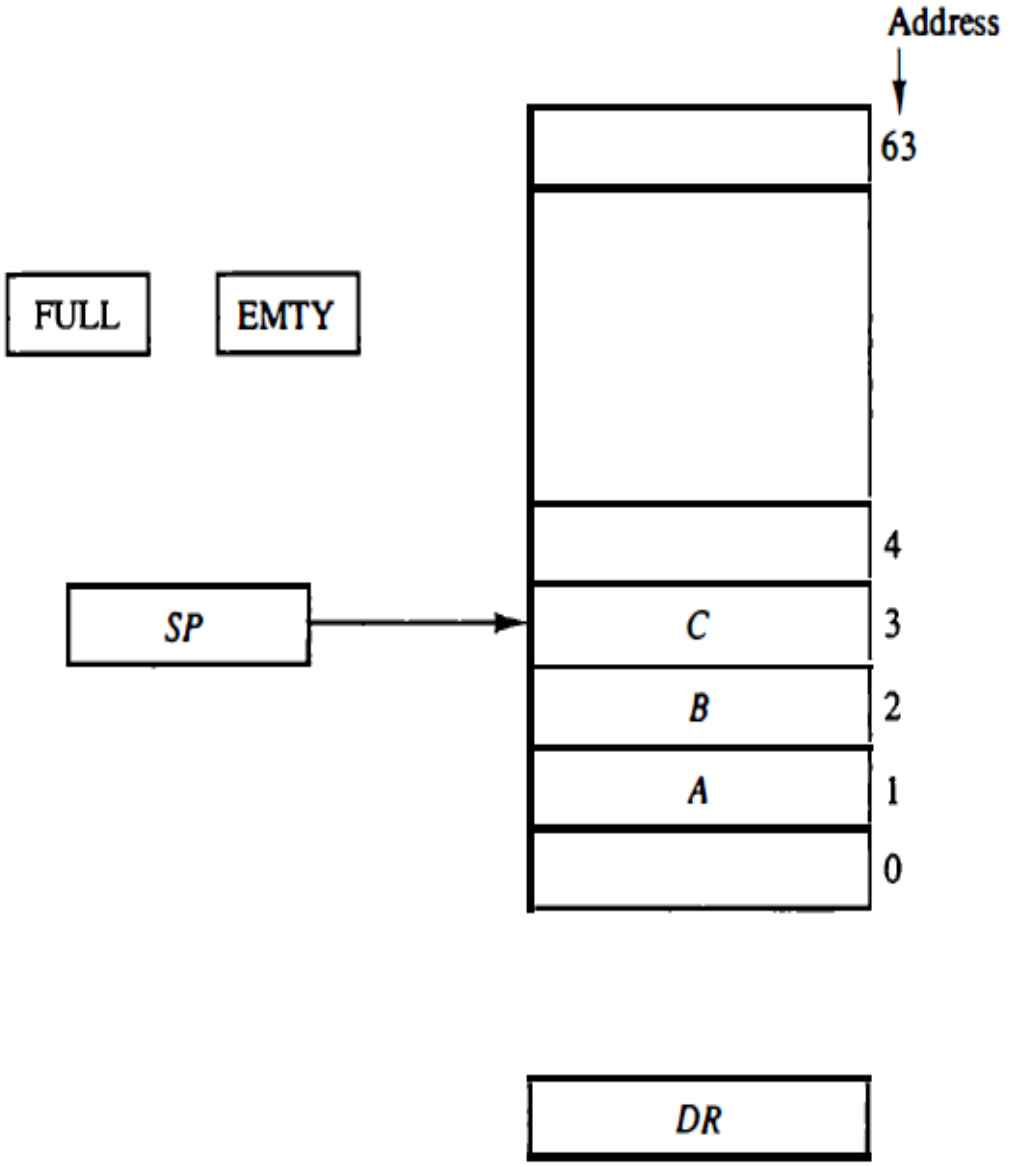



Figure 8-3 Block diagram of a 64-word stack.

