BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2103

COMPUTER ORGANIZATION

Unit 3

Central Processing Unit and Instructions

Table of Contents

1. INTRODUCTION

In the previous unit, we studied basic arithmetic operations. We studied integer addition and subtraction, fixed and floating-point numbers, Signed numbers, and other concepts like Booths Algorithm, hardware implementation, division, restoring and non-restoring algorithms, floating-point representations, and IEEE standards. In this unit, we will introduce the different types of instructions and their characteristics, different types of operations, and different types of addressing modes. The operation of a CPUis determined by the instructions it executes. These instructions are referred to as machine instructions. The complete collection of instructions that are executed by a CPU is referred to as CPU's Instruction Set. The instructions are usually represented in binary and the program is usually written in assembly language.

1.1 Objectives:

After studying this unit, you should be able to:

- *List the characteristics of instructions*
- *List the different types of operands.*
- *Explain different types of operations*
- *Discuss different addressing modes.*
- *List and explain the different types of instruction formats*

2. INSTRUCTION CHARACTERISTICS

Each instruction must contain the information required by the CPU for execution. Elements of a machine Instruction are:

- **Operation Code:** Specifies the operation to be performed. The operation is specified by a binary code, known as the **operation code**, or **opcode**.
- **Source Operand Reference:** The operation may involve one or more source operands; that is, operands that are the inputs for the operation.
- **Result Operand Reference:** Where the result is to be stored.
- **Next Instruction Reference:** This tells the CPU where to fetch the next instruction after the execution of the current instruction is complete.

The next instruction to be fetched is located in the main memory or in the case of a virtual memory system it can be either in the main memory or secondary memory. In most cases, the next instruction to be fetched immediatelyfollows the current instruction. In those cases, there is no explicit reference to the next instruction. But when explicit reference is needed, then the main memory or virtual memory address must be supplied. The form in which the address is supplied is discussed in this section.

Source and Result operands can be in one of the three areas:

- **Main (or virtual) memory:** The memory address must be supplied.
- **CPU register: The** CPU contains one or more registers that may be referenced by machine instructions. If only one register exists, referenceto it may be implicit. If more than one register exists, then each registeris assigned a unique number, and the instruction must contain the number of the desired register.
- **I/O device:** The instruction must specify the I/O module or device for theoperation

Instruction Representation

Each instruction is represented by a sequence of bits. The instruction is divided into fields, corresponding to the constituent elements of the instruction. This layout of the instruction is called **Instruction Format**. With most instruction sets, more than one format is used. It is difficult for a programmer and the reader to deal with binary representations of machine instructions. Hence usually the instructions are written in symbolic representations of machine code using English like language called

Mnemonics

Operation codes or Opcodes in short are represented using mnemonics.The format for this instruction is shown in figure 3.1.

Opcode	Operand Reference	Operand Reference

Fig 3.1: A simple instruction format

In most modern CPUs, the first byte contains the opcode, sometimes including the register reference in some of the instructions. The operand references are in the following bytes (byte 2, 3, etc...). Table 3.1 lists some examples of few mnemonics

Table 3.1: Examples of mnemonics

Mnemonics	***Description***
ADD	add
SUB	subtract
MUL	Multiply
LOAD	Load data from memory
MOV A, B	Move contents of B register to A register,where A & B are user-visible registers of CPU

Example 1: An instruction with a register and memory address operands.

8 bits	8 bits	16/32 bits
Opcode	**Operand Ref.**	**Operand Reference**
	register reference	memory address

Example 2: An instruction may span to the second byte. Eg: 2-byteinstruction can be as follows:

12 bits	4 bits
Opcode	**Operand Ref.**
	register reference

For example, consider an instruction, ADD R, Y; Add the contents of data location Y of the memory to the contents of register R. The operation is performed on the data present in location Y and not on its address i.e., Y itself. Previous to the execution of add instruction we give a list of specifications like X= 153, Y=154, and R=20. And the content of memory at location 153 is 10 and at 154 it is 20. After the execution of the given ADD R,Y the result will be in R and will be equal to {R=20+{[Y=154]=20 }=40.

