Digital Design and Computer Organization

Module 3- Basic Structure of Computer

Topics to be covered

- Basic Operational Concepts
- Bus Structure
- Performance
- Memory Location and Addresses
- Memory Operation
- Instructions and Instruction sequencing
- Addressing Modes

Introduction

- A list of instructions that performs a task is called a program.
- All instructions are executed in sequential order.
- Usually, the program is stored in the memory.
- The processor then fetches the instructions that make up the program from the memory , one after another and performs the desired operations.

Basic Operational Concepts

- The activity in a computer is governed by Instructions.
- To perform a given task , an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.

- Eg: Add LOCA,R0

- Steps:

1. The instruction is fetched from the memory into processor.
2. Operand at LOCA is fetched and added to the contents of R0.
3. Finally resulting sum is stored at R0.

The above instruction is combined with the memory access operations with an ALU operations.

```
Load LOCA, R1
  Add R1,R0
```

- To perform these instructions we use,

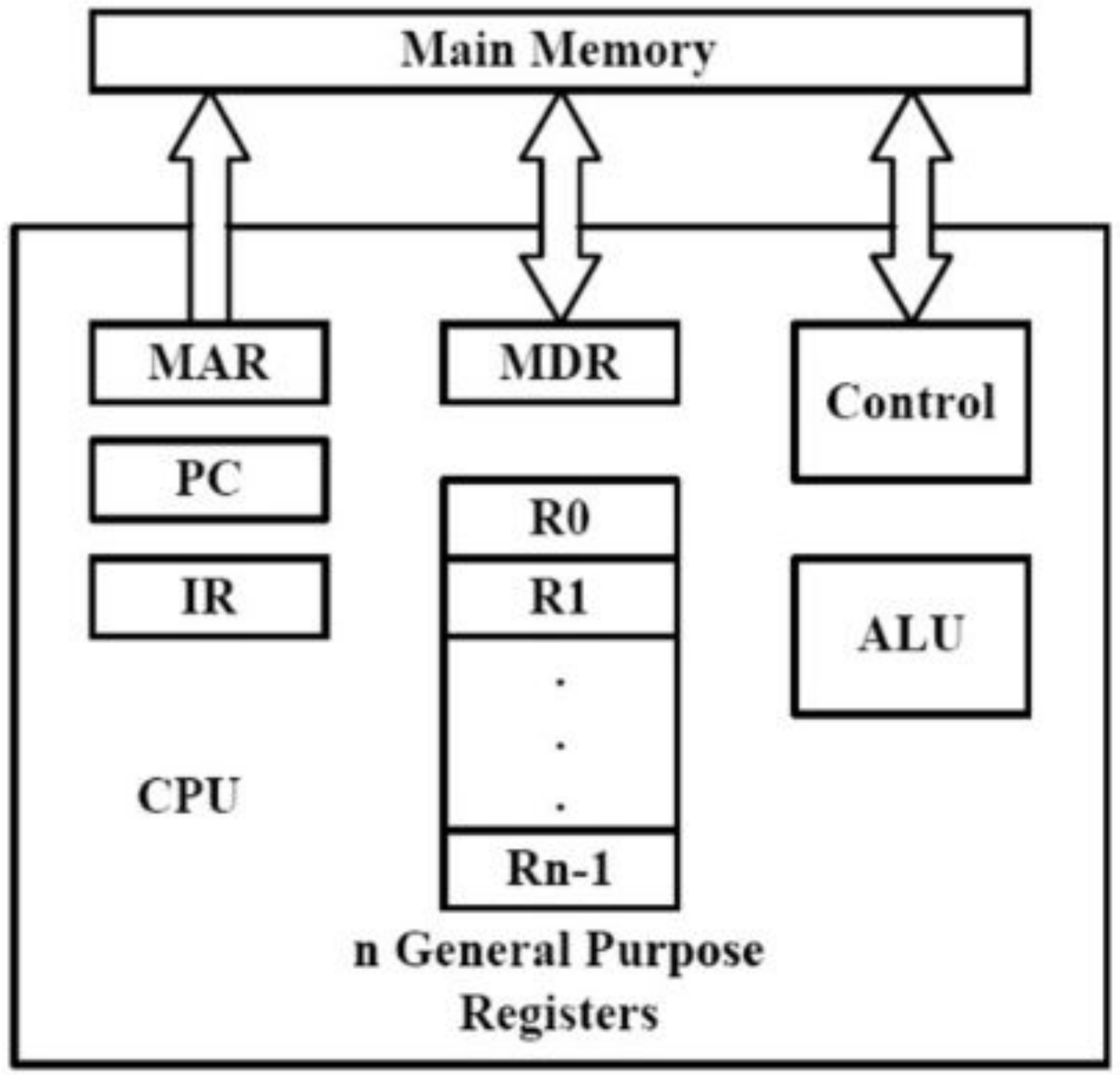
1. ALU
2. CU
3. Different registers.

- Special Functional registers

1. PC [Program Counter]
2. IR [Instruction Register]
3. MAR [Memory address register]
4. MDR [Memory data register]

- **PC -** It keeps track of execution of a program. It contains the memory address of next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.
- IR – Holds the instruction that is currently being executed.
- MAR - It holds the address of the location to be accessed.
- MDR – Data to be written into or read out of the addressed location.

Connection between processor and memory



Some typical Operating steps :

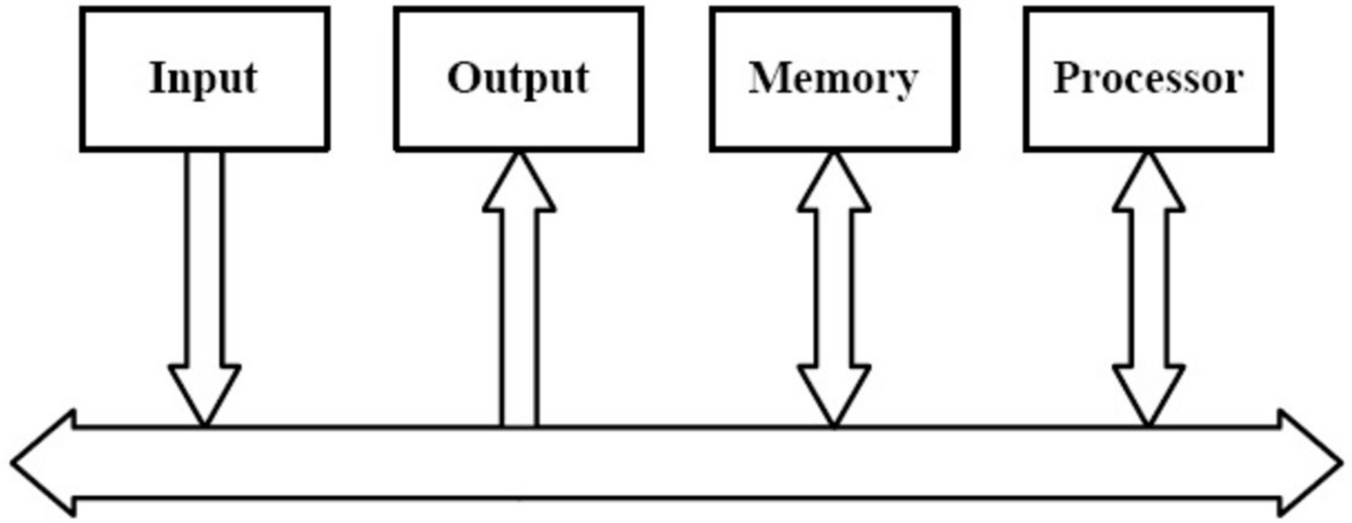
- Program resides in a memory.
- Execution of the program starts when the PC is set to point of first instructions of the program.
- The content of PC is transferred to the MAR and read control signal is sent to the memory.
- After the time elapses , the first instruction of the program is read out of the memory and loaded into the MDR.
- Next, the contents of the MDR are transferred to the IR. The instructions is ready to be decoded and executed.
- If the instructions involves an operations to be performed by the ALU, it is necessary to obtain the required operands.

- If an operand resides in memory , it has to be fetched by sending its address to the MAR and initiating a Read Cycle.
- The operand has been read from memory to MDR, it has transferred from MDR to ALU.
- After one or more operands are fetched. ALU can perform the desired operation.
- If the result of this operation is to be stored in memory, then result is sent to MDR.
- The address of the location where the result is to be stored is sent to MAR and a write cycle is initiated.
- After the execution of instruction, PC are incremented so the PC points to next instruction to be executed.

Interrupt Service routine

- Normally execution of program may be preempted if some device requires urgent servicing.
- E.g., a monitoring device in a computer – controlled industrial process may detect a dangerous condition.
- In order to deal with the immediate situation, the normal execution of current instruction is interrupted by sending the interrupt signal.

Bus Structure



Bus Structures

- A group of lines that serves as a connecting path for several devices is called a Bus.
- In addition to the lines that carry the data, the bus must have lines for address and control purpose.
- The simplest way to interconnect functional units is to use a single bus, as shown in the fig.
- All units are connected to this bus, Because the bus can be used only for one transfer at a time, only two units can actively use the bus at any given time.
- Bus control lines are used to arbitrate multiple requests for use of the bus.

Performance

- The performance of a computer is how quickly it can execute programs.
- The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions.
- Because programs are usually written in a high- level language, performance is also affected by the compiler that translates programs into machine language.
- for the best performance , it is necessary to design the compiler, the machine instruction set, and the hardware in a coordinated way.