# Register Stack

- SP points to the top of stack. Three items are placed in the stack: A, B, and C, in that order.
- Item C is on top of the stack so that the content of SP is now 3.
- The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.
- Two operation can be performed on stack
- 1) Push
- 2) Pop

# Register Stack

- **PUSH Operation**

  $SP \leftarrow SP + 1$

  $M[SP] \leftarrow DR$

- Along with above operation we also want to check if stack is full or not.

$IF\ (SP = 0)\ then\ (FULL \leftarrow 1)$

$EMTY \leftarrow 0$

And while Pop operation we want to check.

- **POP Operation**

  $DR \leftarrow M[SP]$

  $SP \leftarrow SP - 1$

  $IF\ (SP = 0)\ then\ (EMTY \leftarrow 1)$

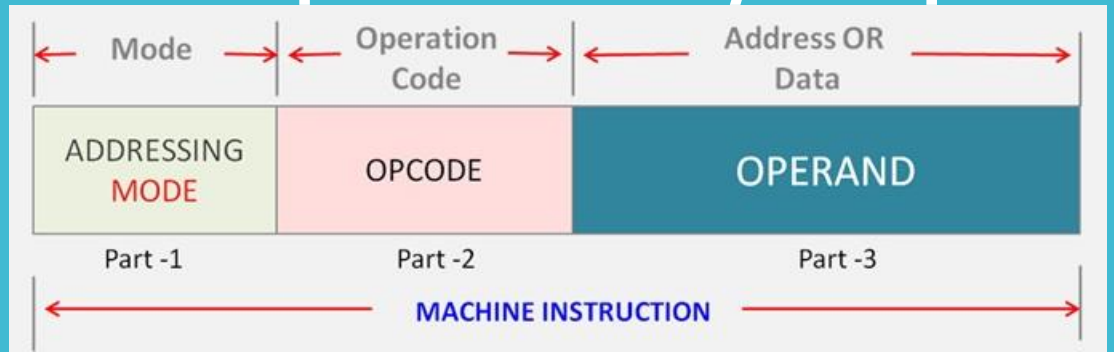
  $FULL \leftarrow 0$

# Instruction Format

# Instruction Format

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.



# Instruction Format

**Instruction are divided or categorized in the number of operand it takes in the operand field in the instruction.**

1. Three Address Instructions
2. Two Address Instruction
3. One Address Instruction
4. Zero Address Instruction
5. RISC Instructions

# Instruction Format – Three Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

for eg, X = (A + B) * (C + D) expression we write assembly language.

```
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
```

**The advantage of the three-address format**

- Results in short programs when evaluating arithmetic expressions.

**The disadvantage is that**

- The binary-coded instructions require too many bits to specify three addresses.

# Instruction Format – Two Address Instructions

- Two-address instructions are the most common in commercial computers.
- Here again each address field can specify either a processor register or a memory word.

for eg, X = (A + B) * (C + D) expression we write assembly language.

```
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
```

- We can observe here Line of code increases in two address.

# Instruction Format – One Address Instructions

- One-address instructions uses accumulator (AC) register for all data manipulation.
- For multiplication and division there is a need for a second register.
- The program to evaluate X = (A + B) * (C + D) is

```
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
```

- All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

# Instruction Format – Zero Address Instructions

- With the help of stack this type of instruction can be implemented.
- Push and pop operation can be used to implement this.
- for example, if we want to perform below expression then it has to be converted to reverse polish notation first and then it has to be pushed or popped from the stack.
- X = (A + B) * (C + D) in reverse polish notation is AB+ CD+ *
- here (TOS stands for top of stack.)

```
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
```

# Instruction Format –RISC (Reduced Instruction set computer)

- The instruction set of a typical RISC processor is restricted to the use of **load and store** instructions when communicating between memory and CPU.
- All other instructions are executed within the registers of the CPU without referring to memory.
- A program for a RISC-type CPU consists of **LOAD and STORE** instructions that have one memory and one register address, and computational-type instructions that have **three addresses** with all three specifying processor registers.

# Instruction Format –RISC (Reduced Instruction set computer)

- The following is a program to evaluate X = (A + B) * (C + D).

```
LOAD    R1, A          R1 ← M[A]
LOAD    R2, B          R2 ← M[B]
LOAD    R3, C          R3 ← M[C]
LOAD    R4, D          R4 ← M[D]
ADD     R1, R1, R2     R1 ← R1 + R2
ADD     R3, R3, R2     R3 ← R3 + R4
MUL     R1, R1, R3     R1 ← R1* R3
STORE   X, R1          M[X] ← R1
```

- As we can see in above code the result of the computations is stored in memory with a store instruction in last.