Instruction Types

The instructions fall into one of the following four categories:

1. **Data processing:** Arithmetic and logic instructions.
2. **Data storage:** Memory instructions.
3. **Data movement:** I/O instructions.
4. **Control:** Test and branch instructions.

Number of Addresses

What is the maximum number of addresses one might need in an instruction? Virtually all arithmetic and logic operations are either unary (oneoperand) or binary (two operands). The result of an operation must be stored, suggesting a third address. Finally, after the completion of aninstruction, the next instruction must be fetched, and its address is needed. This line of reasoning suggests that instruction could be required to contain 4 address references: two operands, one result, and the address of the next instruction. In practice, the address of the next instruction ishandled by the **Program Counter (PC)**; therefore most instructions have one, two, or three operand addresses. Table 3.2 shows the Utilization of Instruction Addresses

Table 3.2: Utilization of Instruction Addresses (Non-branching Instructions)

Number of Addresses	***Symbolic Representation***	***Interpretation***
3	OP A, B, C	A <- B OP C
2	OP A, B	A <- A OP B
1	OP A	AC <- AC OP A, (AC is Aaccumulator)
0	OP	T <- T OP (T-1)

Example 3: Program to execute Y = (A – B) / (C + D x E):

Solution:

A) With One-Address Instructions: (requires an accumulator AC)

Table 3.3: Solution to ex.3

LOAD D	AC <- D
MUL E	AC <- AC x 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

B) With Two-Address Instructions:

Table 3.4: Solution to ex.3

MOVE Y, A	Y <- A
SUB Y, B	Y <- Y – B
MOVE T, D	T <- D
MUL T, E	T <- T x E
ADD T, C	T <- T + C
DIV Y, T	Y <- Y / T

C) With Three-Address Instructions:

Table 3.5: Solution to ex.3

SUB Y, A, D	Y <- A – B
MUL T, D, E	T <- D x E
ADD T, T, C	T <- T + C
DIV Y, Y, T	Y <- Y / T

Instruction Set Design

The design of an instruction set is very complex since it affects many different aspects of the computer system. The instruction set defines many of the functions performed by the CPU. The instruction set is a programmer's means of implementation of the CPU.

The fundamental design issues in designing an instruction set are:

1. **Operation Repertoire:** How many and which operations to provide,and how complex operations should be?
2. **Data Types:** The various types of data upon which operations areperformed.
3. **Instruction Format:** Instruction length (in bits), number of addresses,sizes of various fields, and so on.
4. **Registers:** Number of CPU registers that can be referenced byinstructions and their use.
5. **Addressing:** The mode or modes by which the address of an operandis specified.

These issues are highly interrelated and must be considered together indesigning an instruction set.

Self-Assessment Questions – 1

1. _____specifies the operation to be performed.
2. The layout of the instruction is called_____.
3. The address of the next instruction is handled by the_____.

3. CPU WITH SINGLE BUS

Figure 3.2 shows the connection of a CPU using a single Bus. However this approach has a big drawback: little flexibility in choosing the instruction set; most operations have as an operand the content of the accumulator, andthis is also the place where the result goes. By simple, we mean both hardware simplicity and software simplicity: because one operand is always in the accumulator, and the accumulator is also the destination, the instruction encoding is very simple: the instruction must only specify what is the operation to be performed and which is its second operand.