Processor Cache

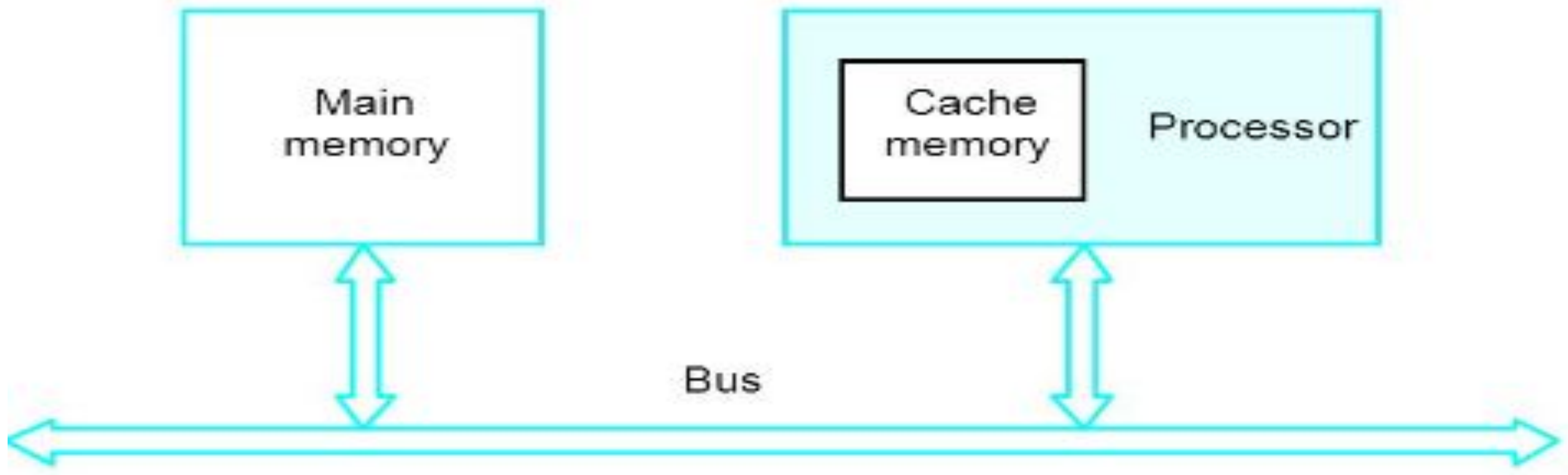


Figure 1.5. The processor cache.

- The cache memory is a part of the processor unit.
- At the start of execution, all program instructions and the required data are stored in the main memory.
- As execution proceeds, instructions are fetched one by one over the bus into processor, and a copy is placed in the cache.
- when the execution of instructions calls for data located in the main memory, data are fetched and a copy is placed in the cache.
- Later , if the same instructions or data items is needed a second time, it is read directly from the cache.
- The time needed to execute a instruction is called the processor time.

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 instructions , the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
- the length P of one clock cycle is an important parameter that affects processor performance.
- Its inverse is the clock rate **R = 1 / P,** which is measured in **cycles per second.**
- In general, **cycles per second is called as Hertz (Hz)**

Basic Performance Equation

- Assume that complete execution of the program requires the execution of N machine languages instructions.
- The number N is the actual number of instruction executions and is not necessarily equal to the number of machine instructions in the object program.
- Some instructions may be executed more than once , which is the case for instructions inside a program loop.
- others may not be executed at all, depending on the input data used.
- Suppose that the average number of basic steps needed to execute one machine instruction is S , where each basic step is completed in one clock cycle.

- If the clock rate is R cycles per second , the program execution time is given by

$$\mathbf{T = N * S / R}$$

This is often referred as the basic performance equation.

- To achieve high performance , the computer designer must seek ways to reduce the value of T, which means reducing N and S, and increasing R.
- The value of N is reduced if the 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 or if the execution of instructions is overlapped.
- Using a higher – frequency clock increases the value or R, which means that the time required to complete a basic execution step is reduced.

Clock Rate

- There are two possibilities for increasing the clock rate, R.
- First , Improving the integrated – circuit (IC) technology makes logic circuits faster , which reduces the time needed to complete a basic steps.
- This allows the clock period P, to be reduced and the clock rate , R to be increased .
- Second, reducing the amount of processing done in one basic step also makes it possible to reduce the clock period, P.
- However , if the actions that have to be performed by an instruction remain the same, the number of basic steps needed may increases.

Performance Measurements

- It is important to assess the performance of a computer.
- Company designers use performance estimates to evaluate the effectiveness of new features.
- Manufacturers use performance indicators in the **marketing process**.
- Buyers use such data to **choose among many available computer models**.
- The **computer community** adopted the idea of measuring computer performance using **benchmark program**.
- **The performance measure is the time it takes a computer to execute a given benchmark**.

- The **only parameter** that properly describes the performance of a computer is the **execution time T**.
- To make comparisons possible, standardized programs must be used.
- A nonprofit organization called **System Performance Evaluation Corporation** (SPEC) selects and publishes representative application programs for different application domains , together with test results for many commercially available computer.
- For general –purpose computers, a suite of benchmark programs was selected in 1989. it was modified somewhat and published in 1995 and again in 2000.

- The SPEC rating is computed as follows

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

The SPEC rating of 50 Means that the computer under test is 50 times faster than the benchmark.

Let SPEC$_i$, be the rating for program i in the suite, The overall SPEC rating for a computer is given by

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

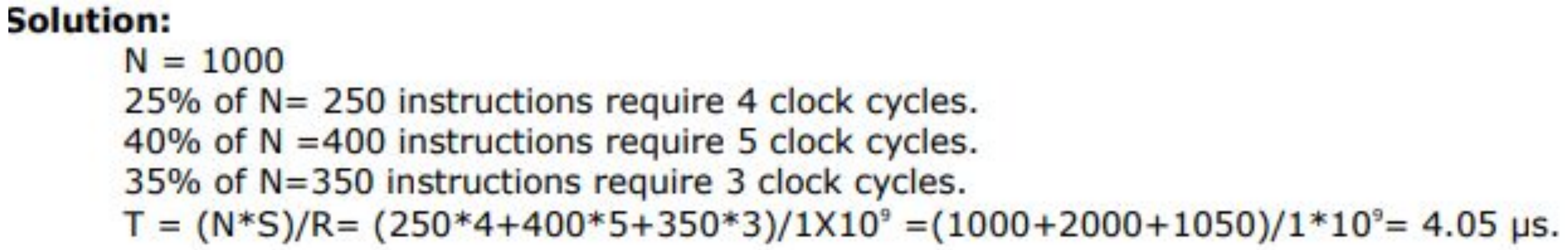
Where n is the number programs in the suite.

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

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

Problems

- **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)/1X10^9 =(1000+2000+1050)/1*10^9= 4.05 μs.

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:

$$T = N * S / R$$

Memory Locations and Addresses

- The number and character operands as well as instructions are stored in the memory of a computer.
- The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.
- The memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation.
- Each group of n bits is referred to as a word of information , and n is called the word length.
- The memory of a computer can be schematically represented as a collection of words.

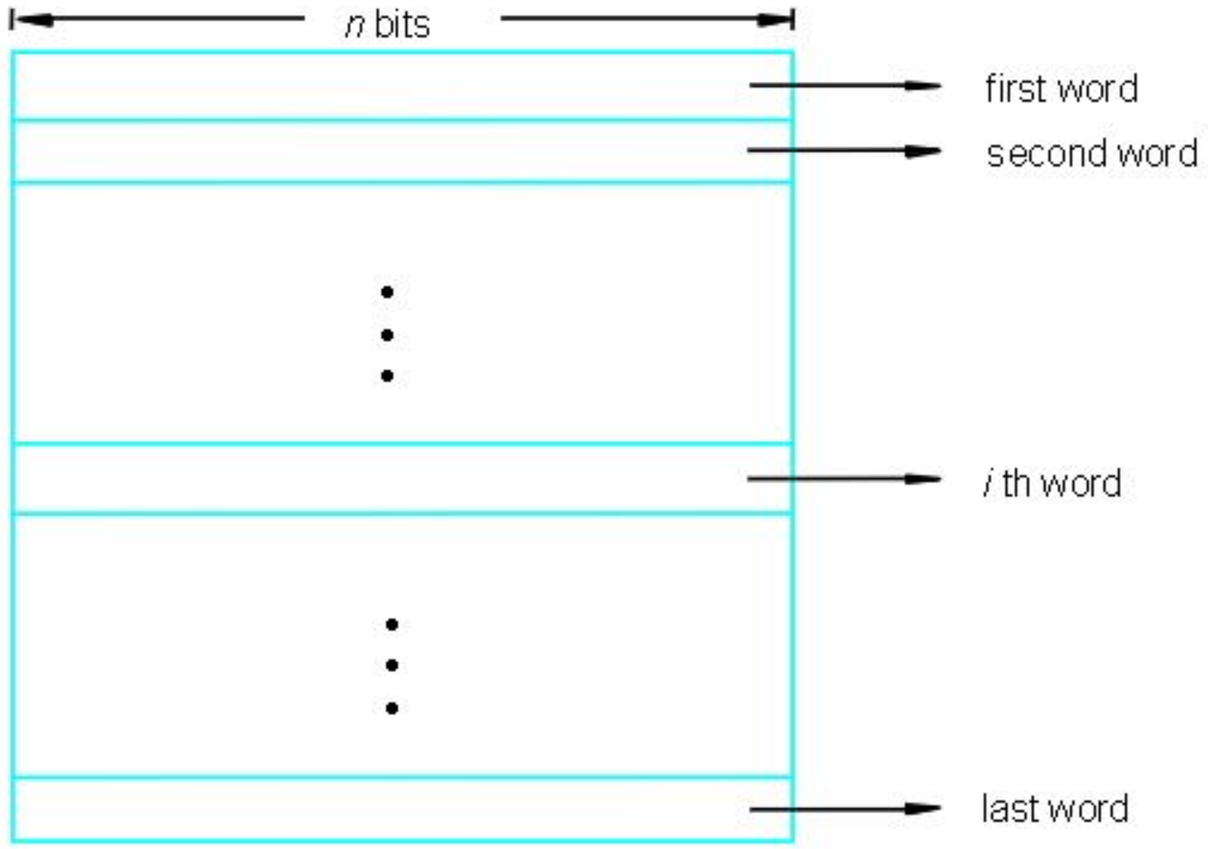


Figure 2.5. Memory words.

