UNIT I
BASIC STRUCTURE OF COMPUTERS

- **Functional units**
- **Basic operational concepts**
- **Bus structures**
- **System software**
- **Performance and metrics**
- **The history of computer development**

Computer organization:

It concerned with the way the hardware components operate and the way they are connected to form a computer system

Computer architecture:

It is concerned with the structure and behavior of computer system. It includes information related instruction set, no of bits used for data, addressing modes.

Functional units of computer system:

A computer consists of five functionally independent main parts input unit ,memory unit , arithmetic logic unit (ALU), output unit and control unit.

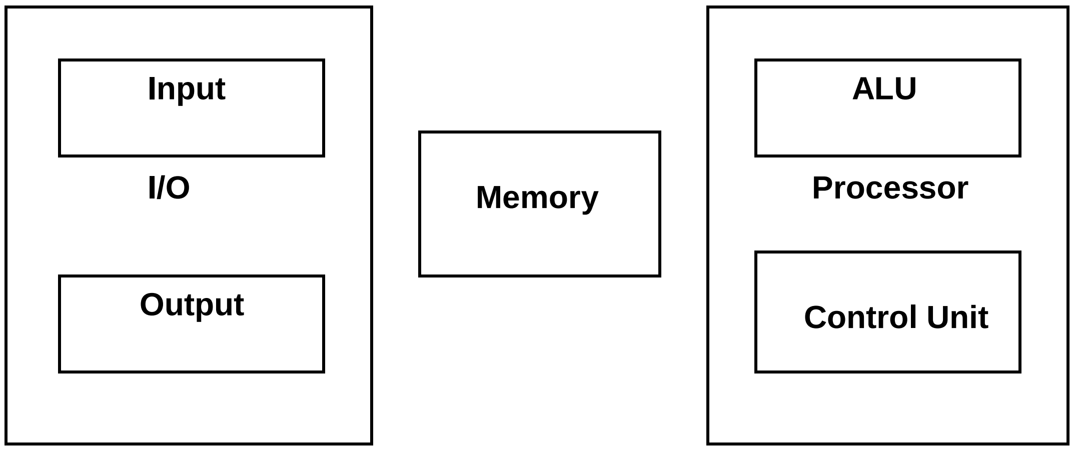


Fig:Functional units of computer

Input device accepts the coded information as source program i.e. high level language. This is either stored in the memory or immediately used by the processor to perform the desired operations. The program stored in the memory determines the processing steps. Basically the computer converts one source program to an object program. i.e. into machine language.

Finally the results are sent to the outside world through output device. All of these actions are coordinated by the control unit.

Input unit: -

The source program/high level language program/coded information/simply data is fed to a computer through input devices keyboard is a most common type. Whenever a key is pressed, one corresponding word or number is translated into its equivalent binary code over a cable & fed either to memory or processor.

Joysticks, trackballs, mouse, scanners etc are other input devices.

Memory unit: -

Its function into store programs and data. It is basically to two types

Primary memory
Secondary memory

1. Primary memory: - Is the one exclusively associated with the processor and operates at the electronics speeds programs must be stored in this memory while they are being executed. The memory contains a large number of semiconductors storage cells. Each capable of storing one bit of information. These are processed in a group of fixed site called word.

To provide easy access to a word in memory, a distinct address is associated with each word location. **Addresses are** numbers that identify memory location.

Number of bits in each word is called word length of the computer. Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of processor.

Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random-access memory (RAM).

The time required to access one word in called memory access time. Memory which is only readable by the user and contents of which can't be altered is called read only memory (ROM) it contains operating system.

Caches are the small fast RAM units, which are coupled with the processor and are aften contained on the same IC chip to achieve high performance. Although primary storage is essential it tends to be expensive.

2 Secondary memory: - Is used where large amounts of data & programs have to be stored, particularly information that is accessed infrequently.

Examples: - Magnetic disks & tapes, optical disks (ie CD-ROM's), floppies etc.,

Arithmetic logic unit (ALU):-

Most of the computer operators are executed in ALU of the processor like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called register. Then according to the instructions the operation is performed in the required sequence.

The control and the ALU are may times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as key boards, displays, magnetic and optical disks, sensors and other mechanical controllers.

Output unit:-

These actually are the counterparts of input unit. Its basic function is to send the processed results to the outside world.

Examples:- Printer, speakers, monitor etc.

Control unit:-

It effectively is the nerve center that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

Basic Operational Concepts Of Computer

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 executes the specified operations. Data to be stored are also stored in the memory.

Examples: - Add LOCA, R_0

This instruction adds the operand at memory location LOCA, to operand in register R_0 & places the sum into register. This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor.
2. The operand at LOCA is fetched and added to the contents of R_0
3. Finally the resulting sum is stored in the register R_0

The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons.

Load LOCA, R1

Add R1, R0

Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.

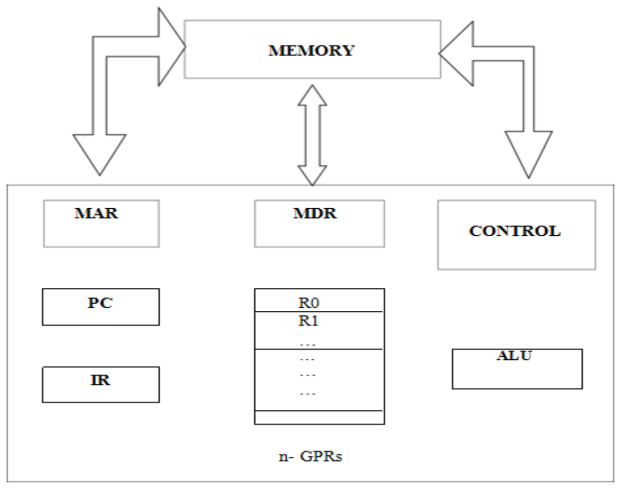


Fig b : Connections between the processor and the memory

The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

The instruction register (IR):- Holds the instructions that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

The program counter PC:-

This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

Besides IR and PC, there are n-general purpose registers R0 through R_{n-1}.
The other two registers which facilitate communication with memory are: -

2. **MAR – (Memory Address Register):-** It holds the address of the location to be accessed.
3. **MDR – (Memory Data Register):-** It contains the data to be written into or read out of the address location.

Operating steps are

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.
10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

Normal execution of a program may be preempted (temporarily interrupted) if some devices require urgent servicing, to do this one device raises an Interrupt signal.

An interrupt is a request signal from an I/O device for service by the processor. The processor provides the requested service by executing an appropriate interrupt service routine.

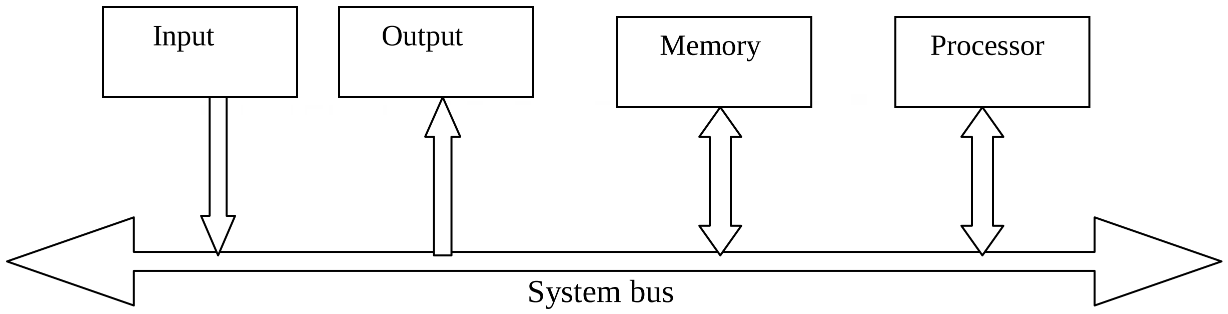
The Diversion may change the internal stage of the processor its state must be saved in the memory location before interruption. When the interrupt-routine service is Completed the state of the processor is restored so that the interrupted program may continue.