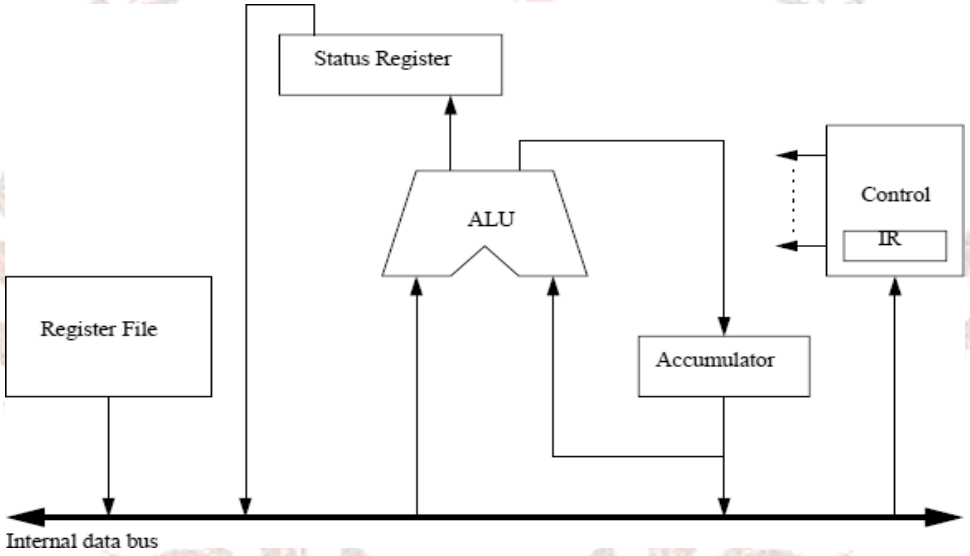

Fig 3.2: CPU using a single internal data bus.

As the technology allowed to move to wider data paths (16, 32, 64, and evenlarger in the future), it has become also possible to specify more complex instruction formats: more explicit operands, more registers, larger offsets, etc.

4. TYPES OF OPERANDS

Each instruction in a program is divided into two parts: Opcode whichspecifies the operation to be done and the operands which specify data to be operated on. So it is clear that the operand is another name for data.

Data Types

The most important general categories of data are:

- Addresses
- Numbers
- Characters
- Logical data.

Addresses

Addresses are also a form of data.

Numbers

All machine languages include numeric data types. Three types of numericaldata commonly used in computers are:

i. Integer or fixed point
ii. Floating point (real numbers)
iii. Decimal (Remember BCD; an instruction set may be able to processBCD numbers.)

i) Integer data type

For unsigned integers: It is a straightforward, simple binary representation.In an N-bit word, the N-bits hold the magnitude of the numbers.

For example

00000000 = 0
00000001 = 1
00101001 = 41
10000000 = 128
11111111 = 255

For signed integers: It involves treating the Most Significant Bit (MSB) in a word as a sign bit. That is if the MSB is 1 the number is considered as negative else positive. In an N-bit word, the (N-1) bit holds the magnitude of the numbers.

For example

00000000 = 0
00000001 = 1
00010010 = 18
10010010 = -18
11111111 = -127

ii) Floating-point numbers: These numbers are represented as $\pm S * B^{\pm E}$

Where Sign: plus or minus

S is significant

B is base for binary it is 2

E is exponent

Figure 3.3 shows the Format of the floating-point number.

Fig 3.3: Format of floating-point number

iii) Decimal: Usual decimal system.

Characters: International Reference Alphabet (IRA) is referred to as ASCII in the USA. Each character in this code is represented by a unique 7-bit pattern, thus 128 different characters can be represented. Some of the patterns represent control characters. ASCII – encoded characters are usually stored and transferred as 8-bits per character. The eight-bit may be set to 1 or 0 for even parity. EBCDIC character set is used in IBM 370 machines. It is an 8-bit code.

Logical Data

With logical data, memory can be used most efficiently for this storage. Logical values are also called as Boolean values which are 1 = true, 0 = false.

Self-Assessment Questions – 2

4. Most operations have content of the_____as an operand.
5. International Reference Alphabet (IRA) is referred to as _____in the USA.

5. TYPES OF OPERATIONS

A set of general types of operations are as follows:

Data transfer

It is the most fundamental type of machine instruction. The data transfermust specify

1. The location of the source and destination. Each location can bememory, register, or top of the stack.
2. The length of the data to be transferred.
3. The mode of addressing for each operand.

If both the operands are CPU registers, then the CPU simply causes data tobe transferred from one register to another. This operation is internal to the CPU. If one or both operands are in memory, then the CPU must perform some or all of the following actions:

1. Calculate the memory address, based on the address mode.
2. If the address refers to virtual memory, translate it from virtual to the actual memory address.
3. Determine whether addressed item is in cache.
4. If not issue a command to memory module.