# Data Transfer and Manipulation

# Data Transfer and Manipulation

**Most computer instructions can be classified into three categories:**

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

- Data transfer instructions cause transfer of data from one location to another without changing the binary information content.
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.
- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

# Data Transfer and Manipulation

## 1. Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

| Name | Mnemonic |
|---|---|
| Load | LD |
| Store | ST |
| Move | MOV |
| Exchange | XCH |
| Input | IN |
| Output | OUT |
| Push | PUSH |
| Pop | POP |

# Data Transfer and Manipulation

## 2. Data Manipulation Instructions

- Data manipulation instructions perform operations on data and gives computational ability to the computer.
- Also these instructions in a typical computer are usually divided into three basic types:
- 1. Arithmetic instructions
- 2. Logical and bit manipulation instructions
- 3. Shift instructions

# Data Transfer and Manipulation

## 2. Data Manipulation Instructions

### 1. Arithmetic Instructions

- The four basic arithmetic operations are addition, subtraction, multiplication, and division.
- The multiplication and division must then be generated by means of software subroutines.

| Name | Mnemonic |
|---|---|
| Increment | INC |
| Decrement | DEC |
| Add | ADD |
| Subtract | SUB |
| Multiply | MUL |
| Divide | DIV |
| Add with carry | ADDC |
| Subtract with borrow | SUBB |
| Negate (2's complement) | NEG |

# Data Transfer and Manipulation

## 2. Data Manipulation Instructions

## 2. Logical and Bit Manipulation Instructions

- Logical instructions perform binary operations on strings of bits stored in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.

| Name | Mnemonic |
|---|---|
| Clear | CLR |
| Complement | COM |
| AND | AND |
| OR | OR |
| Exclusive-OR | XOR |
| Clear carry | CLRC |
| Set carry | SETC |
| Complement carry | COMC |
| Enable interrupt | EI |
| Disable interrupt | DI |

# Data Transfer and Manipulation

## 2. Data Manipulation Instructions

## 3. Shift Instructions

- Instructions to shift the content of an operand are quite useful and are often provided in several variations.
- Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used.

| Name | Mnemonic |
|---|---|
| Logical shift right | SHR |
| Logical shift left | SHL |
| Arithmetic shift right | SHRA |
| Arithmetic shift left | SHLA |
| Rotate right | ROR |
| Rotate left | ROL |
| Rotate right through carry | RORC |
| Rotate left through carry | ROLC |

# Program Interrupt

# Program Interrupt

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed.

# Program Interrupt

**The interrupt is quite similar to a subroutine call except for three variations:**

**(1) The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction (except for software interrupt as explained later);**

**(2) the address of the interrupt service program is determined by the hardware rather than from the address field of an instruction; and**

**(3) an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter. These three procedural concepts are clarified further below.**

# Types of Interrupt

There are three major types of interrupts. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

# Types of Interrupt

## (1) External interrupts

- External interrupts come from input-output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.
- Examples I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure.

## (2) Internal interrupts

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps .
- Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, etc.

# Types of Interrupt

## (3) Software interrupts

- A software interrupt is initiated by executing an instruction.
- Software interrupt is a special call instruction that behaves like an interrupt.

# Reduced Instruction Set Computer (RISC ) & Complex Instruction Set Computer (CISC)

# Reduced Instruction Set Computer (RISC ) & CISC

- A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.
- In the early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often.
- This type of computer is classified as a reduced instruction set computer or RISC.

# CISC Characteristics

1. A large number of instructions-typically from 100 to 250 instructions
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes-typically from 5 to 20 different modes
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory.

