COMPUTER ORGANIZATION AND ARCHITECTURE

MODULE 1

Basic Structure of Computers: Basic Operational Concepts, Bus Structures, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement. Machine Instructions and Programs: Memory Location and Addresses, Memory Operations, Instructions and Instruction Sequencing, Addressing Modes

Textbook 1: Chapter1 – 1.3, 1.4, 1.6 (1.6.1-1.6.4, 1.6.7), Chapter2 – 2.2 to 2.5

BY SADHANA B DEPARTMENT OF ISE ,CEC

Basic Structure of Computers: Basic Operational Concepts, Bus Structures, Performance – Processor Clock,Basic Performance Equation, Clock Rate, Performance Measurement.

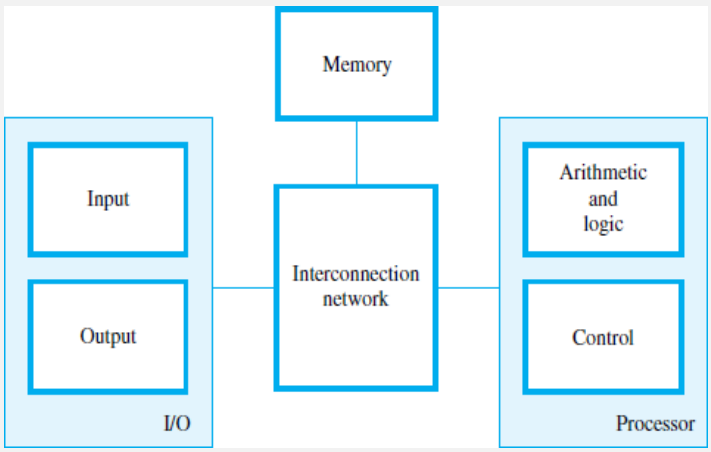
Machine Instructions and Programs: Memory Location and Addresses, Memory Operations, Instructions andInstruction Sequencing, Addressing Modes, Assembly Language, Basic Input and Output Operations, Stacks and Queues, Subroutines, Additional Instructions, Encoding of Machine Instructions

Computer Organization

It describes the function of and design of the various units of digital computer that store and process information.

1.2 Functional Units of computer

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output, and control units, as shown in Figure



MAIN PARTS OF PROCESSOR

1.2.1 Input Unit

Computers accept coded information through input units. The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.

1.2.2 Memory Unit

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

Primary Memory

Primary memory, also called *main memory*, is a fast memory that operates at electronic speeds. Programs must be stored in this memory while they are being executed.

Cache Memory

As an adjunct to the main memory, a smaller, faster RAM unit, called a *cache*, is used to hold sections of a program that are currently being executed, along with any associated data.

Secondary Storage

Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off. Thus additional, less expensive, permanent *secondary storage* is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.

1.2.3 Arithmetic and Logic Unit

Most computer operations are executed in the *arithmetic and logic unit* (ALU) of the processor. Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU.

1.2.4 Output Unit

The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world. A familiar example of such a device is a *printer*. Most printers employ either photocopying techniques, as in laser printers, or ink jet streams.

1.2.5 Control Unit

The memory, arithmetic and logic, and I/O units store and process information and perform input and output operations. The operation of these units must be coordinated in some way. This is the responsibility of the control unit.

Q)With a neat diagram, discuss the basic operational concepts of a computer. **Dec 2015**

Basic Operational Concepts

An Instruction consists of 2 parts,

1) Operation code (Opcode) and

2) Operands.

OPCODE	OPERANDS

The data/operands are stored in memory.

The individual instruction are brought from the memory to the processor. Then, the processor performs the specified operation.

• Let us see a typical instruction

ADD LOCA, R0

• This instruction is an addition operation. The following are the steps to execute the instruction:

Step 1: Fetch the instruction from main-memory into the processor.
Step 2: Fetch the operand at location LOCA from main-memory into the processor.
Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0.
Step 4: Store the result (sum) in

R0. Processor components

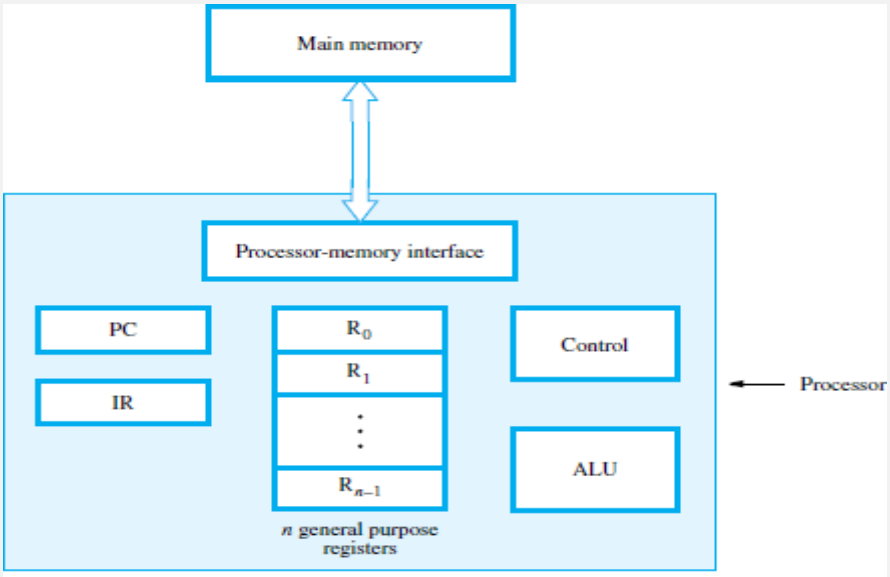


Figure 1.2 Connection between the processor and the main memory.

The **processor** contains ALU, control-circuitry and many registers.

• The processor contains „n" general-purpose registers **R0** through **Rn-1**.
• The **IR** holds the instruction that is currently being executed.
• The **control-unit** generates the timing-signals that determine when a given action is to take place.
• The **PC** contains the memory-address of the next-instruction to be fetched & executed.
• During the execution of an instruction, the contents of PC are updated to point to next instruction.
• The **MAR** holds the address of the memory-location to be accessed.
• The **MDR** contains the data to be written into or read out of the addressed location.
• MAR and MDR facilitates the communication with memory.
(IR : Instruction-Register, PC : Program Counter)
(MAR: Memory Address Register, MDR : Memory Data Register)

STEPS TO EXECUTE AN INSTRUCTION

1) The address of first instruction (to be executed) gets loaded into PC.