Bus Structures:

Bus :A group of lines that serves as the connection path to several devices is called a bus. A bus may be lines or wiresThese lines carry data or address or control signal.

The simplest and most common way of interconnecting various parts of the computer. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time. A group of lines that serve as a connecting port 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. Simplest way to interconnect is to use the single bus as shown



Since the bus can be used for only 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 one bus.

Single bus structure advantages

- Low cost
- Very flexible for attaching peripheral devices

Multiple bus structure certainly increases the performance but also increases the cost significantly.

Software: It a set of instructions or programs instructing a computer to do specific tasks**.**

Software types:

- System software
- Application Software

System Software: It is a collection of programs that are executed as needed to perform function such as,

- Receiving & Interpreting user commands.
- Entering & editing application program and storing them as files in secondary Storage devices.
- Managing the storage and retrieval of files in Secondary Storage devices.
- Running the standard application such as word processor, games and spread sheets 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.
- Linking and running user-written application programs with existing standard library routines.
  Eg: Compiler, Linker , Loaders, operating System ,Editors

Application

Software:

It is written in high level programming language(C,C++,Java, FORTRAN)The programmer using high level language need not know the details of machine program instruction.

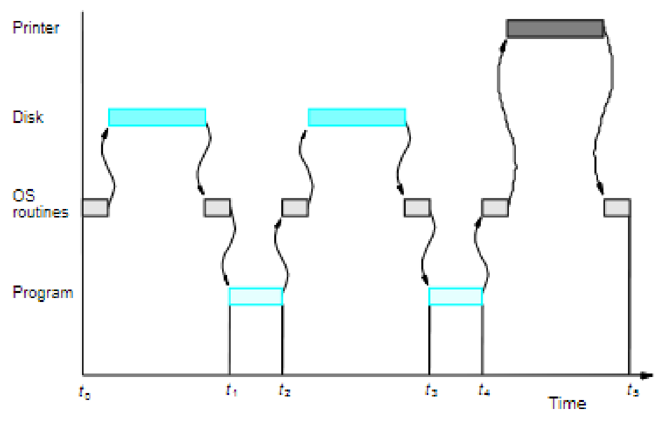


Figure 1.4. User program and OS routine sharing of the processor.

Steps:

- The first step is to transfer the file into memory.
- When the transfer is completed, the execution of the program starts.
- During time period t0 to t1 , an OS routine initiates loading the application program from disk to memory, wait until the transfer is complete and then passes the execution control to the application program & print the results.
- Similar action takes place during t2 to t3 and t4 to t5.
- At t5Operating System may load and execute another application program.
- Similarly during t0 to t1 the Operating System can arrange to print the previous program‟ results while the current program is being executed.
- The pattern of managing the concurrent execution of the several application programs to make the best possible use of computer resources is called the multi- programming or multi-tasking.

Performance:

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

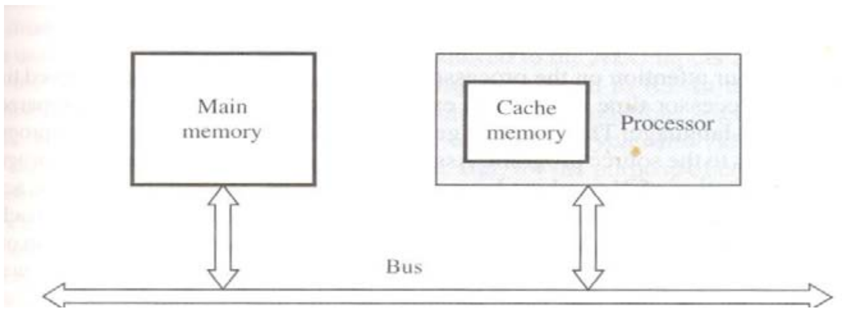
Elapsed Time →The total time required to execute the program is called the elapsed time.

It depends on all the units in computer system.

Processor Time →The period in which the processor is active is called the processor time.

It depends on hardware involved in the execution of the instruction.

Fig: The Processor Cache



A Program will be executed faster if the movement of instruction and data between the main memory and the processor is minimized, which is achieved by using the Cache.

Processor clock:

Clock: The Processor circuits are controlled by a timing signal called a clock.

Clock Cycle: The cycle defines a regular time interval called clock cycle.The length of the clock cycle is an important measure of performance

$$\boxed{\text{Clock Rate, } R = 1 / P}$$

Where, P-length of one clock cycle.

Basic Performance Equation: To access the performance of the processor following is the equation:

$$\boxed{T = (N*S)/R}$$

Where, T-Performance Parameter

R-Clock Rate
in cycles/sec

N-Actual number of instruction execution

S-Average number of basic steps needed to execute one machine instruction.

To achieve high performance,

$N, S < R$

Pipelining and Superscalar operation:

Pipelining: A Substantial improvement in performance can be achieved by overlapping the execution of successive instruction using a technique called pipelining. The process of extraction of next instruction while current instruction is executing

Superscalar Execution :Several instruction can be executed in parallel by creating parallel paths. This mode of operation is called the Superscalar execution.

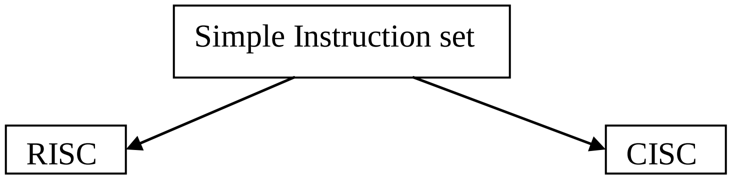
Clock Rate:

There are 2 possibilities to increase the clock rate(R).They are,

- Improving the integrated Chip(IC) technology makes logical circuits faster.
- Reduce the amount of processing done in one basic step also helps to reduce the Clock period P.

Instruction Set: CISC AND RISC:

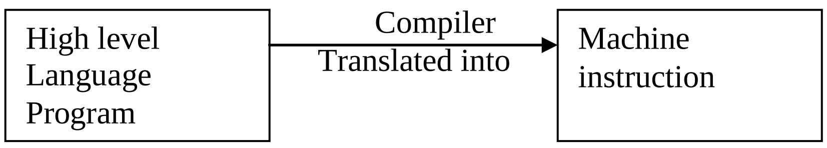
- The Complex instruction combined with pipelining would achieve the best performance.
- It is much easier to implement the efficient pipelining in processor with simple instruction set.



(Reduced Instruction Set Computer)
It is the design of the instruction set of a processor with simple instruction

(Complex Instruction Set Computer)
It is the design of the instruction set of a processor with complex instruction.

Compiler:



Functions of Compiler:

- The compiler re-arranges the program instruction to achieve better performance.
- The high quality compiler must be closely linked to the processor architecture to reduce the total number of clock cycles.

Performance Measurement:

- The Performance Measure is the time it takes a computer to execute a given bench mark.
- A non-profit organization called SPEC (System Performance Evaluation Corporation) selects and publishes representative application program.

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

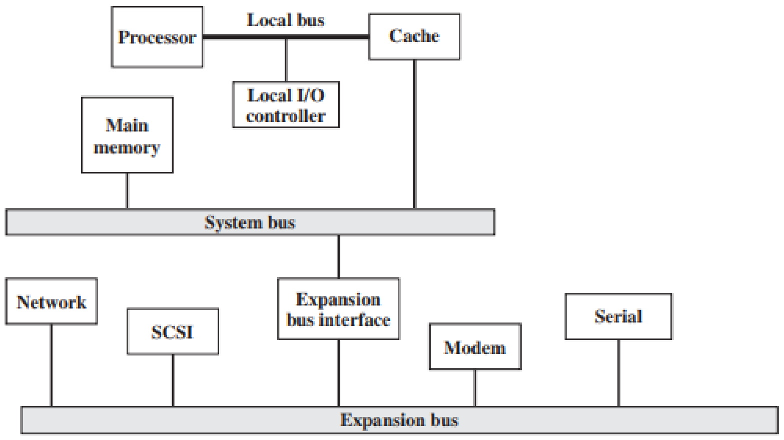
The Overall SPEC rating for the computer is given by,

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

Multiple bus hierarchies:

If a great number of devices are connected to the bus, performance will suffer. There are two main causes:
1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.

