

## **UNIT-1**

### **1. Instruction Formats**

**One address.**

**Two address.**

**Zero address.**

**Three addresses and comparison.**

### **2. Addressing modes with numeric examples.**

### **3. Program control.**

**Status bit conditions.**

**Conditional branch instructions.**

### **4. Program interrupts.**

### **5. Types of Interrupts.**

## **INSTRUCTION FORMATS**

The physical and logical structure of computers is normally described in reference manuals provided with the system. Such manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities. They list all hardware-implemented instructions, specify their binary code format, and provide a precise definition of each instruction. A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.

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 registers.
3. A mode field that specifies the way the operand or the effective address is determined.

Other special fields are sometimes employed under certain circumstances, as for example a field that gives the number of shifts in a shift-type instruction.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address. The various addressing modes that have been formulated for digital computers are presented in Sec. 5.5. In this section we are concerned with the address field of an instruction format and consider the effect of including multiple address fields in an instruction.

Operations specified by computer instructions are executed on some data stored in memory or processor registers. Operands residing in processor registers are specified with a register address. A register address is a binary number of  $k$  bits that defines one of  $2^k$  registers in the CPU. Thus a CPU with 16 processor registers R0 through R15 will have a register address field of four bits. The binary number 0101, for example, will designate register R5.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

- 1 Single accumulator organization.
- 2 General register organization.
- 3 Stack organization.

An example of an accumulator-type organization is the basic computer presented in Chap. 5. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD.

Where  $X$  is the address of the operand. The ADD instruction in this case results in the operation  $AC \leftarrow AC + M[X]$ .  $AC$  is the accumulator register and  $M[X]$  symbolizes the memory word located at address  $X$ .

An example of a general register type of organization was presented in Fig. 7.1. The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

ADD            R1, R2, R3

To denote the operation  $R1 \leftarrow R2 + R3$ . The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

ADD            R1, R2

Would denote the operation  $R1 \leftarrow R1 + R2$ . Only register addresses for R1 and R2 need be specified in this instruction.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction

MOV        R1, R2

Denotes the transfer  $R1 \leftarrow R2$  (or  $R2 \leftarrow R1$ , depending on the particular computer). Thus transfer-type instructions need two address fields to specify the source and the destination.

General register-type computers employ two or three address fields in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized by

ADD            R1, X

Would specify the operation  $R1 \leftarrow R + M[X]$ . It has two address fields, one for register R1 and the other for the memory address X.

The stack-organized CPU was presented in Fig. 8-4. Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction

PUSH            X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction

ADD

In a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

Most computers fall into one of the three types of organizations that have just been described. Some computers combine features from more than one organization structure. For example, the Intel 808- microprocessor has seven CPU registers, one of which is an accumulator register. As a consequence; the processor has some of the characteristics of a general register type and some of the characteristics of an accumulator type. All arithmetic and logic instructions, as well as the load and store instructions, use the accumulator register, so these instructions have only one address field. On the other hand, instructions that transfer data among the seven processor registers have a format that contains two register address fields. Moreover, the Intel 8080 processor has a stack pointer and instructions to push and pop from a memory stack. The processor, however, does not have the zero-address-type instructions which are characteristic of a stack-organized CPU.

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement  $X = (A + B) * (C + D)$ .

Using zero, one, two, or three address instruction. We will use the symbols ADD, SUB, MUL, and DIV for the four arithmetic operations; MOV for the transfer-type operation; and LOAD and STORE for transfers to and from memory and AC register. We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

### **THREE-ADDRESS INSTRUCTIONS**

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates  $X = (A + B) * (C + D)$  is shown below, together with comments that explain the register transfer operation of each instruction.

ADD	R1, A, B	$R1 \leftarrow M[A] + M[B]$
ADD	R2, C, D	$R2 \leftarrow M[C] + M[D]$
MUL	X, R1, R2	$M[X] \leftarrow R1 * R2$

It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses. An example of a commercial computer that uses three-address instructions is the Cyber 170. The instruction formats in the Cyber computer are restricted to either three register address fields or two register address fields and one memory address field.