For example Move, Store, Load (Fetch), Exchange, Clear (Reset), Set, Push, Pop

Arithmetic

Most machines provide basic arithmetic functions like Add, Subtract, Multiply, and Divide. They are invariably provided for signed integer numbers. Often they are also provided for floating-point and packed decimalnumbers. Also, some operations include only a single operand like Absolute, that takes the only absolute value of the operand, Negate that takes the complement of the operands, Increment that increments the value of the operand by 1, Decrement that decrements the value of the operand by 1.

Logical

Machines also provide a variety of operations for manipulating individual bitsof a word often referred to as *bit twiddling.* They are based on Boolean operations like AND, OR, NOT, XOR, Test, Compare, Shift, Rotate, and Set control variables.

Conversion

These instructions are the ones that change the format or operate on the format of the data. Examples are Translate and Convert**.**

I/O, System control

There is a variety of approaches like isolated programmed I/O, memory-mapped I/O, DMA, and so on. Examples: Output (Write), Start I/O, Test I/O.

Transfer of Control

The instruction specifies the operation. In the normal course of events the next instruction to be performed is the one that immediately follows the current instruction in memory. But when the transfer of control type instruction occurs they specify the location of the next instruction that is tobe executed. For example: Jump (Branch), Jump Conditional, Jump to Subroutine, Return, Execute, Skip, Skip Conditional, Halt, Wait (Hold), And No Operation.

System Control

These instructions are reserved for the use of the Operating System. These instructions can only be executed while the processor is in a privileged state,or is executing a program in a special privileged area of memory.

6. ADDRESSING MODES

In general, a program operates on data that reside in the computer's memory. These data can be organized in a variety of ways. If we want to keep track of students' names, we can write them in a list. If we want to associate information with each name, for example, To record telephone numbers or marks in various courses, we may organize this information in the form of a table. Programmers use organizations called ***data structures*** to represent the data used in computations. These include lists, linked lists, arrays, queues, and so on.

Programs are normally written in a high-level language, which enables the programmer to use constants, local and global variables, pointers, andarrays. When translating a high-level language program into assembly language, the compiler must be able to implement these constructs using the facilities provided in the instruction set of the computer in which the program will be run. The different ways in which the location of an operandis specified in instruction are referred to as ***addressing modes***.

There are the following addressing modes:

i) Immediate Addressing
ii) Direct Addressing
iii) Indirect Addressing
iv) Register Addressing
v) Register Indirect Addressing
vi) Displacement Addressing
vii) Stack Addressing

Notation:

A = Contents of an address field in the instruction.

R = Contents of an address field in the instruction that refers to aregister.

EA = Effective (actual) Address of the location containing the referencedoperand.

(X) = Contents of location X.

Table 3.6: Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operandmagnitude
Direct	EA = A	Simple	Limited addressspace
Indirect	EA = (A)	Large address space	Multiple memoryreferences
Register	EA = R	No memory reference	Limited addressspace
Register Indirect	EA = (R)	Large address space	Extra memoryreference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top ofstack	No memory reference	Limited capability

Direct Addressing Mode

Figure 3.4 illustrates the Direct Addressing Mode

- EA = A.
- Address field contains address of operand.
- Effective address (EA) = address field (A).e.g. ADD A
- Add contents of cell 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

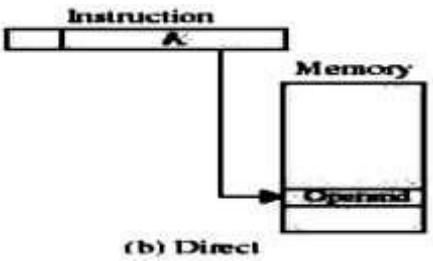

Fig 3.4: Direct Addressing Mode

The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called *Direct*.)