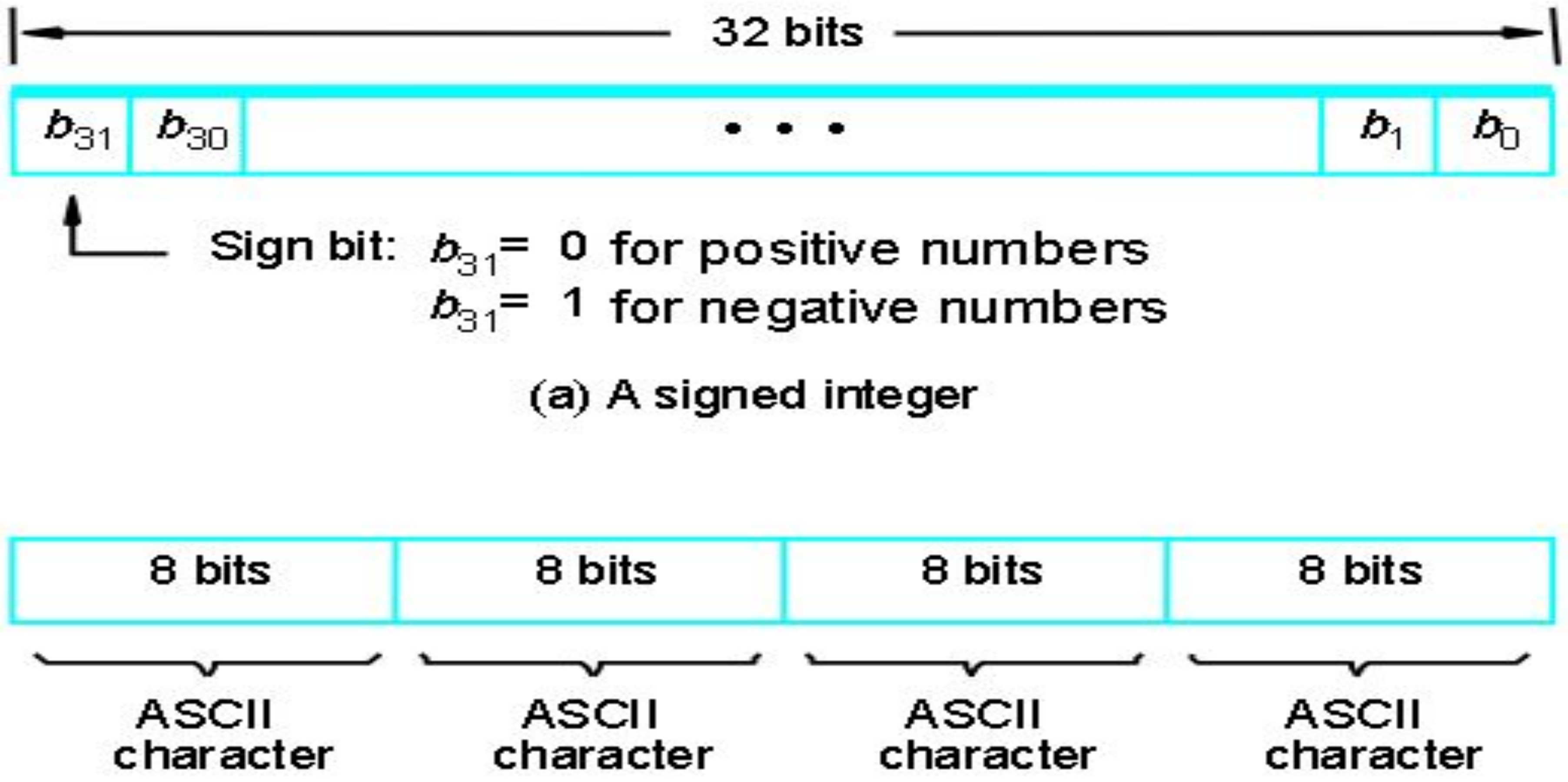
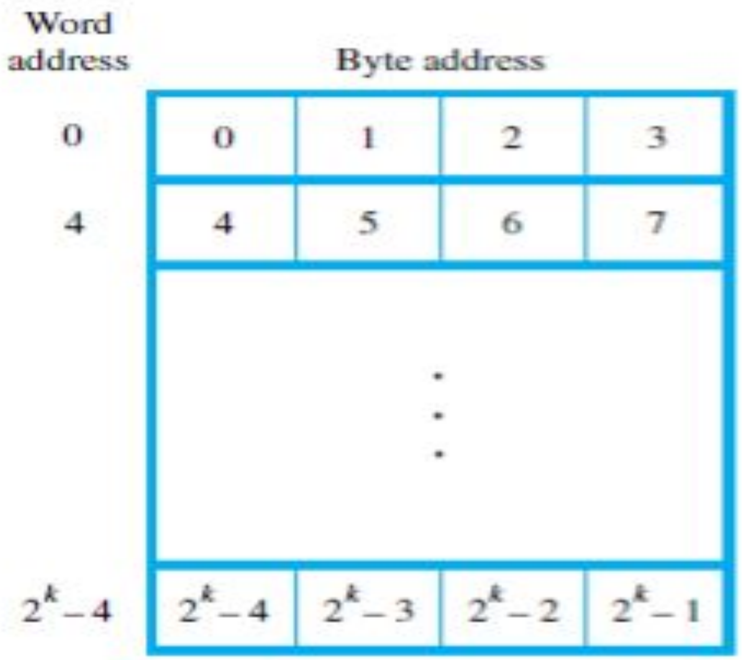


Figure 2.6. Examples of encoded information in a 32-bit word.

- Accessing the memory to store or retrieve a single item of information , either a word or a byte , requires distinct names or addresses for each item location.
- It is customary to use numbers from 0 through 2^K -1, for suitable value of k, as the addresses of successive locations in the memory.
- 2^ K addresses constitute the address space of the computer , and the memory can have up to 2 ^ k addressable locations.

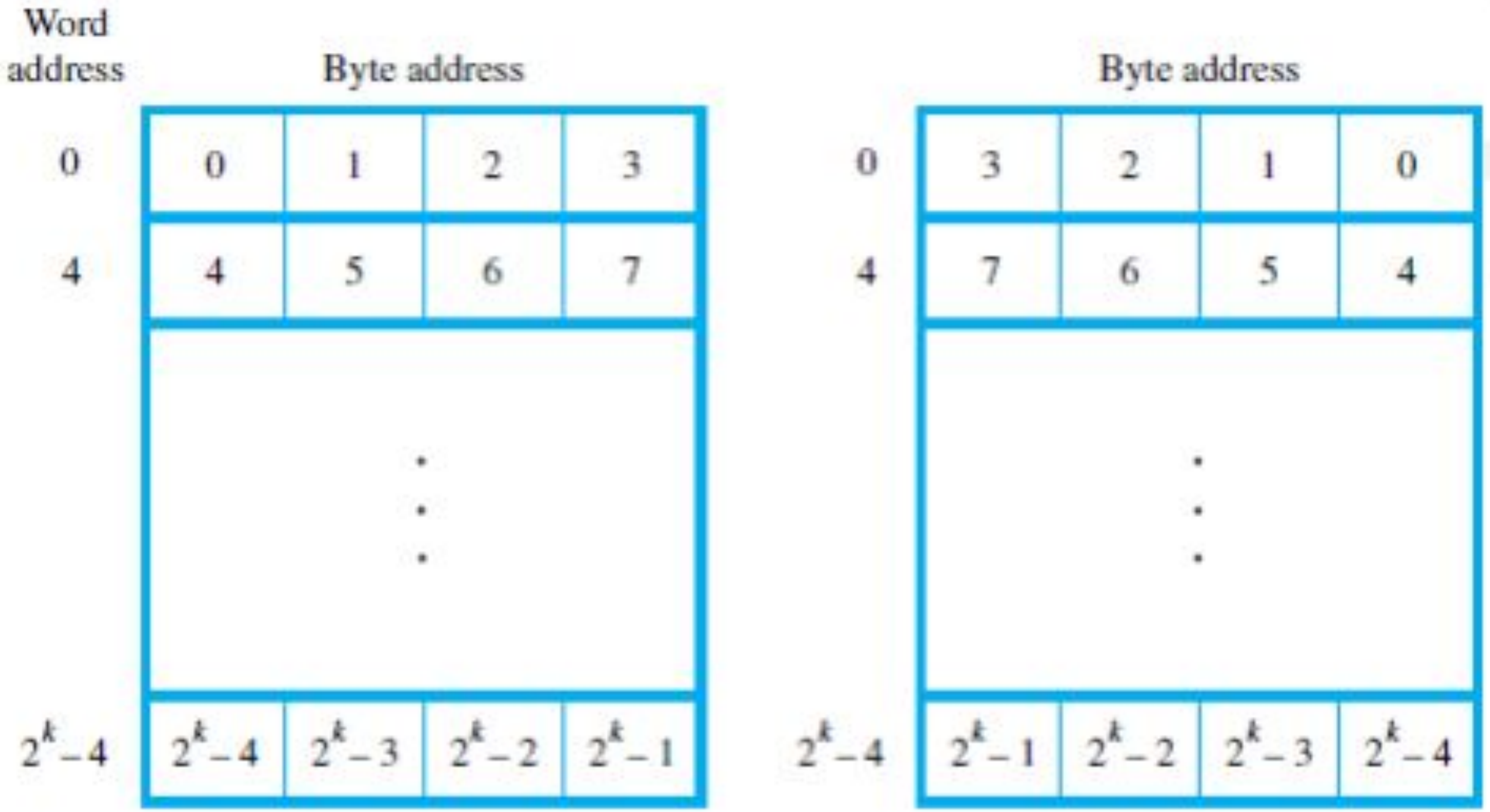
Byte Addressability



- The are 3 basic information quantities ,
  - bit
  - byte and
  - word.
- The word length typically ranges from 16 to 64 bits.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory.
- The term byte addressable memory is used for this assignment.
- Byte locations have addresses 0,1,2,3,…….
- If the word length of machine is 32 bits , successive words are located at addresses 0,4,8,…., with each word consisting of four bytes.

