UNIT 9 INSTRUCTION SET ARCHITECTURE

9.0 Introduction
9.1 Objectives
9.2 Instruction Set Characteristics and Design Considerations
  9.2.1 Operand Data Types
  9.2.2 Types of Instructions
  9.2.3 Stored Program Organization
9.3 Number of Addresses and Instruction size
9.4 Instruction Set and Format Design Issues
9.5 Addressing Schemes
  9.5.1 Immediate Addressing
  9.5.2 Direct Addressing
  9.5.3 Indirect Addressing
  9.5.4 Register Addressing
  9.5.5 Register Indirect Addressing
  9.5.6 Relative Addressing Scheme
  9.5.7 Base Register Addressing
  9.5.8 Indexed Addressing Scheme
  9.5.9 Stack Addressing
9.6 Summary
9.7 Answers/Solutions

9.0 INTRODUCTION

In the previous two blocks, you have learnt the concepts of data representation, memory organization and Input/output organisation. This Unit discusses about one of the most fundamental aspect of a general-purpose computer – the instruction. A computer can do general purpose or specific tasks using the instructions. Instructions are the mediator between the programmer and hardware. Hardware is the computer architecture, and software is the instruction set architecture. Instructions set architecture is the only way you can interact with the computer machines. An instruction set architecture consists of a complete set of instruction to do a task on a specific computer system.

In this unit, details of instructions format, operands data types, instruction types and various addressing modes have been discussed.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Define various characteristics of instruction set of a computer
- Appreciate various components of instructions
- Explain the design of different instruction
- Define the term instruction format
- Explain various addressing modes used in an instruction.

9.2 INSTRUCTION SET CHARACTERISTICS AND DESIGN CONSIDERATIONS

A programmer writes the program instructions in assembly language, which are easily understandable for the humans. But the assembly language is not understandable by the computer machines, as these computer machines are made of digital modules, which understand only binary logic. Hence, during execution, program is converted into binary codes which is understandable by the computer machines. A binary coded instruction is in format, which is interpreted and executed by the machine. Instruction format is discussed next.

Instruction format

A simple instruction format is shown in Figure 9.1. It consists of three components: the operand address, the operation code or opcode and the mode. The number of bits is allocated to each component. Basic definition of these components are:

Opcode: In instruction, opcode decides the operation to be performed on the data.

Operand: In Instruction, operands are the data on which operation is to be performed.

Effective address: In instruction, it is the memory address where the data/operand is stored. For example, if data is stored in register, register is the effective address. Effective address is decided by the addressing schemes, which are discussed in details in section 9.5. The mode field of the instruction determines, how the effective address would be computed in a machine.

31	**30 24**	**23 0**
I	Opcode	Operand

Figure 9.1: Instruction format components

The total number of operations, which can be performed by a computer machine is decided by the bits allocated for the operation code of an instruction. The n-bits operation code can be used to code 2^n distinct operations. For example, the 7 bits opcode, as given in Figure 9.1, can be used to code $2^7 = 128$ distinct operations. The computer can be designed using these opcodes, such that a typical opcode may represent, a typical operation. An example opcode can be designed as:

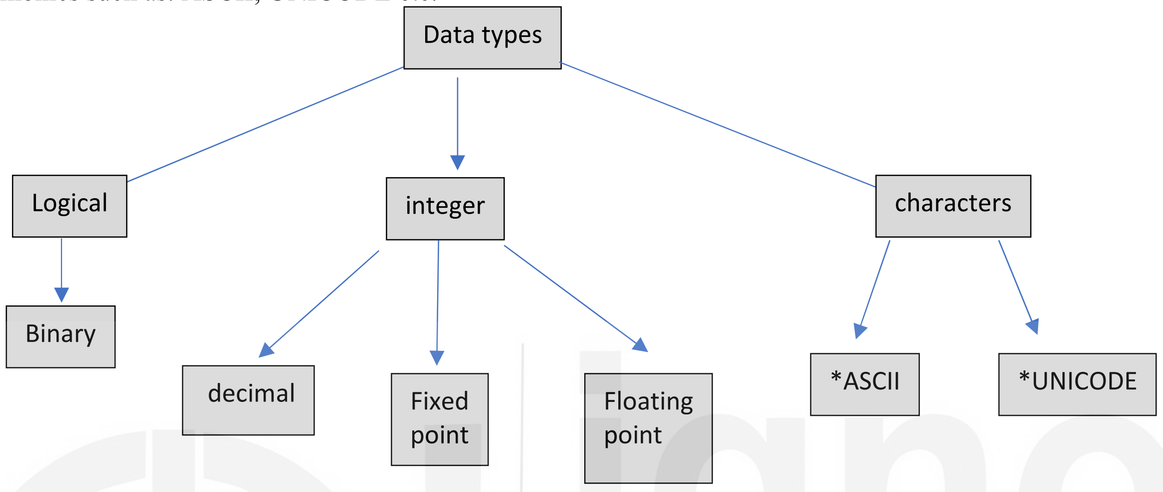
ADD operation: 1100100 (7-bit opcode)

The number of bits allocated to operand field, of the instruction in Figure 9.1, depends on the memory address size or the data bits. In Figure 9.1, the 31st bit "I" decides the mode if address is in direct or indirect mode. In direct mode, the address given in the instruction is the effective address of the operand. In indirect mode, the instruction holds the memory address where the effective address of operand is saved. It may be noted that this instruction format just allows direct and indirect addressing mode and a single operand in each instruction. However, different instruction formats may support different types of addressing modes and different number of operands.

