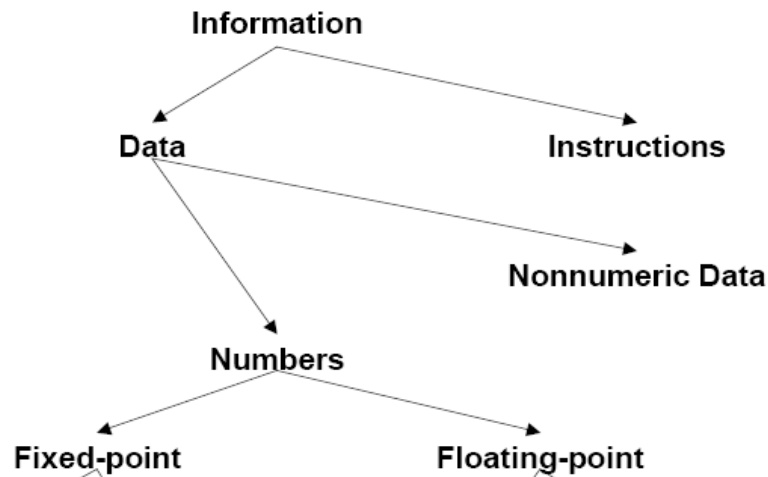


2.1.Information types

SOME BASIC INFORMATION TYPES



Figure(2.1) basic information types.

DATA FORMATS

In selecting the number representation the following factors should be taken into account:

1. TYPES of NUMBERS to be represented e.g., integers, real numbers, complex numbers
2. The RANGE of VALUES to be encountered
3. The PRECISION of the NUMBER
4. THE COST of HARDWARE required to store and process the numbers

INSTRUCTION FORMATS

In selecting the instruction format(s) the following factors should be considered.

1. The number of instructions to be represented.
2. The addressability and addressing modes.
3. The ease of decoding.
4. Type of instruction field (fixed or variable)
5. The cost of hardware required to decode and execute instructions.

2.2 Instruction formats

Instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation code operation part.

OP-CODE	ADDRESS(ES)
---------	-------------

0-, 1-, 2- or 3-addressable instruction formats

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.

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.

2.3 Instructions Types

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

2.3.1 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	$M[X] \leftarrow R1 * R2$

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.

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

2.3.2.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	$M[X] \leftarrow R1$

2.3.3.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 register 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	$M[X] \leftarrow AC$