# RISC Characteristics

The major characteristics of a RISC processor are:

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed-length, easily decoded instruction format
6. Single-cycle instruction execution
7. Hardwired rather than microprogrammed control

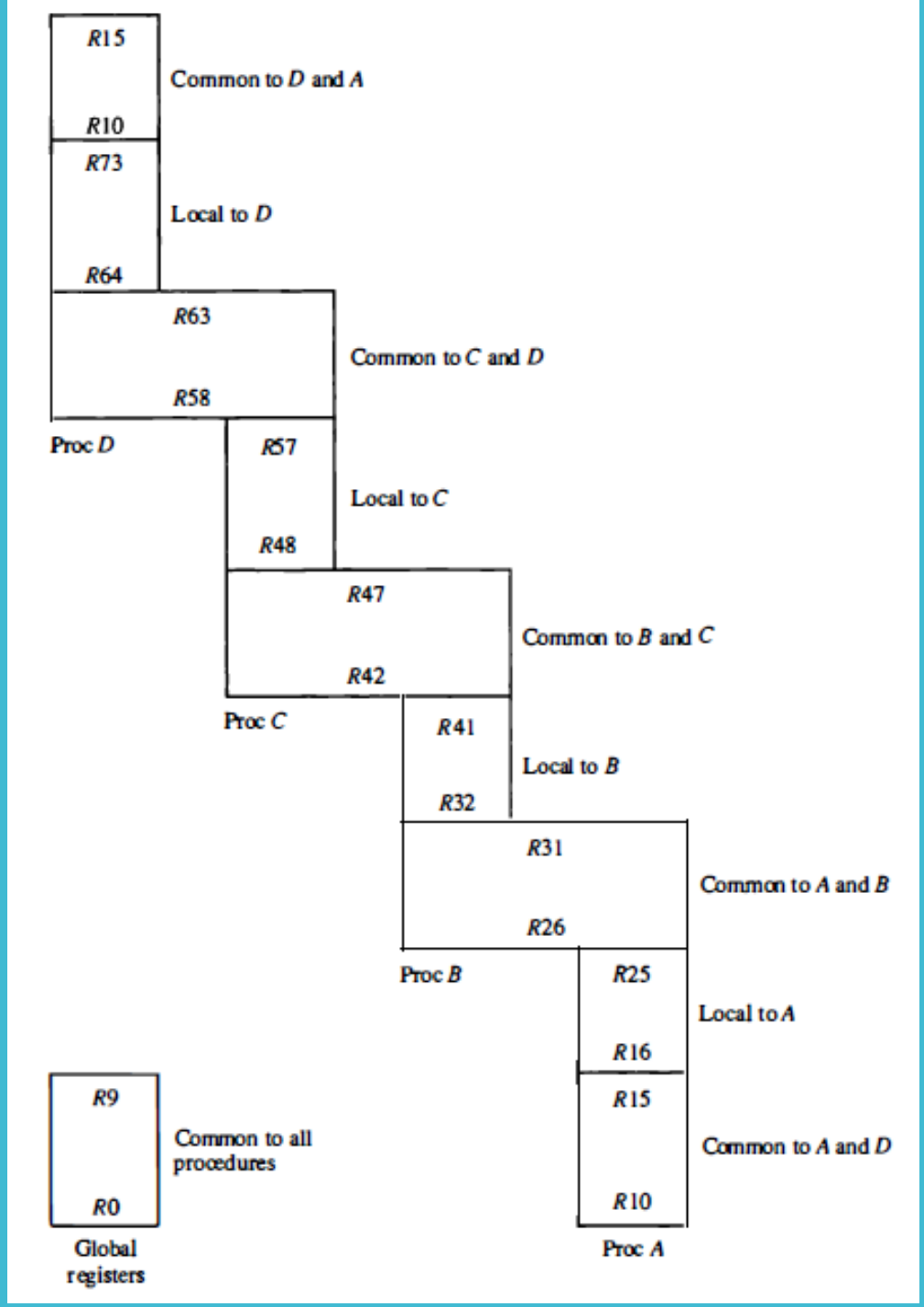
| CISC | RISC |
| --- | --- |
| Complex Instruction Set Computer | Reduced Instruction Set Computer |
| Large no. of Instructions from 100 to 250 | Relatively fewer Instructions (30 to 40) |
| Complex Instruction Set | Simple Instruction Set |
| Large no. of addressing modes from 5 to 20 | Relatively fewer addressing modes |
| Variable length instruction format | Fixed length instruction format |
| Instruction decoding is complex | Instruction decoding is simple |
| Pipeline implementation is complex | Pipeline implementation is simple |
| It uses special purpose register and flags | It uses large number of general purpose register. |
| Microprogrammed Controlled | Hardwired Controlled |