A point that can be noted for the instruction format of Figure 9.1 is that size of operand filed of instruction is 24 bits. In case, this operand is a direct operand, then the size of the main memory addresses that can be supported by the machine having instruction format, as given in Figure 9.1 is $2^{24} = 16$ M memory words.

9.2.1 Operand Data Types

In computer, operands are the data bits on which operation is to be performed. As shown in ***Figure 9.1***, data types can be categorised as logical, integer and characters. Logical data is Boolean which may be 1/0 or true/false. Integer data may be further divided into: decimal, fixed point, and floating point. And the character data have mnemonics such as: ASCII, UNICODE etc.



*ASCII : **A**merican **S**tandard **C**ode For **I**nformation **I**nterchange

*UNICODE: **Universal Character Encoding**

Figure 9.1: operand data types

9.2.2 Types of Instructions

To evaluate any computable function, a computer must have a group of instructions created by the designer in machine language. The set of instructions include various types such as: arithmetic instructions, logical instructions, shift instructions, instructions to transfer content from memory to processor register, program control instructions and input/output instructions. Broadly, instruction can be categorized as below:

1. Instructions to transfer the data
2. Instruction to manipulate the data
3. Instructions for program control

Instructions to transfer the data: In computer machines data transfer takes place between processor registers, between memory and processor register, between processor register and input or output interface. The instructions with data in the central processor register are faster than that of instructions with data in memory. Each instruction has a mnemonic which can be used as part of the assembly language. Please note that these mnemonics may vary for different machines. Data transfer instructions are listed below:

Table 9.1: Some Data transfer instructions of computer

Instruction	*Mnemonics*	*Remarks*
Load	LD	To load data from memory to accumulator register
Store	ST	To store data from processor register to memory
Move	MOV	To transfer data between processor registers
Exchange	XCH	Exchange data between processor registers or between a register and memory
Input	IN	Data transfer between processor register and input terminal
Output	OUT	Data transfer between processor register and output terminal
Push	PUSH	Data transfer from register to memory stack
Pop	POP	Data transfer from memory stack to register

Instruction to manipulate the data: Data manipulation instructions are categorized based on the type of operation the perform on the data. The following are the basic categories of data manipulation instructions:

- Arithmetic
- Logical and bit manipulation
- Shift instructions

Arithmetic: These instructions perform arithmetic operations on the data. Various arithmetic operations are: addition, subtraction, multiplication, division etc.

Logical and bit manipulation: The logical and bit manipulation functions are used to perform logical operations or manipulation by setting/resetting a single data bit. Some of the logical operations are -AND, OR, XOR etc.

Bit manipulation - selective set and mask: Let's assume A= 1010, and you need to set the least significant bit (LSB), keeping all the remining bits unchanged. Since you just need to set the LSB, therefore, the second operand for selective set would be B=0001 and you will use OR operator, as shown below:

A	1010
B	0001
A OR B	1011

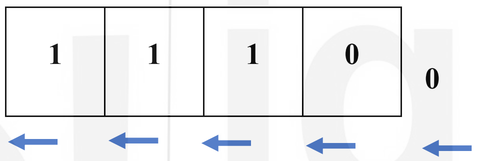
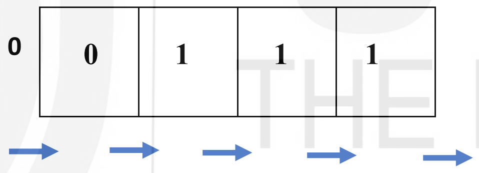
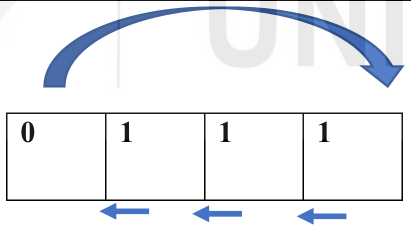
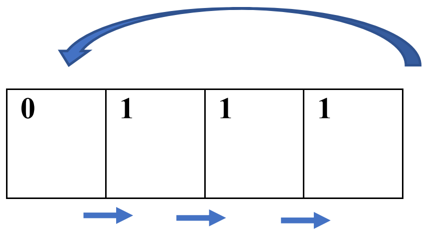
You may please notice that upper three bits in the result (A OR B) remains unchanged, whereas, the LSB is set to 1. Hence, least significant bit of A is set to 1 by performing the OR operation. Similarly, AND function can be used to clear the selective bit. For example, if for the given A value (1011), you just want to use the

LSB or in other words you would like to make the upper three bits as 0's. This is performed with the help of AND. For this example, the value of B would be selected as 0001. This operation is also called the mask operation and is shown below:

A	1011
B	0001
A AND B	0001

Shift instructions: Shift instructions are to shift the register data bits in left or right direction. In *shl R*, data bits of the register are shifted left where data bit input takes place at the rightmost bit and leftmost bit is lost. Whereas in *shr R*, data bits are shifted right and data is input from the leftmost bit. In *shr R*, rightmost bit is lost. In case of circular shift no bit is lost. In *cil R*, data bits move to the right and rightmost bits is stored in the leftmost bit. In *cir R*, data bits move to the left and leftmost bit is stored in the rightmost bit. The Table 9.2 shows these two types of shift operations.

