UNIT-1

Chapter 1:Basic Structure Of Computers

Computer:

- It is fast electronic device that accepts digitized input information, processes it according to a list of internally stored instructions and produces the resulting output information.

Computer program:

The list of instructions that performs the task is called a computer program.

Types of Computers

Computers are classified based on size, cost and computational power.

- Microcomputers
- Mainframes
- Minicomputers
- Servers
- Super computers

Microcomputer

- The most common computer is microcomputer.

e.g. Desktop computer.

- Microcomputers are designed to be used by individuals, whether in the form of PCs, workstations, or notebook computers.
- A Microcomputer contains a CPU on a microchip (the microprocessor).
- Desktop computers have processing and storage units, display and audio output units, and keyboard that can all be located easily on a desk.
- The storage media includes hard-disks, CD-ROMs.

Laptop Computer

- Portable notebook computers are a compact version of the personal computer..
- All components (keyboard, mouse, etc.) are in one compact unit.
- Usually more expensive than a comparable desktop.
- Sometimes called a Notebook.

Workstation

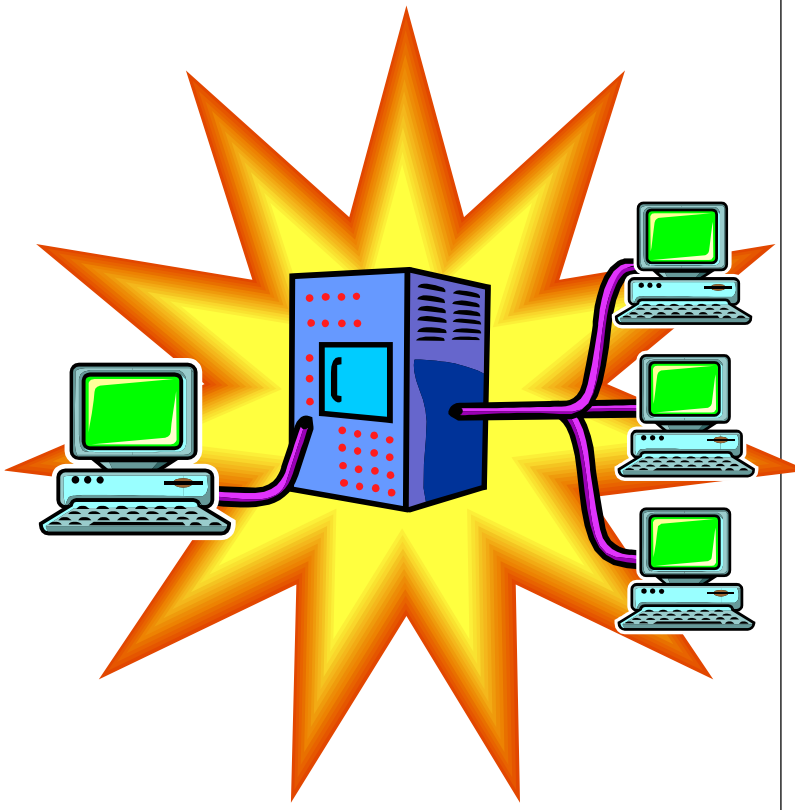
- Powerful desktop computer designed for specialized tasks.
- It has more computational power than personal computers.
- Workstations are often used in engineering applications.
- Workstations generally come with a large high-resolution graphics screen.
- Most workstations also have mass storage devices such as disk drive, but a special type of workstation, called diskless workstation, comes without a disk drive.
- Like personal computers, most workstations are **single-user computers.**

Mainframe

- A very large and expensive computer capable of supporting hundreds or even thousands of users simultaneously.
- Mainframes are mainly Used to store, manage, and process large amounts of data.
- Mainframes are used in medium to large corporations that require more computing power and storage capacity than workstations can provide.

Server

- Purpose is to "serve."
- A computer that has the purpose of supplying its users with data; usually through the use of a LAN (local area network).



Supercomputer

- A Super computers is typically used for scientific and engineering applications that **must handle very large databases** or do a great amount of computation.
- It is used for large- Scale numerical calculations required in applications such as weather forecasting and aircraft design.
- super computers are fastest, costliest and most powerful computers.
- It can execute million instructions per second.
- Most super computers are really multiple computers that perform parallel processing.

Functional Units Of Computer

A computer consists of five functionally independent main parts.

- Input unit
- Output unit
- Memory unit
- Arithmetic and logic unit
- Control unit

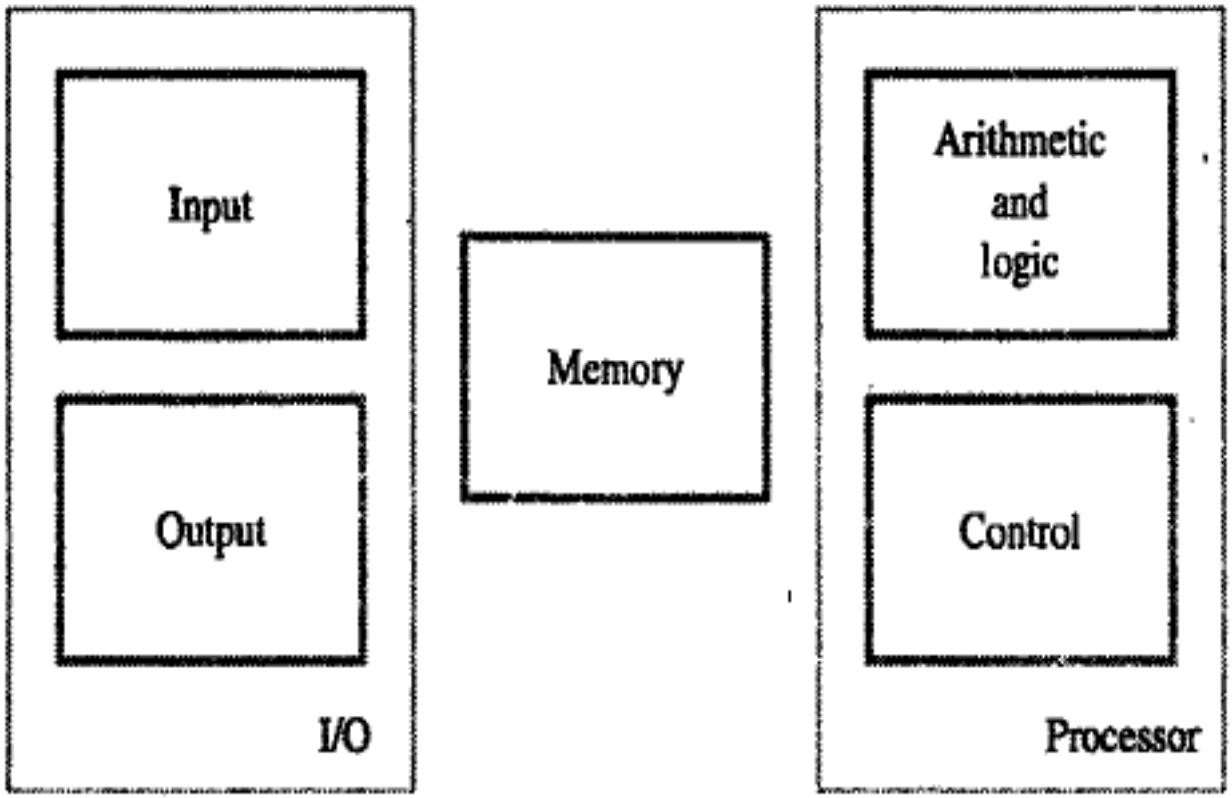


Figure 1.1 Basic functional units of a computer.

Input Devices

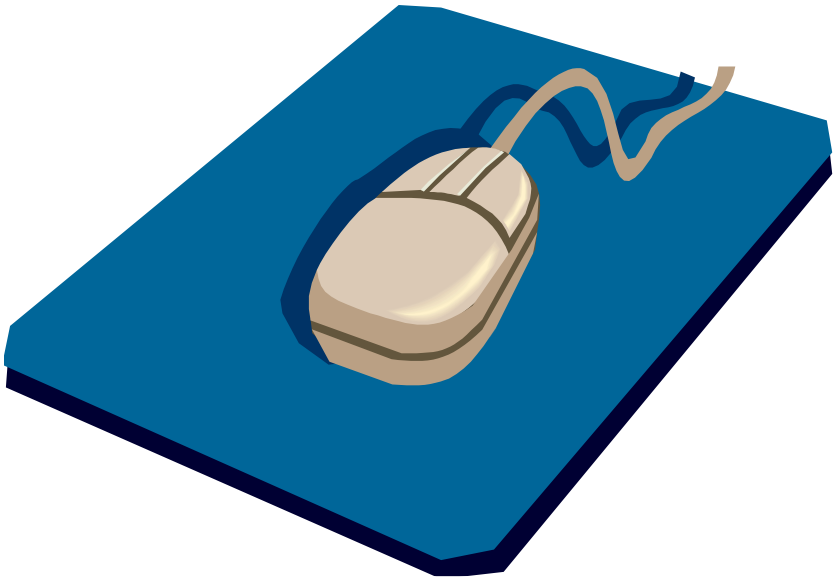
- The input unit accepts coded information from users
- The information received is either stored in the computer's memory or immediately executed by the ALU.
- The most well-known input device is the key-board.
- Whenever key is pressed, the corresponding letter or digit is **automatically translated into its corresponding binary code** and transmitted over a cable to either the memory or the processor.

Keyboard

- An arrangement of letters, numbers, and special function keys that act as the primary input device to the computer.

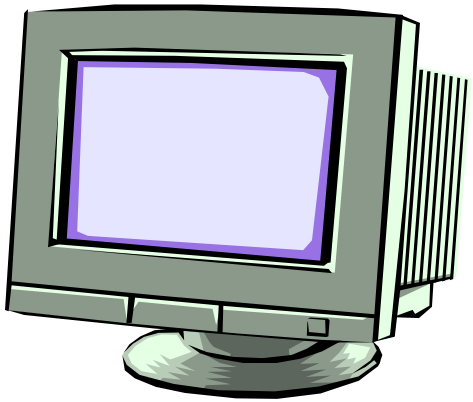
Mouse

- An input device that allows the user to manipulate objects on the screen by moving the device along the surface of a desk.

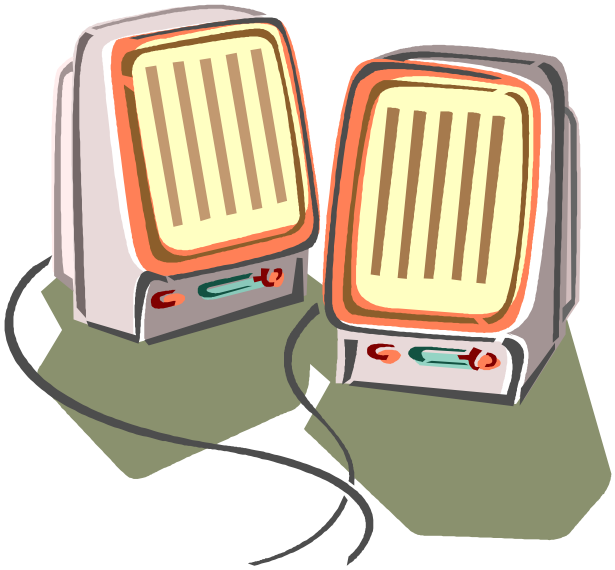


Output Devices

- The output unit is the counterpart of the input unit.
- Its function is to send processed results to the outside world.



Speakers



- Output devices that receive signals from the computer's sound card to play music, narration, or sound effects.

Printer

- Output device that produces text or graphical images on paper.

Storage Devices

- Used to keep data when the power to the computer is turned off.
- Different forms
  - Hard disk
  - Floppy or zip disks
  - CD-Writer

Memory Unit

- The function of memory unit is to store **the programs** and **data.**
- There are two classes of storage

  1) Primary Storage

  2) Secondary storage

Primary Storage:

- It is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed.