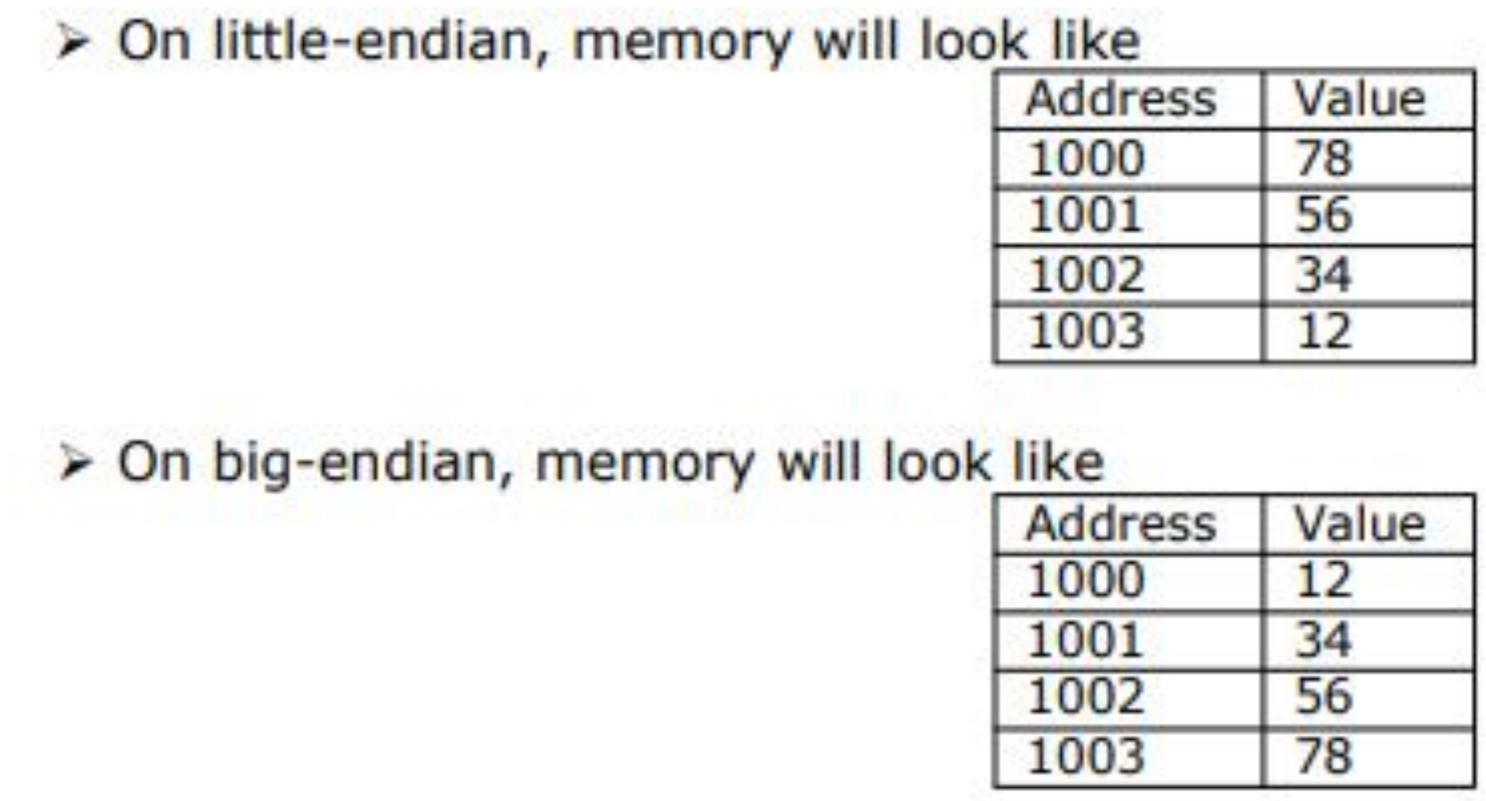
Big- Endian and Little – Endian Assignments

- There are 2 ways that byte addresses can be assigned across words.



(a) Big-endian assignment (b) Little-endian assignment

- Consider a 32-bit integer (in hex): 0x12345678 which consists of 4 bytes: 12, 34, 56, and78.
- 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 of the 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 its word 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 last character in the string.

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

Memory Operations

- 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 the processor.

 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

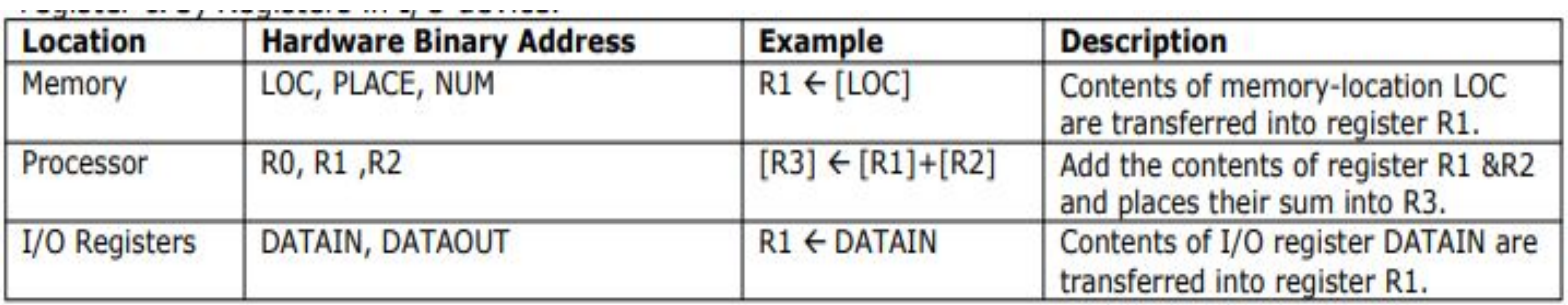
- A computer must have instructions capable of performing 4 types of operations:

1) Data transfers between the memory and the registers
2) Arithmetic and logic operations on data
3) Program sequencing and control
4) I/0 transfers.

Register Transfer Notation

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

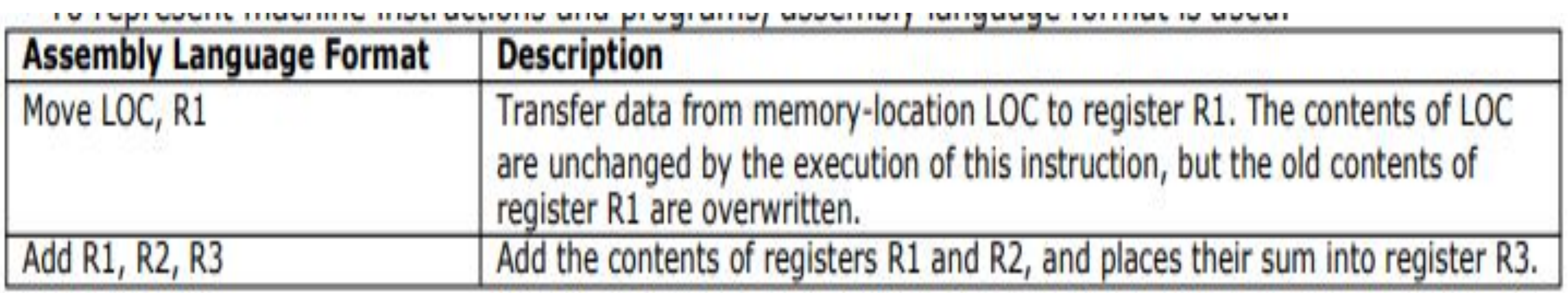
1) Memory locations
2) Processor registers &
3) Registers in I/O device.



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

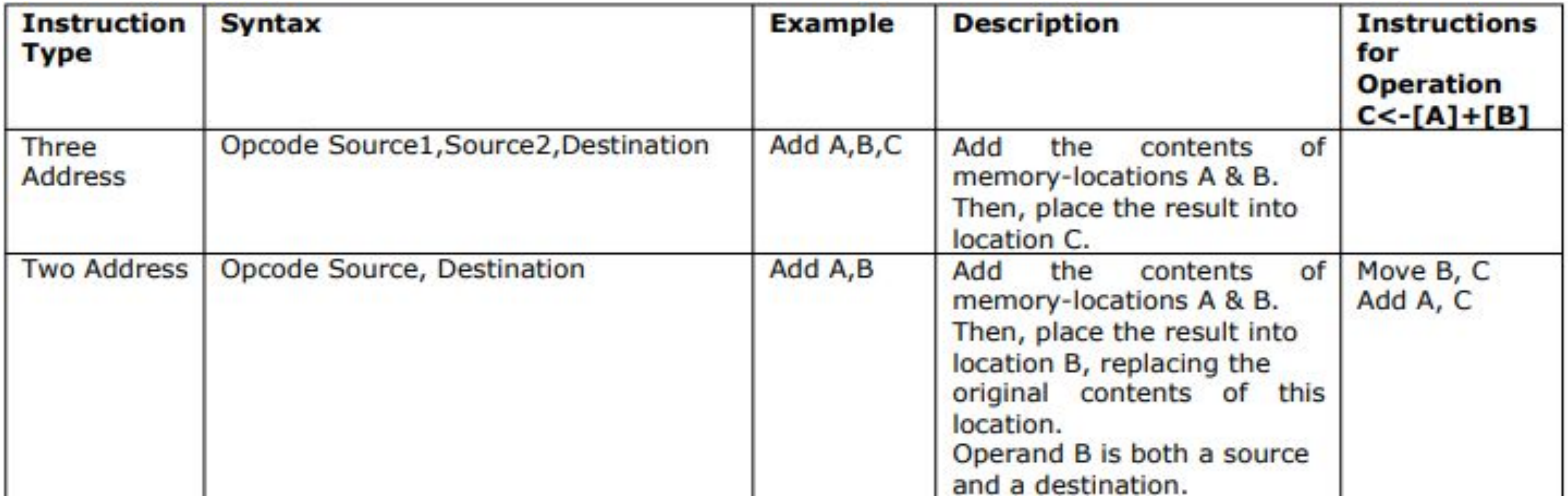
- To represent machine instructions and programs, assembly language format is used.