Table 9.2: Shift instructions

Symbolic representation	**Description**	**Functioning**	**Remarks**
R ← shl R	Shift left register R		Left most bit is lost Answer is 1100
R ← shr R	Shift right register R		Right most bit is lost Answer is 0011
R ← cil R	Register circular shift left		Left most bit is moved to rightmost bit, no bit loss. Answer is 1110
R ← cir R	Register circular shift right		Right most bit is moved to left most bit, no bit loss Answer is 1011

Program control instructions: The address of the next executable instruction is stored in the program counter register. Program control instructions contains the condition, which may cause address alteration in the program counter. Hence, the execution of program control instruction results into change in program counter address breaking the sequence.

Table 9.3 : Program control instructions

Instruction	Mnemonics	Function	Remarks
Branch	BR	To branch at particular address	conditional or unconditional
Jump	JMP	Jump to a specific address	unconditional
Skip	SKP	To skip the next instruction	conditional
Call	CALL	Call is used with subroutines	To call a function
Return	RET	To return back to sequence after function call	To return after executive a function

Branch (BR) and Jump (JMP): Branch and jump instructions are used to change the flow of control of a program to a new instruction address specified as the target of branch and jump instruction. These instructions may be conditional or unconditional. For example,
JMP: is an unconditional branch to an instruction address. It may be used to implement simple loops.
JNE: (jump not equal) is a conditional branch instruction. This instruction checks the zero-flag register to determine, if two operands are equal or not (How the flags would be set? This will be explained in Block 4). The jump to the specified address of instruction will take place, only if the zero flag is not set.

Some of the conditional branch instructions are:
BRP X: branch to memory location X if the result of most recent operation is positive
BRN X: branch to memory location X if the result of most recent operation is negative

In the following example, conditional branch has been utilized in which, PC will be loaded with memory location 707 if the content of accumulator register (AC) is zero. And an unconditional JUMP instruction is to execute the program from a particular memory location 901.

```
DR  ←   0           ; Assign 0 to the memory branch register DR
AC   ←    M[X]      ; Read a value from memory location X and store in AC
BRZ 707             ; Branch to location 707 if AC is zero (Conditional branch)
ADD AC, DR          ; Add the content of DR to AC and store result to AC
…
JUMP 901            ; jump to memory location 901 for further processing.
```

SKIP: The SKIP instruction skips the next instruction to be executed in sequence. Hence, it increments the value of PC by one instruction length. The SKIP can also be conditional. For example, the instruction ISZ skips the next instruction only if the result of the most recent operation is zero.

Subroutine call and return instruction : The subroutine call instcution holds the opcode and address of the start of subroutine. On execution of a subroutine call instruction, first the return address, which is the addresss of the next instruction to subroutine call stored in PC is moved to the memory, and then PC is loaded with the subroutine address from the subroutine call instruction. Once subroutine execution is completed, program has to retun back to the calling program to execute the next instruction after the subroutine call instruciton, which was stored as return address. Therefore, all the subroutines end with the return instruction. A subroutine call is implemented as below:

SP ← SP-1	stack pointer is decreased by one
M[SP]← PC	PC content is pushed onto the stack to store return address
PC ← effective address	From the subroutine call instruction the effective address is moved to the program counter (PC) register.

While the return instruction would be executed as:

PC ← M[SP]	The stored return address is assigned to PC
SP ← SP+1	stack pointer is increased by one

In subroutine call, the return address is saved at one location. Once call function execution is over, PC is loaded with the return address. A typical example of subroutine call and return in the context of 8086 microprocessor is explain in Block 4 of this course.

9.2.3 Stored Program Organization

A computer is organized to have processor registers and memory unit. Memory holds the program instructions as well as the operands/data on which operation is to be performed. For example, in a 16-bit memory of size 4096 words, as shown in Figure 9.3, an instruction of size one word includes an opcode and one operand address. Since the size of the memory is 2^{12} or 4096 words, the operand address would be 12-bits and the remaining 4-bits would represent an opcode, which defines the operation to be performed. Operand address defined in instruction is the memory location which holds 16-bit data.

Accumulator is the processor register. The operations are performed on the memory content and the accumulator data. A detailed and enhanced example of this structure is given in Block 4, which provides details on 8086 microprocessor.