# Overlapped Register Windows

# Overlapped Register Windows

- A characteristic of some RISC processors is their use of overlapped register windows to provide the passing of parameters and avoid the need for saving and restoring register values.
- The system has a total of 74 registers.
- Registers R0 through R9 are global registers that hold parameters shared by all procedures.
- The other 64 registers are divided into four windows to accommodate procedures A, B, C, and D.
- Each register window consists of 10 local registers and two sets of six registers common to adjacent windows.

# Overlapped Register Windows



# Overlapped Register Windows

- A characteristic of some RISC processors is their use of overlapped register windows to provide the passing of parameters and avoid the need for saving and restoring register values.
- The system has a total of 74 registers.
- Registers R0 through R9 are global registers that hold parameters shared by all procedures.
- The other 64 registers are divided into four windows to accommodate procedures A, B, C, and D.
- Each register window consists of 10 local registers and two sets of six registers common to adjacent windows.

# Overlapped Register Windows

- Local registers are used for local variables.
- Common registers are used for exchange of parameters and results between adjacent procedures.
- The common overlapped registers permit parameters to be passed without the actual movement of data.
- Only one register window is activated at any given time with a pointer indicating the active window.
- Each procedure call activates a new register window by incrementing the pointer.

# Overlapped Register Windows

- As an example, suppose that procedure A calls procedure B.
- Registers R26 through R31 are common to both procedures, and therefore procedure A stores the parameters for procedure B in these registers.
- Procedure B uses local registers R32 through R41 for local variable storage.
- If procedure B calls procedure C, it will pass the parameters through registers R 42 through R 47.
- When procedure B is ready to return at the end of its computation, the program stores results of the computation in registers R26 through R31 and transfers back to the register window of procedure A.

# Overlapped Register Windows

- Note that registers RIO through R 15 are common to procedures A and D because the four windows have a circular organization with A being adjacent to D .