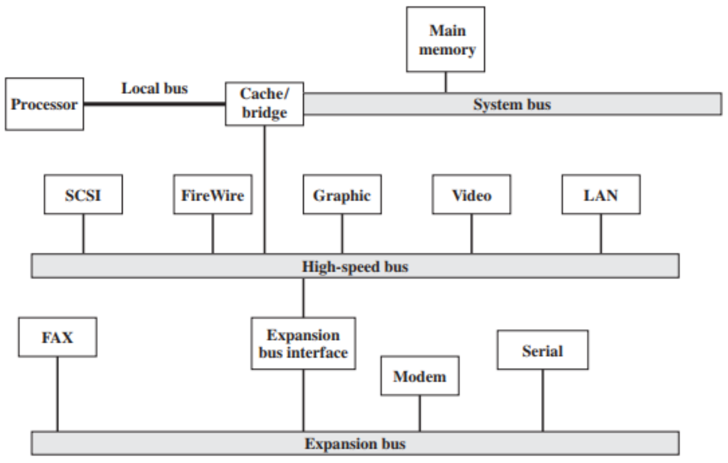
Accordingly, most computer systems use multiple buses, generally laid out in a hierarchy.



(a) Traditional bus architecture

A typical **Traditional bus structure** is shown in Figure (a). There is a local bus that connects the processor to a cache memory and that may support one or more local devices. The cache memory controller connects the cache not only to this local bus, but to a system bus to which are attached all of the main memory modules. The use of a cache structure insulates the processor from a requirement to access main memory frequently. Hence, main memory can be moved off of the local bus onto a system bus. In this way, I/O transfers to and from the main memory across the system bus do not interfere with the processor's activity. It is possible to connect I/O controllers directly onto the system bus. A more efficient solution is to make use of one or more expansion buses for this purpose .An expansion bus interface buffers data transfers between the system bus and the I/O controllers on the expansion bus. This arrangement allows the system to support a wide variety of I/O devices and at the same time insulate memory-to-processor traffic from I/O traffic. Figure 3.18a shows some typical examples of I/O devices that might be attached to the expansion bus. Network connections include local area networks (LANs) Ethernet and connections to wide area networks (WANs),SCSI (small computer system interface) is itself a type of bus used to support local disk drives and other peripherals. A serial port could be used to support a printer or scanner. This traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices.

High Speed bus architecture:



(b) High-performance architecture

In response to these growing demands, a common approach taken by industry is to build a high speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as a mezzanine architecture. Figure 3.18b shows a typical realization of this approach.Again, there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device, that connects to the high-speed bus. This bus supports connections to high-speed LANs, such as Fast Ethernet at 100 Mbps, video and graphics workstation controllers, as well as interface controllers to local peripheral buses, including

SCSI and FireWire. The latter is a high-speed bus arrangement specifically designed to support high-capacity I/O devices. Lower-speed devices are still supported off an expansion bus, with an interface buffering traffic between the expansion bus and the high-speed bus. The advantage of this arrangement is that the high-speed bus brings high demand devices into closer integration with the processor and at the same time is independent of processor.

History of Computer Development:

Development of technologies used to fabricate the processors, memories and I/O units of computers have been divided into the following generations

First generation (1945-1955):

- Program and data are reside in the same memory
- ALP was made used to write programs
- Vacuum tubes were used to implement CU,ALU
- Magnetic core and Magnetic tape storage devices are used
- Using electronic Vacuum tubes,as the switching components.

Second Generation (1955-1965):

- Transistors were used to design ALU,CU
- High level language (FORTRAN) is used
- To convert HLL to MLL compiler were used
- Separate I/O Processor were developed to operate in parallel with CPU, thus improving performance
- Invention of transistors which was faster, smaller and required considerably less power to operate

Third Generation (1965-1975):

- Integrated circuit(IC) technology improved
- Improved IC technology helped in designing low cost, high speed processor and memory modules
- Multiprogramming, pipelining concept were incorporated
- DOS allowed efficient and coordinate operation of computer system with multiple users
- Cache and virtual memory concepts were developed
- More than one circuit on single silicon chip became available

Forth generation (1975-1985):

- CPU termed as microprocessor
- INTEL,MOTOROLA,TEXAS,NATIONAL semiconductors started developing microprocessor
- Work stations, microprocessor &Notebook computers were developed

- Interconnection of different computer for better communication LAN/MAN/WAN.
- Computational speed increased by 1000 times
- Specialized processors like Digital Signal processor were also developed

Fifth generation (1985-till now):

- E-Commerce, E-Banking, Home Office
- ARM,AMD,INTEL,MOTOROLA
- High Speed processor-GHz speed
- Because of submicron IC technology lot of added features in small size.

UNIT 2

Machine Instruction and Programs

- Instructions and instruction sequencing
- Register transfer notation
- Assembly language notations
- Basic Instruction Types
- Addressing modes
- Basic input output operations
- Role of Stacks and Queues in computer programming equation
- Component of Instructions: Logic ,Shift& Rotate Instructions

Instruction And Instruction Sequencing

Tasks carried out by a computer program consist of a sequence of steps such as adding two numbers, testing for particular condition, reading a character from keyboard or sending a character to be display on display screen.A computer must have instruction capable of performing the following operations. They are,

- Data transfer between memory and processor register.
- Arithmetic and logical operations on data.
- Program sequencing and control.
- I/O transfer.

Register Transfer Notation:

Register Transfer and Micro operations

- **A microoperation** is an elementary operation performed on the information stored in one or more registers

  Eg: shift, count, clear, and load

Register transfer language

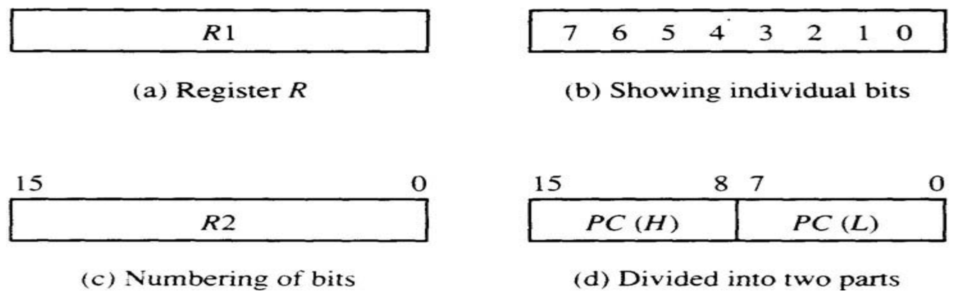
- The symbolic notation used to represent micro operations called a *register transfer language*

Register Transfer

- Designate computer registers by capital letters to denote its function
- The register that holds an address for the memory unit is called MAR
- The program counter register is called PC
- IR is the instruction register and R1 is a processor register
- The individual flip-flops in an *n*-bit register are numbered in sequence from 0 to *n*-1

Figure 4.1 for the different representations of a register

Figure 4-1 Block diagram of register.



Designate information transfer from one register to another by

R2 ← R1//Content of register R1 is transferred to R2

Control function:

The control function is a Boolean variable either o or 1.If the transfer is to occur only under a predetermined control condition, designate it by

If (P = 1) *then* (R2 ← R1)

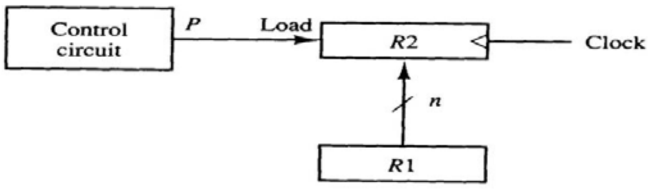
(or)

P: R2 ← R1

Where P is a control function that can be either 0 or 1.

Every statement written in register transfer notation implies the presence of the required hardware construction

Figure 4-2 Transfer from $R1$ to $R2$ when $P = 1$.