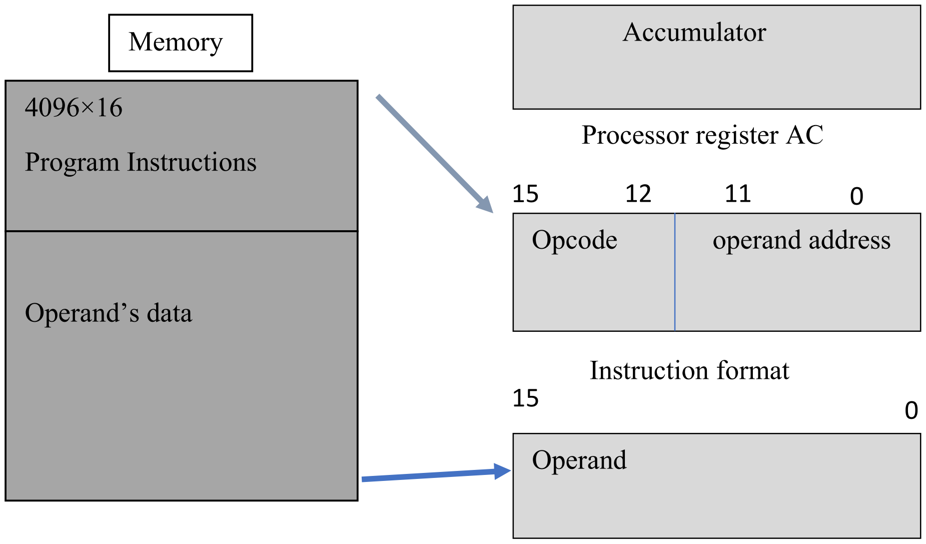


Figure 9.2: Schematic representation of stored program

☞ Check Your Progress – 1

1. What operation can be performed on A =0011 1001 to get a
    a. 1001 1100
    b. 0100 1110
    c. 0010 0111

----

2. Define opcode and operand.

----

3. Consider that A contains 0101 1110 and B contains 0000 1111, then what would be the value of
    (a) Selective Set of A using B
    (b) Masking of A by B

9.3 NUMBER OF ADDRESSES AND INSTRUCTION SIZE

There are instructions set with zero address, one address, two address, three address, and RISC (Reduced Instruction Set Computers). What is the impact of number of addresses in an instruction on instruction size and program? It will be discussed with the help of program that evaluates the arithmetic function:

$$A = (W+X) \times (Y+Z)$$

Zero address instruction: The instructions used for stack organized computers do not have the address field in the instruction. As you can see in the following instructions that no address is assigned in the instruction except for the PUSH and POP commands. The operands are implicit and are taken from the top of the stack. Hence, these are called Zero address instructions. The program to evaluate the value of A is given in Table 9.4.

Table 9.4: Zero Address instructions

PUSH W	The stack top position← W
PUSH X	The stack top position← X
ADD	The stack top position← W+X
PUSH Y	The stack top position← Y
PUSH Z	The stack top position← Z
ADD	The stack top position← Y+Z
MUL	The stack top position← (W+X)×(Y+Z)
POP A	A ← The stack top position

One address instruction: in these instructions, only one address field is present while the second operand is an implicit register operand - the AC accumulator register. The set of instructions that can solve the stated mathematical operation are discussed in

Table 9.5: One address instructions

Instruction	Function	Discussion
LOAD W	AC← M[W]	Data of Memory Address W is loaded in accumulator register
ADD X	AC←AC +M[X]	Data of Memory Address X is added with accumulator data
STORE A	M[A] ←AC	Accumulator data containing (W+X) is stored in Memory Address A
LOAD Y	AC← M[Y]	Data from Memory Address Y is loaded in accumulator
ADD Z	AC←AC +M[Z]	Data Memory Address Z is added with accumulator data
MUL A	AC←AC *M[A]	Memory address A is loaded in accumulator
STORE A	M[A]← AC	Accumulator data is stored in Memory address A

Table 9.5. You can observer that all the instructions in the Table 9.5 have just one operand address specified in the instruction. We can see, instructions are only with one address.

Two addresses instruction: Here, one instruction holds two addresses which may be memory address or processor register. As listed in the following table, all the instructions hold two addresses one of these is a processor register and the other is the memory location.

Table 9.6: Two-addresses instructions

Instruction	Function	Discussion
MOV R1, W	R1← M[W]	Data of Memory Address W is moved to register R1
ADD R1, X	R1←R1 +M[X]	Data of Memory Address X is added with data of register R1 and result is saved in R1
MOV R2, Y	R2← M[Y]	Data of Memory Address Y is moved to register R2
ADD R2, Z	R2←R2 +M[Z]	Data of Memory Address z is added with data of register R2 and result is saved in R2
MUL R1, R2	R1←R1 *R2	Data of registers R1 and R2 is multiplied and saved in R1
MOV A, R1	M[A]←R1	Results of R1 is moved to Memory address A

Three addresses Instructions: These instructions hold three addresses which may be processor register and/or memory. As shown in the following table, all the instructions hold three addresses.

Table 9.7: 3-address instructions

Instruction	Function	Discussion
ADD R1, W, X	R1← M[W]+M[X]	Data of memory location W and X are added and result is stored in processor register R1
ADD R2, Y, Z	R2←M[Y] +M[Z]	Data of memory location Y and Z are added, and result is stored in processor register R2
MUL A, R1, R2	M[A]←R1*R2	Data of processor registers R1 and R2 is multiplied and saved in memory location A .

Advantage of increasing the number of addresses: As you can observe from Tables 9.4, 9.5, 9.6 and 9.7 that number of instructions that are required to perform arithmetic operation decreases.
Disadvantage of increasing the number of addresses: The number of bits required to define an instruction keeps on increasing, thus, increasing the length of an instruction.