# Parallel Processing

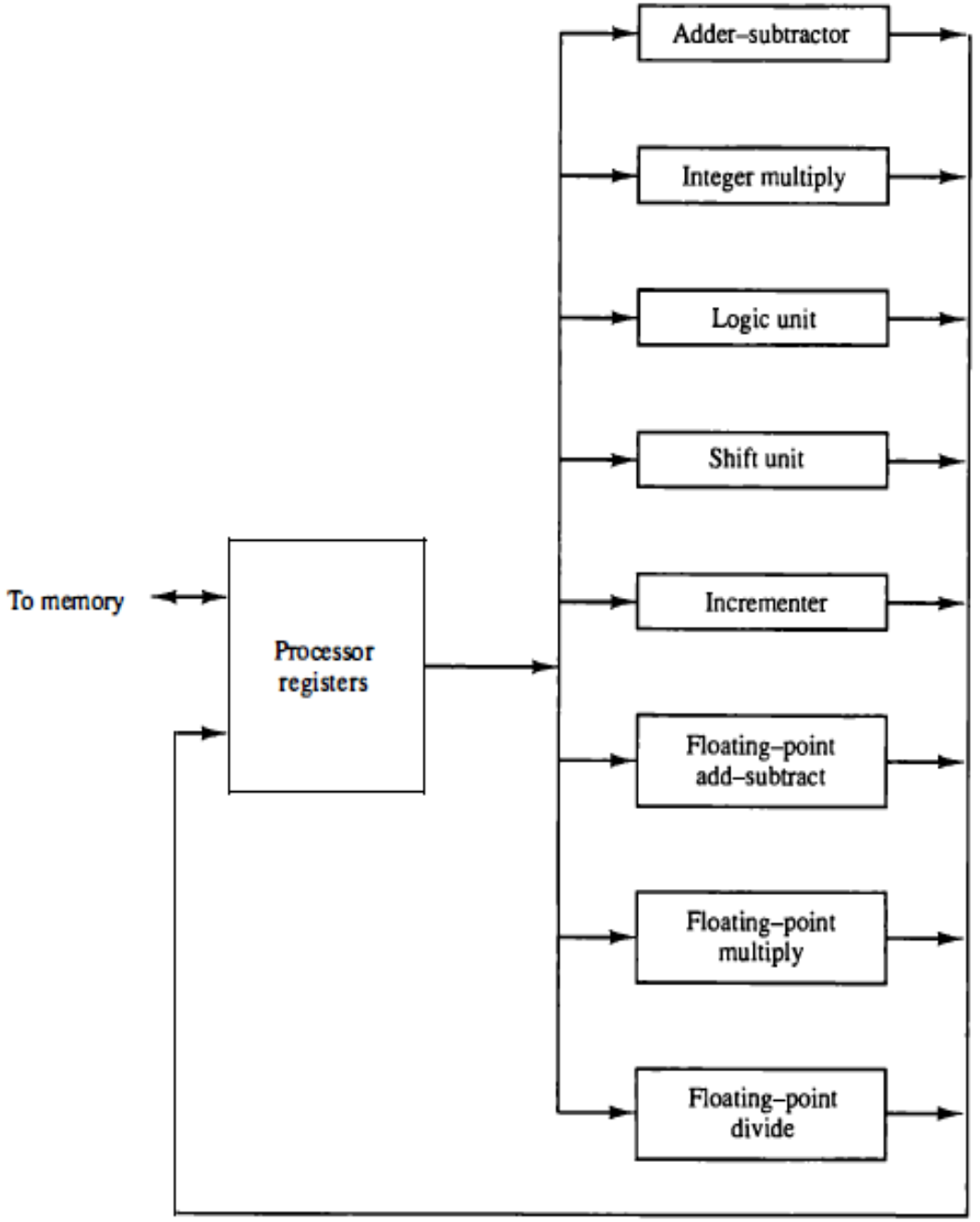
# Parallel Processing

- Parallel processing system is able to performing concurrent data processing to achieve faster execution time.
- The system may have two or more processors operating concurrently and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.
- Parallel processing by having a multiplicity of functional units that perform identical or different operations simultaneously.
- For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

# Parallel Processing

- Figure shows one possible way of separating the execution unit into eight functional units operating in parallel.
- The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands.

Figure 9-1 Processor with multiple functional units.



# Parallel Processing

- There are a variety of ways that parallel processing can be classified.
- One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

❖The sequence of instructions read from memory constitutes an instruction stream.

❖The operations performed on the data in the processor constitutes a data stream.

# Parallel Processing

- Flynn's classification divides computers into four major groups as follows:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

# Parallel Processing

## 1. Single instruction stream, single data stream (SISD)

- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.
- Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

# Parallel Processing

## 2. Single instruction stream, multiple data stream (SIMD)

- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

# Parallel Processing

## 3. Multiple instruction stream, single data stream(MISD)

- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

# Parallel Processing

## 4. Multiple instruction stream, multiple data stream (MIMD)

- MIMD organization refers to a computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category.

# Addressing Modes

# Addressing Modes

- The way the operands are used during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for accommodating one or both of the following provisions:
  - 1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
  - 2. To reduce the number of bits in the addressing field of the instruction.

# Addressing Modes

- Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all.
- These are the implied and immediate modes.
- Total 10 types of addressing modes are supported by our system.

# Addressing Modes

(1) Implied/Implicit Mode

(2) Immediate Mode

(3) Register Mode

(4) Register Indirect Mode

(5) Auto Increment/Decrement Mode

(6) Direct Address Mode

(7) Indirect Address Mode

(8) Relative Address Mode

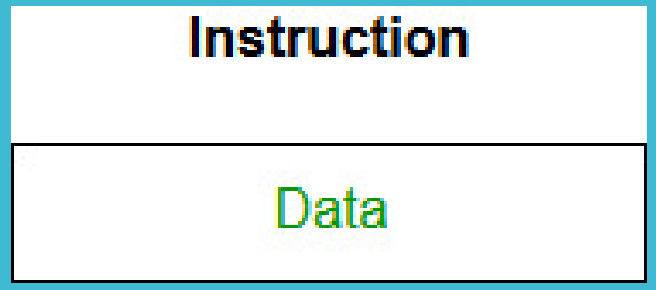
(9) Indexed Addressing Mode

(10)Base Register Addressing Mode

# Addressing Modes

## (1) Implied/implicit Mode:

- In implied addressing the operand is specified in the instruction itself.
- In this mode the data is 8 bits or 16 bits long and data is the part of instruction.
- Zero address instruction are designed with implied addressing mode.