2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.
3) After certain amount of elapsed time, the first instruction is read out of memory and placedinto MDR.
4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded &executed.
5) To fetch an operand, its address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.
6) Likewise required number of operands is fetched into processor.
7) Finally, ALU performs the desired operation.
8) If the result of this operation is to be stored in the memory, then the result is sent to theMDR.
9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.
10) At some point during execution, contents of PC are incremented to point to next instruction in the program.

BUS STRUCTURE

- A bus is a group of lines that serves as a connecting path for several devices.
- A bus may be lines or wires.
- The lines carry data or address or control signal.
- There are 2 types of Bus structures:

1) Single Bus Structure and 2) Multiple Bus Structure.

1) Single Bus Structure

Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus.

Advantages:

1) Low cost &
2) Flexibility for attaching peripheral devices.

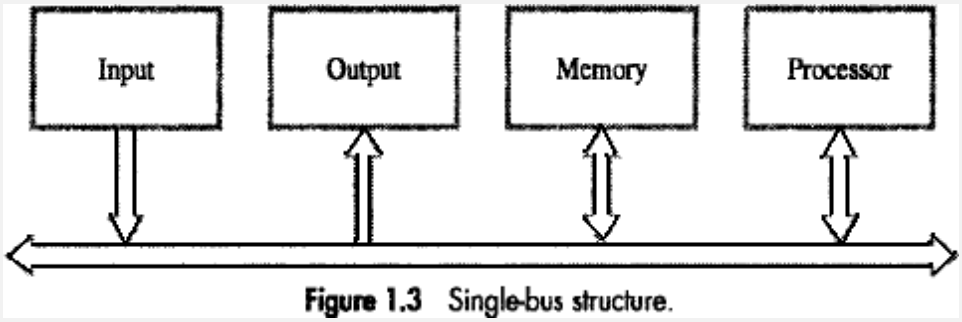


Figure 1.3 Single-bus structure.

2) Multiple Bus Structure

Systems that contain multiple buses achieve more concurrency in operations. Two or more transfers can be carried out at the same time.

Advantage: Better performance.
Disadvantage: Increased cost.

- The devices connected to a bus vary widely in their speed of operation.
- To synchronize their operational-speed, buffer-registers can be used.

- **Buffer Registers**

→ are included with the devices to hold the information during transfers.
→ prevent a high-speed processor from being locked to a slow I/O device during data transfers.

Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement.

PERFORMANCE

- The most important measure of performance of a computer is how quickly it can execute programs.

- The speed of a computer is affected by the design of
  1) Instruction-set.
  2) Hardware & the technology in which the hardware is implemented.
  3) Software including the operating system.

- Because programs are usually written in a HLL, performance is also affected by the compiler that translates programs into machine language. (HLL :High Level Language).

- For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.

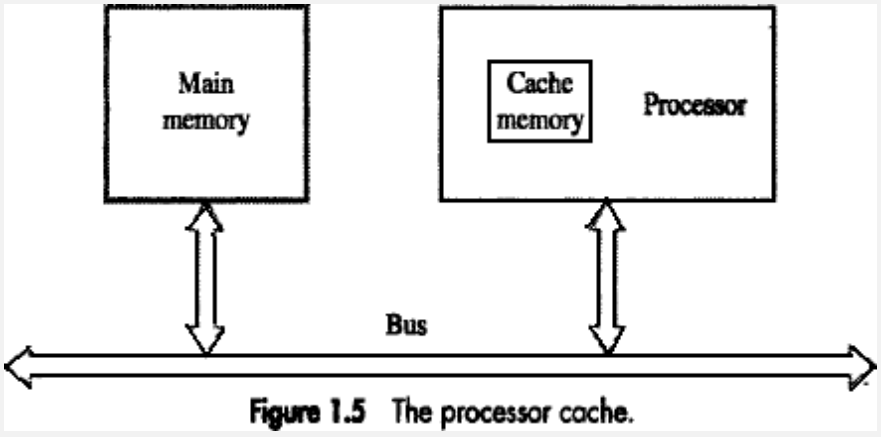


Figure 1.5 The processor cache.

- Let us examine the flow of program instructions and data between the memory & the processor.
- At the start of execution, all program instructions are stored in the main-memory.
- As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.

• Later, if the same instruction is needed a second time, it is read directly from the cache.
•A program will be executed faster if movement of instruction/data between the main-memory and the processor is minimized which is achieved by using the cache.

PROCESSOR CLOCK

• Processor circuits are controlled by a timing signal called a **Clock**.
• The clock defines regular time intervals called **Clock Cycles**.
• To execute a machine instruction, the processor divides the action to be performed into asequence of basic steps such that each step can be completed in one clock cycle.
• Let P = Length of one clock cycle
R = Clock rate.
• Relation between P and R is given by $R = \frac{1}{P}$
• R is measured in cycles per second.
• Cycles per second is also called Hertz (Hz)

Q) Write the basic performance equation. explain the role of each of the parameters in the equation on the performance of the computer. **July 2014**

BASIC PERFORMANCE EQUATION

• Let T = Processor time required to executed a program. N = Actual number of instruction executions.
S = Average number of basic steps needed to execute one machine instruction. R = Clock rate in cycles per second.
• The program execution time is given by $T = \frac{N \times S}{R}$ (1)

• Equ1 is referred to as the basic performance equation.
• To achieve high performance, the computer designer must reduce the value of T, whichmeans reducing N and S, and increasing R.

The value of N is reduced if source program is compiled into fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform. The value of R can be increased by using a higher frequency clock. Care has to be taken while modifying values since changes in one parameter may affect the other.

CLOCK RATE

There are 2 possibilities for increasing the clock rate R:
1) Improving the IC technology makes logic-circuits faster.
This reduces the time needed to compute a basic step. (IC: integrated circuits).
This allows the clock period P to be reduced and the clock rate R to be increased.

2) Reducing the amount of processing done in one basic step also reduces the clock period

P. In presence of a cache, the percentage of accesses to the main-memory is small.
Hence, much of performance-gain expected from the use of faster technology can be realized.

The value of T will be reduced by same factor as R is increased „." S & N are not affected.