2.3.4.Zero'Address Instructions

A stack-organized computer does not use an address held for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address held 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 ← A
PUSH B      TOS ← B
ADD         TOS ← (A+B)
PUSH C      TOS ← C
PUSH D      TOS ← D
ADD         TOS ← (C+D)
MUL         TOS ← (A+B)*(C+D)
POP  X      M[X] ← TOS

```

2.4.Addressing Modes

The most common fields found in instruction formats are:

Instruction format with mode field.

Opcode	Mode	Address
--------	------	---------

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.

- The **mode field** is used to locate the operands needed for the operation.

(i.e. A mode field that specifies the way the operand or the effective address).

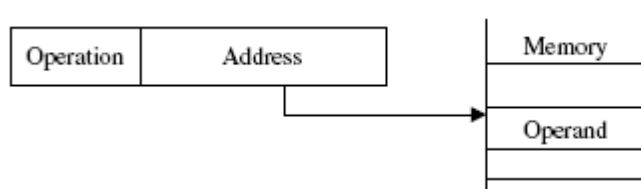
- Operands residing in processor registers are specified with a **register address**

2.4.1. Immediate Mode

According to this addressing mode, the value of the operand is (immediately) available in the instruction itself. Consider, for example, the case of loading the decimal value 1000 into a register R_i. This operation can be performed using an instruction such as the following: MOV R₁, 1000 In this instruction, the operation to be performed is to load a value into a register. The source operand is (immediately) given as 1000, and the destination is the register R₁. It should be noted that in order to indicate that the value 1000 mentioned in the instruction is the operand itself and not its address (immediate mode), it is customary to prefix the operand by the special character . As can be seen the use of the immediate addressing mode is simple. The use of immediate addressing leads to poor programming practice. This is because a change in the value of an operand requires a change in every instruction that uses the immediate value of such an operand. A more flexible addressing mode is explained below.

2.4.2.Direct (Absolute) Mode

According to this addressing mode, the address of the memory location that holds the operand is included in the instruction. Consider, for example, the case of loading the value of the operand stored in memory location x into register R₁. This operation can be performed using an instruction such as MOV R₁,X In this instruction, the source operand is the value stored in the memory location whose address is x, and the destination is the register R₁.Figure (2.2) shows an illustration of the direct addressing mode. For



Figure(2.2)Illustration of the direct memory address mode

example, if the content of the memory location whose address is x was (50) at the time when the instruction MOV R₁, X is executed, then the result of executing such instruction is to load the value 50 into register R₁.

Direct (absolute) addressing mode provides more flexibility compared to the immediate mode. However, it requires the explicit inclusion of the operand address in the instruction. A more flexible addressing mechanism is provided through the use of the indirect addressing mode. This is explained below.

2.4.3. Indirect Memory Mode

In the indirect mode, what is included in the instruction is not the address of the operand, but rather a name a memory location that holds the (effective) address of the operand. In order to indicate the use of indirection in the instruction, it is customary to include the name of the memory location in parentheses. Consider, for example, the instruction `MOV R1,[X]`. This instruction has the memory location `x` enclosed in parentheses, thus indicating indirection. The meaning of this instruction is to load register `R1` with the contents of the memory location whose address is stored at memory address `x`. This types of addressing mode is illustrated in Figure (2.3).

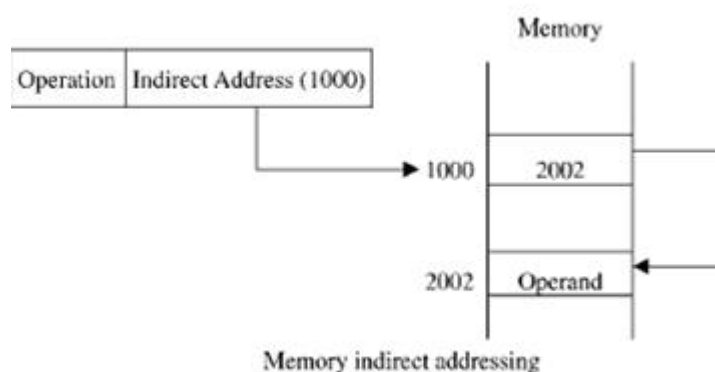


Figure (2.3) Illustration of the indirect memory address mode

2.4.4. 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.

2.4.5. Register Indirect Mode

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

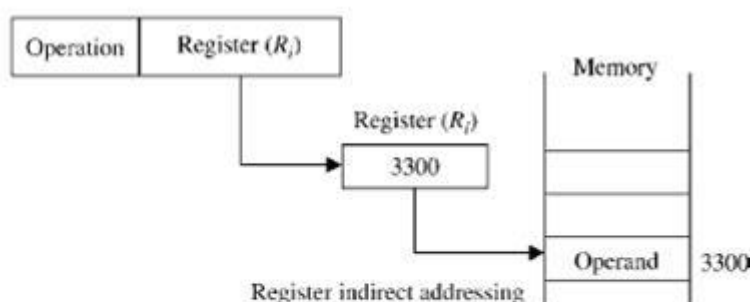


Figure (2.4) Illustration of the indirect register address mode

2.4.6-2.4.7 Autoincrement and Autodecrement Modes:

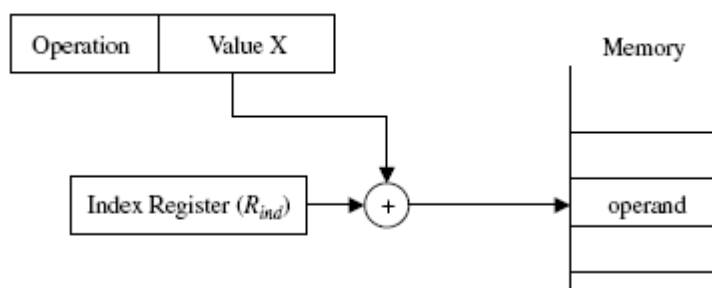
This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.

--- A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following computation: effective address = address part of instruction + content of CPU register

2.4.8. Indexed Mode

In this addressing mode, the address of the operand is obtained by adding a constant to the content of a register, called the index register. Consider, for example, the instruction `MOV R1, X(Rind)`. This instruction loads register R1 with the contents of the memory location whose address is the sum of the contents of register Rind and the value X. Index addressing is indicated in the instruction by including the name of the index register in parentheses and using the symbol X to indicate the

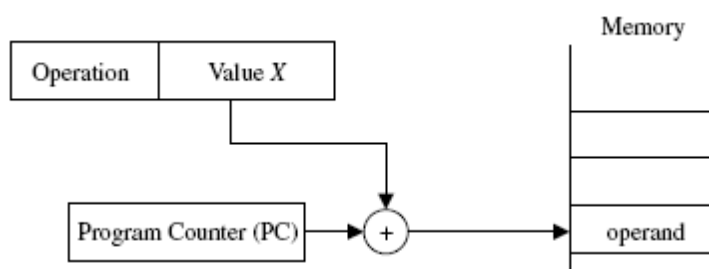
constant to be added. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Figure 2.5 illustrates indexed addressing. As can be seen, indexing requires an additional level of complexity over register indirect addressing.



Figure(2.5)Illustration of the indexed addressing mode

2.4.9.Relative Mode

Recall that in indexed addressing, an index register, R_{ind} , is used. Relative addressing is the same as indexed addressing except that the program counter (PC) replaces the index register. Figure (2.6) illustrates the relative addressing mode.



Figure(2.6)Illustration of relative addressing mode

2.4.10.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.

2.4.11. 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. In this mode the operand is specified in the instruction itself.

Q1//If there is an instruction which is stored on two words at address 200 and 201. The PC has the value 200 for fetching this instruction. The content of processor register R1 is 400. Index register XR has content 100. Compute the effective address and the content of AC after execution

Address	Memory
200	Load to AC Mode
201	Address = 500
202	Next instruction
399	450
400	700
500	800
600	900
702	325
800	300

Q2// A computer uses a memory unit 256k words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.

a-How many bits are there in the operation code, the register code part, and the address part?

b-Draw the instruction word format and indicate the number of bits in each part.

c-How many bits are there in the data and address inputs of the memory

Dr.Eman S. Alshamery