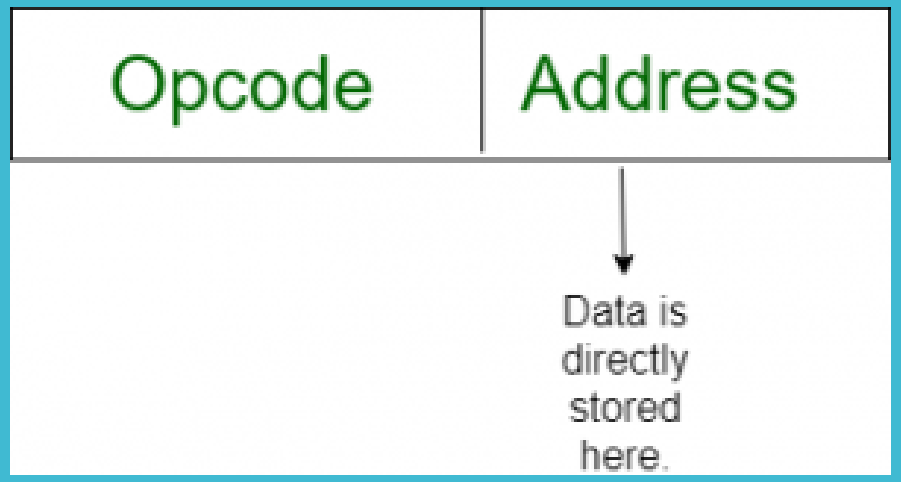



- Example:  CLC (used to reset Carry flag to 0)

# Addressing Modes

## (2) Immediate Mode (symbol #) :

- In this mode data is present in address field of instruction.
- Designed like one address instruction format.

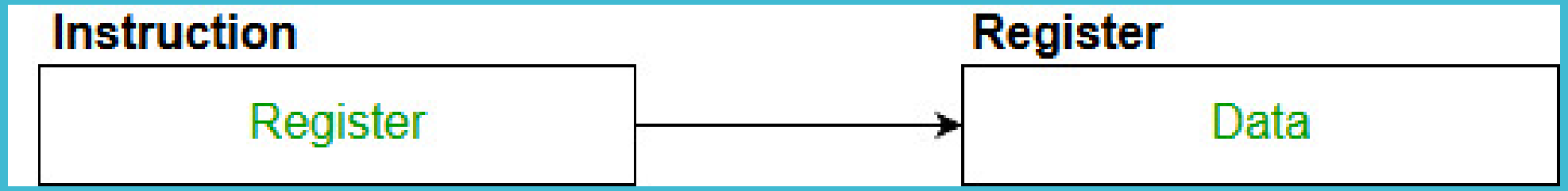



- **Example:** MOV AL, 35H (move the data 35H into AL register)

# Addressing Modes

## (3) Register Mode:

- In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers.
- The data is in the register that is specified by the instruction.
- Here one register reference is required to access the data.




- **Example:** MOV AX,CX (move the contents of CX register to AX register)

# Addressing Modes

## (4) Register Indirect Mode:

- In this addressing the operand's offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction.
- The effective address of the data is in the base register or an index register that is specified by the instruction.
- Here two register reference is required to access the data.


- **Example:-** MOV AX, [BX](move the contents of memory location s addressed by the register BX to the register AX)