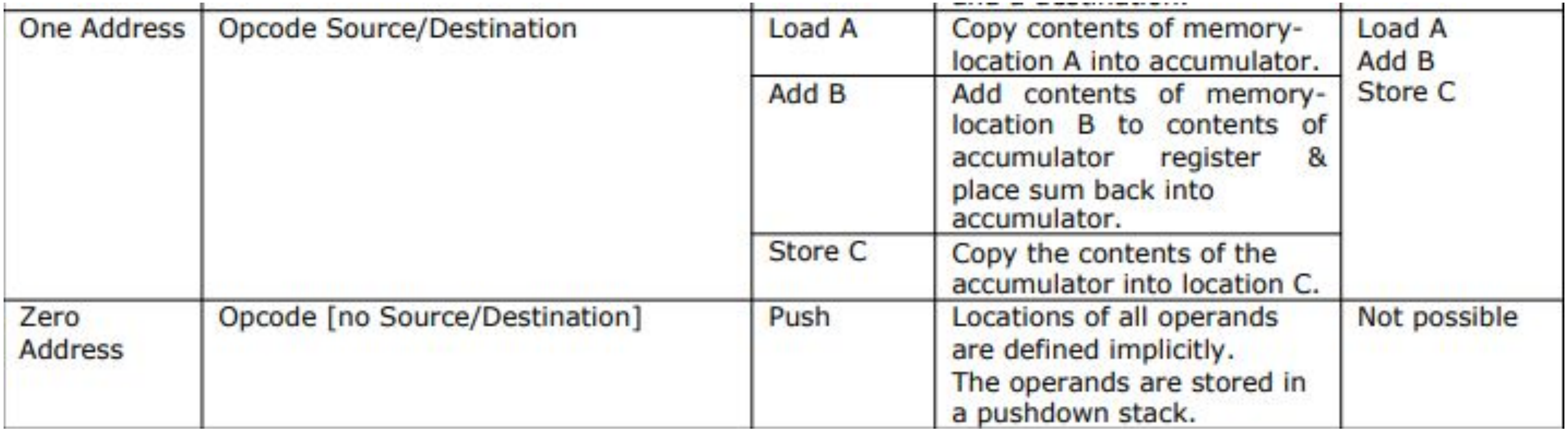
To represent machine instructions and programs, assembly language format is used.

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

- Access to data in the registers is much faster than to data stored in memory-locations.
- Let Ri represent a general-purpose register.

The instructions:

Load A,Ri

Add A,Ri

Store Ri,A

are generalizations of the Load, Store and Add Instructions for the single-accumulator case, in which register Ri performs the function of the accumulator.

Example :

- In processors, where arithmetic operations as allowed only on operands that are in registers, the task

C<-[A]+[B]

can be performed by the instruction sequence:

```
Move A,Ri
Move B,Rj,
Add Ri,Rj
Move Rj,C
```

Instruction execution and straight-line sequencing

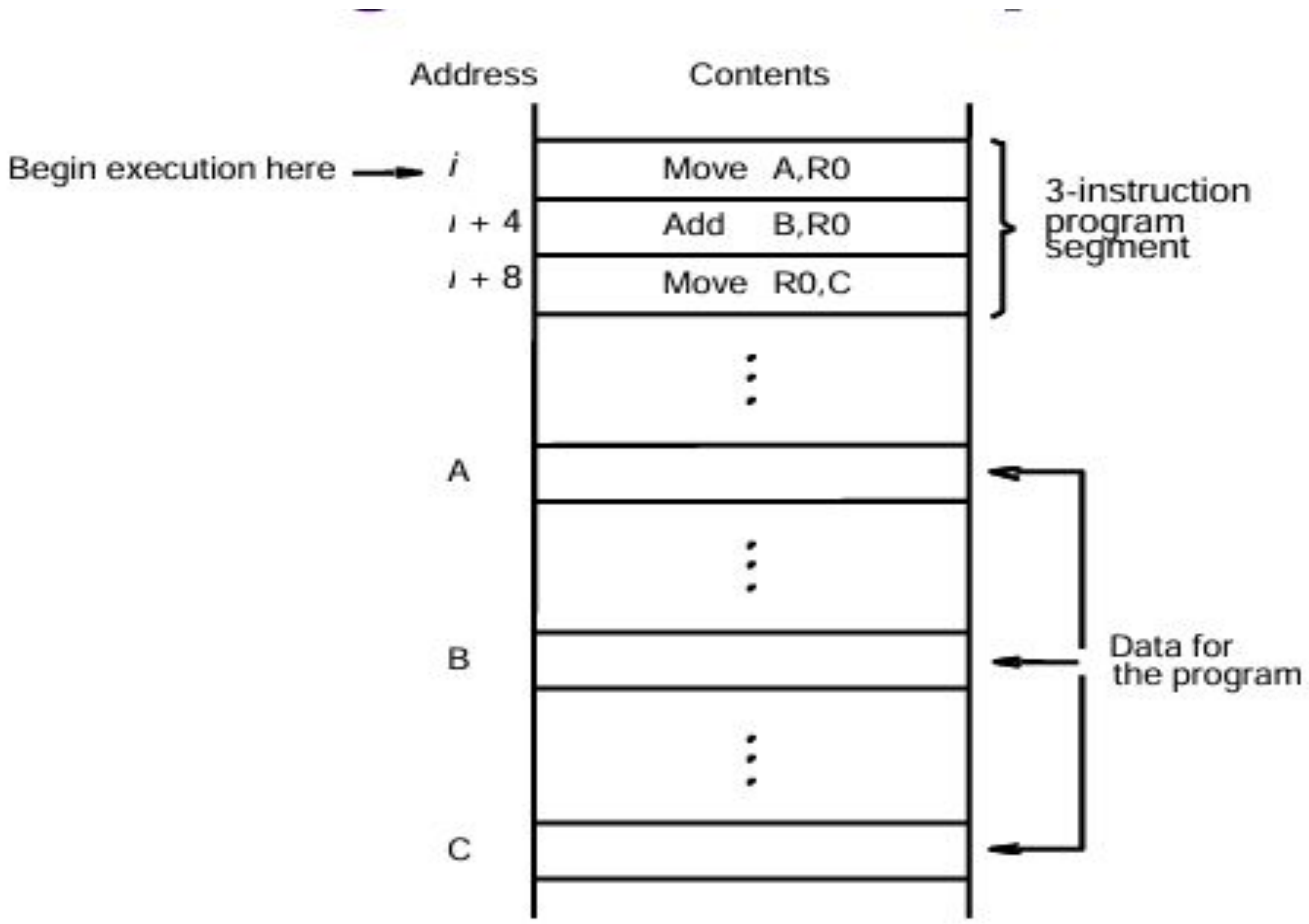


Figure 2.8. A program for C ← [A] + [B].