PERFORMANCE MEASUREMENT

- Benchmark refers to standard task used to measure how well a processor operates.
- The Performance Measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC: System Performance Evaluation Corporation).
- SPEC Rating is given by

$$\textbf{SPEC rating} = \frac{Running\ time\ on\ the\ reference\ computer}{Running\ time\ on\ the\ computer\ under\ test}$$

- SPEC rating = 50 :The computer under test is 50 times as fast as reference-computer.
- The test is repeated for all the programs in the SPEC suite. Then, the geometric mean of the results is computed.
- Let SPECi = Rating for program „i' in the suite.

Overall SPEC rating for the computer is given by

$$SPEC\ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{1/n}$$

Where n is the number of programs in the suite

Geometrical mean= Is defined as the nth root of the product of n numbers

---

Problem 1:

List the steps needed to execute the machine instruction:

Load LOC A, R0

Solution:

1. Transfer the contents of register PC to register MAR.
2. Issue a Read command to memory. And, then wait until it has transferred the requested word into register MDR.
3. Transfer the instruction from MDR into IR and decode it.
4. Transfer the address LOCA from IR to MAR.
5. Issue a Read command and wait until MDR is loaded.
6. Transfer contents of MDR to the ALU.
7. Transfer contents of R0 to the ALU.
8. Perform addition of the two operands in the ALU and transfer result into R0.
9. Transfer contents of PC to ALU.
10. Add 1 to operand in ALU and transfer incremented address to PC.

Problem 2:

List the steps needed to execute the machine instruction:

Add R1, R2, R3

Solution:

1. Transfer the contents of register PC to register MAR.
2. Issue a Read command to memory. And, then wait until it has transferred the requested word into register MDR.
3. Transfer the instruction from MDR into IR and decode it.
4. Transfer contents of R1 and R2 to the ALU.
5. Perform addition of two operands in the ALU and transfer answer into R3.
6. Transfer contents of PC to ALU.
7. Add 1 to operand in ALU and transfer incremented address to PC.

Problem 3:

(a) Give a short sequence of machine instructions for the task "Add the contents of memory- location A to those of location B, and place the answer in location C".

Instructions:

Load R*i*, LOC
and
Store R*i*, LOC

are the only instructions available to transfer data between memory and the general purpose registers.

Is it possible to use fewer instructions of these types to accomplish the task in part (a)? If yes, give the sequence.

Solution:

(a)

Load A, R0
Load B, R1
Add R0, R1
Store R1, C

(b) Yes;

Move B, C
Add A, C

Problem 4:

A program contains 1000 instructions. Out of that 25% instructions requires 4 clock cycles,40% instructions requires 5 clock cycles and remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine.

Solution:
N = 1000

25% of N= 250 instructions require 4 clock cycles.
40% of N =400 instructions require 5 clock cycles. 35% of N=350 instructions require 3 clock cycles.

T = (N*S)/R=

=(250*4+400*5+350*3)/1X109

=(1000+2000+1050)/1*109

= 4.05 µs.

Problem 5:

For the following processor, obtain the performance. Clock rate = 800 MHz
No. of instructions executed = 1000
Average no of steps needed / machine instruction = 20

Solution:

$$T = \frac{N \times S}{R} = (1000*20)/800 * 10^{6} = 25 \text{ micro sec or } 25*10^{-6} \text{ sec}$$

Problem 6:
(a) Program execution time T is to be examined for a certain high-level language program. The program can be run on a RISC or a CISC computer. Both computers use pipelined instruction execution, but pipelining in the RISC machine is more effective than in the CISC machine. Specifically, the effective value of S in the T expression for the RISC machine is 1.2, bit it is only 1.5 for the CISC machine. Both machines have the same clock rate R. What is the largest allowable value for N, the number of instructions executed on the CISC machine, expressed as a percentage of the N value for the RISC machine, if time for execution on the CISC machine is to be longer than on the RISC machine?

(b) Repeat Part (a) if the clock rate R for the RISC machine is 15 percent higher than that for the CISC machine.

Solution:
(a) Let TR = (NR X SR)/RR & TC = (NC X SC)/RC be execution times on RISC and CISC processors.

Equating execution times and clock rates, we have 1.2NR = 1.5NC

Then

NC/NR = 1.2/1.5 = 0.8

Therefore, the largest allowable value for NC is 80% of NR.

(b) In this case,

$$1.2NR/1.15 = 1.5NC/1.00$$

Then

$$NC/NR = 1.2/(1.15 \times 1.5) = 0.696$$

Therefore, the largest allowable value for NC is 69.6% of NR.

----------MACHINE INSTRUCTIONS & PROGRAMS

MEMORY-LOCATIONS & ADDRESSES

- **Memory** consists of many millions of storage cells (flip-flops).
- Each cell can store a bit of information i.e. 0 or 1 (Figure 2.1).
- Each group of n bits is referred to as a **word** of information, and n is called the **word length**.
- The word length can vary from 8 to 64 bits.
- A unit of 8 bits is called a **byte**.
- Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through 2^k-1 as the addresses of successive-locations in the memory).

- If 2^k = no. of addressable locations;

Then 2^k addresses constitute the address-space of the computer.

For example, a 24-bit address generates an address-space of 2^{24} locations (16 MB).

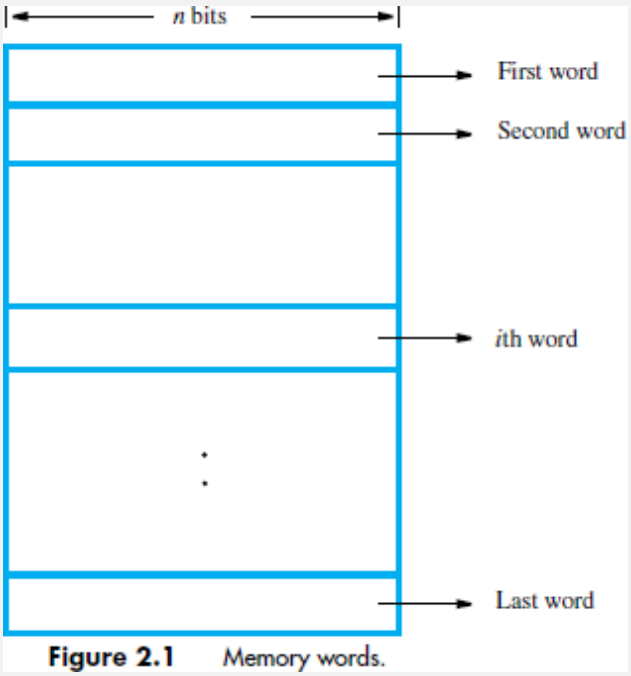


Figure 2.1 Memory words.