(a) Block diagram

- It is assumed that all transfers occur during a clock edge transition
- To represent more than one micro operation micro operations are separated by comma(,)
- All micro operations written on a single line are to be executed at the same time

T: R2 ← R1, R1 ← R2

TABLE 4-1 Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	*R2* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

Assembly Language Notation: This is another type of notation used to represent machine instructions and

programs

Eg:

Assembly Language Format	**Description**
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Based on number of address fields used the instructions are categorized into 4 types

1) Three address instructions
2) Two address instructions
3) One address instructions

4) Zero address instructions

Execution of C=A+B in four types of instructions is as shown bellow

Instruction Type	**Syntax**	**Eg**	**Explanation**	**Description**
Three Address	Operation Source1,Source2,Destination	Add A,B,C	C ← [A]+[B]	Add values of variable ,B & place the result into C.
Two Address	Operation Source, Destination	Add A,B	B ← [A]+[B]	Add the values of A,B place the result into B.
One Address	Operation Operand	Add B	AC ← [AC]+[B]	Content of accumulator add with content of B and store t result in Accumulator.

Zero address	---	PUSH A
		PUSH B
		ADD
		POP C
			TOS ← [A]
			TOS ← [B]
			TOS ← [A]+[B]
			C ← [TOS]
				Stack is used to store operands, result

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

<u>THREE-ADDRESS INSTRUCTIONS</u>

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

```
ADD    A, B, R1     R1 ← M [A] + M [B]
ADD    C, D, R2     R2 ← M [C] + M [D]
MUL    R1, R2, X    M [X] ← R1 ∗ R2
```

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

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.
The **disadvantage** is that the binary-coded instructions require too many bits to specify three addresses.

<u>TWO-ADDRESS INSTRUCTIONS</u>
Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) ∗ (C + D) is as follows:

```
MOV    A, R1      R1 ← M [A]
ADD    B,R1       R1 ← R1 + M [B]
MOV    C,R2       R2 ← M [C]
ADD    D,R2       R2 ← R2 + M [D]
```

```
MUL    R2,R1           R1 ← R1*R2
MOV   R1,X             M [X] ← R1
```

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

ONE-ADDRESS INSTRUCTIONS

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

```
LOAD     A      AC ← M [A]
ADD      B      AC ← A [C] + M [B]
STORE    T      M [T] ← AC
LOAD     C      AC ← M [C]
ADD      D      AC ← AC + M [D]
MUL      T      AC ← AC ∗ M [T]
STORE    X      M [X] ← AC
```

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

ZERO-ADDRESS INSTRUCTIONS

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

```
PUSH     A      TOS ← A
PUSH     B      TOS ← B
ADD             TOS ← (A + B)
PUSH     C      TOS ← C
PUSH     D      TOS ← D
ADD             TOS ← (C + D)
MUL             TOS ← (C + D) ∗ (A + B)
POP      X      M [X] ← TOS
```

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

ADDRESSING MODES

The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

Generic Addressing Modes:

- Implied addressing mode

- Immediate addressing mode
- Register addressing mode
- Register Indirect addressing mode
- Direct addressing mode
- Indirect addressing mode
- Relative addressing mode
- Index addressing mode
- Base register addressing mode
- Auto-increment &Auto-decrement addressing mode

Implied Addressing mode:
It is also called inherent addressing mode.
• The operand is implied by the instruction.
• The operand is hidden/fixed inside the instruction.
Eg: Complement Accumulator CMA (Here accumulator A is implied by the instruction)
Complement Carry Flag CMC (Here Flags register is implied by the instruction)
Set Carry Flag STC (Here Flags register is implied by the instruction)

Immediate Addressing mode:
The operand is specified within the instruction.
Operand itself is provided in the instruction rather than its address.
Eg:Move Immediate MVI 15h,A A ← 15h Here 15h is the immediate operand

Register addressing Mode:
The operand is specified within one of the processor register.
Instruction specifies the register in which the operand is stored.
Eg: Move MOV R1,R2 R2 ← R1 Here A is the operand specified in register
Register indirect addressing mode:
The instruction specifies the register in which the memory address of operand is placed.
• It do not specify the operand itself but its location within the memory where operand is placed.
Eg:Move MOV [R2], R1 // R1 ← [R2] It moves the data from memory location specified
by R2 register to R1

Direct addressing mode:
The instruction specifies the direct address of the operand.
• The memory address is specified where the actual operand is.
Eg:Load Accumulator LDA 2805h // A ← [2805] It loads the data from memory location 2805 to A
Indirect addressing mode:
The instruction specifies the indirect address where the effective address of the operand is placed
• The memory address is specified where the actual address of operand is placed.

Eg:Move MOV 2802h , A // A ← [[2802]] It moves the data from memory location specified by the location 2802 to A

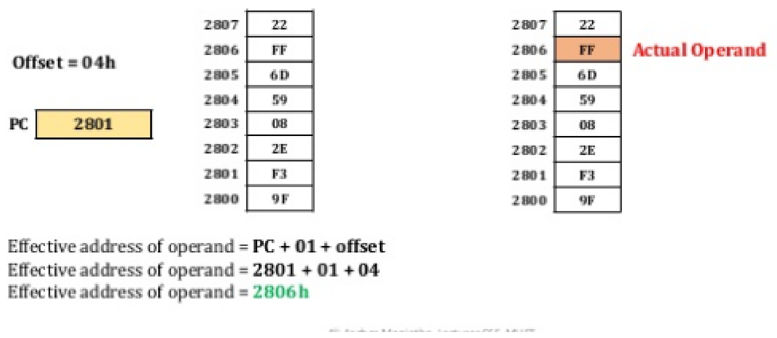
Relative addressing mode:

In relative addressing mode, contents of Program Counter PC is added to address part of instruction to obtain effective address.

• The address part of the instruction is called as offset and it can +ve or –ve.

• When the offset is added to the PC the resultant number is the memory location where the operand will be placed.

Effective address of operand = PC + 01 + offset



Effective address of operand = **PC + 01 + offset**
Effective address of operand = **2801 + 01 + 04**
Effective address of operand = **2806h**

Index addressing mode,: Here the contents of Index register is added to address part of instruction to obtain effective address.

• The address part of instruction holds the beginning/base address and is called as base.

• The index register hold the index value, which is +ve.

• Base remains same, the index changes.

• When the base is added to the index register the resultant number is the memory location where the operand will be placed

Eg:ADD R1,XR[50]
Effective address=Content of index register XR+50

Base register addressing mode

In base register addressing mode, contents of base register is added to address part of instruction to obtain effective address.

• It is similar to the indexed addressing mode except the register now is called as base instead of index.

• The base register hold the beginning/base address.

• The address part of instruction holds the offset.

• Offset remains same, the base changes.

• When the offset is added to the base register the resultant number is the memory location where the operand will be placed.

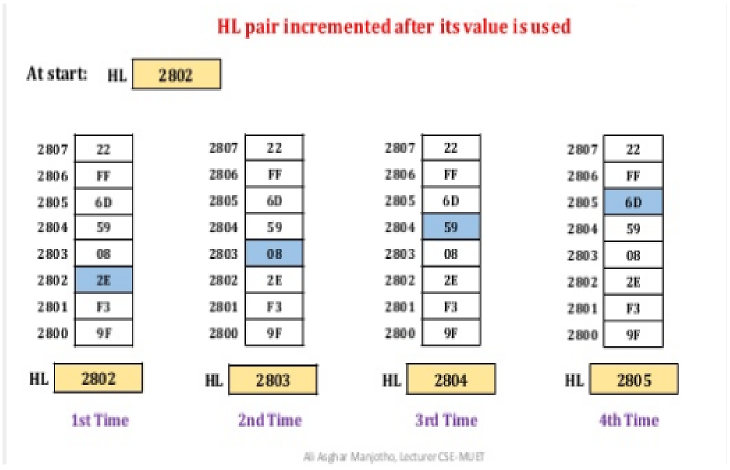
Eg:ADD R1,BR[50]

Effective address=Content of index register XR+50

Auto-increment &Auto-decrement addressing mode:

It is similar to register indirect addressing mode. • Here the register is incremented or decremented before or after its value is used

Eg:



Basic Input Output Operations:

There are three ways by which the data transfer between memory and I/O Devices happens.