- The memory contains a large number of semiconductor cells, each capable of storing in one bit of information.
- Because a single bit represents a very small amount of information, bits are not handled individually.
- The memory is organized so that the **contents of one word, containing 'n' bits** can be stored or retrieved in one basic operation.
- To provide easy access to any word in the memory, a **distinct address is associated in with each word**.
- A word address can be used to access the word.

Word Length:

- The number of bits in each word is called as the word length of the computer.
- Typical word lengths range from 16-64 bits.

Memory Access Time:

- The time required to access one word is called the memory access time.
- This time is fixed, independent of the location of the word being accessed.
- It typically ranges from a few nanoseconds (ns) to about 100 ns.
- The primary memory is also called as main memory.

- Primary memories are small in capacities and expensive.
- Thus additional, cheaper secondary storage is used when large amounts of data and many programs have to be stored.

Secondary storage devices:

- Magnetic disks
- Magnetic tapes
- CD-ROMs

Arithmetic And Logic Unit

- It performs the arithmetic operations such as addition, subtraction, multiplication, division and logic operations such as AND, OR, Not.
- Suppose two numbers located in the memory are to be added, **they are brought into the processor,** and addition is carried out by the ALU .
- The SUM may then be stored in the memory.
- When the operands are brought into the processor, they are stored in high-speed storage elements called **registers.**
- Each register can store one word of data.

Control Unit

- The memory unit, arithmetic and logic and input and output units store and process information and perform input and output operations.
- The operation of these units must be coordinated in some way.
- Control unit **coordinates and controls** the activities among the functional units.
- It sends the **control signals** to other units and senses their states.

The operation of a computer can be summarized as follows:

- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
- Information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities inside the machine are directed by control unit.

Basic Operational Concepts

- To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which execute the specified operation.

e.g. Consider the addition of two operands

Add LOCA, R0

this instruction adds the operand at memory location LOCA to the operand in a register in the processor,R0 and places the sum into register R0.

- **This instruction requires the performance of several steps:**

  1) the **instruction** is fetched from the memory into the processor.

  2) the **operand** at LOCA is fetched and added to the contents of R0.

  3) Finally, the sum is placed in register R0.

the same task can be performed using two instructions

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

the first instruction transfers the contents of memory location LOCA into the register R1.

Second instruction adds the contents of registers R1 and R0 and places the Sum into R0.

- Figure shows how the memory and the processor can be connected.
- Processor contains number of registers used for several purposes.

Instruction Register:

It holds the instruction that is currently being executed.

Program Counter:

It is another specialized register, which holds the address of next instruction to be fetched and executed.(it points to the next instruction).

General Purpose Registers:

These registers can be used for temporary storage of data. Processor has 'n' general purpose registers.

MAR & MDR:

These two registers provide communication with the memory.

Memory address Register: (MAR)

It holds the address of the location to be accessed.

Memory Data Register:

It contains the data to be written into or read out of the addressed location.

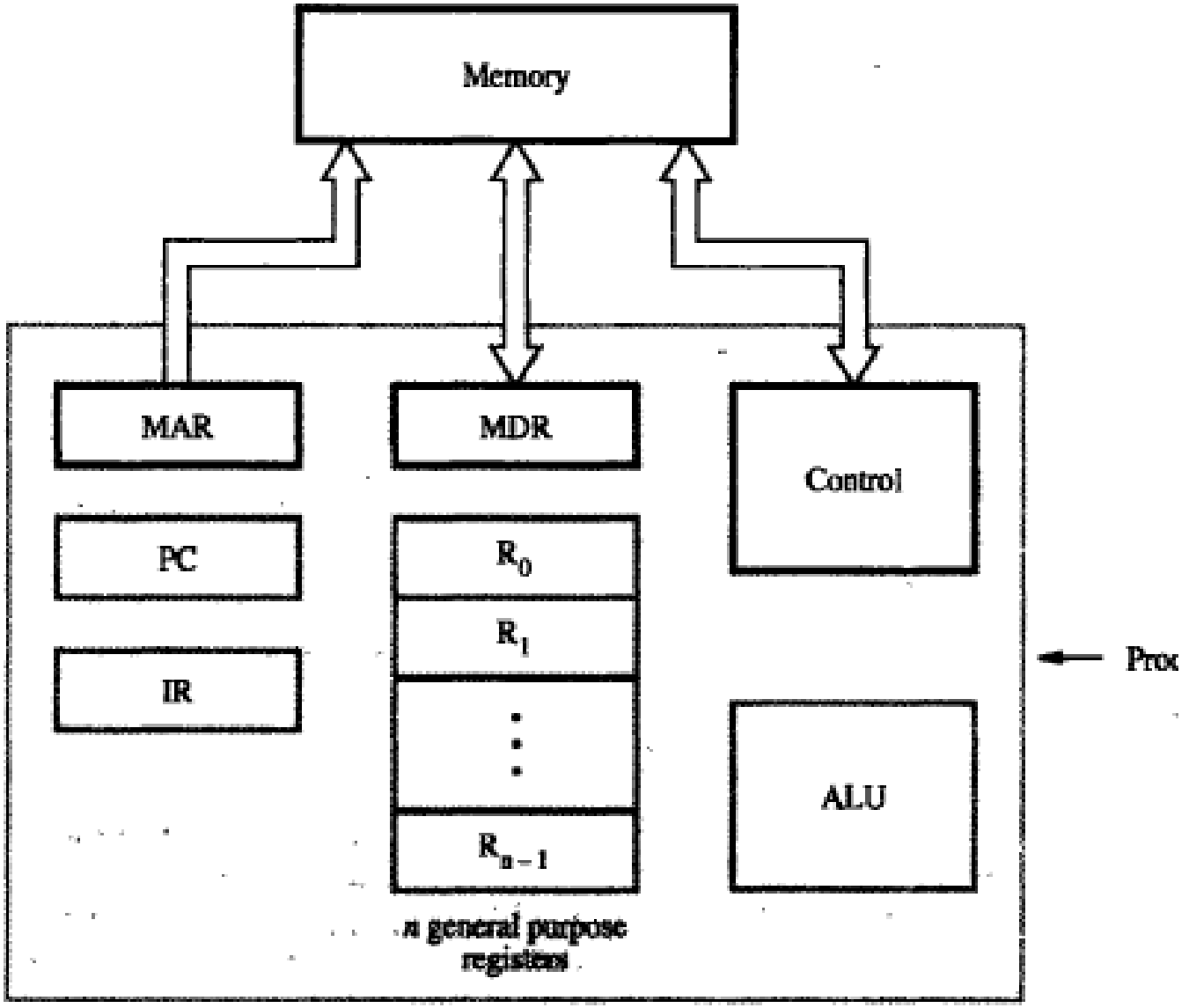


Figure 1.2 Connections between the processor and the memory.

Operating steps:

- Execution of program starts when the PC is set to point to the first instruction of the program.
- The contents of the PC are transferred to the MAR and a read control signal is sent to the memory.
- After some time, the addressed word is read out of the memory and loaded into the MDR.
- Next, the contents of the MDR are transferred to the IR.
- At this point, the instruction is ready to be executed.

- If the instruction involves an operation to be performed by the ALU, it is necessary to **obtain the required operands.**
- If operand resides in the memory it has to be fetched by **sending its address to the MAR** and initiates read cycle.
- When the operand has been read from the memory into the MDR, it is transferred from the MDR to the ALU. Then the **ALU can perform the desired operation**.
- If the **result of this operation** is to be stored in the memory, then the result is sent to the MDR.
- The address of the location where the **result is to be stored is sent to the MAR**, and write cycle is initiated.
- At some point during the execution of the current instruction , the **contents of the PC are incremented** so that PC points to the next instruction to be executed.

Bus Structures

- So far we have discussed the functions of individual parts of a computer.
- To form an operational system. These parts must be connected in some organized way.
- A group of lines that serves as a connecting path for several devices is called a BUS.
- Bus contains three sets of lines

  Data lines
  Address lines
  control lines.

- The simplest way to interconnect functional units is to use a single bus as shown in the figure below.
- All units are connected to this bus.
- Because bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.
- The advantage of the single bus-structure is its **low cost** and its **flexibility** for attaching peripheral devices.
- Multiple buses can be used to interconnect large number of devices.

- The devices connected to bus vary widely in their speed of operation, some devices **such as keyboards and printers are relatively slow.**
- **Memory and processor** operate at high speeds.
- To synchronize the speeds of these devices.
- A common approach is to include **buffer registers** with the devices to hold the information during transfers.

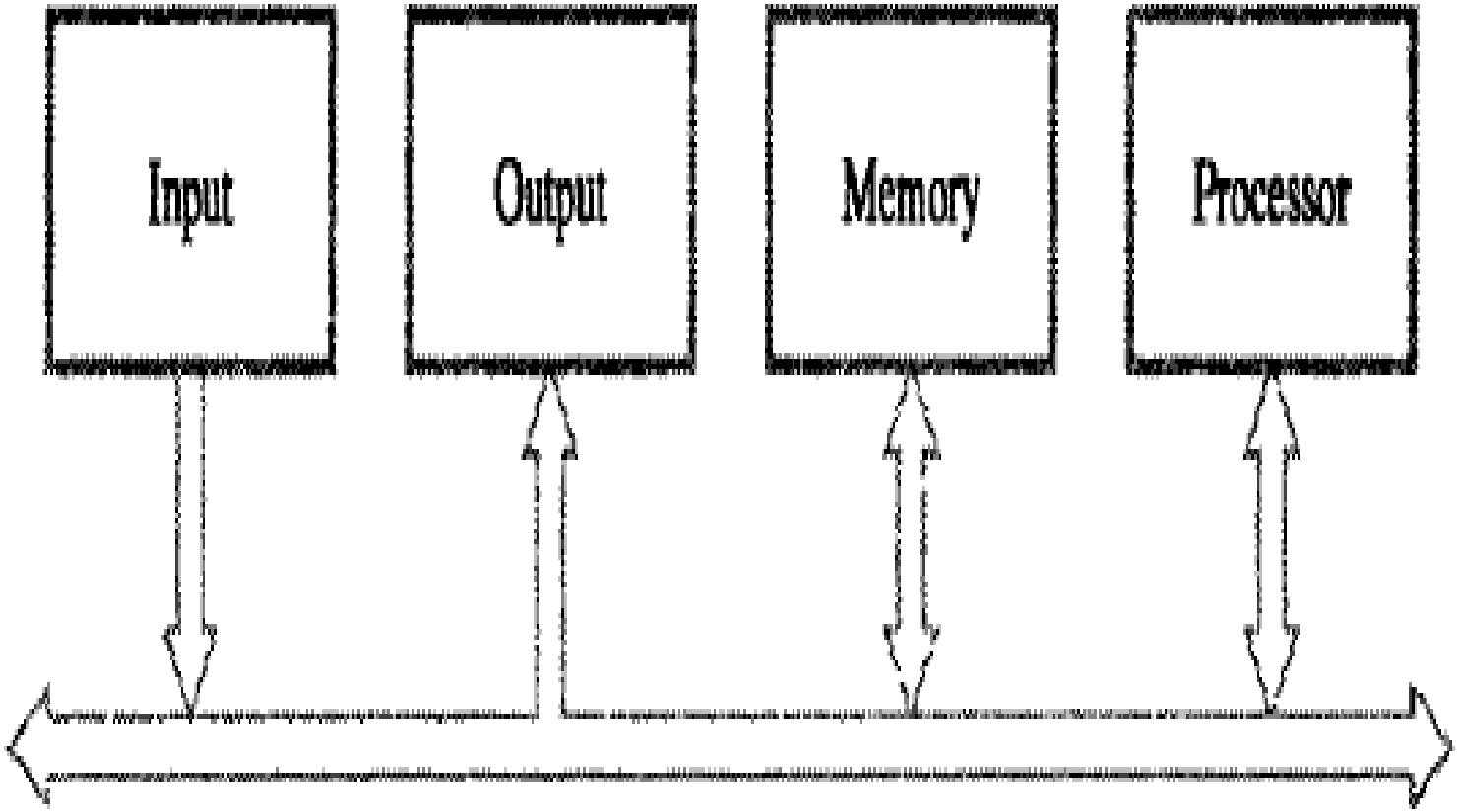


Figure 1.3 Single-bus structure.