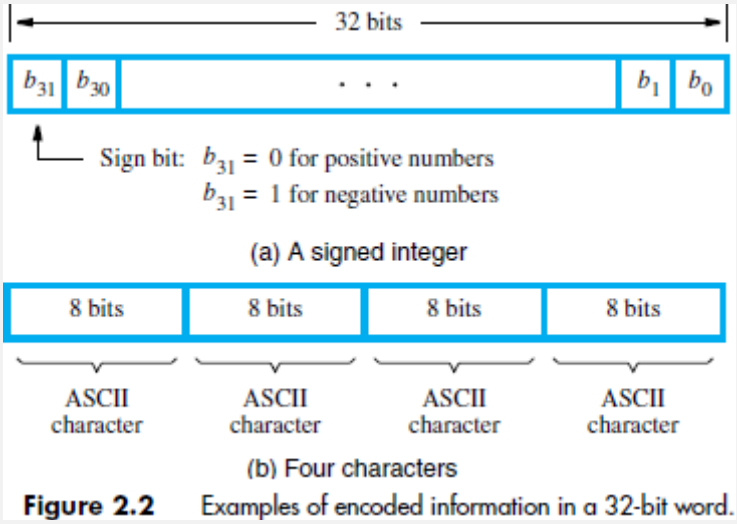


Figure 2.2 Examples of encoded information in a 32-bit word.

BYTE-ADDRESSABILITY

A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits.

- In byte-addressable memory, successive addresses refer to successive byte locations in the memory.
- Byte locations have addresses 0, 1, 2. . . . .
- If the word-length is 32 bits, successive words are located at addresses 0, 4, 8. . with each word having 4 bytes.

Q) Discuss two ways in which byte addresses are assigned **June 2015**
Q) Big-Endian and Little-Endian assignment, explain with necessary figure. **Jan 2014**

BIG-ENDIAN & LITTLE-ENDIAN ASSIGNMENTS

- There are two ways in which byte-addresses are arranged (Figure 2.3).

1) **Big-Endian:** Lower byte-addresses are used for the more significant bytes of the word.

2) **Little-Endian:** Lower byte-addresses are used for the less significant bytes of the word

- In both cases, byte-addresses 0, 4, 8....are taken as the addresses of successive words in the memory.

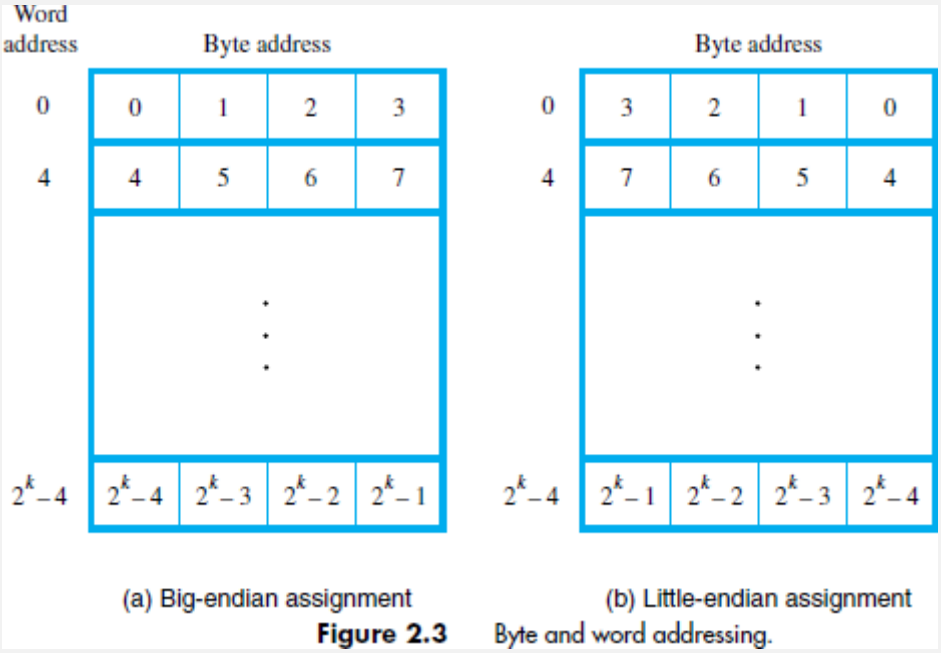


Figure 2.3 Byte and word addressing.

Consider a 32-bit integer (in hex): 0**x**12345678 which consists of 4 bytes: 12, 34, 56, and 78. Hence this integer will occupy 4 bytes in memory.
Assume, we store it at memory address starting 1000. On little-endian, memory will look like

Address	Value
1000	78
1001	56
1002	34
1003	12

On big-endian, memory will look like

Address	Value
1000	12
1001	34
1002	56
1003	78

WORD ALIGNMENT

• Words are said to be **Aligned**in memory if they begin at a byte-address that is a multiple ofthe number of bytes in a word.

• For example,
If the word length is 16(2 bytes), aligned words begin at byte-addresses 0, 2, 4 . . . . .
If the word length is 64(2 bytes), aligned words begin at byte-addresses 0, 8, 16 . . . . .

• Words are said to have **Unaligned Addresses**, if they begin at an arbitrary byte-address.

ACCESSING NUMBERS, CHARACTERS & CHARACTERS STRINGS

• A number usually occupies one word. It can be accessed in the memory by specifying itsword address. Similarly, individual characters can be accessed by their byte-address.

• There are two ways to indicate the length of the string:

1)A special control character with the meaning "end of string" can be used as the lastcharacter in the string.

2)A separate memory word location or register can contain a number indicating the length ofthe string in bytes.

MEMORY OPERATIONS

Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containingthe instruction to be transferred from the memory to the processor. Operands and results must also be moved between the memory and the processor. Thus, two basic operations involving the memory are needed, namely, *Read* and *Write*.

• Two memory operations are:

1) Load (Read/Fetch) &

2) Store (Write).

• The **Load** operation transfers a copy of the contents of a specific memory-location to theprocessor.

The memory contents remain unchanged.

• Steps for Load operation:

1) Processor sends the address of the desired location to the memory.
2) Processor issues „read“ signal to memory to fetch the data.
3) Memory reads the data stored at that address.
4) Memory sends the read data to the processor.