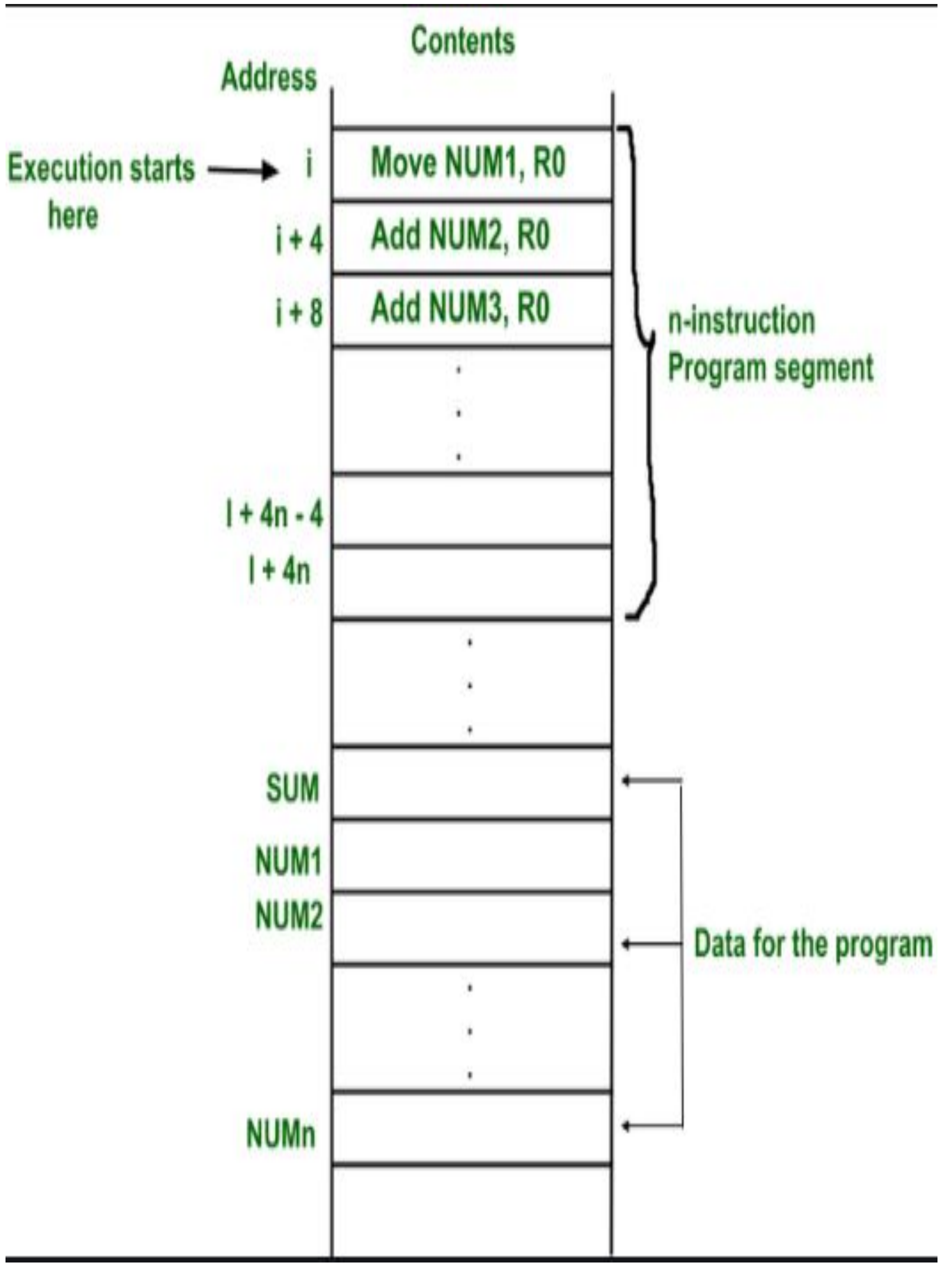
How the program is executed

- The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next.
- To begin executing a program, the address of its first instruction, placed into the PC.
- 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.
- During the execution of each instruction , the PC is incremented by 4 to point to the next instruction.
- Thus , after the Move instruction at location i + 8 is executed, the PC contains the value i + 12
- Executing a given instruction in 2 phase

1. Instruction fetch
2. Instruction execute

A straight-line Program

- Consider the task of adding a list of n numbers.
- The addresses of memory locations containing the n numbers are symbolically given as NUM1, NUM2.,………NUMn, and
- 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.
- Instead of using a long list of Add instruction , it is possible to place a single Add instruction in a program loop.



Branching

- The loop is a straight- line sequence of instructions executed as many times as needed.
- It starts at location LOOP and ends at the instruction Branch > 0.
- During each pass through this loop , the address of the next list entry is determined , and the entry is fetched and added to R0.

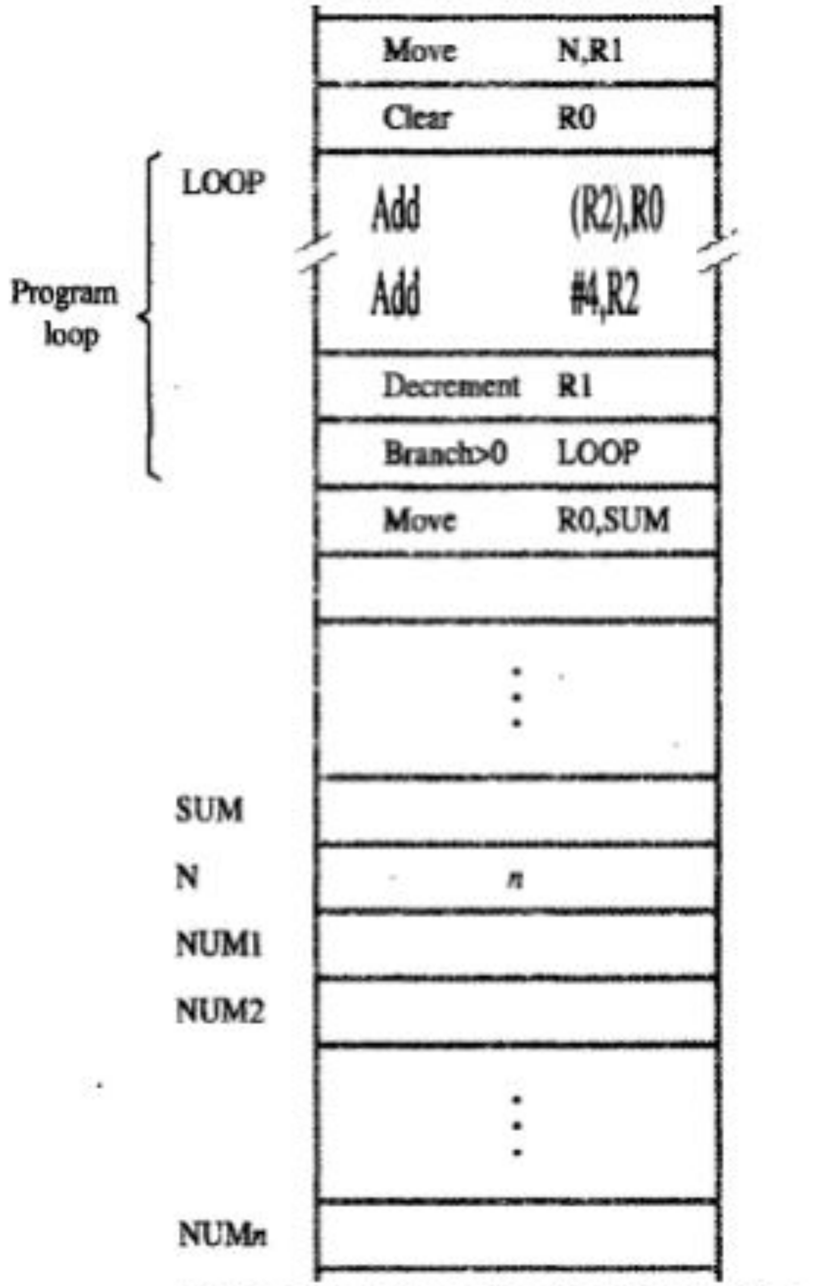


Figure 2.10 Using a loop to add *n* numbers.

Addressing Modes

- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.
- Variables and constants are the simplest data types and are found in almost every computer program.
- In assembly language , a variables is represented by allocating a register or a memory location to hold its value.
- Thus, the value can be changed as needed using appropriate instructions.
- Constants are always represented by #.

Table 2.1
Generic addressing modes

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

Register mode

- 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 ;

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, R

Immediate Mode

- The operand is given explicitly in the instruction.
- For example, the instruction

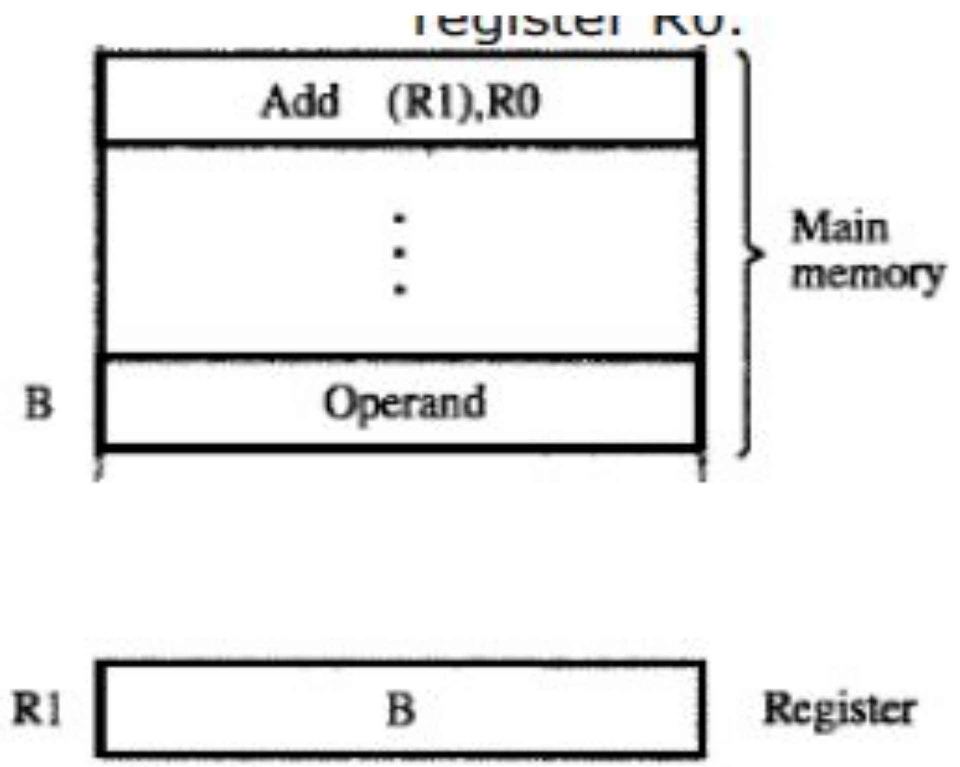
  Move #200, R0 ;Place the value 200 in register R0.
- Clearly, the immediate mode is only used to specify the value of a source-operand.