The instruction Move LOC, R2 uses these two modes. Processor registers are used as temporary storage locations where the data in a register are accessed using the Register mode. The Absolute mode can represent global variables in a program. A declaration such as Integer A, B; in a high-level language program will cause the compiler to allocate a memory location to each of the variables A and B. Whenever they are referenced later in the program, the compiler can generate assembly language instructions that use the Absolute mode to access these variables. Next, let us consider the representation of constants. Address and data constants can be represented in assembly language using the immediate mode.

Immediate Addressing Mode

The operand is actually present in the instruction. (Refer to figure 3.5).Operand = A.

- Can be used to define and use constants, or set initial values.
- Operand is part of instruction
- Operand = address field
- e.g. ADD 5
- Add 5 to contents of accumulator
- 5 is operand
- No memory reference to fetch data
- Fast
- Limited range

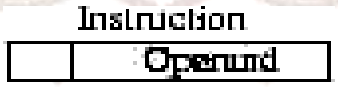

Fig 3.5: Immediate

For example, the instruction Move 200*immediate,* R0 places the value 200in register R0. Clearly, the immediate mode is only used to specify the valueof a source operand. Using a subscript to denote the immediate mode is not appropriate in assembly languages. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand.

Hence, we write the instruction above in the form Move # 200, R0. Constant values are used frequently in high-level language programs. For example, the statement A = B + 6 contains

the constant 6. Assuming that A and B have been declared earlier as variables and may be accessed using the Absolutemode; this statement may be compiled as follows:

Move B, R1

Add #6, R1

Move R1, A

Constants are also used in assembly language to increment a counter, testfor some bit pattern, and so on.

Indirect Addressing Mode

- The memory cell pointed to by the address field contains the address of (pointerto) the operand (refer to figure 3.6).
  EA = (A)
- Look in A, find address (A), and look there for operand
  e.g. ADD (A)
- Add contents of cell pointed to by contents of A to accumulator
- Large address space
- 2n where n = word length
- May be nested, multilevel, cascaded
  e.g. EA = (((A))) Draw the diagram yourself
- Multiple memory accesses to find operand

Hence slower

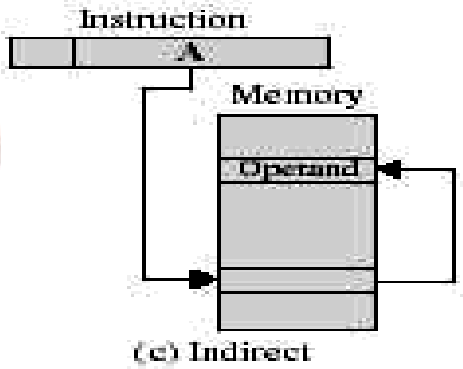

Fig 3.6: Indirect Addressing Mode

Indirect mode – The effective address of the operand is the contents of a register or memory location whose address appears in the instruction. We denote indirection by placing the

name of the register or the memory address given in the instruction in parentheses as illustrated in figure 3.7.

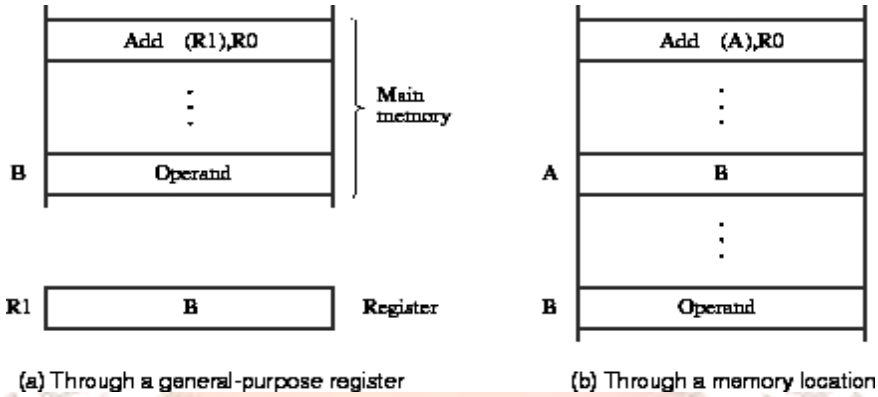

Fig 3.7: Indirect mode

To execute the Add instruction in figure 3.7a the processor uses the value B,which is in register R1, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register R0. Indirect addressing through a memory location is also possible as shown in figure 3.7b.