• The **Store** operation transfers the information from the register to the specified memory- location. This will destroy the original contents of that memory-location.

• Steps for Store operation are:

1) Processor sends the address of the memory-location where it wants to store data.
2) Processor issues „write“ signal to memory to store the data.
3) Content of register (MDR) is written into the specified memory-location.

INSTRUCTIONS & INSTRUCTION SEQUENCING

The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

• A computer must have instructions capable of performing 4 types of operations:

1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
3) Program sequencing and control (CALL.RET, LOOP, INT).
4) I/0 transfers (IN, OUT).

REGISTER TRANSFER NOTATION (RTN)

Here we describe the transfer of information from one location in a computer to another.Possible locations that may be involved in such transfers are memory locations, processorregisters, or registers in the I/O subsystem. Most of the time, we identify such locationssymbolically with convenient names.

• The possible locations in which transfer of information occurs are:

1) Memory-location
2) Processor register &
3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	R1 ← [LOC]	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1 ,R2	[R3] ← [R1]+[R2]	Add the contents of register R1 &R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	R1 ← DATAIN	Contents of I/O register DATAIN are transferred into register R1.

ASSEMBLY LANGUAGE NOTATION

• To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES

Instruction Type	Syntax	Example	Description	Instructions for Operation C<-[A]+[B]
Three Address	Opcode Source1,Source2,Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

• The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC (Figure 2.8).

2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses.

This is called *Straight-Line sequencing*.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

• There are 2 phases for Instruction Execution:

1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.

2) Execute Phase: The contents of IR is examined to determine which operation is to beperformed. The specified-operation is then performed by the processor.

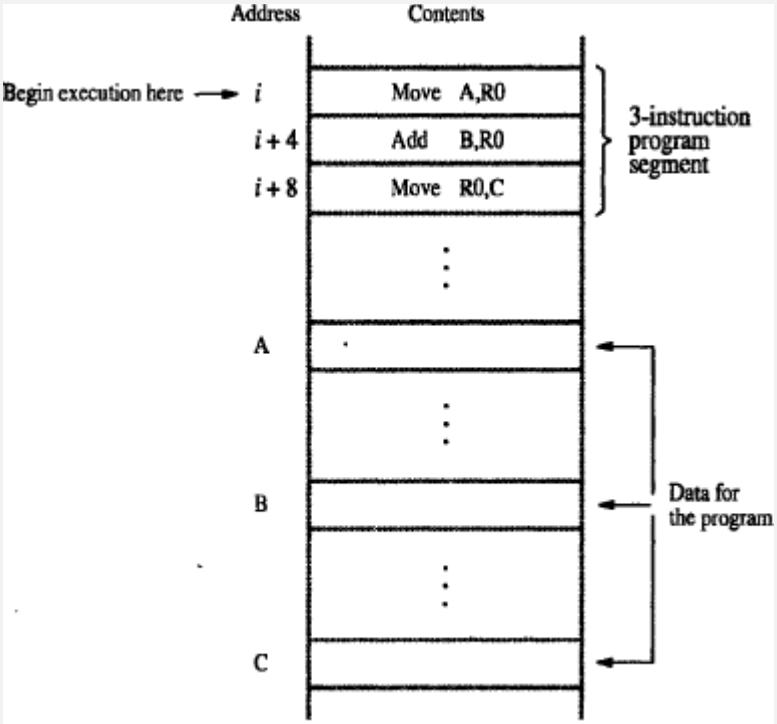


Figure 2.8 A program for C ← [A] + [B].

Program Explanation

- Consider the program for adding a list of n numbers (Figure 2.9).
- The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2…..NUMn.
- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory-location SUM.

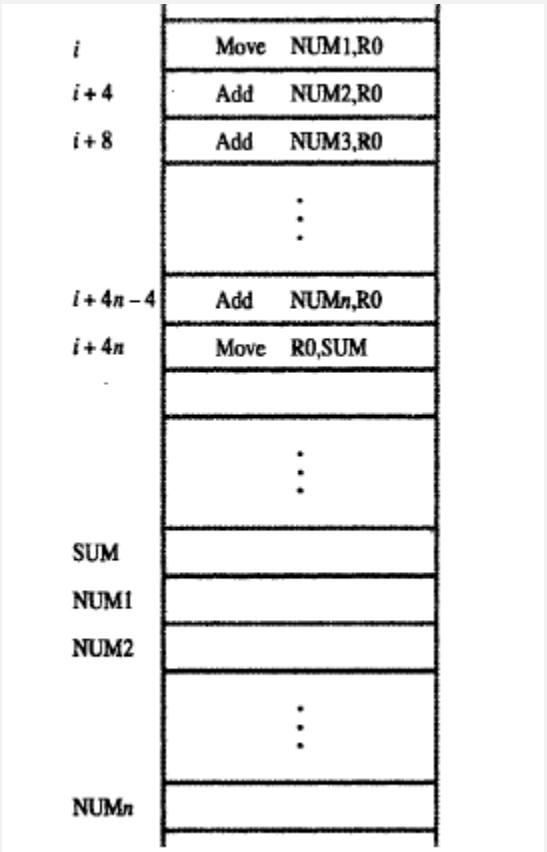


Figure 2.9 A straight-line program for adding *n* numbers.

BRANCHING [Diagram 2.10]

- Consider the task of adding a list of „n" numbers (Figure 2.10).
- Number of entries in the list „n" is stored in memory-location **N**.
- Register **R1** is used as a counter to determine the number of times the loop is executed.
- Content-location N is loaded into register R1 at the beginning of the program.
- The **Loop** is a straight line sequence of instructions executed as many times as needed.

The loop starts at location LOOP and ends at the instruction Branch>0.

- During each pass,

→ address of the next list entry is determined and
→ that entry is fetched and added to R0.

- The instruction *Decrement R1* reduces the contents of R1 by 1 each time through the loop.

- Then **Branch Instruction** loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address called the **Branch Target**.