Software

- In order for a user to enter and run the application program, the computer must already contain some ***system software*** in its memory.
- System software is a collection of programs that are executed as needed to perform functions such as:
  - Receiving and interpreting user commands
  - Entering and editing application programs and storing them as files in secondary storage devices
  - Managing the storage and retrieval of files in secondary storage devices
  - Running standard application programs such as word processors, spreadsheets, or games, with data supplied by the user
  - Controlling I/O units to receive input information and produce output results
  - Translating programs from source form prepared by the user into object form consisting of machine instructions
  - Linking and running user-written application programs with existing standard library routines, such as numerical computation packages

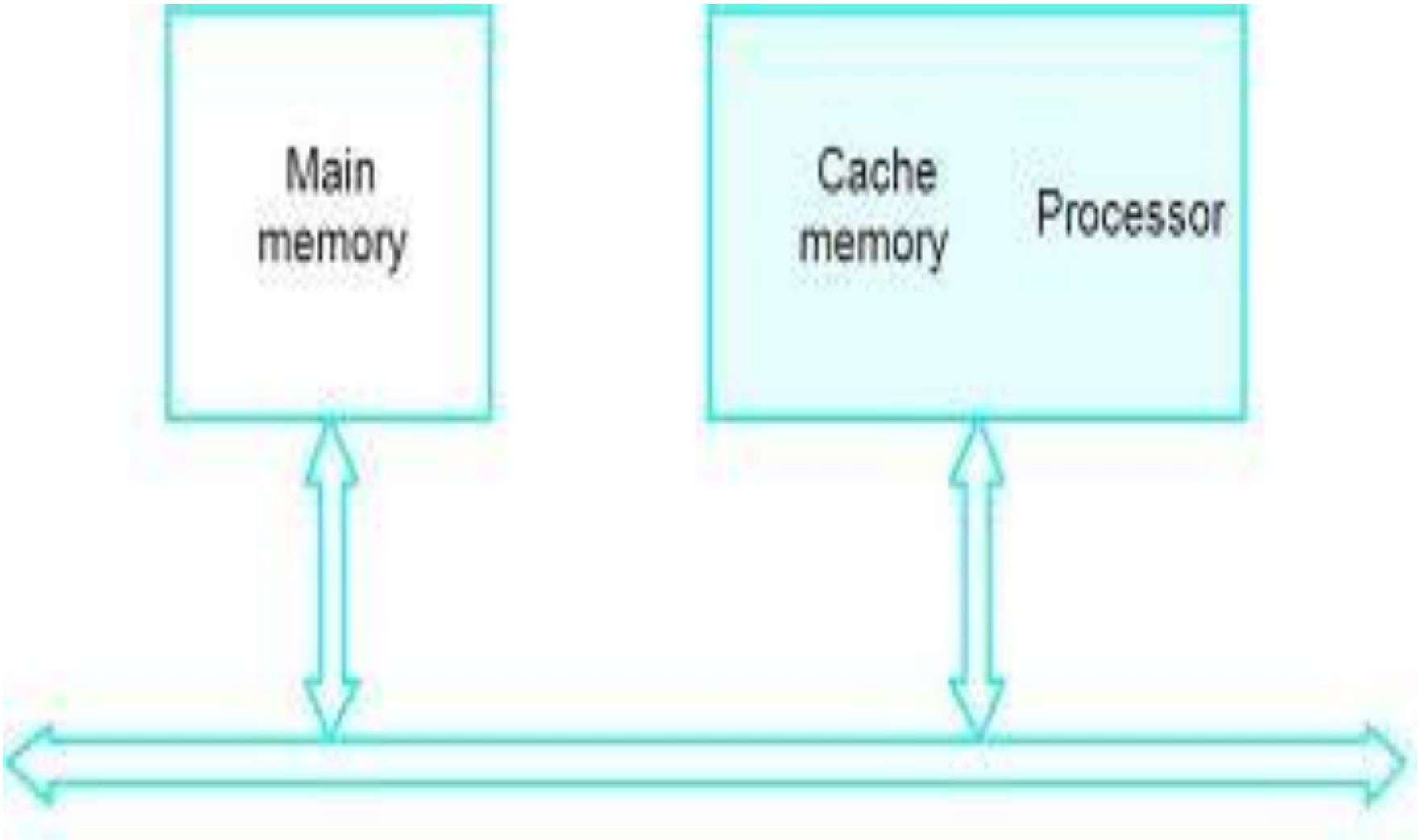
- System software is thus responsible for the coordination of all the activities in a computing system.
- The purpose of this section is to introduce some basic aspects of system software.
- Compliers, Text Editor, Operating Systems are some of the examples of system software.

General Reading: Computer Organization by Carl Hamacher 5th Edition Page 11

Performance of a computer

- The most important measure of a computer is how quickly it can execute the programs.
- The speed with which a computer executes the program is affected by the design of its hardware and its machine language instructions.
- Performance of a computer is also affected by the compiler.
- For the best performance, it is necessary to design the compiler, the machine instruction set, and the hardware in a coordinated way.

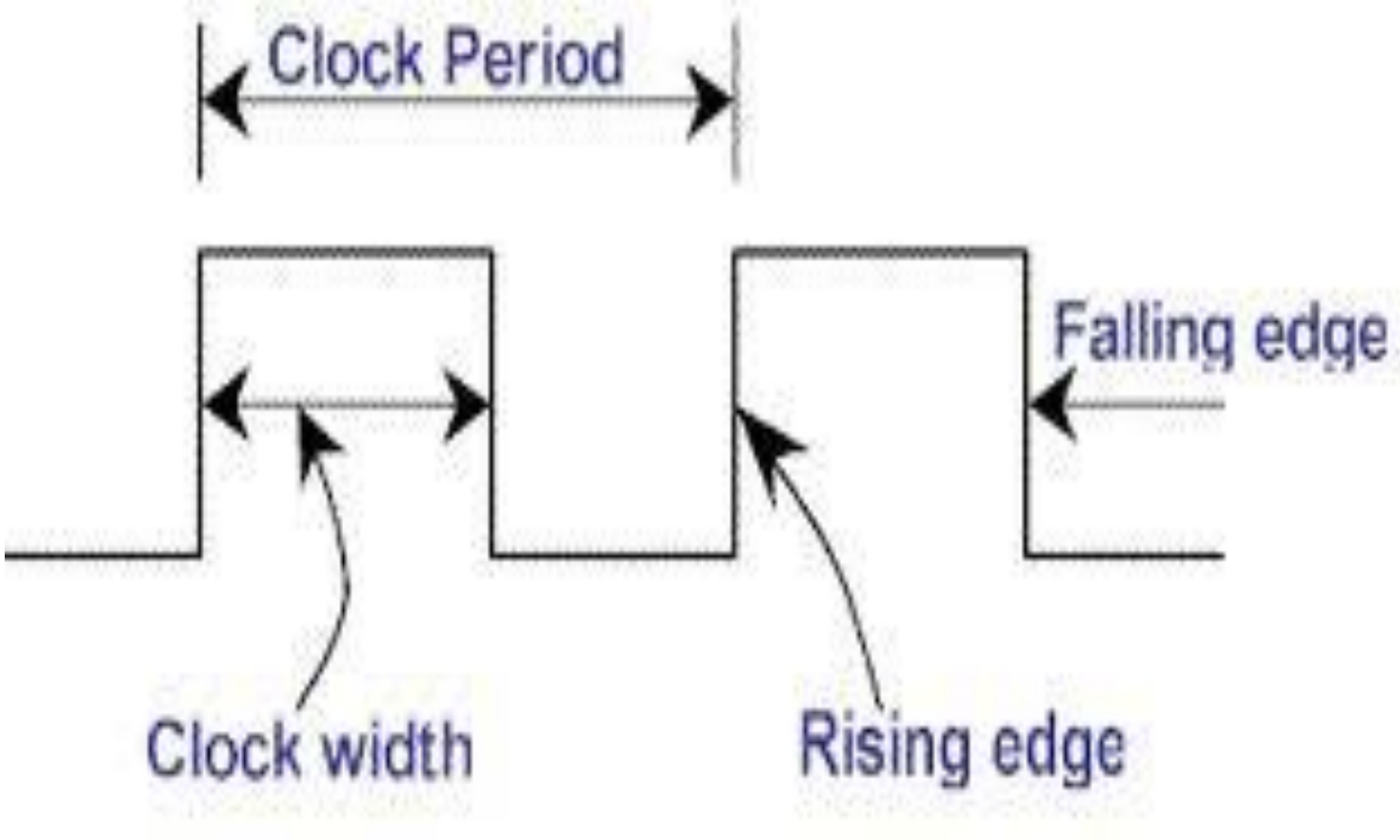
- The processor time depends on the **hardware involved** in the execution of individual machine instructions.
- This hardware comprises the processor and the memory, which are usually connected by bus.
- **Let us discuss the flow of program instructions and data between the memory and processor**:
- At start of execution, all program instructions and required data are stored in main memory.
- As execution proceeds, instructions are fetched one by one over the into the processor, and copy is placed in the cache.
- When execution of an instruction calls for data located in the main memory., the data are fetched and a copy is placed in the cache.
- If the **same instruction or data item** is needed a second time, it is read directly from **the cache**.


Main
memory
Cache
memory
Processor

- A program will be executed faster if the movement of instructions and data between the main memory is minimized.
- For example, suppose a number of instructions are **executed repeatedly over short period of time**, as happens in the program loop.
- If these instructions are available in the cache memory, **they can be fetched quickly** during the period of repeated use.

Processor clock

- Processor circuits are controlled by a timing signal called clock.
- The clock defines regular time intervals, called clock cycles.
- To execute a machine instruction, **the processor divides the action to be performed into sequence of basic steps**, such as each step to be completed in one clock cycle.
- The length '*p*' of one clock cycle is an important parameter that affects the processor performance.


Clock Period
Falling edge
Clock width
Rising edge

Clock rate:

- The number of clock cycles per second is called clock rate.
- It is denoted by '*R*'.

$$R=1/p$$

- It is measured in cycles per second.
- Cycles per second is called hertz (Hz).
- Ex: 500MHz, 1.2GHz

The term "million" is denoted by the prefix *Mega* (M), and "billion" is denoted by the prefix *Giga* (G). Hence, 500 million cycles per second is usually abbreviated to 500 Megahertz (MHz), and 1250 million cycles per second is abbreviated to 1.25 Gigahertz (GHz). The corresponding clock periods are 2 and 0.8 nanoseconds (ns), respectively.

Basic Performance Equation

$$T = (N*S)/R$$

- T→ processor **time** required to execute the program.
- N→ the **actual number of instruction executions**, and is not necessarily equal to the number of machine language instructions in the object program.
- Because some instructions may be executed more than once, which is the case for instructions inside a program loop.
- S→ number of **steps** needed to execute one machine instruction.
- R→ clock rate

- 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.
- Using higher- frequency clock increases the value of 'R'. which means that the time required to complete a basic execution step is reduced.

Possibilities for increasing the clock rate

- Improving the integrating –circuit (IC) technology makes logic circuits faster, which reduces the time needed to complete a basic step. this allows the clock period 'p' to be reduced and the clock rate 'R' to be increased.
- Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period.

Performance Measurement

- The previous discussion suggests that the only parameter that properly describes the performance of a computer is the execution time (T).
- Computing the value of 'T' is not simple.
- For this reason the computer community adopted the idea of measuring computer performance using benchmark programs.
- The performance measure is the time it takes a computer to execute given benchmark.