In this case, the processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand. The register or memory location that contains the address of an operand is called a *pointer*. Indirection and the use of pointersare important and powerful concepts in programming. Consider the analogy of a treasure hunt: In the instructions for the hunt, you may be told to go to a house at a given address. Instead of finding the treasure there, you find a note that gives you another address where you will find the treasure. By changing the note, the location of the treasure can be changed, but the instructions for the hunt remain the same. Changing the note is equivalent tochanging the contents of a pointer in a computer program. For adding a list of numbers, indirect addressing can be used to access successive numbers in the list, resulting in the program shown in figure 3.8. Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value *n* from memory location N into R1 and uses the

immediate addressing mode to place the address value NUM1, which is the address of the first number in the list, into R2. Then it clears R0 to 0.

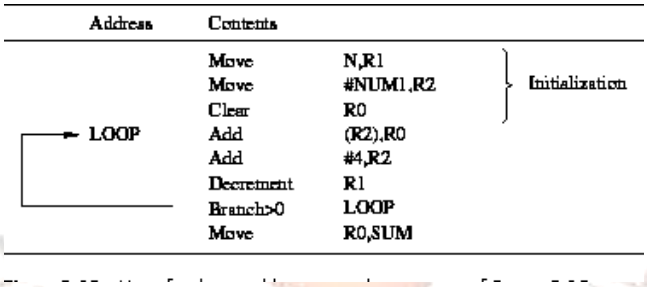

Fig 3.8: Resultant of accessing successive numbers

The instruction, Add (R2), R0 fetches the operand at location NUM1 and adds it to R0. The second Add instruction adds 4 to the contents of the pointer R2 so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop. Consider the C-language statement A=&B; where B is a pointer variable. This statement may be compiled into

Move B, R1

Move (R1), A.

Using indirect addressing through memory, the same action can be achieved with Move (B), A. Despite its apparent simplicity, indirect addressing through memory has proven to be of limited usefulness as an addressing mode, and it is seldom found in modern computers. Indirect addressing through registers is used extensively. The program in Figure shows the flexibility it provides. Also, when absolute addressing is not available, indirect addressing through registers makes it possible to access global variables by first loading the operand's address in a register.

Register Addressing Mode

- The operand is held in a register named in the address field (Refer to figure 3.9)
- EA = R
- Limited number of registers
- Very small address field needed
- Shorter instructions
- Faster instruction fetch

- No memory access
- Very fast execution
- Very limited address space
- Multiple registers helps performance
- Requires good assembly programming or compiler writing

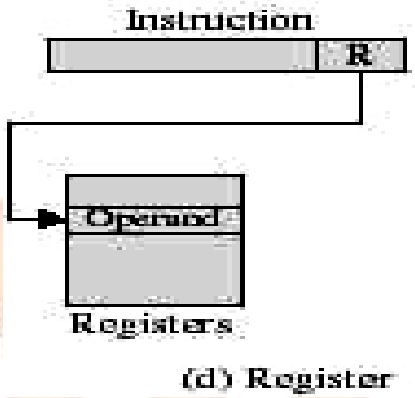

Fig 3.9

Register Indirect Addressing Mode

- Register indirect addressing (refer to figure 3.10)
- EA = (R)
- Operand is in memory cell pointed to by contents of register R
- Large address space (2n)
- Fewer memory access than indirect addressing