1) Program controlled I/O
2) Memory Mapped I/O
3) Isolated I/O

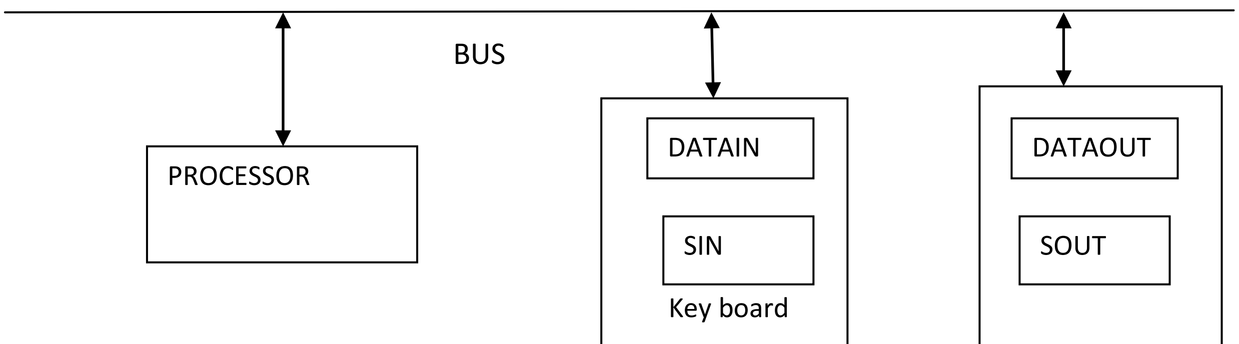
Program Controlled I/O:

In this, the processor involves continuously in the transfer of every byte between memory and I/O Devices.

Every input output devices has two components. They are 1) Buffer 2) Flag register

Buffer register: It is used to hold the contents temporary.
Flag register: It is used to indicate whether the device is ready or not.

The following diagram shows the inter connection among processor, keyboard and monitor.



The action of striking a key does not automatically cause the corresponding character to be displayed on the screen .Instead there is a block of instructions in the IO program which transfer a character into the processor and another block of instruction causes the character to be displayed on the screen.

Striking a key transfers corresponding character code into DATAIN register (buffer) of the keyboard. The Flag register SIN is set to 1 to indicate that valid character is available in DATAIN .When the character is transferred to processor it is reset to 0.when the next character is ready in the DATAIN, it is again set to 1.

The same process takes place when the character are transferred from processor to monitor. When SOUT equals to 1 the device is ready to receive a character from the processor. When SOUT is '1' the processor sends the character to monitor. After this SOUT is set to '0'.the SOT is set to '1' ,when the monitor is ready to receive the character from the processor.

Memory mapped I/O vs Isolated I/O:

ISOLATED I/O	MEMORY MAPPED I/O
Memory and I/O have seperate address space	Both have same address space
All address can be used by the memory	Due to addition of I/O addressable memory become less for memory
Separate instruction control read and write operation in I/O and Memory	Same instructions can control both I/O and Memory
In this I/O address are called ports.	Normal memory address are for both
More efficient due to seperate buses	Lesser efficient
Larger in size due to more buses	Smaller in size
It is complex due to separate separate logic is used to control both.	Simpler logic is used as I/O is also treated as memory only.

The role of stacks and Queues in Computer Systems

Stack:

A stack is a list of data elements with the accessing restriction that elements can be added or removed at only one end this end is called "Top" of stack.

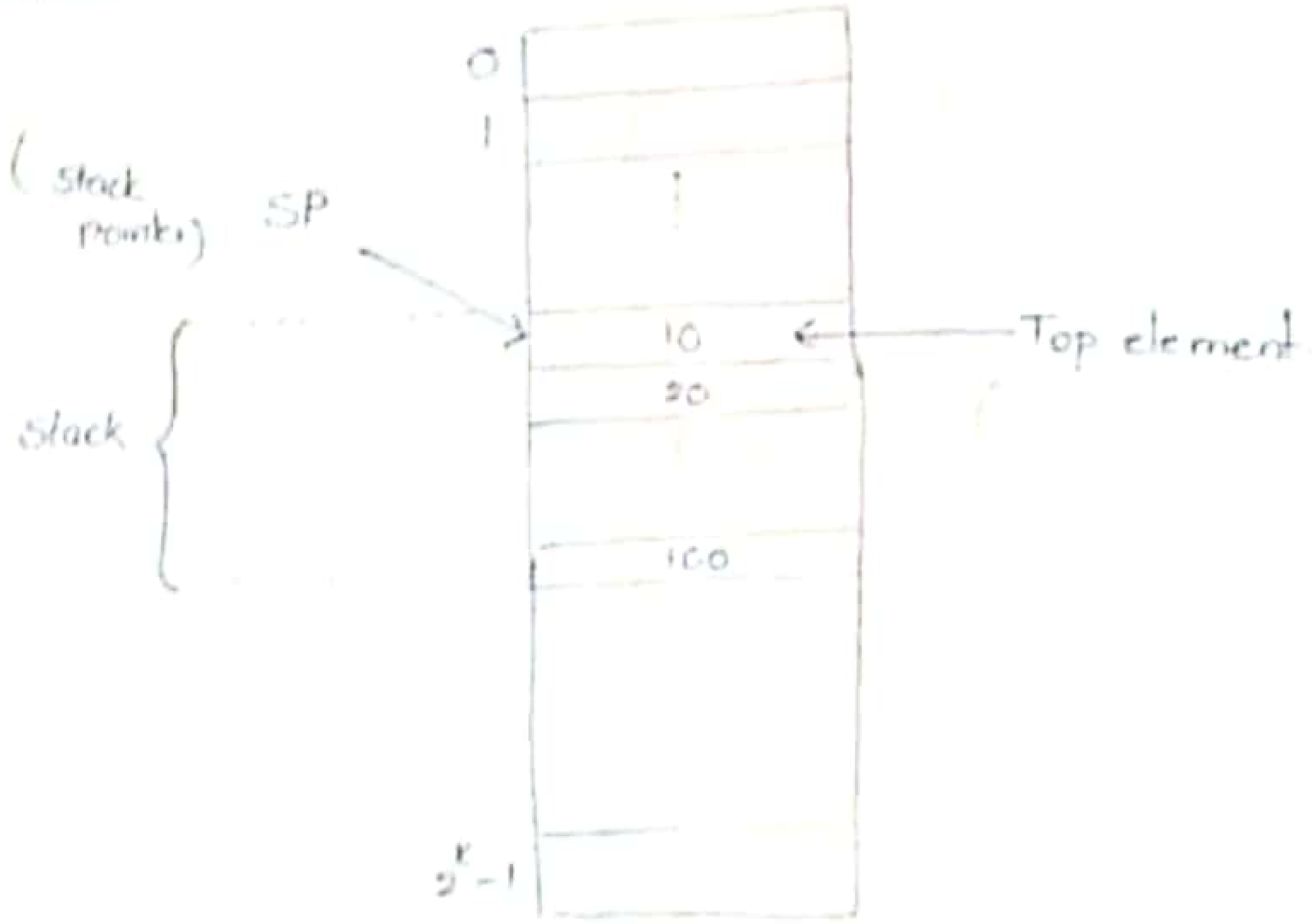
It has a characteristic called "LIFO" (Last In First Out) The operations that are performed on stack are (i) Push (ii) Pop

Push is used to insert the element into the stack.

Pop is used to delete the element out of the stack.

The stack has a special purpose register called as "Stack Pointer" simply refered to as SP SP is used to keep track of the address of the top element of the stack

The location of stack in the main memory is visualized as follows



Implementation of PUSH and POP operations

Let us make an assumption that each location takes 4 bytes of memory.

The "push" operation can be implemented as follows:

```
PUSH : Subtract #4, SP
       Move NEWITEM, (SP)
```

In the above, the "subtract" operation subtract the value of SP by 4 so that it points to newly inserted data item and the new data item ~~is~~ is inserted by "Move" operation.

The "POP" operation can be implemented as follows.