- An organization called system performance evaluation corporation(SPEC) selects and publishes programs for different application domains.

- SPEC (System Performance Evaluation Corporation) is an non-profitable organization, that measures performance of computer using SPEC rating. The organization publishes the application programs and also time taken to execute these programs in standard systems.

$$SPEC = \frac{Running\ time\ of\ reference\ Computer}{Running\ time\ of\ computer\ under\ test}$$

Pipelining and Superscalar Operation

- Fetching the next instruction when the current instruction is in execution. This concept is called **Pipelining.**
- A higher degree of concurrency can be achieved if **multiple instruction pipelines are implemented** in the processor.
- With such an arrangement, it becomes possible to start the execution of several instructions in every clock cycle. This mode of operation is called **superscalar execution**.

Instruction Set: CISC and RISC

Chapter 2: Machine Instructions and programs:

Number Representation

- Computers are built using logic circuits that operate on information represented by two valued electrical signals. We label the two values as 0 and 1.
- The most natural way to represent a number in a computer system is by **string of bits**, called **binary number.**
- A text character can also be represented by a string of bits called **character code.**

- We obviously need to represent both positive and negative numbers.
- **Three systems are used for representing such numbers.**

    1. Sign- and –magnitude.
    2. 1's Complement.
    3. 2's Complement.

- In all three systems, the left most bit is 0 for **positive numbers** and 1 for **negative numbers**.
- Positive values have identical representations in all three systems, but **negative values** have different representations.

Sign- and - magnitude:

- In sign and magnitude system, negative values are represented by changing the most significant bit from 0 to 1 in the corresponding positive value.

  For example +5 is represented by 0101, and -5 is represented by 1101.

1's complement:

- In 1's complement representation, negative values are obtained by complementing each bit of the corresponding positive value.

  Thus, the representation for -3 is obtained by complementing each bit in the vector 0011 to get 1100.

2’ s complement :

- In 2’s complement, negative values are obtained by adding 1 to the 1’s complement of that number.
- 2’s complement is the most efficient way to carry out addition and subtraction operations.
- It is one most often used in computers.

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Addition and subtraction of signed numbers

Addition:

- To add two signed numbers: add their n-bit representations, **ignoring the carry-out** signal from the *most significant bit (MSB) position.*
- The sum will be algebraically correct value in the 2's complement representation as long as the answer is in the range -2^{n-1} to $+2^{n-1}-1$.

Subtraction of signed numbers:

- To subtract two numbers X and Y, that is to perform X-Y, **form the 2's complement of Y and then add it to X,** the result will be the algebraically correct value in the 2's complement representation system if the answer is in the range -2^{n-1} through $+2^{n-1}-1$.

(a)

0 0 1 0	(+2)
+ 0 0 1 1	(+3)
0 1 0 1	(+5)

(b)

0 1 0 0	(+4)
+ 1 0 1 0	(−6)
1 1 1 0	(−2)

(c)

1 0 1 1	(−5)
+ 1 1 1 0	(−2)
1 0 0 1	(−7)

(d)

0 1 1 1	(+7)
+ 1 1 0 1	(−3)
0 1 0 0	(+4)

(e)

1 1 0 1	(−3)
− 1 0 0 1	(−7)

⟹

1 1 0 1	
+ 0 1 1 1	
0 1 0 0	(+4)

(f)

0 0 1 0	(+2)
− 0 1 0 0	(+4)

⟹

0 0 1 0	
+ 1 1 0 0	
1 1 1 0	(−2)

(g)

0 1 1 0	(+6)
− 0 0 1 1	(+3)

⟹

0 1 1 0	
+ 1 1 0 1	
0 0 1 1	(+3)

(h)

1 0 0 1	(−7)
− 1 0 1 1	(−5)

⟹

1 0 0 1	
+ 0 1 0 1	
1 1 1 0	(−2)

(i)

1 0 0 1	(−7)
− 0 0 0 1	(+1)

⟹

1 0 0 1	
+ 1 1 1 1	
1 0 0 0	(−8)

(j)

0 0 1 0	(+2)
− 1 1 0 1	(−3)

⟹

0 0 1 0	
+ 0 0 1 1	
0 1 0 1	(+5)

Overflow in integer arithmetic

- In the 2's complement number representation system, n bits can represent values in the range -2^{n-1} **through** $+2^{n-1}-1$. for example, using four bits, the range of numbers that can represented is -8 through +7.
- When the **result of an arithmetic operation is outside regrettable range**, an overflow has occurred.

For example, when using 4- bit signed numbers, if we try to add the numbers +7 and +4, the output sum vector , S, is 1011. which is code for -5, an incorrect result.

Similarly, if we try to ad -4 and -6, we get S= 0110= +6, another incorrect result

- Thus , overflow **may** occur if both summands have the same sign.
- A simple way to detect overflow is to examine the signs of the two summands X and Y and the sign of the result. When both operands X and Y have the same sign, an overflow occurs when the sign of S is not the same as the signs of X and Y.

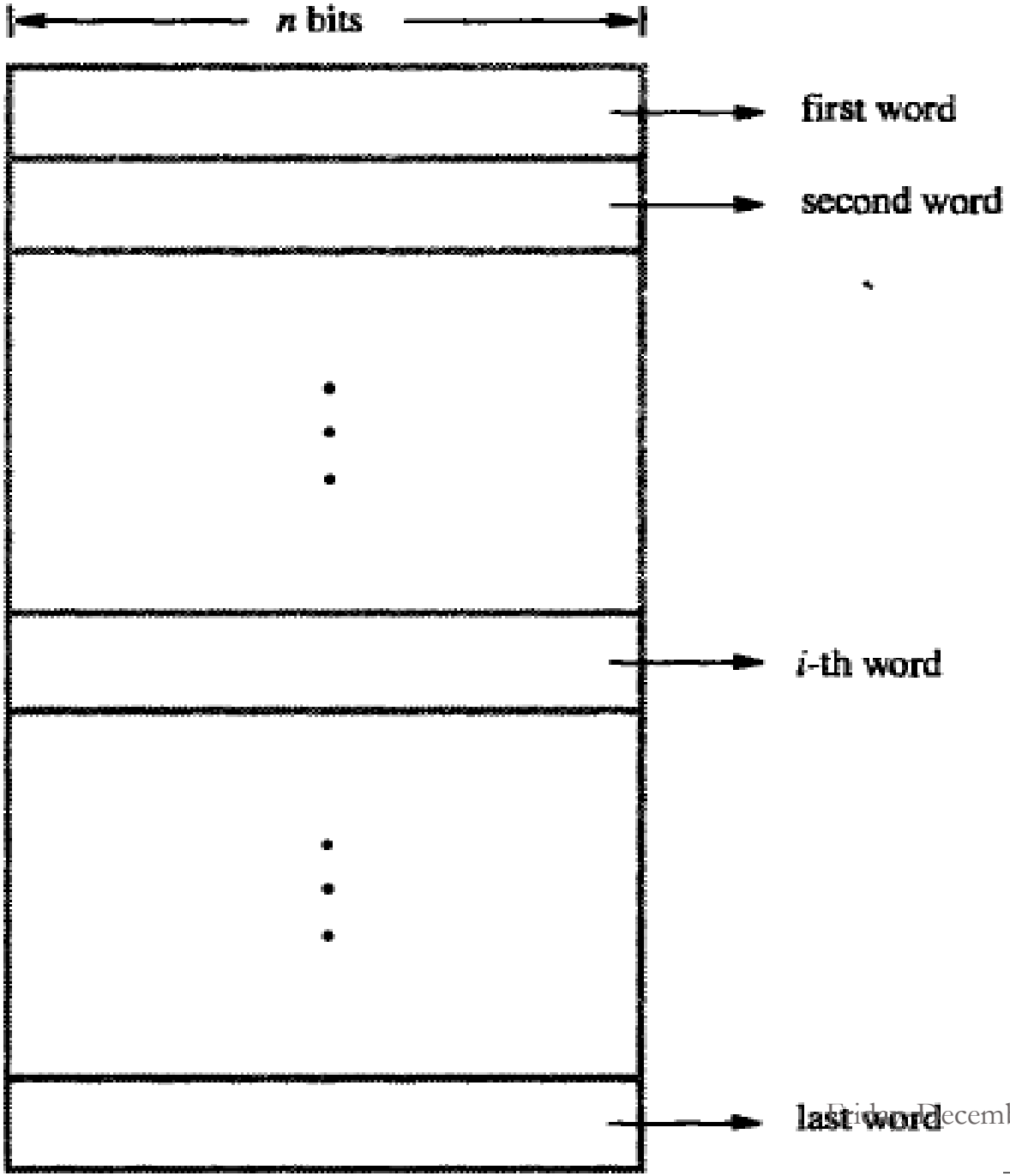
Characters :

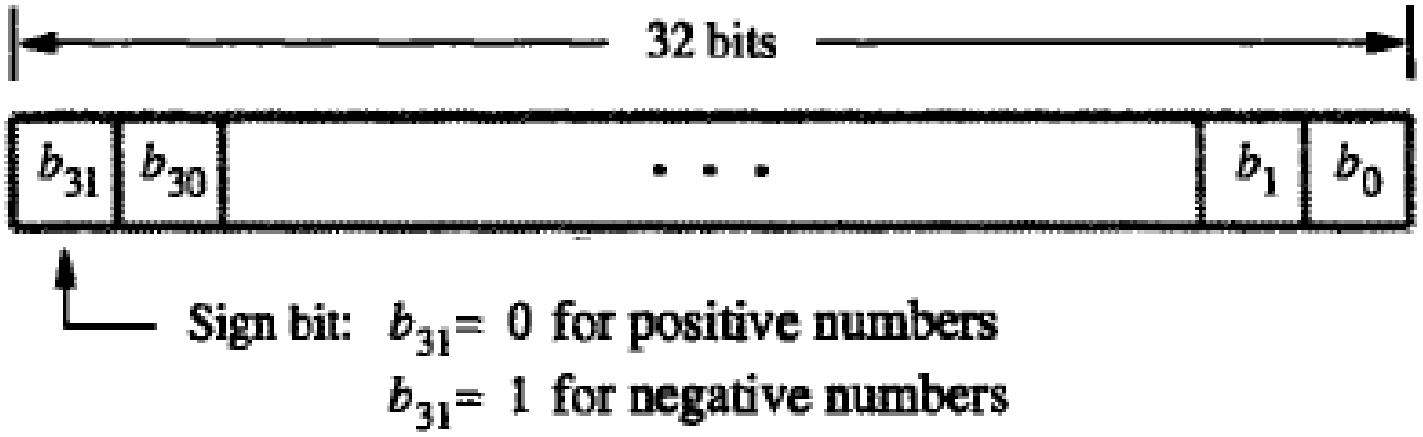
- In addition to numbers computers must be able to handle nonnumeric text information consisting of characters. Characters can be letters of the alphabet, punctuation marks, and so on.
- They are represented by codes that usually eight bits long.
- One of the most widely used codes is the **American Standards Committee on Information Interchange (ASCII).**

Memory locations and Address

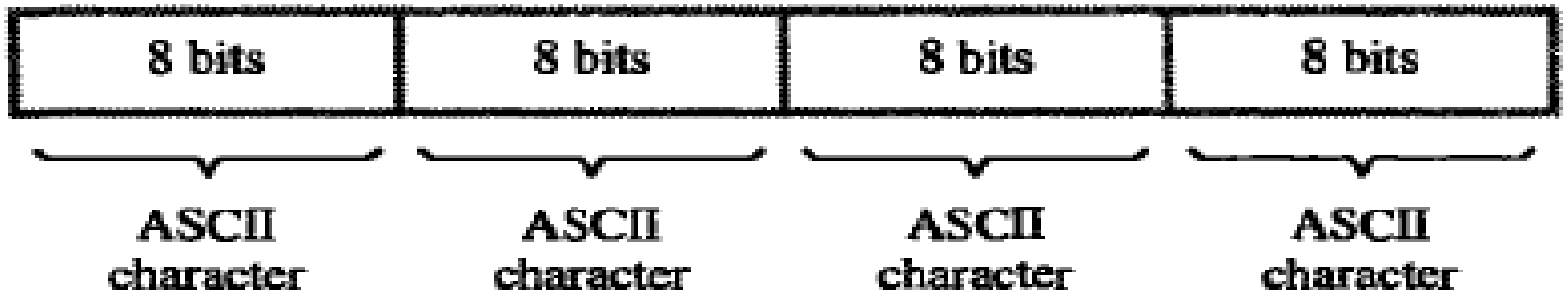
- **Number** and **character operands**, as well as **instructions**, are stored in the memory of a computer.
- We will now consider how the memory is organized. The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.
- Because a single bit represents a very small amount of information , bits are not handled individually. The usual approach is to deal with them in groups of fixed size.
- For this purpose, 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 ***word length.***
- The memory of a computer can be schematically **represented as a collection of words.**
- If the word length of a computer is 32 bits, a single word can **store a 32- bit 2's complement number** or **four ASCII characters**, each occupying 8 bits .
- A unit of 8 bits is called a byte.


n bits
first word
second word
i-th word
last word



(a) A signed integer



(b) Four characters

Byte addressability:

- We now have three basic information quantities to deal with the bit, byte, and word.
- A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits.
- It is **impractical to assign different address** to individual bit locations in the memory.
- The most practical assignment is to have successive address refer to successive byte locations memory. this is called ***BYTE ADDRESSABLE*** *memory.*

Big-Endian and Little –Endian assignments

- There are two ways that byte address can be assigned across words as shown in the figure.