Reduced Instruction Set computers (RISC): RISC instructions use three processor registers for operations that perform computations, whereas one memory address and one processor register for the input/output instructions, such as load and store instructions. Instructions performing computational operation do not use memory. The following table illustrate a program based on RISC machine.

Table 9.8: reduced instruction set

Instruction	Function	Working
LOAD R1, W	R1← M[W]	Data of Memory Address W is loaded in processor register R1
LOAD R2, X	R2← M[X]	Data of Memory Address X is loaded in processor register R2
LOAD R3, Y	R3← M[Y]	Data of Memory Address Y is loaded in processor register R3
LOAD R4, Z	R4← M[Z]	Data of Memory Address Z is loaded in processor register R4
ADD R1, R1, R2	R1←R1+R2	Data of processor register R1 and R2 are added and result is stored in R1
ADD R3, R3, R2	R3←R3+R4	Data of processor register R3 and R4 are added and result is stored in R3
MUL R1, R1, R3	R1←R1* R3	Data of processor register R1 and R3 is multiplied added and result is stored in R1
STORE A, R1	M[A]←R1	Final result is stored from register R1 to memory address A

*LOAD: to load data from memory to processor register

*STORE: To store data from processor register to memeory

Advantages of RISC instruction set:

- Memory access is limited to load and store instruction only
- Even though three address instructions are used, however as most of the operands are registers, therefore instruction length is small.
- Single cycle executable instructions
- All operations are performed within processor registers

9.4 INSTRUCTION SET AND FORMAT DESIGN ISSUES

In Figure 9.1 of Section 9.2 an example instruction format is shown. This instruction format stores only one operand address, which is stored in the lowest 24 bits (bit 0 to bit 23) for the instruction. In case, this address is a direct operand address, then such an instruction format may support only 2^{24} = 16 M words memory. This computation assumes that each memory address is an address of one memory word. The opcode size is 7 bits (bit 24 to bit 30). Therefore, in general, there would be 2^7= 128 possible operation codes for this machine. The most significant bit (bit 31) is an addressing mode bit, which in this instruction format specifies the direct or indirect memory addressing mode. An instruction of a computer can have several addressing modes, which are explained in the next section. Thus, in an instruction format, there are three components: opcode, operand and the addressing mode. Hence, instruction length depends on the number of bits allocated to each component. The following are the issues relating to the instruction format design:

Instruction Length: Instruction length is critical in instruction format. There is trade of in smaller vs longer instructions.

a) More operands in one instruction will result into smaller programs, as discussed three address instructions have smaller program in comparison to instructions having lesser number of addresses. However, it may increase the length of an instruction.
b) Addressing modes: Its length is very crucial, as addressing modes give the flexibility of implementing different function.

Bits allocation to operand and opcode: Bits allocated to operand depends on the addressing mode used. For example, register addressing would require lesser number of bits than a memory address. Even the number of opcode depends on the flexibility, for example, an instruction set may have many different add operation codes for different addition operations. For example, a machine may have several addition operations like memory and one immediate operand, one memory and one register operand, two memory operand etc. each being assigned a different opcode. The addressing mode bits can facilitate bringing down such instructions. RISC computers use only register addressing (except for memory read and write instruction), thus, may simplify the instruction format.

Addressing mode: The bits allocated to addressing mode depends on the number of addressing modes used in the instruction set. The number of bits allocated will be high if number of addressing modes been used are high.

Variable length instruction: Computers use several different types of instruction formats. In certain machines these formats can be of different length. For example, an instruction format using one register operand may be a smaller format, whereas an instruction format using one direct memory operand would be of different instruction length. Complex Instruction sets comes under variable length instructions, abbreviated as CISC.

☞ Check Your Progress – 2

1. A computer with memory unit 512K words of 32 bits each. The computer has 32 registers. A binary instruction code is stored in one word memory, where instructions consist of an indirect bit, an opcode, a memory operand address and a register address.
   - Find the bits of opcode, register address, and memory address.
   - Draw the instruction word format.
   - Find out the bits required for the address and data input of the memory?

   ……………………………………………………………………………………
   ……………………………………………………………………………………

2. Give one instruction for each of the following:
   i. Zero address
   ii. One address
   iii. Two address
   iv. Three address

   ……………………………………………………………………………………
   ……………………………………………………………………………………

9.5 ADDRESSING SCHEMES

There is concept of addressing schemes which can be used by the programmer to get flexibility to do task. There are a number of addressing modes. A machine can support a number of these addressing modes.

Implied mode: In this mode, the operand is an implied operand for the given instruction. For example, CPLA is an instruction, which complements the accumulator register. The instruction does not contain the address of the operand register; however, it has one implied operand, i.e. accumulator register. Few other examples of implied addressing mode instructions are: INCA and DECA, where accumulator content is processed.

INCA	Increments accumulator register content by one
DECA	Decrements accumulator register content by one

9.5.1 Immediate Addressing

In this, the instruction holds the operand on which operation is to be operated, is called direct immediate operand. For example, the following instruction loads the value 20 in the accumulator register. The # sign in this instruction indicates that value 20 is an immediate operand. Thus, in immediate addressing mode an operand is part of the instruction. Suh addressing modes are very useful, when you want to initialize a register to a constant value.