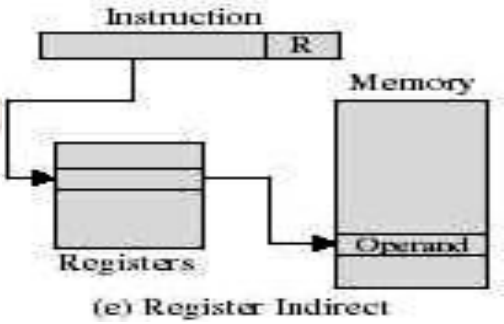

Fig 3.10: Register Indirect Addressing Mode

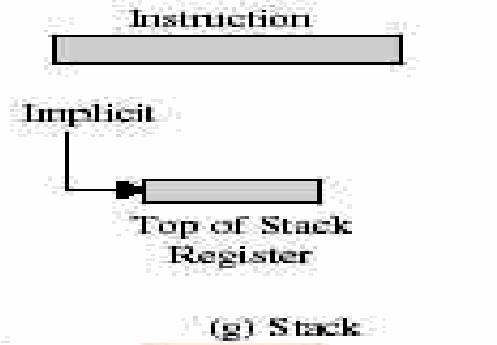

Fig 3.11: Stack Addressing

Indexing and pointers

The next addressing mode we discuss provides a different kind of flexibility for accessing operands. It is useful in dealing with lists and arrays.

Index mode – The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be either a special register provided for this purpose or, more commonly, it may be any one of a set of general-purpose registers in the processor. In either case, it is referred to as an ***index register***. We indicate the Index mode symbolically as X(R*i)* where X denotes the constant value contained in the instruction and R*i* is the name of the register involved. The effective address of the operand is given by EA = X + [R*i*]. The contents of the indexregister are not changed in the process of generating the effective address.

In an assembly language program, the constant X may be given either as anexplicit number or as a symbolic name representing a numerical value. When the instruction is translated into machine code, the constant X is given as a part of the instruction and is usually represented by fewer bits than the word length of the computer. Since X is a signed integer, it must be sign-extended to the register length before being added to the contents of the register.

Figure 3.12 illustrates two ways of using the Index mode. In figure 3.12a, theindex register, R1, contains the address of a memory location, and the valueX defines an ***offset*** (also called a ***displacement***) from this address to the location where the operand is found. An alternative use is illustrated in figure3.12b. Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand. In either case,the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register.

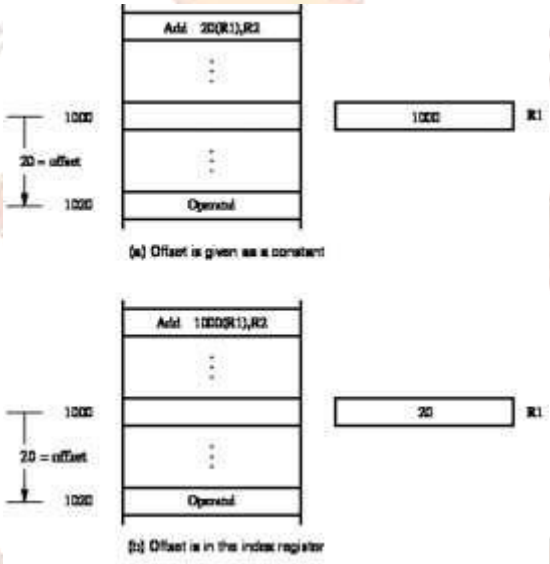

Fig 3.12: Ways of using the Index mode

Displacement Addressing Mode

- EA = A + (R)
- Address field hold two values
- A = base value
- R = register that holds displacement
- or vice versa

Relative Addressing Mode

- A version of displacement addressing
- R = Program counter, PC
- EA = A + (PC)
- i.e. get operand from A cells from current location pointed to by PC

Base-Register Addressing

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
- e.g. segment registers in 80x86

Indexing

- A = base
- R = displacement
- EA = A + R
- Good for accessing arrays
- EA = A + R
- R++

Stack Addressing

- The operand is (implicitly) on top of the stack(refer to figure 3.11)
- e.g. ADD Pop top two items from stack and add

Other additional addressing modes

The two modes described next are useful for accessing data items in successive locations in the memory.

Autoincrement mode – The 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 item in a list. We denote the Autoincrement mode by putting the specified register in parentheses, to show that the contents of the register are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed. Thus, the Autoincrement mode is written as *(*R*i)*+.

Self-Assessment Questions – 3

6. Machines also provide a variety of operations for manipulatingindividual bits of a word often referred to as _____.
7. _____ instructions are reserved for the use of Operating System.
8. The different ways in which the location of an operand is specified in instruction are referred to as_____.
9. In _____the effective address of the operand is generatedby adding a constant value to the contents of a register.