- A **Conditional Branch Instruction** causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

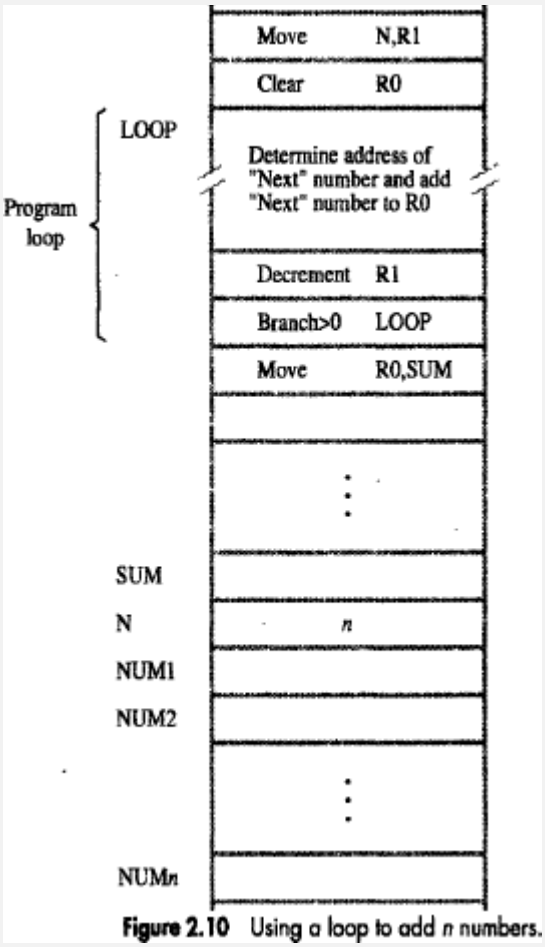


Figure 2.10 Using a loop to add *n* numbers.

CONDITION CODES

• The processor keeps track of information about the results of various operations. This is accomplished by recording the required information in individual bits, called **Condition Code Flags**.

• These flags are grouped together in a special processor-register called the condition code register (or statue register).

• Four commonly used flags are:

1) N (negative) set to 1 if the result is negative, otherwise cleared to 0.
2) Z (zero) set to 1 if the result is 0; otherwise, cleared to 0.
3) V (overflow) set to 1 if arithmetic overflow occurs; otherwise, cleared to 0.
4) C (carry) set to 1 if a carry-out results from the operation; otherwise cleared to 0.

ADDRESSING MODES

Q) What is an addressing mode .Explain different generic addressing modes with an example for each **Dec 2014/Jan 2015**

Q) What is the need for an addressing mode? Explain the following addressing modes with examples: immediate, direct, indirect, index, relative **Dec 2014 /July 2015**

• The different ways in which the location of an operand is specified in an instruction are referred to as

Addressing Modes (Table 2.1).

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	–(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

1) **Immediate Mode**
• The operand is given explicitly in the instruction.
• For example, the instruction
Move #200, R0; Place the value 200 in register R0.
• Clearly, the immediate mode is only used to specify the value of a source-operand.

2) **Register Mode**
• The operand is the contents of a register.
• The name (or address) of the register is given in the instruction.
• Registers are used as temporary storage locations where the data in a register are accessed.
• For example, the instruction
Move R1, R2; Copy content of register R1 into register R2.

3) **Absolute (Direct) Mode**
• The operand is in a memory-location.
• The address of memory-location is given explicitly in the instruction.

• The absolute mode can represent global variables in the program.
• For example, the instruction
Move LOC, R2 ;Copy content of memory-location LOC into register R2.

4) INDIRECTION AND POINTERS

• Instruction does not give the operand or its address explicitly.
• Instead, the instruction provides information from which the new address of the operand canbe determined.
• This address is called **Effective Address (EA)** of the operand.

Indirect Mode

• The EA of the operand is the contents of a register(or memory-location).
• The register (or memory-location) that contains the address of an operand is called a **Pointer**.

• We denote the indirection by
→ name of the register or
→ new address given in the instruction.

E.g: *Add (R1),R0*;The operand is in memory. Register R1 gives the effective-address (B) ofthe operand. The data is read from location B and added to contents ofregister R0.

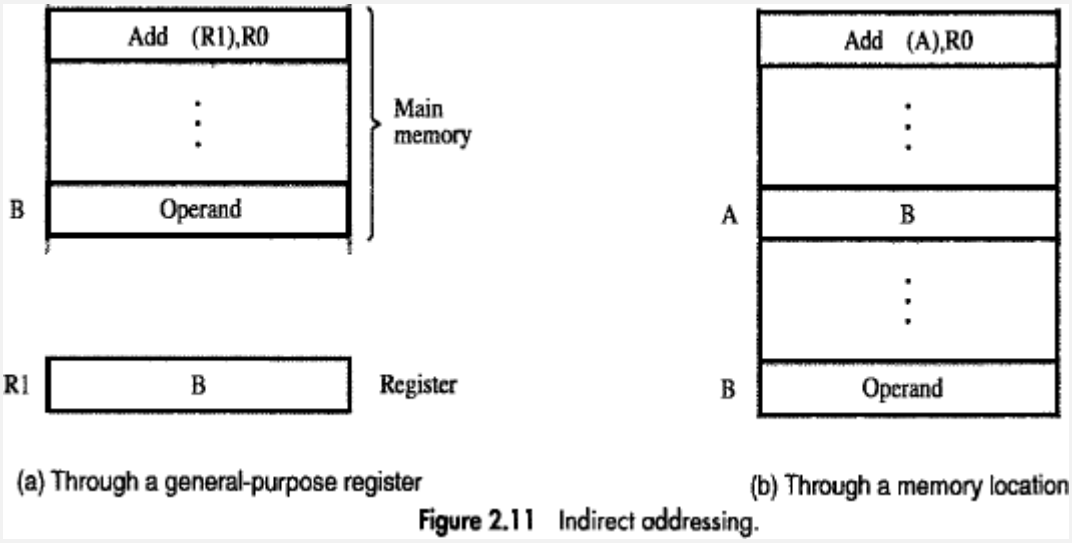


Figure 2.11 Indirect addressing.

• To execute the Add instruction in fig 2.11 (a), the processor uses the value which is in register R1, as the EA 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 fig 2.11(b). 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.

Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Figure 2.12 Use of indirect addressing in the program of Figure 2.10.

Program Explanation

• In above program, 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 Ninto 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.

• The first two instructions in the loop implement the unspecified instruction block starting at LOOP.

• The first time through the loop, 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.

5) INDEXING AND ARRAYS

• A different kind of flexibility for accessing operands is useful in dealing with lists and arrays.

Index mode

• The operation is indicated as X(Ri)

Where X=the constant value which defines an offset(also called a displacement).

Ri=the name of the index register which contains address of a new location.

• The effective-address of the operand is given by EA=X+[Ri]

• The contents of the index-register are not changed in the process of generating the effectiveaddress.