Load AC, #20	Loads an immediate value *20* accumulator register.
Load R, #30	Loads an immediate value *30* in processor register R.

9.5.2 Direct Addressing

In this, the instruction specifies the memory address, where the operand is stored. This is one of the most fundamental addressing modes and is present in most of the machines, including RISC machine. For example, the following example, the content of memory location 200 would be loaded in the accumulator register.

Load AC, 200

In this instruction, memory address 200 holds the operand which is stored in the processor register accumulator. Therefore, instruction address is the effective address. Figure 9.4 illustrates this instruction at location 20. As per Figure 9.4, the location 200 contains the operand 350.

Thus, Effective Address of operand = 200
The value of operand which would be loaded in AC = 350.

Similarly, for an instruction *Load R, 350* in Figure 9.4, the effective address would be 350 and the operand value 10 that is stored in the memory address 300 would be loaded in processor register *R.*

9.5.3 Indirect Addressing

In this addressing mode, the address given in the instruction is the memory address where effective address of operands is stored. For example, consider following instruction in the Figure 9.4:

Load AC (200)

This instruction is stored at memory location 21, as shown in Figure 9.4. The instruction address 200 contains 350 which is the effective address of the operand. The operand stored at 350 is 10. Thus, in the indirect addressing mode for the given example,

Effective Address of operand = Content of location (200) = 350
The value of operand which would be loaded in AC = 10.

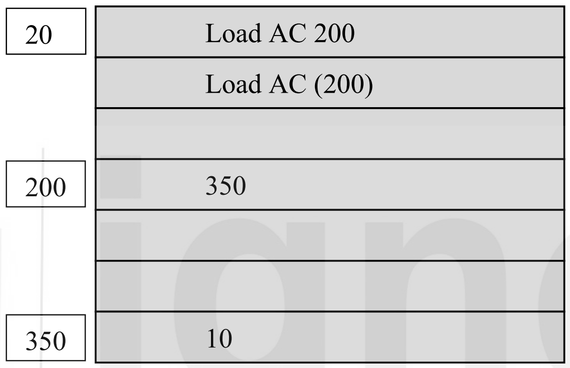


Figure 9.4: Schematic representation of Direct and Indirect addressing modes

9.5.4 Register Addressing

Operands are in register that reside within CPU. For example, consider the instruction:

LD R1

Which loads the content of register R1 to the accumulator register (AC). In this instruction, R1 is a processor register. In case, this instruction is executed on the machine shown in Figure 9.5, the effective address in this case is the address of register R1 itself. On execution of this instruction the content of R1, which is 201, will be loaded in the AC. Therefore, the content of the AC would be 201, after execution of this instruction.

9.5.5 Register Indirect Addressing

In the register indirect addressing, the instruction holds the processor register which carries the address of the operand in memory. For example, the instruction

LD (R1)

In this instruction the operands from memory location whose address in in R1 register is loaded in the accumulator.

AC $\leftarrow$ M[R1]

For example, given the values stored in different registers and memory locations, as shown in Figure 9.5, the instruction LD(R1), which is a register indirect addressing mode will be interpreted as follows:

- The register R1 contains the value 201, therefore, the effective address of operand is memory address 201, i.e. EA=201
- The operand value stored in memory address 201 is 175. Hence, content of memory location 201, which is 175 is loaded in AC register.

Therefore, content of the AC after execution of this instruction would be 175.

9.5.6 Relative Addressing Scheme

In this mode, the operands are stored at the memory address obtained by adding address given in the instruction with the current program counter (PC) location. For example, the instruction:

LD $ADR ; ADR is the value of the operand address field of instruction.

This instruction results in the operation: AC $\leftarrow$ M[PC+ADR], which means that the operands are located at memory address (effective address) PC+ADR. Assuming this is the instruction given in location 100 of the Figure 9.5, which can be written as:

LD $100 ; Please note ADR=100 in this instruction.

The effective address is calculated as: content of PC + ADR of the instruction. Since the current instruction is at PC value 100, on fetch of this instruction PC would be incremented to the address of next instruction, which is 102. Thus, the effective address would be 102+100 = 202, which means that operand is at the memory location 202.

EA =202

Therefore, on execution of this instruction, the content of memory location 202, which is 130, would be loaded in the accumulator.

Therefore, content of the AC after execution of this instruction would be 130.

9.5.7 Base Register Addressing

In the base register addressing scheme a base register and an offset from this register is specified in the instruction. Effective address is found by adding the base register content with the address part of the instruction. For example,

LD BR, ADR

The instruction, as given above, assumes that the instruction specifies the base register (BR in this instruction) and offset from the base register in the address part of the instruction (ADR). Assuming that the instruction at the location 100 is using base register addressing, the instruction can be written as:

LD BR, 100

Since, the value of the base register BR=100 and ADR=100. The effective address would be:

EA=BR+ADR = 100+100 = 200

Therefore, on execution of the instruction the content of the memory location 200, which is 170, would be loaded in the accumulator.
AC is loaded with 170

Therefore, content of the AC after execution of this instruction would be 170.