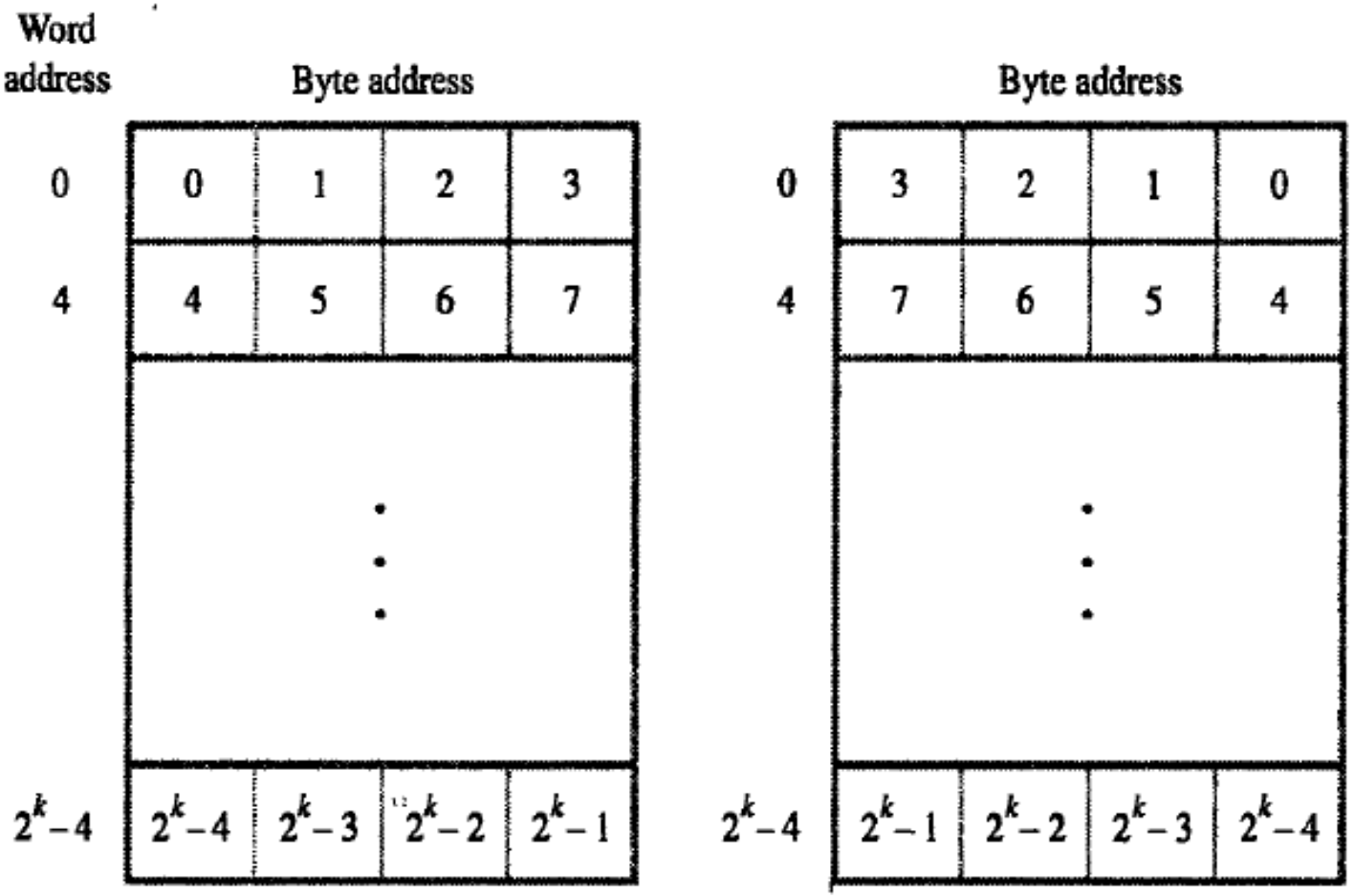
Big-Endian:

- The name big- endian is used when **lower byte** addresses are used for the **most significant bytes**(left most bytes) of the word.

Little- Endian:

- The name little-endian is used for the opposite ordering, where the **lower byte** addresses are used for the **least significant bytes**(the right most bytes)of the word.

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



Eg – store a word "JOHNSENA" in memory starting from word 1000, using Big Endian and Little endian.

Bigendian -

1000	J **1000**	O **1001**	H **1002**	N **1003**
1004	S **1004**	E **1005**	N **1006**	A **1007**

Little endian -

1000	N **1000**	H **1001**	O **1002**	J **1003**
1004	A **1004**	N **1005**	E **1006**	S **1007**

Word Alignment:

- In case of a 32- bit word, boundaries occur at addresses 0,4,8,….. As shown in the figure.
- Thus , we say that the word locations have *aligned addresses.*
- if the word length is 16(2 bytes), aligned words begin at byte addresses 0,2,4,…..for a word length of 64(8 bytes), aligned words begin at addresses 0,8,16,……

- Each group of n bits is referred to as a word of information, and n is called ***word length.***
- The memory of a computer can be schematically represented as a **collection of words.**

For 16 bit CPU

4000	34H
4002	65H
4004	86H
4006	93H
4008	45H

(Here, no. of bytes of a word is 2, and the address of word is in multiples of 2)

For 32 bit CPU

4000	34H
4004	65H
4008	86H
4012	93H
4016	45H

(Here, no. of bytes of a word is 4, and the address of word is in multiples of 4)

For 64 bit CPU

4000	34H
4008	65H
4016	86H
4024	93H
4032	45H

(Here, no. of bytes of a word is 8, and the address of word is in multiples of 8)

Accessing numbers, characters , and character 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.**
- In many applications, it is necessary to handle character strings of variable length.
- The beginning of the string is indicated by giving the address of the **byte containing its first character**. Successive byte locations contain successive characters of the string.

- **There are two ways to indicate the length of the string:**

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

  2)A processor register can contain a number indicating the length of the string in the bytes.

Memory operations

- Both program instructions and data operands are stored in the memory.
- To execute an instruction, the processor has to fetch that instruction from memory.
- The result of the operation may also be copied to the memory.
- **Thus, two basic operations are performed with the memory:**

1. Load(read or fetch)
2. store (write)

Load(Read) Operation

- The load operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged.

the steps are:

1. processor sends the **address** of the desired memory location.
2. processor **issues a read control** signal to memory to request the data .
3. Memory **sends the data** to the processor.
4. Data gets **stored in processor register**.

Store(Write) Operation

- The store operation transfers an item of information from the processor to a specific memory location, destroying the former contents of that location.

the steps are:

- Processor **sends the address** of the memory location where it wants to store data.
- A **write signal is issued** by the processor.
- The content of the processor register is written into the **specified memory location**.

Instructions & instruction sequencing

- A computer performs its task according to the program stored in memory.
- A program is a collection of instructions which tell the processor to perform a basic operation like addition, reading a character from keyword.
- **A computer should have instructions to perform at least four types of operations listed below:**

  1. data transfer between memory and processor registers.
  2. Arithmetic and logic operations on data.
  3. program sequencing and control.
  4. I/O transfers.

- To illustrate the operations, we need to know how to represent the operations and operands for an instruction.
- **In this section two types of notations:**

Register Transfer Notation(RTN):

- In this notation, we identify a memory location by a symbolic name in the upper case letters, the name indicates its address. For example, LOCA, A, VAR2; processor register names may be R0, R5; and I/O register names may be DATAIN, DATAOUT.

R1← [LOCA]

- The contents of a location are denoted by placing square brackets around the name of the location.
- Thus the expression means that the contents of memory location LOCA are transferred into processor register R1.
- As another example, consider the operation that adds the contents of the register R1and R2, and then places their sum into register R3. this action is indicated as

R3← [R1]+[R2] **register transfer notation**

This type of notation is known as.

Note that the Right- hand side of an RTN expression always denotes a value and the Left-hand side is the name of the location where the value is to be placed, overwriting the old contents of that location.

Assembly language notation:

- We need an another type of notation to represent machine instructions and programs.
- For this, we use an *assembly language* format.
- For example, an instruction that causes the transfer from memory location LOCA to processor register R1, is specified by the statement.

MOV LOCA, R1

The contents of LOCA are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.

The second example of adding two numbers contained in the processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement .

ADD R1,R2,R3

Basic instruction types

A statement like C = A+B in a high – level language program informs the computer to add the values of the two variables called A and B and assign the sum to a third variable called C. when the program containing this statement is compiled, each variable is assigned a different address in memory. The contents of these locations represents the values of three variables.

Hence, the above high- level language statement requires the action

$$C \leftarrow [A] + [B]$$

To carry out this kind of operation, assembly language provides three types of instruction formats:

Three address instruction:

- The instruction has the format

operation Source1, source2, Destination

Using this format the above operation can be completed using a single machine instruction as

Add A,B,C

A and B are called source operands, C is the destination operand and Add is the operation to be performed. This type of instruction has the disadvantage the instruction code will be too large to fit in one word location in memory.

Two address instruction:

- The general format is

operation Source, Destination

An add instruction of this type is

Add A,B

Which performs the operation B ← [A] +[B]. when the sum is calculated, the result is sent to the memory and stored in location B, replacing the original contents of this location. This means that operand B is both source and destination.

- A single two – address instruction cannot be used to solve our original problem, which is to add the contents of locations A and B, without destroying either of them, and to place the sum in location C.
- This problem can be solved by using **another two address instruction** that copies the contents of one memory location into another.

  such an instruction is Move B,C which performs the operation C← [B]. Leaving the contents of location unchanged.
- The operation C← [A] +[B] can now be performed by the two – instruction sequence.

```
Move  B,C
Add   A,C
```

One -Address instruction:

- The general format is

operation source/destination

e.g. **Add A**

This instruction adds the contents of the memory location A to the contents of the accumulator register and place the sum back into the accumulator.

Load A

The load instruction copies the contents of the memory location A into the accumulator.

Store A

The store instruction copies the contents of the accumulator into memory location A.