• The constant X may be given either

→ as an explicit number or

→ as a symbolic-name representing a numerical value.

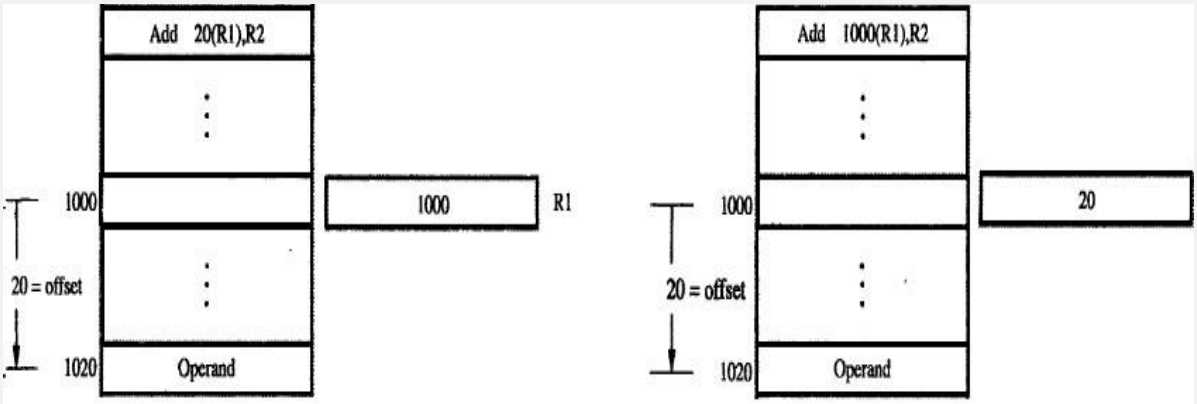


Figure 2.13 Indexed addressing.

• Fig(a) illustrates two ways of using the Index mode. In fig(a), the index register, R1, contains the address of a memory-location, and the value X defines an offset(also called a displacement) from this address to the location where the operand is found.

• To find EA of operand:

Eg: Add 20(R1), R2
EA=>1000+20=102
0

• An alternative use is illustrated in fig(b). 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.

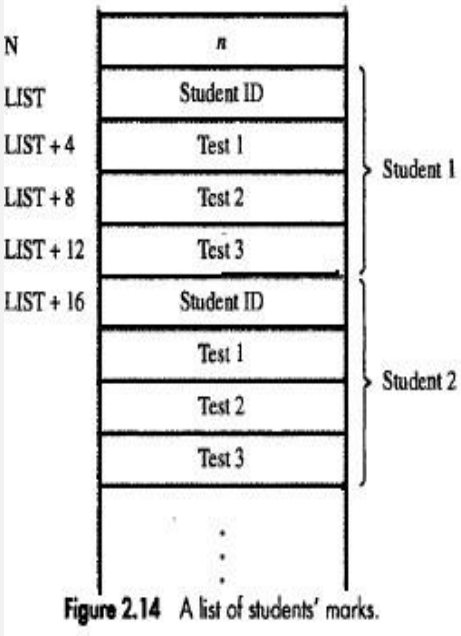


Figure 2.14 A list of students' marks.

```
        Move    #LIST,R0
        Clear   R1
        Clear   R2
        Clear   R3
        Move    N,R4
LOOP    Add     4(R0),R1
        Add     8(R0),R2
        Add     12(R0),R3
        Add     #16,R0
        Decrement R4
        Branch>0  LOOP
        Move    R1,SUM1
        Move    R2,SUM2
        Move    R3,SUM3
```

Figure 2.15 Indexed addressing used in accessing test scores in the list in Figure 2.14.

6) Base with Index Mode

• Another version of the Index mode uses 2 registers which can be denoted as(Ri, Rj)

• Here, a second register may be used to contain the offset X.

• The second register is usually called the *base register*.
• The effective-address of the operand is given by EA=[Ri]+[Rj]
• This form of indexed addressing provides more flexibility in accessing operands because both components of the effective-address can be changed.

7) Base with Index & Offset Mode

• Another version of the Index mode uses 2 registers plus a constant, which can be denoted as X(Ri, Rj)
• The effective-address of the operand is given by EA=X+[Ri]+[Rj]
•This added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (Ri, Rj) part of the addressing-mode. In other words, this mode implements a 3-dimensional array.

8) RELATIVE MODE

• This is similar to index-mode with one difference:
The effective-address is determined using the PC in place of the general purpose register Ri.

• The operation is indicated as X(PC).
• X(PC) denotes an effective-address of the operand which is X locations above or below the current contents of PC.
• Since the addressed-location is identified "relative" to the PC, the name Relative mode is associated with this type of addressing.
• This mode is used commonly in conditional branch instructions.

• An instruction such as

Branch > 0 LOOP; Causes program execution to go to the branch target location identified by name LOOP if branch condition is satisfied.

9) Auto Increment Mode

Effective-address of operand is contents of a register specified in the instruction (Fig: 2.16). After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
Implicitly, the increment amount is 1.
This mode is denoted as

(Ri)+ ;where Ri=pointer-register.

10) Auto Decrement Mode

The contents of a register specified in the instruction are first automatically decremented and are then used as the effective-address of the operand.

This mode is denoted as

-(Ri) ;where Ri=pointer-register.

These 2 modes can be used together to implement an important data structure called a stack.

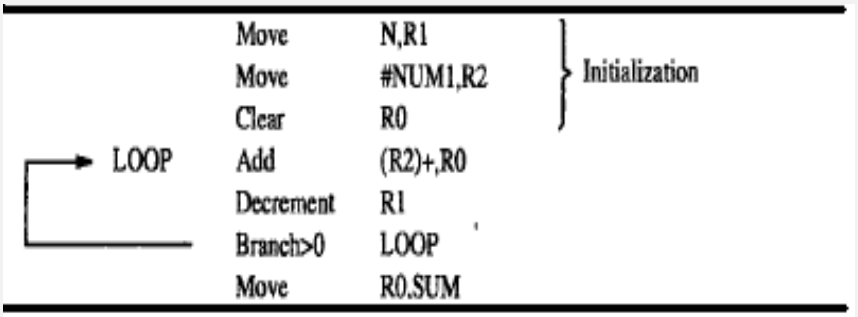


Figure 2.16 The Autoincrement addressing mode used in the program of Figure 2.12.