7. INSTRUCTION FORMATS

Instruction Format is defined as the layout of bits in instruction in terms of its constituent parts. An Instruction Format must include opcode implicitly or explicitly and one or more operand(s). For, most instruction sets have usually more than one instruction format.

Instruction Length

The most important design issue is the length of an instruction. It is affected by and affects Memory size, Memory organization, Bus structure, CPU complexity, and CPU speed. There is a trade-off between powerful instruction repertoire and saving space. Apart from this tradeoff, there are other considerations. Either the instruction length should be equal to the memory transfer length or one should be a multiple of the other. A related consideration is the memory transfer rate.

Allocation of Bits

The factors that determine the use of addressing bits are:

- **Number of addressing modes:** Sometimes addressing mode is implicit in the instruction or maybe certain opcodes call for indexing. In other cases the addressing mode must be explicit and one or more bits are needed.
- **Number of operands:** Typically today's machines provide two operands.Each operand may require its own mode indicator or the use of the indicator is limited to one of the address fields.
- **Register versus memory:** A machine must have registers so that the data can be brought into the CPU for processing. One operand address is implicit. The more the registers are used to specify the operands less the number of bits needed.
- **Number of register sets:** Almost all machines have a set of general-purpose registers, with typically 8 or 16 registers in it. These registers can be used to store data or addresses for displacement addressing etc.
- **Address range:** For addresses that refer to the memory locations, the range of addresses is related to the number of address bits. Because of this limitation, direct addressing is rarely used.
- **Address granularity:** It is concerned with addresses that refer to the memory other than registers. In a system with 16 or 32-bit words, an address can refer to a word or a

byte at the designer's choice. Byte addressing is convenient for character manipulation but requires fixed-size memory, and hence more address bits.

Variable-Length Instruction

The designer might provide a variety of instruction formats of different lengths. Addressing can be more flexible, with various combinations ofregisters and memory references plus addressing modes.

The price that needs to be paid is an increase in the complexity of the CPU.

1. Due to the varying number of operands,
2. Due to varying lengths of opcode in some CPUs.

Self-Assessment Questions – 4

10. Most instruction sets have usually more than one instruction format.(State True or False?)
11. A machine needs not to have registers to bring the data into the CPUfor processing. (State True or False?)
12. _____is concerned with addresses that refer to thememory other than registers.

8. SUMMARY

In this unit, we learned the representation of instructions and discussed Arithmetic and logic instructions, Memory instructions, I/O instructions, and Test and branch instructions. We have also seen different types of addressing modes like direct, indirect, immediate, and register, including the important concepts of pointers and indexed addressing. Here we studied the different data types like Addresses, Numbers, Characters, and Logical data. Numbers can be integers or decimal (floating point).

9. TERMINAL QUESTIONS

1. Define the following:
   a. Op-code.
   b. Machine instruction.
2. Discuss the different categories of instructions.
3. Write a note on different data types with examples for each.
4. What do you mean by addressing modes? List the different types ofaddressing modes.
5. Explain the indirect and register addressing modes.
6. Write a note on Instruction formats.

10. ANSWERS

Self-Assessment Questions

1. Opcode
2. Instruction Format
3. Program Counter (PC)
4. Accumulator
5. ASCII
6. bit twiddling
7. System control
8. addressing modes
9. Index mode
10. True
11. False
12. Address granularity

Terminal Questions:

1. Operation Code specifies the operation to be performed. Refer Section 3.2 for details.
2. The different categories of instructions are: Data processing, Data storage, Data movement and Control. Refer Section 3.2 for more details.
3. The most important general categories of data are: Numbers,Characters etc., Refer to section 3.4.
4. The different ways in which the location of an operand is specified in instruction are referred to as addressing modes. Refer to section 3.6 for details.
5. In indirect mode the effective address of the operand is the contents of aregister or memory location whose address appears in the instruction. Refer to section 3.6.
6. Instruction format is defined as the layout of bits in an instruction in terms of its constituent parts. Refer to section 3.7.