- Using one address- instructions, the operation C ← [A]+[B] can be performed by executing the sequence of instructions.

```
Load   A
Add    B
Store  C
```

- In processors, where arithmetic operations are allowed only on operands that are in the processor registers , the C= A+B task can be performed by the instruction sequence

mov A, R_i

mov B, R_j

Add R_i, R_j

mov R_j, C

- In processors where one operand may be in the memory but the other must be in a register, an instruction an instruction sequence for the required task would be

```
Mov A, Ri
Add B, Ri
mov Ri, C
```

Instruction & straight –line sequencing

- So far we have discussed the instruction formats.
- We used the task C← [A]+[B] for illustration.
- Figure shows the possible program segment for this task as it appears in the memory of a computer.
- We have assumed that the computer allows one memory operand per instruction and has a number of processor registers.

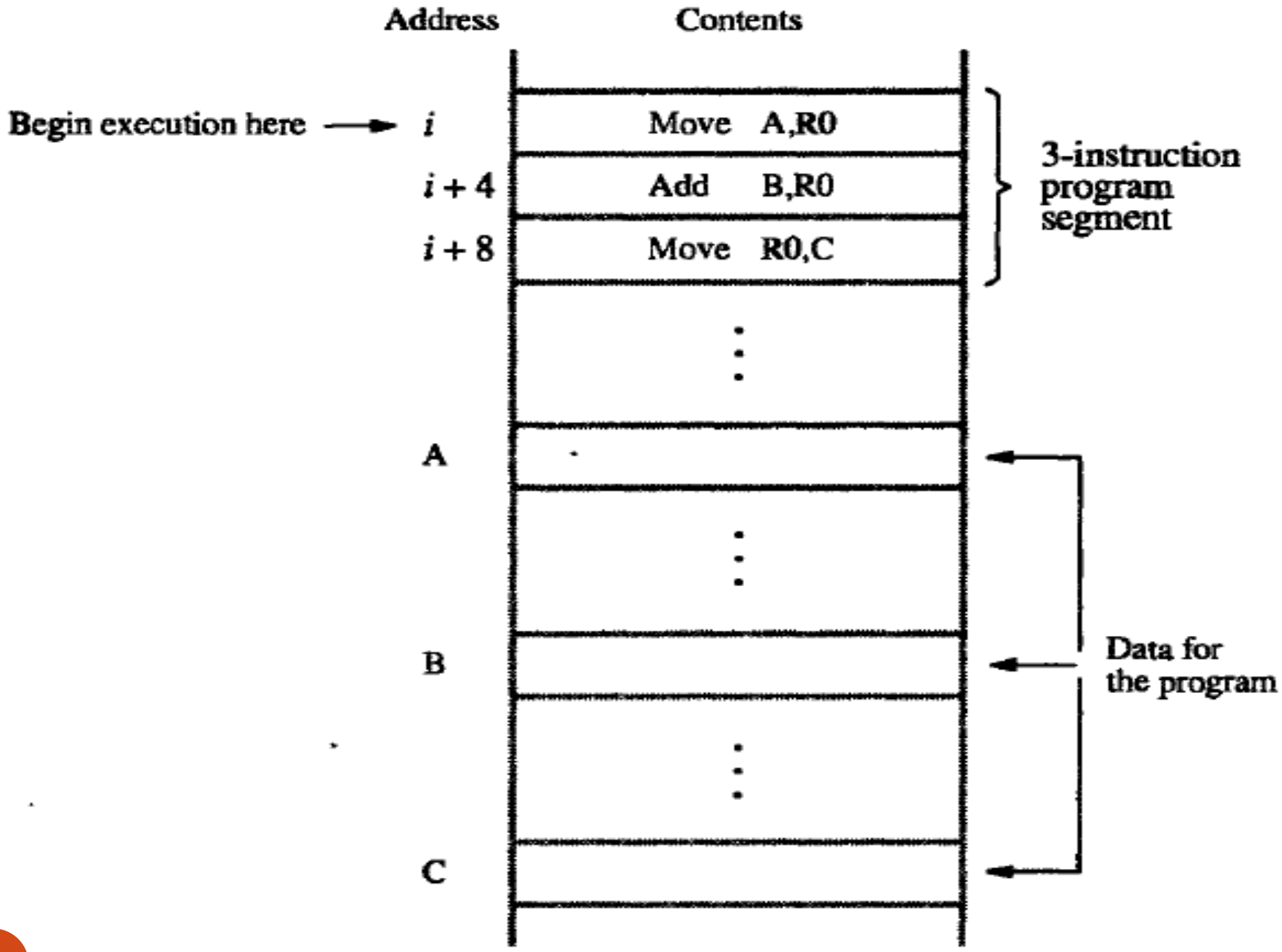


Figure 2.8 A program for C ← [A] + [B].

- We assume that the word length is 32 bits and the memory is Byte addressable. The three instructions of the program are in successive word locations, starting at location i. since each instruction is 4bytes long, the second and third instructions start at addresses i+4 and i+8.
- Let us consider how this 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 (i in our example), must be placed into the PC. then the processor 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. move instruction at location i+8 is executed , the PC contains the value i+12. which the address of the first instruction of the next program.
- Executing a given instruction is a two- phase procedure.

First phase(instruction fetch):

The instruction is fetched from the memory location whose address is in the PC. this instruction is placed in the *Instruction Register(IR).*

Second phase (Execute phase):

- At the start of second phase, called instruction execute, the instruction in IR is examined to determine which operation is to be performed.
- The specific operation is then performed by the processor.

Branching

- Consider the task of adding list of 'n' numbers.
- The addresses of the memory locations containing the 'n' numbers are symbolically given as NUM1, NUM2,.....NUM n, 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.

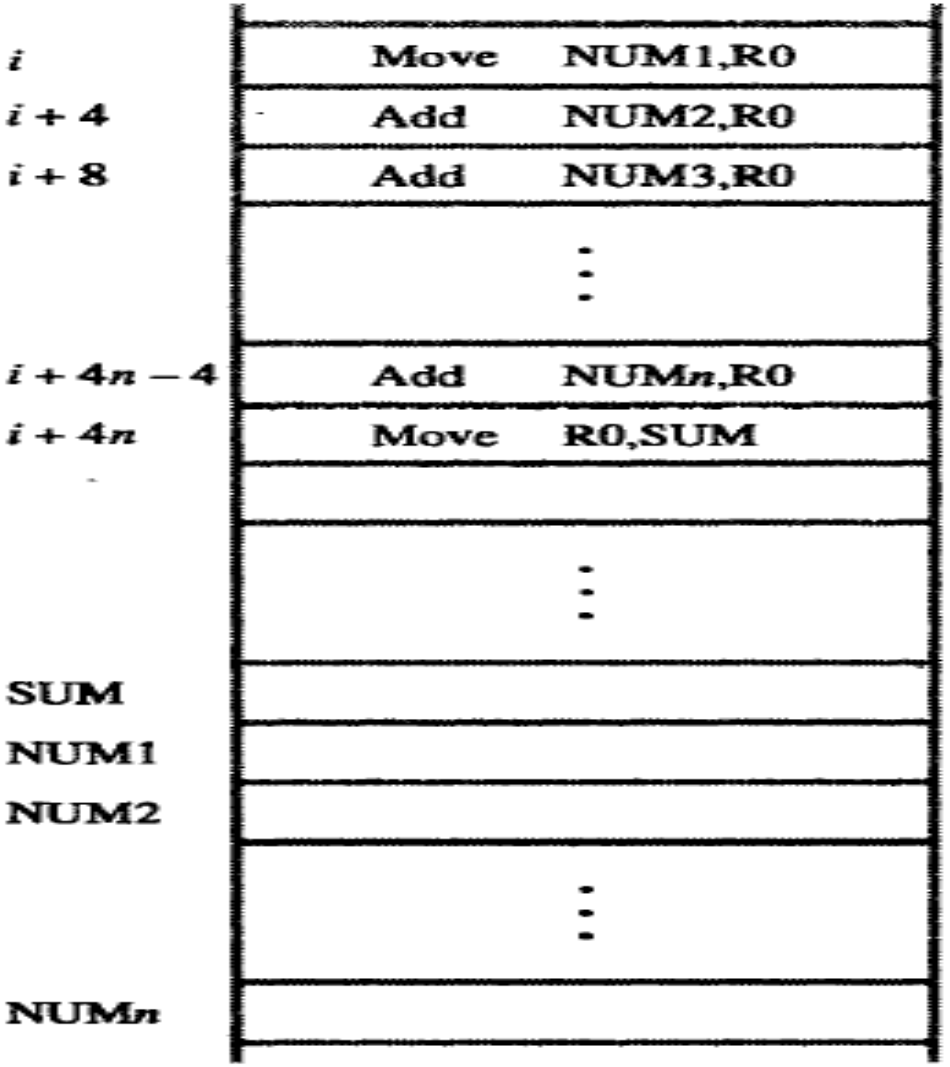


Figure 2.9 A straight-line program for adding n numbers.

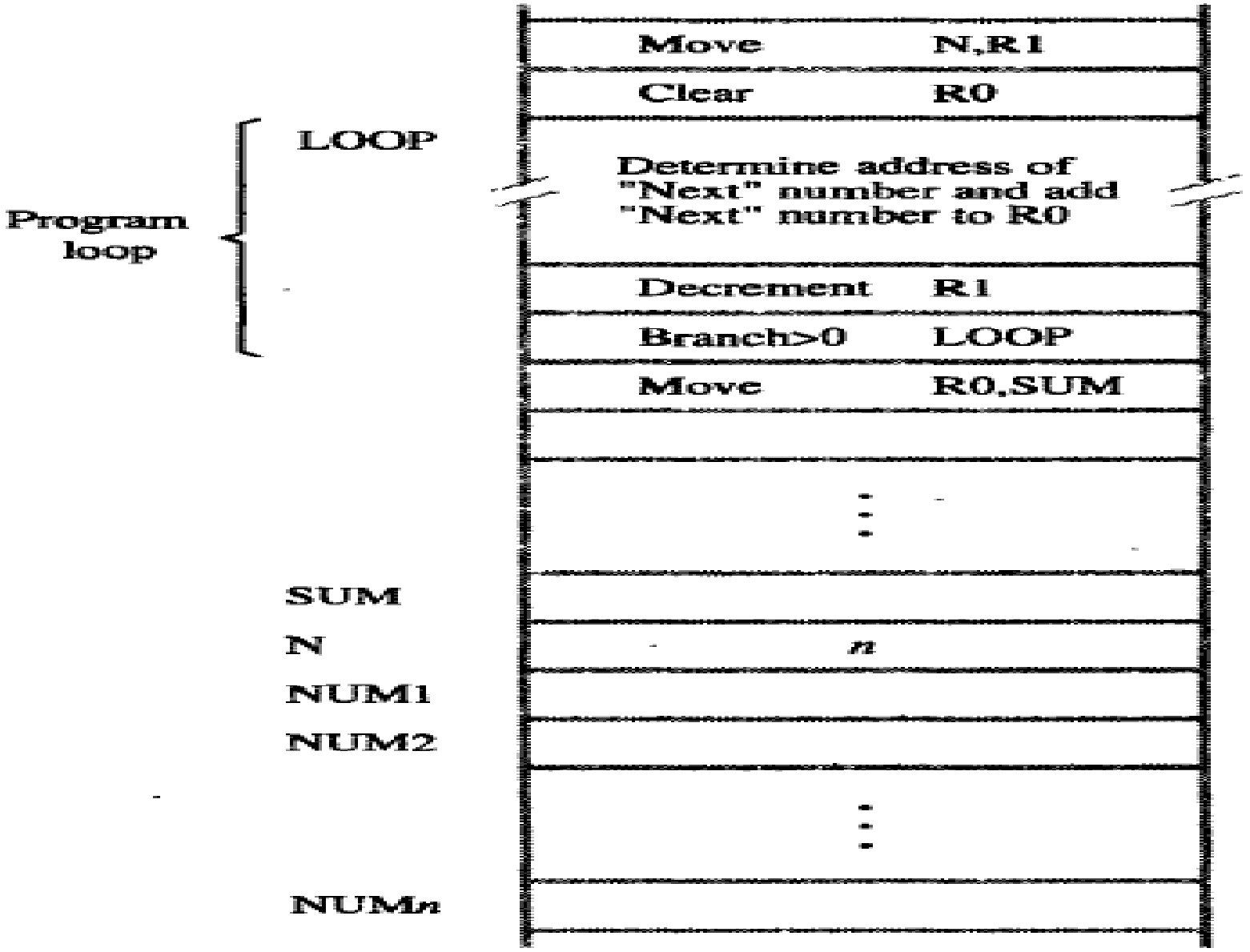


Figure 2.10 Using a loop to add *n* numbers.