# Addressing Modes

## (5) Auto increment Mode:

- Effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.(R1)+

- **Example:-** ADD R1, (R2)+

# Addressing Modes

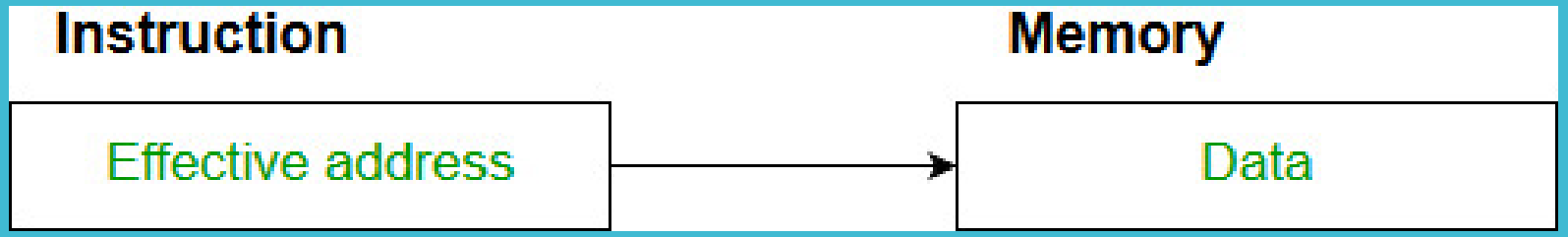
## (5) Auto decrement Mode:

- Effective address of the operand is the contents of a register specified in the instruction.
- Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. –(R1)
- **Example:-** ADD R1, -(R2)

# Addressing Modes

## (6) Direct Address Mode (Symbol [ ]):

- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.




- **Example:-** ADD AL,[0301] //add the contents of offset address 0301 to AL

# Addressing Modes

## (7) Indirect Address Mode (Symbol [@ or () ):

- In this mode address field of instruction contains the address of effective address.
- Here two references are required:
  - 1st reference to get effective address.
  - 2nd reference to access the data.
- **Example:-** ADD 457

# Addressing Modes

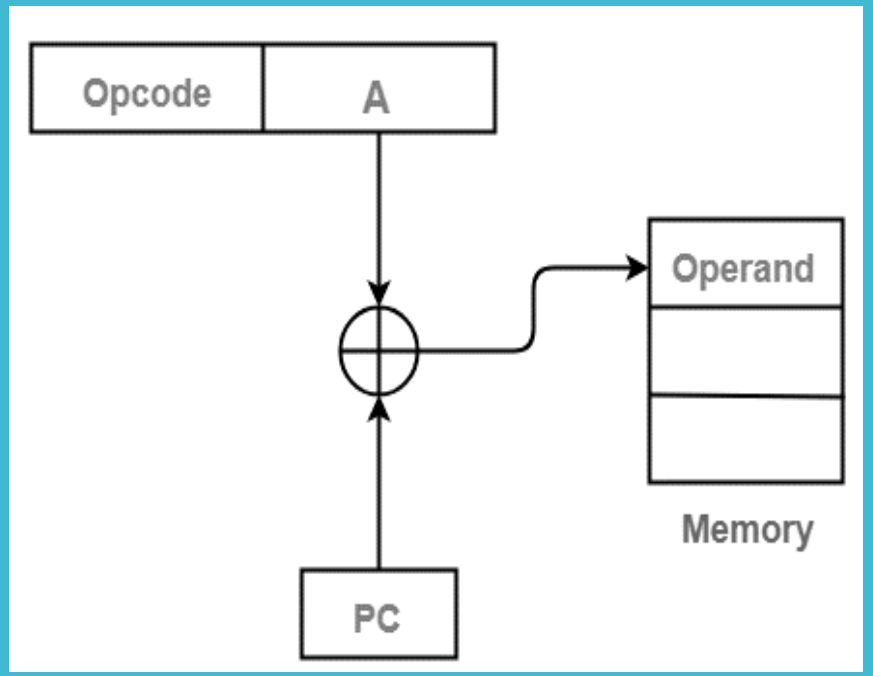
## (8) Relative Address Mode:

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (in 2' s complement representation) which can be either positive or negative.
- When this number is added to the content of the program counter, the result produces an effective address of the next instruction.
- assume that the program counter contains the number 825 and the address part of the instruction contains the number 24.

# Addressing Modes

## (8) Relative Address Mode:

- The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826.
- The effective address computation for the relative address mode is 826 + 24 = 850.
- This is 24 memory locations forward from the address of the next instruction.



# Addressing Modes

## (9) Indexed Addressing Mode:

- In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- The address field of the instruction defines the beginning address of a data array in memory.
- Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value.

# Addressing Modes

## (10) Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.