### **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. The program to evaluate  $X = (A + B) * (C + D)$  is as follows:

MOV	R1, A	$R1 \leftarrow M[A]$
ADD	R1, B	$R1 \leftarrow R1 + M[B]$
MOV	R2, C	$R2 \leftarrow M[C]$
ADD	R2, D	$R2 \leftarrow R2 + M[D]$
MUL	R1, R2	$R1 \leftarrow R1 * R2$
MOV	X, R1	$M[X] \leftarrow R1$

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

### **ONE-ADDRESS INSTRUCTIONS**

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second and assume that the AC contains the result of all operations. The program to evaluate  $X = (A + B) * (C + D)$  is

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$

STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$

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.

### **ZERO-ADDRESS INSTRUCTIONS**

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how  $X = (A + B) * (C + D)$  will be written for a stack organized computer. (TOS stands for top of stack)

PUSH	A	$TOS \leftarrow A$
PUSH	B	$TOS \leftarrow B$
ADD		$TOS \leftarrow (A + B)$
PUSH	C	$TOS \leftarrow C$
PUSH	D	$TOS \leftarrow D$
ADD		$TOS \leftarrow (C + D)$
MUL		$TOS \leftarrow (C + D) * (A + B)$
POP	X	$M[X] \leftarrow TOS$

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name “zero-address” is given to this type of computer because of the absence of an address field in the computational instructions.

### **ADDRESSING MODES**

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode of the instruction specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. Computers use addressing mode techniques for the purpose of 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.
- 3 The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.

To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the computer. The control unit of a computer is designed to go through an instruction cycle that is

divided into three major phases:

1. Fetch the instruction from memory
2. Decode the instruction.
3. Execute the instruction.

There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction. Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.

An example of an instruction format with a distinct addressing mode field is shown in Fig. 1. The operation code specifies the operation to be performed. The mode field is used to locate the operands needed for the operation. There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register. Moreover, as discussed in the preceding section, the instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.

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.

**1 Implied Mode:** In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction “complement accumulator” is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions.

Op code	Mode	Address
---------	------	---------

**Figure 1:** Instruction format with mode field

Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

**2 Immediate Mode:** In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate-mode instructions are useful for initializing registers to a constant value.

It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in the register mode.

**3 Register Mode:** In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of  $2^k$  registers.

**4 Register Indirect Mode:** In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the

operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address for the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

**5 Auto increment or Auto decrement Mode:** This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access.

The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational-type instruction. It is the address where control branches in response to a branch-type instruction. We have already defined two addressing modes in previous chapter.

**6 Direct Address Mode:** 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. In a branch-type instruction the address field specifies the actual branch address.

**7 Indirect Address Mode:** In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

**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 whose position in memory is relative to the address of the next instruction. To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to  $826 + 24 = 850$ . This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

**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. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation. Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

**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. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. 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. The base register addressing mode is used in computers to facilitate the relocation of programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of the base register requires updating to reflect the beginning of a new memory segment.

### Numerical Example

	Address	Memory	
<i>PC</i> = 200	200	Load to AC	Mode
	201	Address = 500	
<i>R1</i> = 400	202	Next instruction	
<i>XR</i> = 100			
	399	450	
<i>AC</i>	400	700	
	500	800	
	600	900	
	702	325	
	800	300	

Figure 8-7 Numerical example for addressing modes.



**TABLE 8-4** Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

### **Program Control**

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed

**Typical Program Control Instructions**

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

### **Status Bit Conditions**

It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions can be stored for further analysis. Status bits are also called condition-code bits or flag bits.

The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit C (carry) is set to 1 if the end carry C<sub>8</sub> is 1. It is cleared to 0 if the carry is 0.
2. Bit S (sign) is set to 1 if the highest-order bit F<sub>7</sub> is 1. It is set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise.

In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.