- Instead of using a long list of add instructions, it is possible to place a single Add instruction in a program loop as shown in figure.
- 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 that entry is fetched and added to R0.
- Assume that the number of entries in the list , n is stored in memory location N, as shown. Register R1 is used as a counter to determine the number of times the loop is executed.
- Hence the contents of location n are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction

Decrement R1

Reduces the contents of R1 by each time through the loop.

Execution of the loop is repeated as long as the result of the decrement operation is greater than Zero.

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, often called *condition code flags*.
- These flags are usually grouped together in a special processor register called the *condition code register or status register.*
- Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

ADDRESSING MODES

- The different ways in which the location of an operand is specified in an instruction are referred to as ***addressing modes.***
- We can access an operand by specifying the ***name of the register*** or the ***address of the memory location*** where the operand is located.

Register Mode:

The operand is the contents of a processor register, the name of the register is given in the instruction.

e.g. MOV R1,R2

The instruction copies the contents of register R1 to register R2.

Absolute mode:

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

E.g. **MOV LOCA,R2**

Immediate mode:

- The operand is given explicitly in the instruction.

  e.g. **MOV $200_{immediate}$,R0**

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

Hence we write the instruction above in the form

MOV #200,R0

Indirect mode:

- In this addressing mode, the instruction does not give the operand or its address explicitly, instead it provides information from which the memory address of the operand can be determined. we refer to this address as the ***effective address(EA)*** of the operand.
- The ***effective address*** of an operand is the contents of register or memory location whose address (or name) appears in the instruction.
- To execute the add instruction in the figure(A) the processor uses the value B, which is in the register R1, as the effective address of the operand. the value read is desired operand, which the processor adds to the contents or register R0.

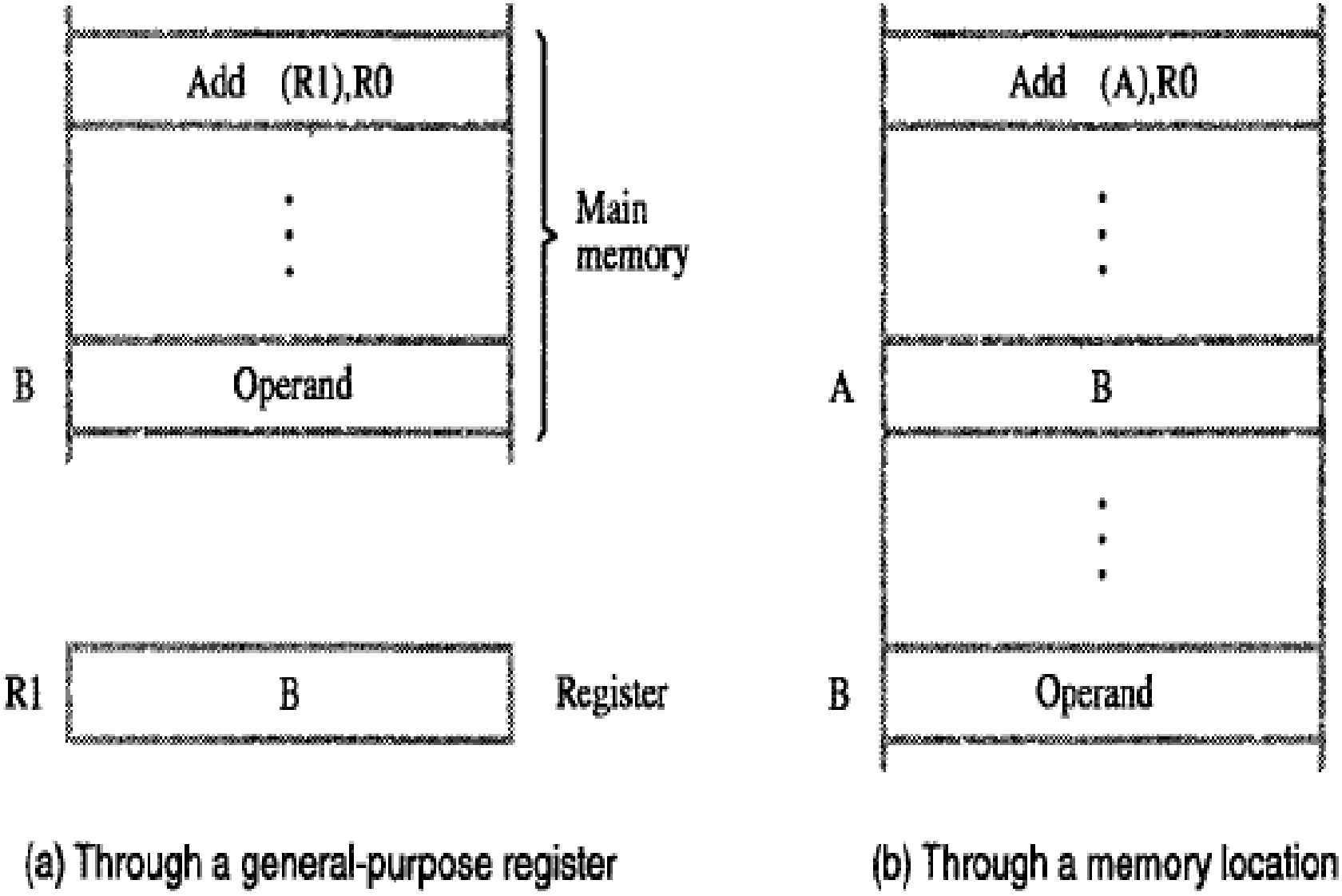


Figure 2.11 Indirect addressing.

- Indirect addressing through a memory location is also possible as shown in the figure(b). In this case processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand.
- The ***register or memory location*** that contains the address of an operand is called a ***pointer.***

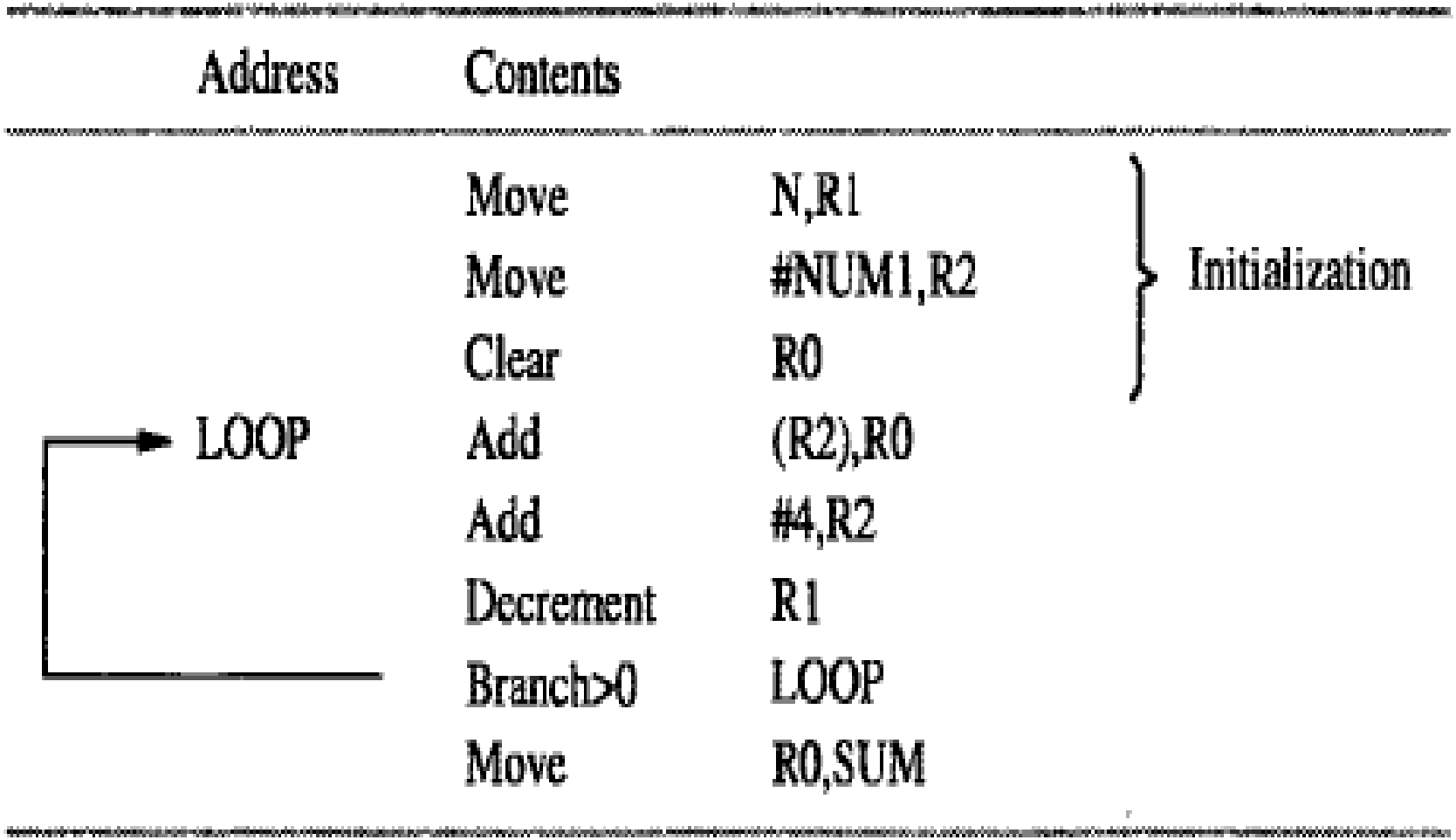


Figure 2.12 Use of indirect addressing in the program of Figure 2.10.

- Let us now return to the program of adding list of numbers.
- Indirect addressing can be used to access successive numbers in the list, resulting in the program shown in the figure.
- Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.

- The initialization section of the program loads the counter value *'n' from memory location 'N' into R1* and uses the immediate addressing mode to place the address value NUM1, which is the address of first number in the list into R2.
- Then it clears R0 to 0.
- The first 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.