ASSEMBLY LANGUAGE

- We generally use symbolic-names to write a program.
- A complete set of symbolic-names and rules for their use constitute an **Assembly Language**.
- The set of rules for using the mnemonics in the specification of complete instructions and programs is called the **Syntax** of the language.
- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an **Assembler.**
- The user program in its original alphanumeric text formal is called a **Source Program**, and the assembled machine language program is called an **Object Program**.

For example:

MOVE R0,SUM;

The term MOVE represents OP code for operation performed by instruction.

ADD #5,R3;

Adds number 5 to contents of register R3 & puts the result back into registerR3.

ASSEMBLER DIRECTIVES

- **Directives** are the assembler commands to the assembler concerning the program being assembled.
- These commands are not translated into machine opcode in the object-program.

	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	START

Figure 2.18 Assembly language representation for the program in Figure 2.17.

• **EQU** informs the assembler about the value of an identifier (Figure: 2.18).
Ex*: SUM EQU 200; Informs* assembler that the name SUM should be replaced by the value 200.

• **ORIGIN** tells the assembler about the starting-address of memory-area to place the datablock.
Ex*:* ORIGIN 204 ;Instructs assembler to initiate data-block at memory-locations starting from 204.

• **DATAWORD** directive tells the assembler to load a value into the location.
Ex: *N DATAWORD 100; Informs* the assembler to load data 100 into the memory-location N(204).

• **RESERVE** directive is used to reserve a block of memory.
Ex: *NUM1 RESERVE 400 ;*declares a memory-block of 400 bytes is to be reserved for data.

• **END** directive tells the assembler that this is the end of the source-program text.

• **RETURN** directive identifies the point at which execution of the program should beterminated.
• Any statement that makes instructions or data being placed in a memory-location may begiven a

label. The label(say N or NUM1) is assigned a value equal to the address of that location.

GENERAL FORMAT OF A STATEMENT
• Most assembly languages require statements in a source program to be written in the form:

Label	Operation	Operands	Comment

1) **Label** is an optional name associated with the memory-address where the machine language instruction produced from the statement will be loaded.
2) **Operation Field** contains the OP-code mnemonic of the desired instruction or assembler.
3) **Operand Field** contains addressing information for accessing one or more operands, depending on the type of instruction.
4) **Comment Field** is used for documentation purposes to make program easier to understand.

ASSEMBLY AND EXECUTION OF PROGRAMS

So far, we have used normal words, such as Load, Store, Add, and Branch, for the instruction operations to represent the corresponding binary code patterns. When writing programs for a specific computer, such words are normally replaced by acronyms called *mnemonics*, such as LD, ST, ADD, and BR. A shorthand notation is also useful when identifying registers, such as R3 for register 3. Finally, symbols such as LOC may be defined as needed to represent particular memory locations. A complete set of such symbolic names and rules for their use constitutes a programming language, generally referred to as an *assembly language*.

2.5.1 Assembler Directives

In addition to providing a mechanism for representing instructions in a program, assembly language allows the programmer to specify other information needed to translate the source program into the object program. We have already mentioned that we need to assign numerical values to any names used in a program. Suppose that the name TWENTY isused to represent the value 20. This fact may be conveyed to the assembler program through an *equate* statement such as

TWENTY EQU 20

• **Assembler Program**

→ replaces all symbols denoting operations & addressing-modes with binary-codes used in machine instructions.
→ replaces all names and labels with their actual values.
→ assigns addresses to instructions & data blocks, starting at address given in ORIGIN directive
→ inserts constants that may be given in DATAWORD directives.
→ reserves memory-space as requested by RESERVE directives.

• **Two Pass Assembler** has 2 passes:

1) **First Pass:** Work out all the addresses of labels.
As the assembler scans through a source-program, it keeps track of all names of numerical values that correspond to them in a symbol-table.

2) **Second Pass:** Generate machine code, substituting values for the labels.
When a name appears a second time in the source-program, it is replaced with its value from the table.

• The assembler stores the object-program on a magnetic-disk. The object-program must be loaded into the memory of the computer before it is executed. For this, a **Loader Program** is used.

• **Debugger Program** is used to help the user find the programming errors.

• Debugger program enables the user
→ to stop execution of the object-program at some points of interest &
→ to examine the contents of various processor-registers and memory-location.

Problem 1:
Write a program that can evaluate the expression A*B+C*D In a single-accumulator processor. Assume that the processor has Load, Store, Multiply, and Add instructions and that all values fit in the accumulator

Solution:

A program for the expression is:

```
Load          A
Multiply      B
Store  RESULT
Load          C
Multiply  D Add
RESULT
Store RESULT
```

Problem 2:

Registers R1 and R2 of a computer contains the decimal values 1200 and 4600. What is the effective address of the memory operand in each of the following instructions?

(a) Load 20(R1), R5
(b) Move #3000,R5
(c) Store R5,30(R1,R2)
(d) Add -(R2),R5
(e) Subtract (R1)+,R5

Solution:

(a) EA = [R1]+Offset=1200+20 = 1220
(b) EA = 3000
(c) EA = [R1]+[R2]+Offset = 1200+4600+30=5830 (d) EA = [R2]-1 = 4599
(e) EA = [R1] = 1200

Problem 3:

Registers R1 and R2 of a computer contains the decimal values 2900 and 3300. What is the effective address of the memory operand in each of the following instructions?

(a) Load R1,55(R2)
(b) Move #2000,R7
(c) Store 95(R1,R2),R5
(d) Add (R1)+,R5
(e) Subtract-(R2),R5

Solution:

a) Load R1,55(R2) This is indexed addressing mode.

So EA = 55+R2=55+3300=3355.

b) Move #2000,R7 This is an immediate addressing mode. So, EA = 2000

c) Store 95(R1,R2),R5 This is a variation of indexed addressing mode, in which contents of 2 registers are added with the offset or index to generate EA. So, 95+R1+R2=95+2900+3300=6255.

d) Add (R1)+,R5 This is Autoincrement mode. Contents of R1 are the EA so, 2900 is the EA.

e) Subtract -(R2),R5 This is Auto decrement mode. Here, R2 is subtracted by 4 bytes(assuming 32-bt processor) to generate the EA, so, EA= 3300-4=3296.

Problem 4:

Given a binary pattern in some memory-location, is it possible to tell whether this pattern represents a machine instruction or a number?

Solution:
No; any binary pattern can be interpreted as a number or as an instruction.