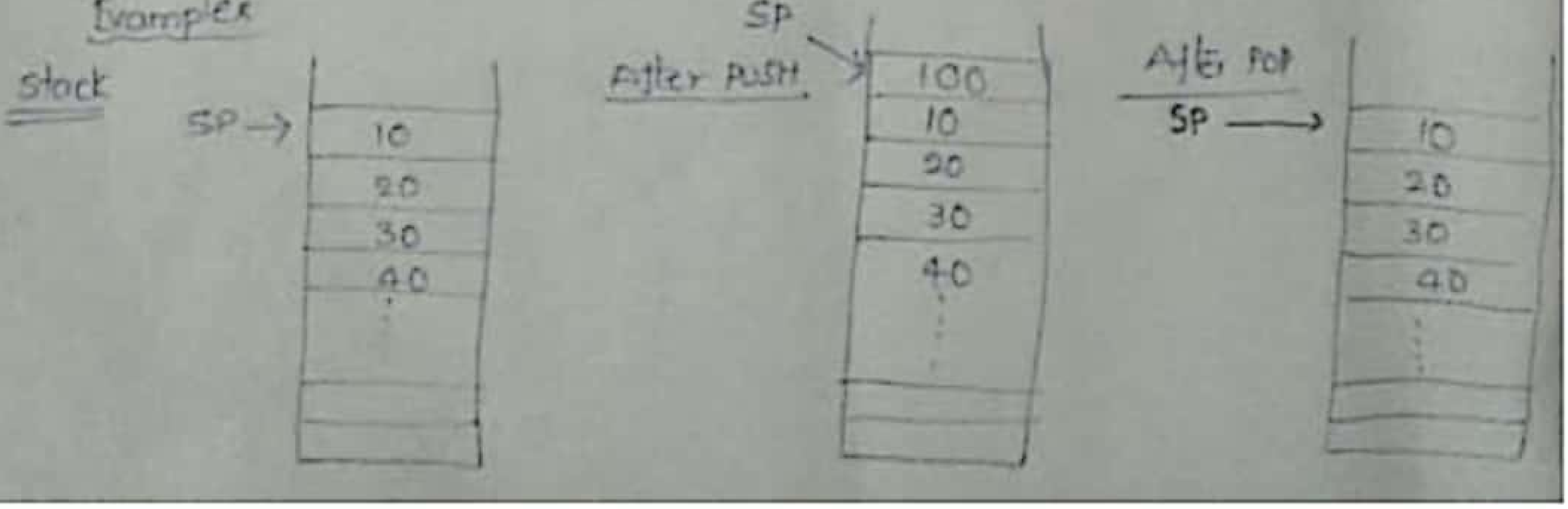
```
POP : Move (SP), NEWITEM
      Add #4, SP
```

The above two operations move the top value into NEWITEM and increments the SP by 4 so that it now points to the new "TOP" element.

If the processor has Auto increment and Autodecrement modes, the push operation and pop operations can be implemented by single instruction each.

```
PUSH : Move NEWITEM, -(SP)
POP :  Move (SP)+, NEWITEM
```

Examples



Implementation of SAFE PUSH and Safe POP

when a stack is used in the program, it is usually allocated a fixed amount of space in the memory. when the element is pushed to stack, the SAFEPUSH operation checks whether there exists atleast ~~code~~ one location available. when the element is taken out of the stack, the SAFEPOP operation checks whether there exists atleast one element available.

Let us make an assumption that our stack ~~ab~~ spawns from 2000 to 1500 locations. According to this, if the stack pointer SP contains ~~less~~ greater than 2000, it indicates that the STACK IS EMPTY. If the SP contains less than 1500, it indicates that the STACK IS FULL

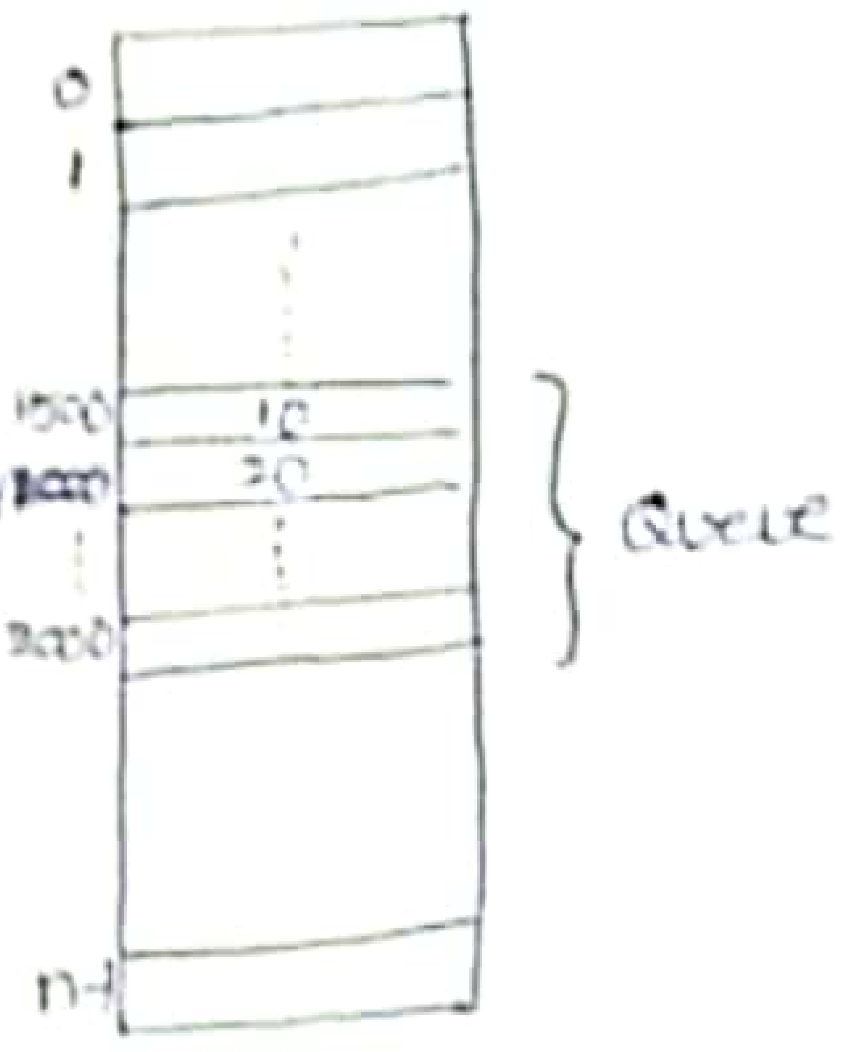
```
SAFEPUSH :  Compare #2000, SP
            Branch >0  EMPTYERROR
            MOVE (SP)+, ITEM

SAFEPOP :   Compare #1500, SP
            Branch ≤ 0  FULLERROR
            Move  NEWITEM, -(SP)
                                        X
```

```
SAFEPUSH : CMP #1500, SP
           Branch ≤ 0  FULL_ERROR
           Move NEWITEM, -(SP)
SAFEPOP :  CMP #2000, SP
           Branch > 0  EMTPY_ERROR
           Move (SP)+, ITEM.
```

Queue: A Queue is an ordered collection of elements in which an element first inserted is the element first removed out.

It has two components. They are (i) front
(ii) rear

* The element "front" is used to delete the elements from the queue. The element "rear" is used insert the elements into the queue.



Method 1

```
ENQUEUE : Add #4, rear
          Move NEWITEM, (rear)

DEQUEUE : Move (front), ITEM
          Add #4, front.
```

Method 2

```
SAFEENQUEUE : CMP #2000, rear
              BEQ FULLERROR
              Add #4, rear
              Move NEWITEM, (rear)

SAFEDEQUEUE : CMP rear+4, #front
              BEQ EMPTYERROR
              Add #4, front
              Move (front), ITEM.
```

Logic Instructions

The logic operations such as AND, OR, and NOT are applied to individual bits

AND. The ouput of AND operation is "1" only when both inputs are "1" otherwise "0".