Index mode :

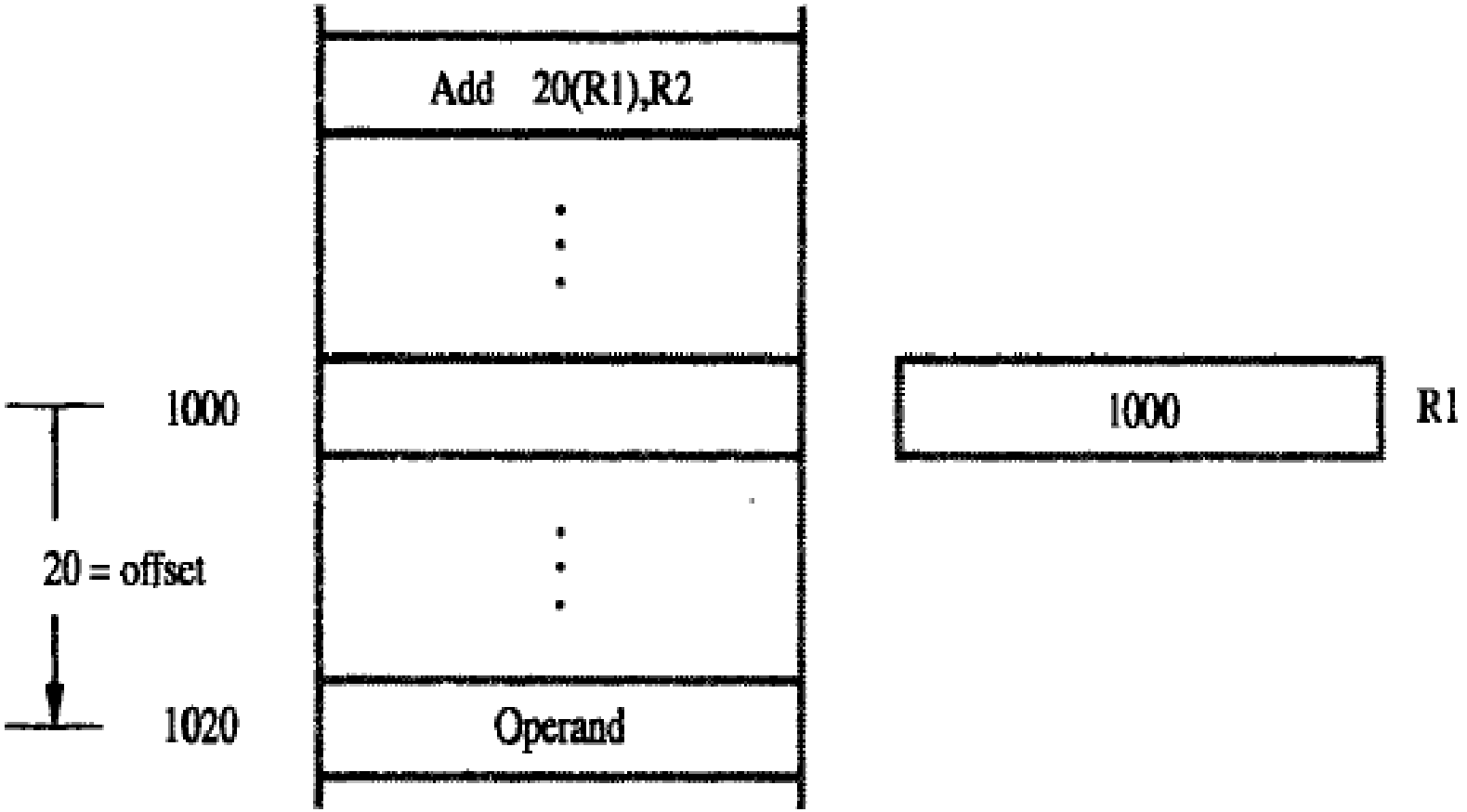
- The effective address of the operand is generated by ***adding a constant value*** to the contents of a ***register***.
- The register used may be either a special register provided for this purpose, or more commonly, it may be any one of a set of general-purpose registers in the processor.
- In either case, it is referred to as an ***index register***.
- We indicate the index mode symbolically as

$$X(R_i)$$

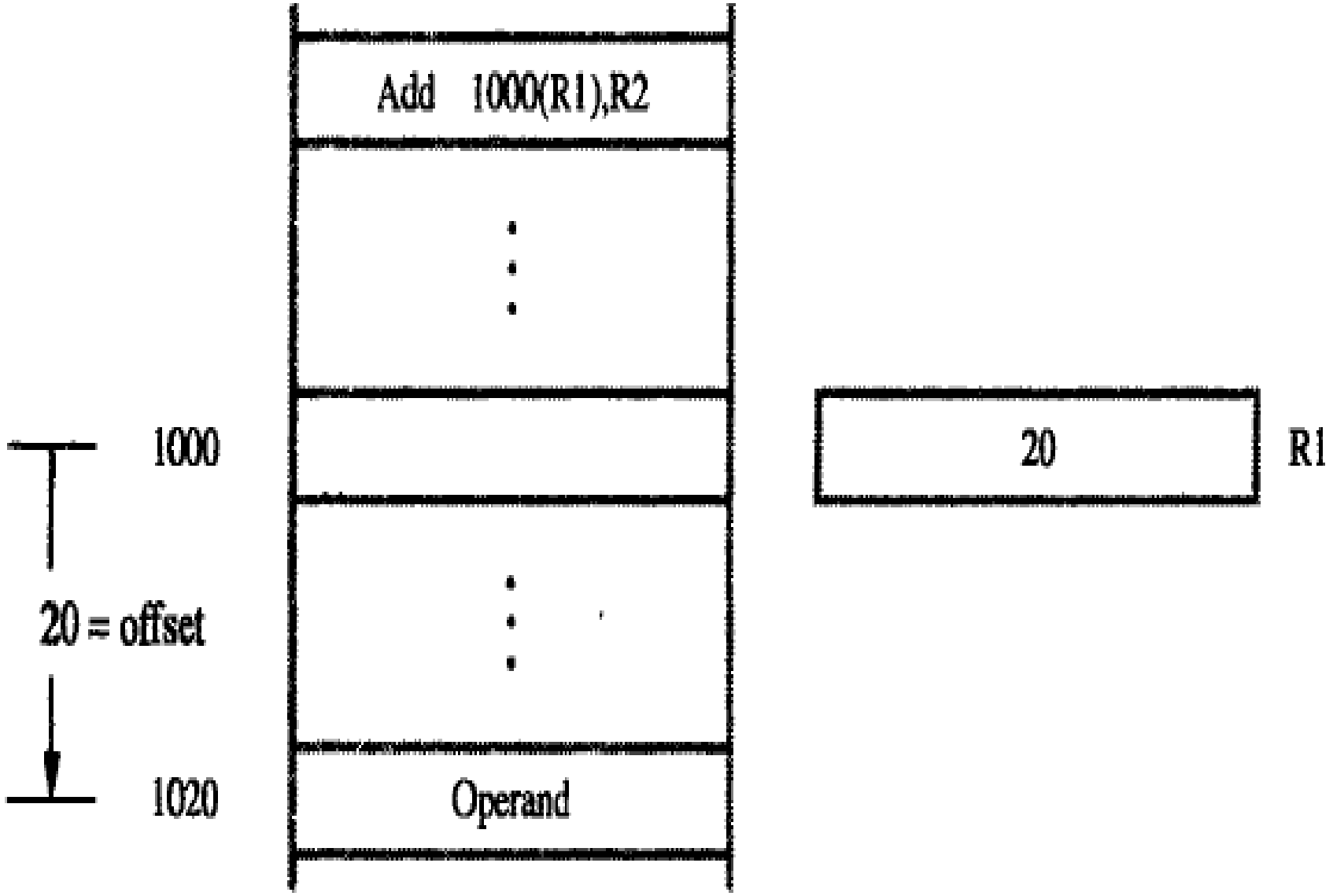
X→ denotes the constant value in the instruction, Ri is name of the register involved.

Effective address of the operand is given by

$$EA = X+[Ri]$$



(a) Offset is given as a constant



(b) Offset is in the index register

- Figure illustrates two ways of using the index mode.
- In figure A the index register R1, contains the address of a memory location and the value X defines an offset(also called displacement) from this address to the location where the operand is found.
- An alternative use is illustrated in figure 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 the two values, one is given explicitly in instruction and the other is stored in a register.

Base with Index Mode

- In this addressing mode Effective Address is the sum of two registers.

Syntax:

(R_i, R_j)

Effective Address:

$EA = [R_i] + [R_j]$

The first register is called Index Register.

The second register is called Base Register.

Base with Index and Offset

- In this addressing mode effective address is the sum of the constant(X) and the contents of the two registers.

Syntax:

$X(R_i, R_j)$

$EA=[R_i]+[R_j]+X$

Relative Addressing:

- We have discussed the index mode using general-purpose register.
- A useful version of this obtained if **program counter(PC)** is used instead of a general purpose register.
- Then X(PC) can be used to address a memory location that is X bytes away from the location pointed by the program counter.

$$EA=[PC]+X$$

The effective address is determined by the index mode using the program counter in the place of the general purpose register.

Additional Modes

- So far we have discussed the six basic addressing modes- register, absolute, immediate, index, indirect and relative.
- Many computers provide two additional modes:

1) Autoincrement Mode
2) Autodecrement Mode

Autoincrement mode:

- The ***effective address*** of an operand *is contents of register* specified in the instruction. After accessing the operand, the contents of the register are **automatically incremented** to point to the next item in the list.

- The autoincrement mode is written as

(Ri)+

Autodecrement 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.
- We denote the autodecrement mode by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the register are to be decremented before being used as the effective address. Thus we write

-(Ri)

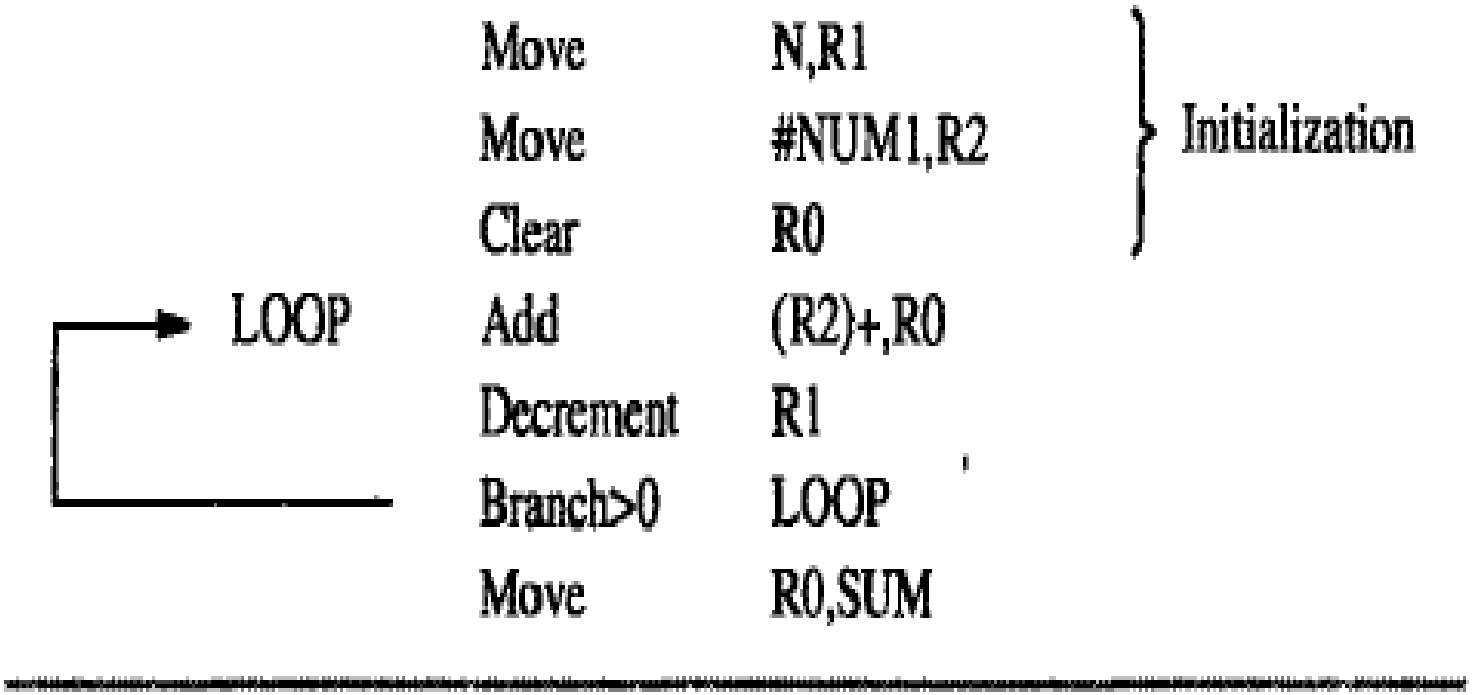


Figure 2.16 The Autoincrement addressing mode used in the program of Figure 2.12.

8. Register R1 and R2of computer contain the decimal value 1200 and 4600. What is the effective address of the source operand in each of the following instructions?

i)Load 20(R1), R5 ii) Move #3000, R5 iii) Store R5, 30(R1,R2)

iv) Add –(R2), R5 v) subtract (R1)+, R5