  A=B+6

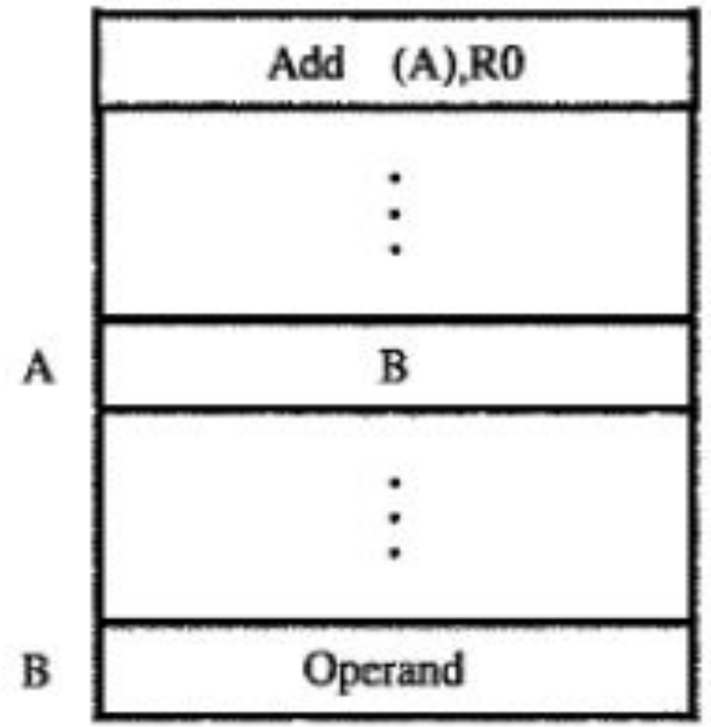
Indirect Addressing Modes

- The address field in the instruction contains the **memory location or register** where the **effective address** of the operand is present
- The **effective address** is a term that describes the address of an operand that is stored in memory
- For example, in the instruction “**MOV R1, (R2)”,** the memory location specified by the contents of the R2 register is accessed to obtain the actual address of the operand

Indirect Addressing Modes

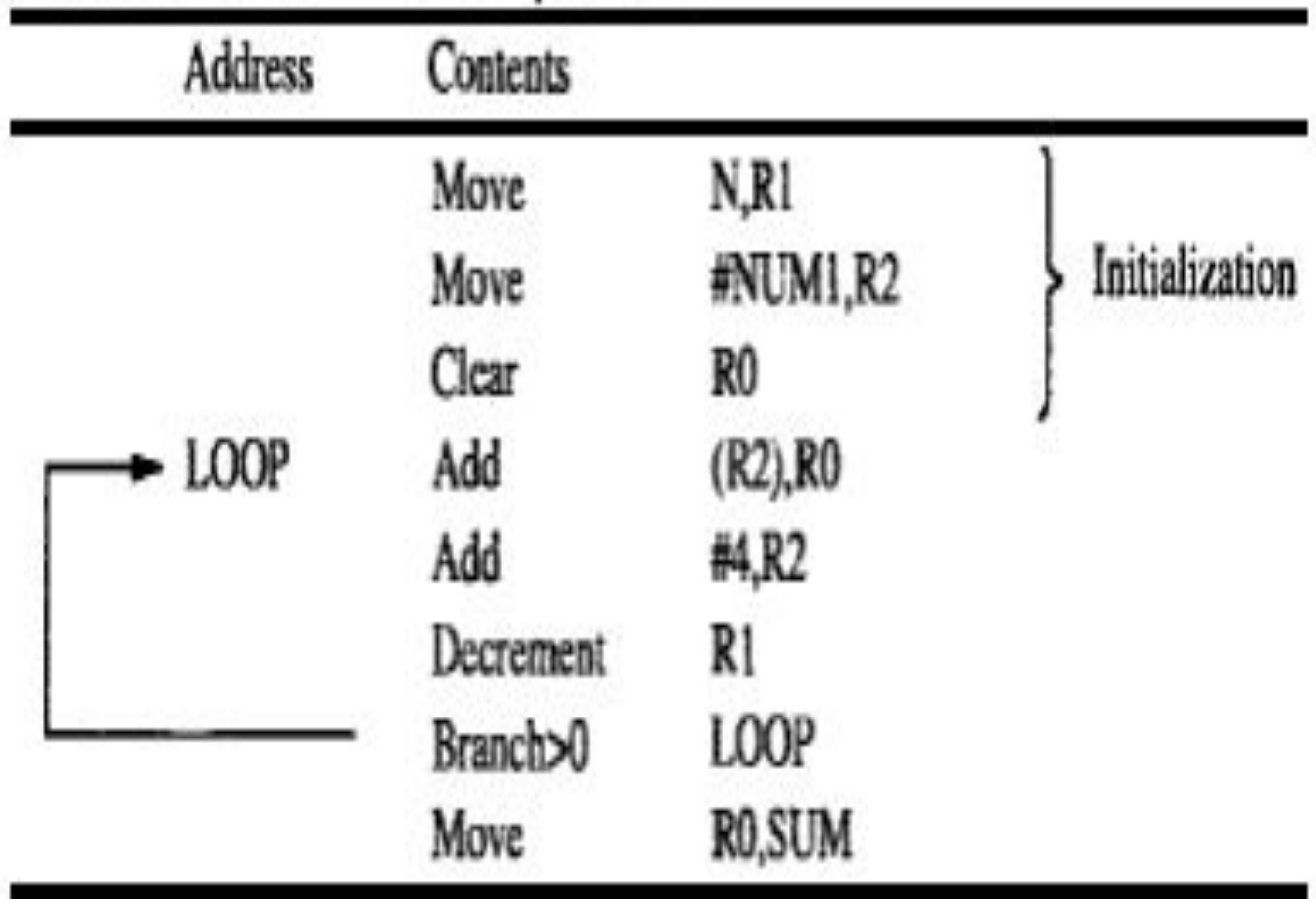


(a) Through a general-purpose register



(b) Through a memory location

Figure 2.11 Indirect oddressing.



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	

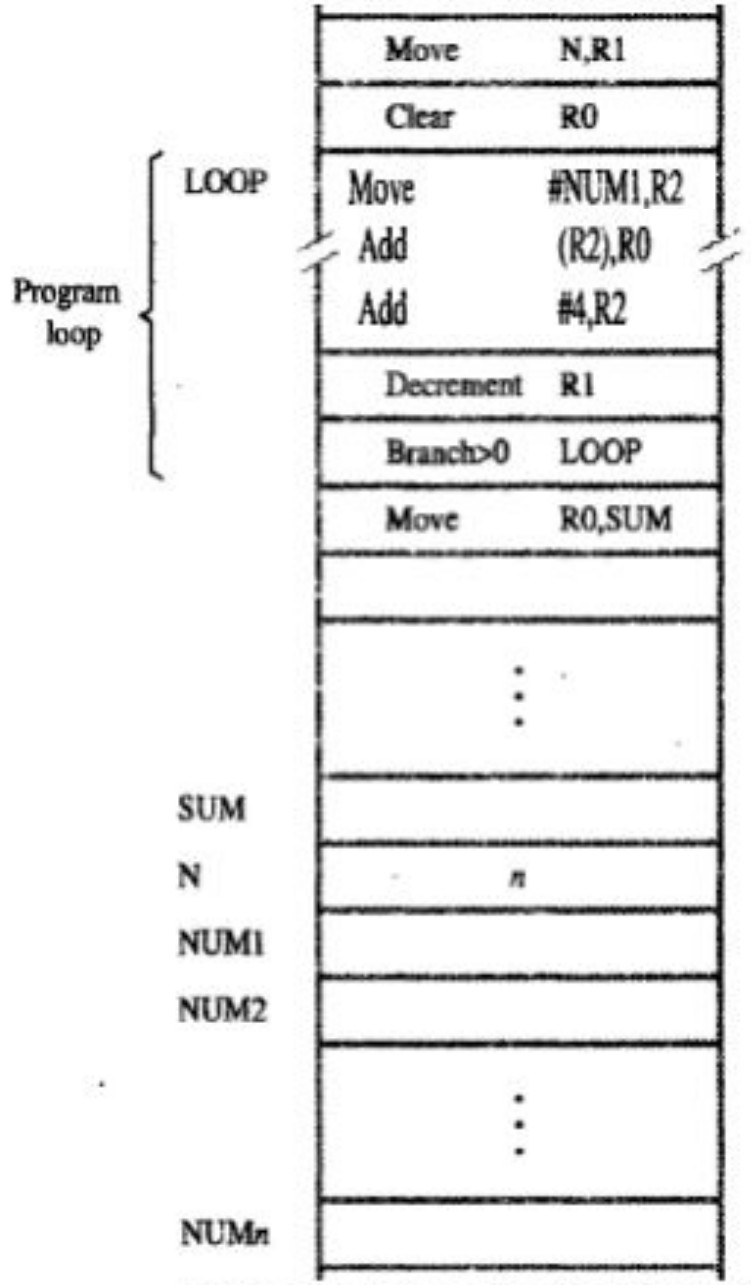


Figure 2.10 Using a loop to add *n* numbers.

Indirect Addressing Modes

- The register or memory location that contains the address of an operand is called a **pointer**
- Consider A = *B

```
        MOV B, R1
        MOV (R1), A
Through memory
        MOV (B), A
```

Indexed Addressing mode

- Indexed addressing mode is used to access elements in arrays, tables, or sequential data which are stored in memory at consecutive locations.
- The effective address of the operand is generated by adding a constant value to the content of a register
- This addressing mode will allow the program to access a location by incrementing or decrementing the index value

Continue…

- The register used may be special register provided or any one set of general – purpose register in the processor which is considered as **indexed register**
- Representation of **Indexed AM**

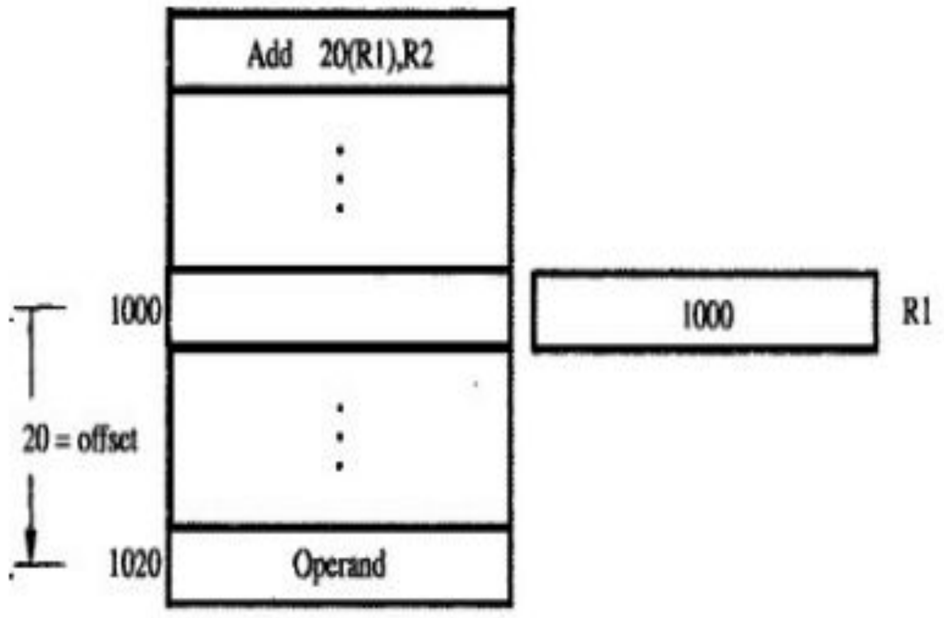
  X(Ri)