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Example AND R1, R2 gives output 0000 if
R1 = 0101 R2 = 1010

OR The output of OR operation is "0" only when both inputs are "0" otherwise "1"

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Example OR R1, R2 gives output 1111
if R1 = 0101 R2 = 1010

NOT The output of NOT operations the Complement of its input.

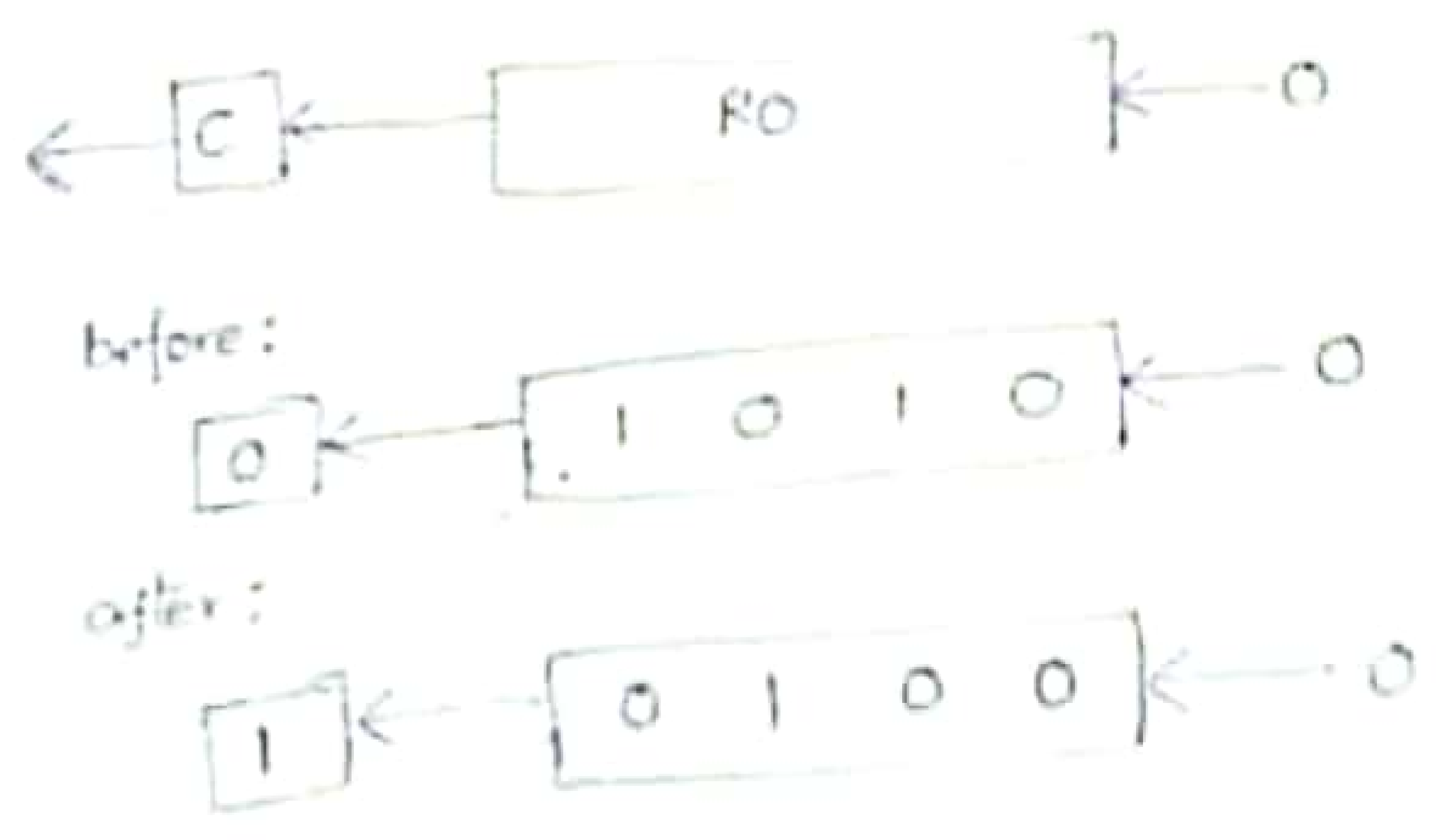
A	NOT A
0	1
1	0

Example NOT R1 gives output 1010
if R1 = 0101.

Shift and Rotate Instructions

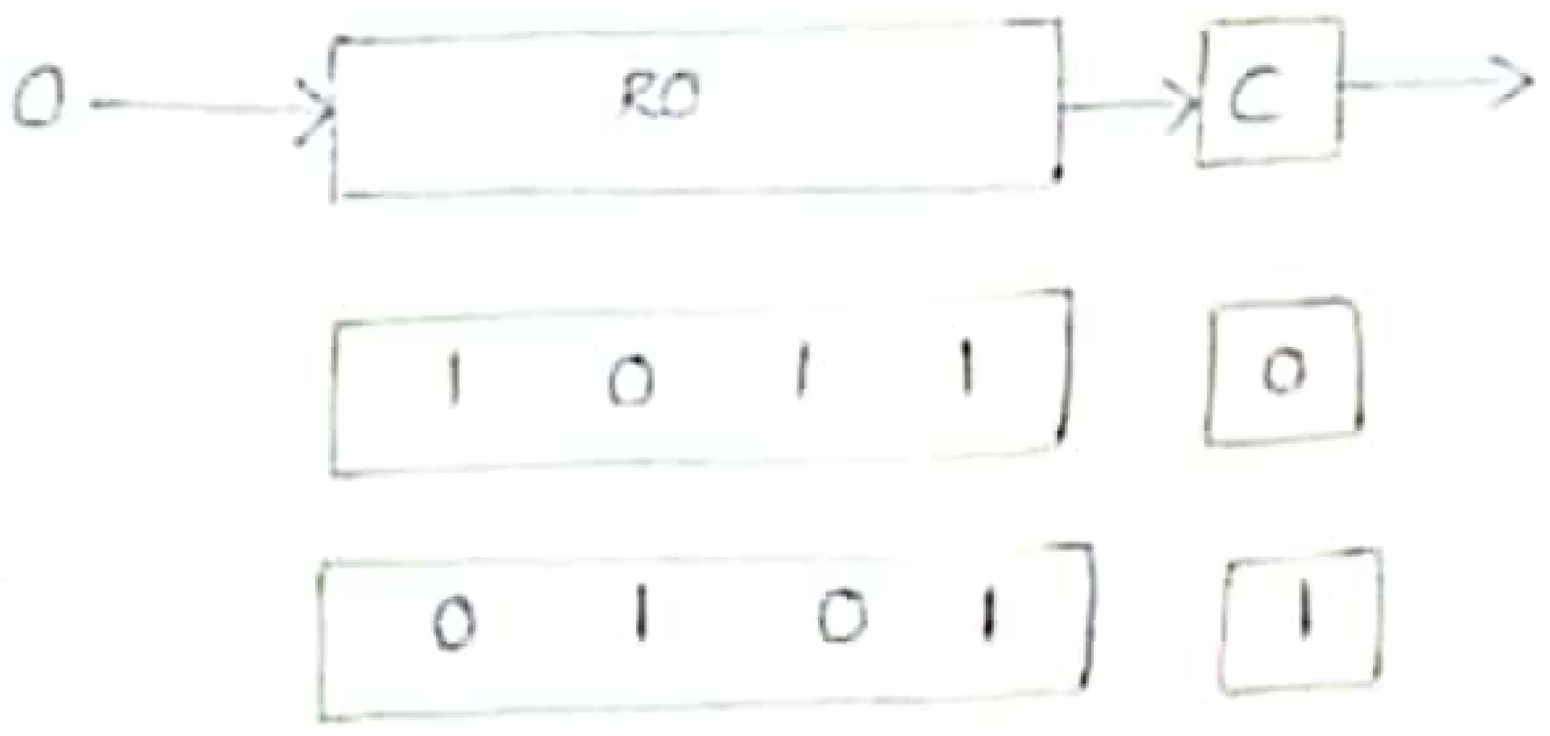
There are 4 types of shift operations they are as foll
(i) LShiftL (ii) LShiftR (iii) AShiftL (iv) AShiftR

LShiftL It multiplies the given number by "2" by shifting the bits in left direction.