9.5.8 Indexed Addressing Scheme

In the indexed addressing scheme an address of a memory location is defined and the value of an index register, which represents an offset, is added to it to find the effective address of an operand. This addressing scheme is very useful for accessing an array, where an index register points to offset in an array of values. For example, the following instruction uses the index register XR with the original offset of data is stored in ADR, which is the operand field of the instruction.

LD ADR(XR)

The following operation defines the operation performed by the instruction.

AC ← M[ADR+XR]

For example, as ADR portion of the instruction is 100 and if the index register is referring to 200th element of an array that is XR=200, then effective address would be:

EA =100+200 = 300

Therefore, on execution of this instruction, the content of Memory location 300 would be loaded in the accumulator. Thus, AC will be loaded with 140 (Refer to Figure 9.5).

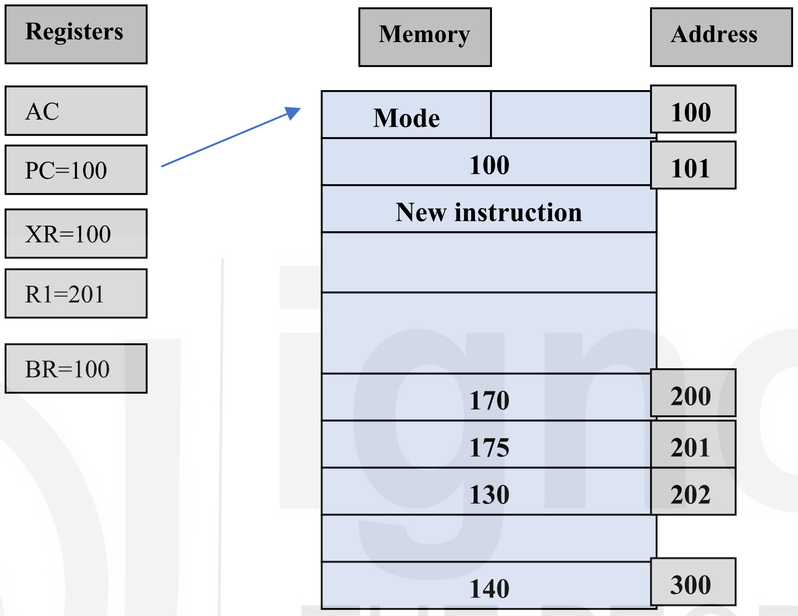


Figure 9.5: An example for addressing modes

9.5.9 Stack Addressing

Stack organization operated on Last in First out (LIFO), e.g. as you arrange books one above other in a bookshelf, while taking out the books last placed book will be lifted first. A stack is collection of finite number of words, as shown in Figure 9.6. It is a 64-word stack of memory locations. A register called Stack Pointer (SP) is used to hold the word address of top of the stack. Data register DR holds the data to be written or read from the stack. Where FULL and EMPTY are one –bit registers to find out if the stack is full or empty. Where data bit FULL is 1 when stack is full, and EMPTY bit is 1 when stack is empty. A new data is inserted in the stack using PUSH, when stack is not full, i.e. FULL=0, the PUSH operation is shown below:

SP ← SP+1	Stack pointer is incremented to an empty location
M[SP] ←DR	Data is written into stack top

Whereas, POP deletes a data item from the top of the stack and puts it in DR register:

DR ←M[SP]	data of top-most stack location is read to DR
SP←SP-1	Stack pointer is decremented by 1

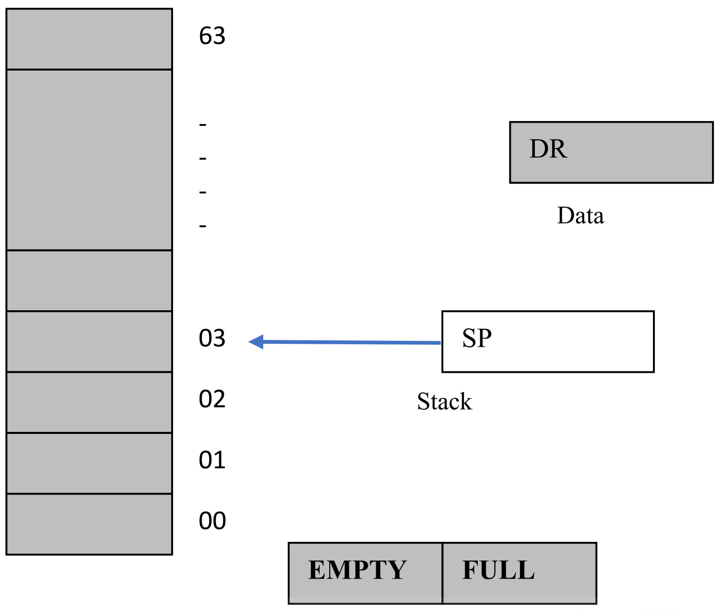


Figure 9.6: Stack organization

Check Your Progress 3:

1. Find the effective address in case of each addressing modes using the Figure 9.7.

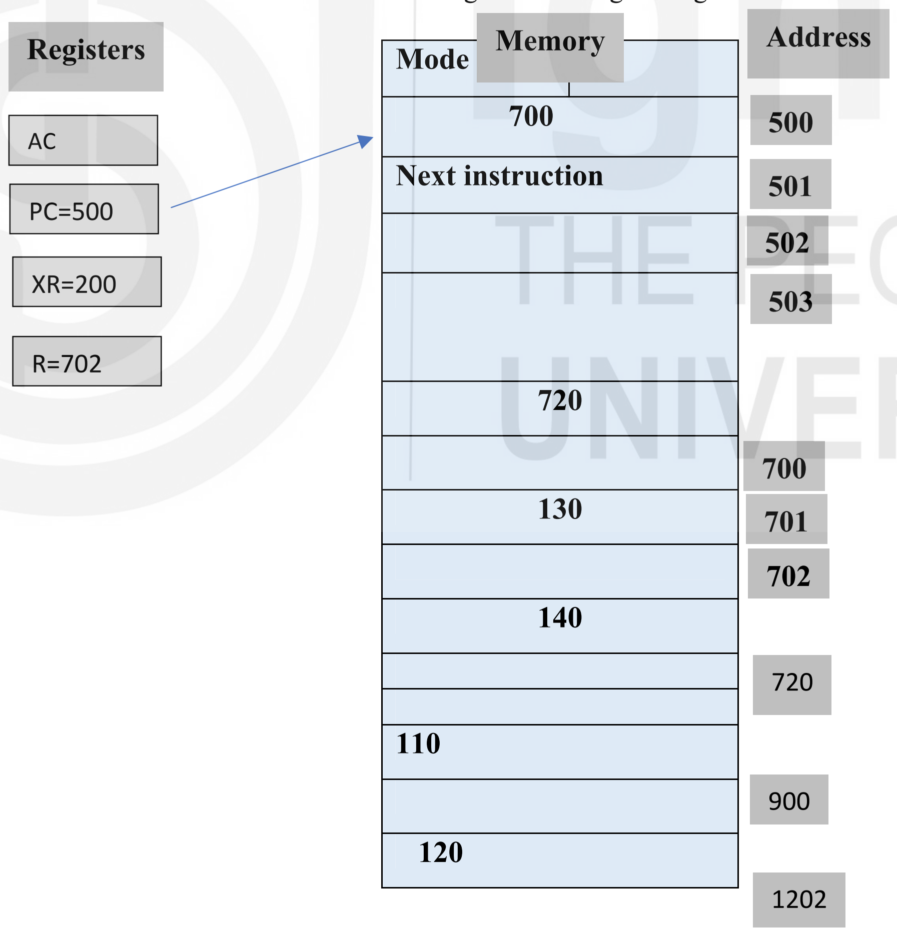


Figure 9.7: The Example Data

You may assume that the instruction is at location 500 and 501. Please note AC is accumulator register; PC is Program counter register; XR is index register; and R is any processor register. The address of the instruction to be executed, is stored in processor register program counter.

Q 2: which instruction set is with zero address?

9.6 SUMMARY

In this unit, basics of instructions such as operand data types, instruction length, and instruction set has been discussed. The instruction format with details of operands and opcode had been explained. In addition to that, various addressing modes, and related instruction size (w.r.t. number of addresses) has been discussed. Depending on the addressing modes, the number of addresses in one instruction and the program complexity has been explained. The flexibility provided to the programmer because of the addressing modes has also been deliberated.

9.7 ANSWER TO CHECK YOUR PROGRESS

Check Your Progress 1

1. a) cir r 1-time
   b) cir r 2-time
   c) cir r 3 –times
2.
   - *Opcode*: opcode is to decides the operation to be performed on the data.
   - *Operand*: operands are the data on which operation is to be performed
3. Consider that A contains 0101 1110 and B contains 0000 1111
   Selective set : A OR B 0101 1111 (to set LSB 4)
   Masking A AND B 0000 1110 (masked MSB 4 bits)

Check Your Progress 2

1 Memory unit =512K

$= 2^{19}$

Hence, size of memory address =19 bits
Register address for selecting one of 32 registers $=2^5$
= 5 bits

Indirect bit =1

Opcode is : 32-(19+5+1)=7 bits

Indirect bit	Opcode	Register	Address
31	30 24	23 19	18 0

1. The instructions are as follows:

i. Zero address: In stack organised computers after two PUSH instructions, which will put two operands on a stack, the zero address ADD instruction will add the content of the top two operands.

ii. One address: In an accumulator-based machine, which uses one of the operands as AC and stores result in AC. The one address instruction: ADD X will add the content of memory location X to the AC register and result of addition will be in AC.

iii. Two address: An instruction like ADD R1, R2 would add registers R1 and R2 and store the result in R1 register.

iv. Three address: An instruction like ADD R3, R1, R2 would add registers R1 and R2 and put the result in R3 register.

Check Your Progress 3

Q1:

Addressing mode	**Effective address**	**Accumulator content**	**Remarks**
Immediate	500	700	Instructions holds the operand data
Direct	700	720	Instruction holds the memory address where operand is stored
Indirect	720	140	Instruction holds the memory address where the operand address is stored
Relative Address	1202	120	Relative address holds the operands at: PC+ instruction content (502+700)
Index Address	900	110	Index address holds the operands at : XR+700 (Index register is added with instruction content)
Register	-	702	Operand is stored in CPU Register

Register indirect	702	130	Register holds the Memory address where operand is stored

Question 2: Stack organization instructions PUSH and POP