  X is the constant value (signed integer of fewer bits) which is considered as offset or displacement

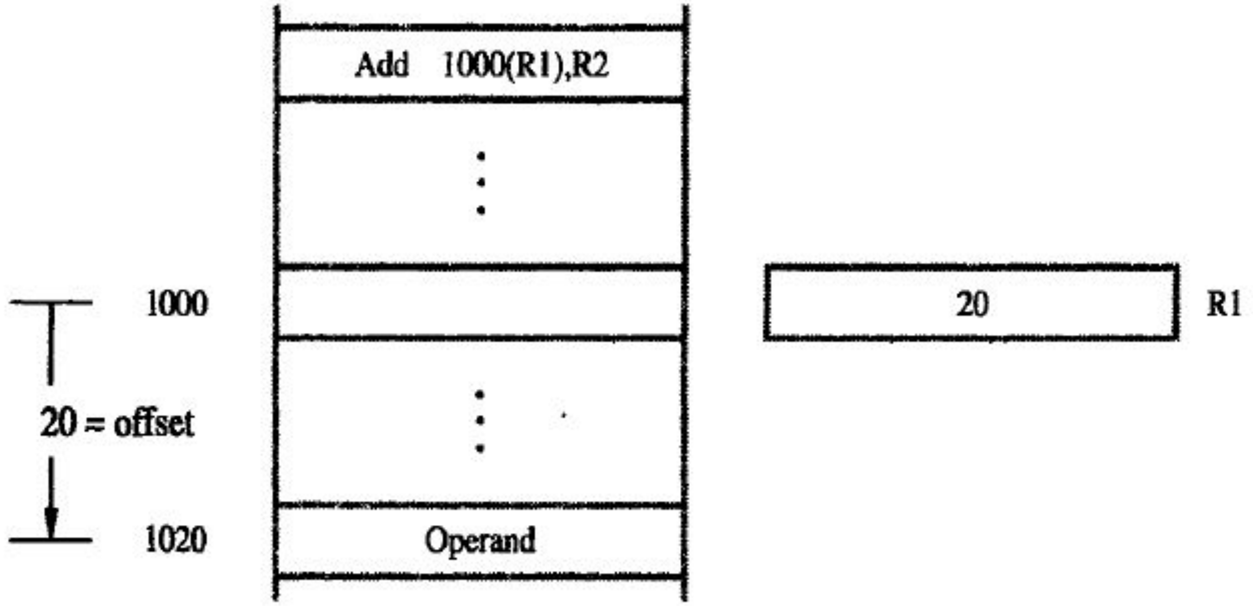
  Ri is the register
- The effective address is

  EA= X + [Ri]

Indexed Addressing mode



(a) Offset is given as a constant



(b) Offset is in the index register

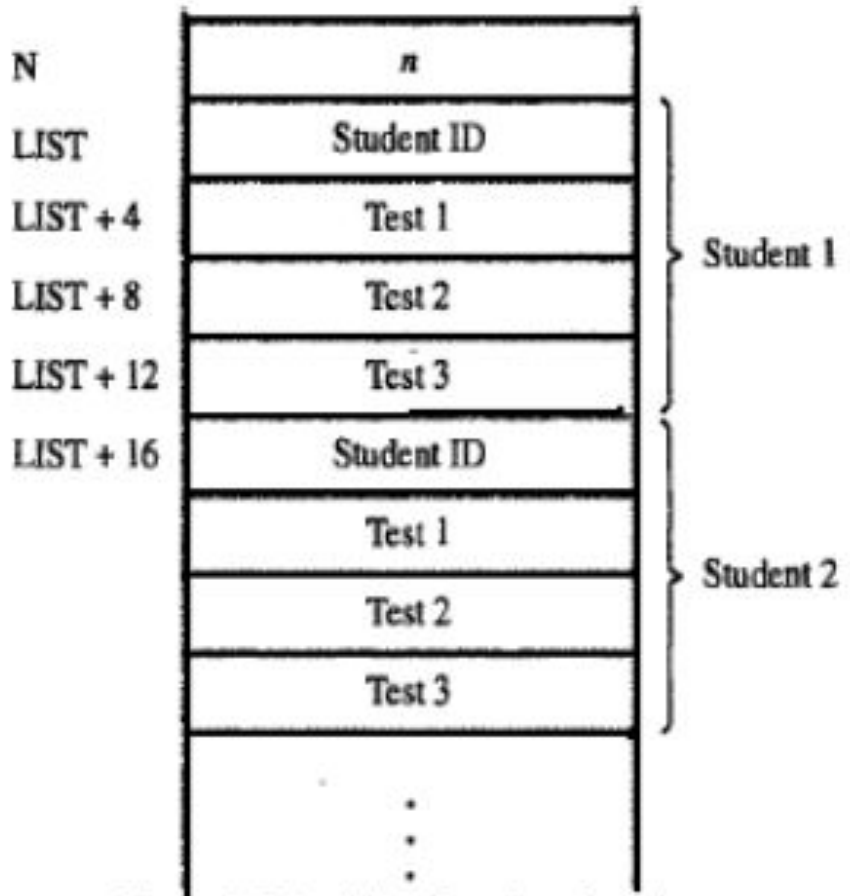


Figure 2.14 A list of students' marks.

```
        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.

Base with Index Mode

- Based with indexed mode is a combination of two addressing modes —base addressing mode and indexed addressing mode—used in assembly language programming.
- This addressing mode is particularly useful for accessing complex data structures like 2 - dimensional arrays that depend on both a base address and an index.

Base with Index Mode

- Components of Based with Indexed Mode:
    - **Base register**: Holds the starting (or base) address of a memory location.
    - **Index register**: Holds an offset that is added to the base address.
- **Formula for Effective Address:**

Effective Address=Base + Index

Base with Index Mode

- Example :

```
MOV R1, (R2 + R3)
```

R2: The base register, holding the base address (e.g., the starting address of an array).

R3: The index register, holding the offset (e.g., which element in the array to access).

Base with Index and Offset Mode

- Based with indexed mode and offset is a combination of two addressing modes —base addressing mode and indexed addressing mode and offset value (Constant value)—used in assembly language programming.
- This addressing mode is particularly useful for accessing complex data structures like 3 - dimensional arrays that depend on both a base address, an index and offset.

Base with Index and Offset Mode

- Components of Based with Indexed Mode:
  - **Base register**: Holds the starting (or base) address of a memory location.
  - **Index register**: Holds an offset that is added to the base address.
  - **Optional constant displacement**: A fixed value that can be added to the sum of the base and index registers.
- **Formula for Effective Address:**

Effective Address=Base + Index + Displacement

Base with Index and Offset Mode

- Example :

```
MOV R1, [R2 + R3 + 4]
```

R2: The base register, holding the base address (e.g., the starting address of an array).

R3: The index register, holding the offset (e.g., which element in the array to access).

4: A constant displacement, added to the calculated address.

RELATIVE MODE

- This is similar to index-mode with one difference:
- The effective-address is determined using the Program Counter (PC) in place of the general purpose register Ri.
- Since the addressed-location is identified "relative" to the PC, the name Relative mode
- This mode is used commonly in conditional branch instructions.

RELATIVE MODE

- **Key Concepts of Relative Addressing Mode:**
- Program Counter (PC): A register that holds the address of the next instruction to be executed.
- Offset/Displacement: A value, either positive or negative, that is added to (or subtracted from) the current program counter to compute the target address.

Effective Address=PC + Offset

RELATIVE MODE

- When a **branch** or **jump** instruction is executed, the relative addressing mode calculates the target address by adding a constant displacement to the value of the program counter (which points to the next instruction).
- Example in Assembly:
- Assume you are at **memory location 1000h**, and the instruction at this location is:

```
JMP 100h
```

This could represent a relative jump where the 100h is added to the current PC, so the new address to jump to would be 1100h (1000h + 100h).

ADDITIONAL ADDRESSING MODES

1) Auto Increment Mode

- Effective-address of operand is 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.
- Implicitly, the increment amount is 1.
- This mode is denoted as

    (Ri)+ ;

    where Ri=pointer-register

Example : Add (R1)+, R2

```
           Move        N,R1          }
           Move        #NUM1,R2      }  Initialization
           Clear       R0            }
LOOP       Add         (R2)+,R0
           Decrement   R1
           Branch>0    LOOP
           Move        R0.SUM
```

2) 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

Example : Add -(R1), R2

Exercise Examples

1. Registers R1 and R2 of a computer contain 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

a) Load 20(R1),R5

This instruction means load the content from memory at the address R1+20.

Effective address:

R1=1200

20(R1)=1200+20=1220

b) Move #3000,R5

This is an immediate mode instruction, meaning the value 3000 is directly moved to register R5.

Effective address: None, as it's not a memory reference instruction. The operand is immediate.

c) Store R5, 30(R1,R2)

This instruction uses indexed addressing, where the effective address is calculated by adding the values in R1, R2, and the offset (30).

Effective address:

R1=1200 , R2=4600

30(R1,R2)=1200+4600+30=5830

- **d) Add -(R2),R5**

  This uses register indirect with pre-decrement. The content of R2 is decremented first, then used as the effective address for the operation.

  Effective address:

  R2=4600−1=4599

- **e) Subtract (R1)+,R5**

  This is register indirect with post-increment. The effective address is the current value of R1, but after accessing it, R1 is incremented by 1.

  Effective address:

  R1=1200 (before increment).