Example
LShiftL R1, #2
gives the output 0100
if R1 contains 0001

LShiftR It divides the given number by "2" by shifting the bits in right direction.

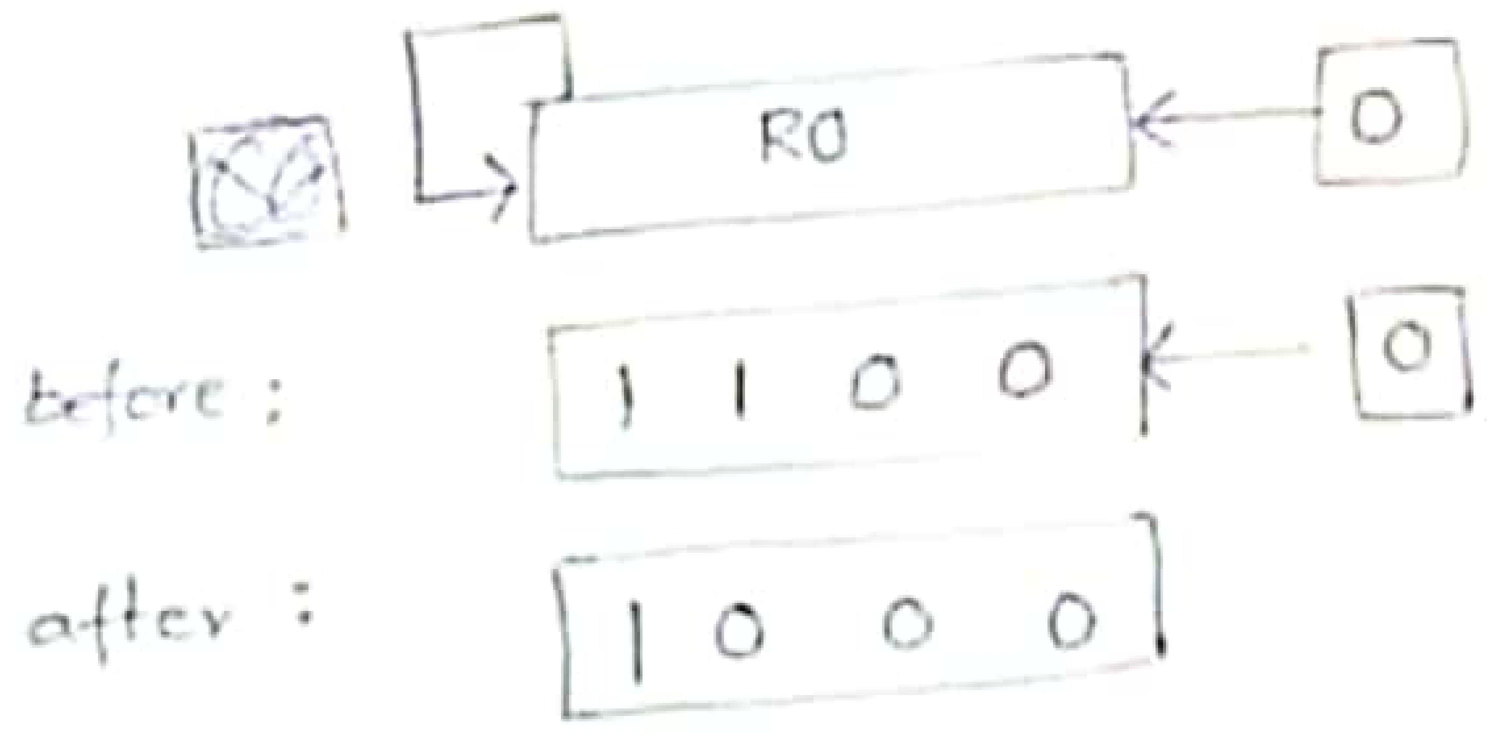


Example LShiftR R0,#1 gives the output 0101 if R0 contains 1011

AShiftL

Arithmetic shift operations not only work on unsigned numbers but also works on signed numbers.

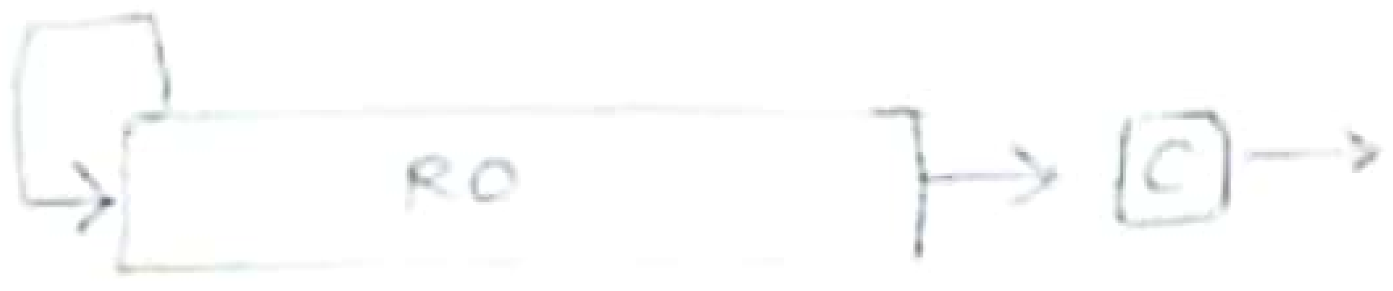
AShiftL operation is used to shift the rightmost (n-1) bits in the left direction and rightmost vacant bit is filled with "0".

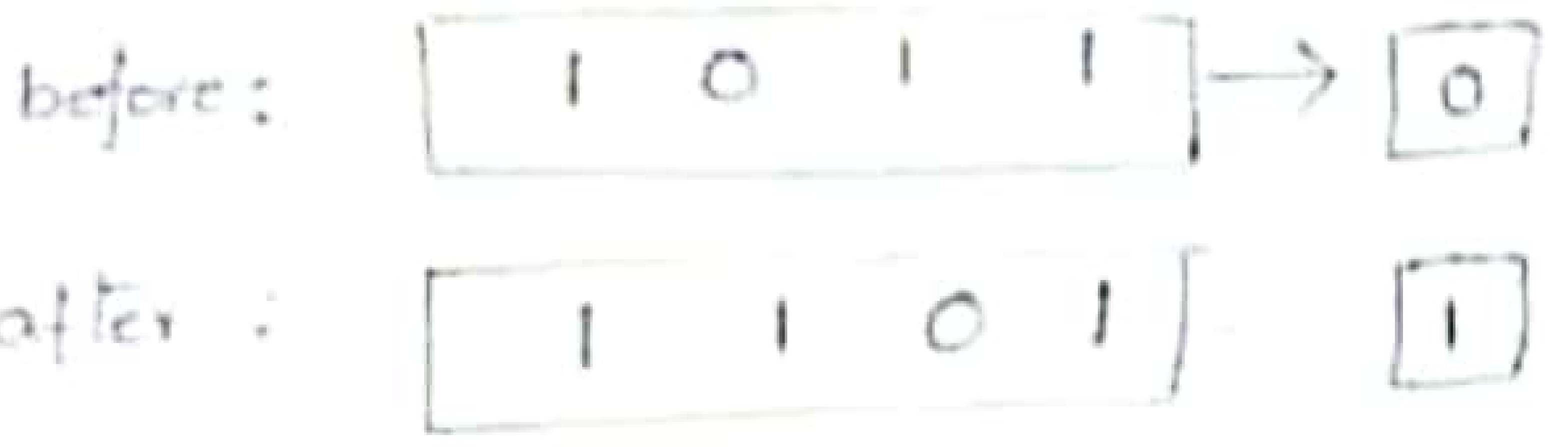


Example AShiftL R0,#1 produces the output 1000 (-8) if the R1 contains 1100 (-4)

AShiftR

It is used to shift the leftmost (n-1) bits in right direction and the leftmost vacant bit is filled with previous sign bit.





Example AShiftR #1, R0 produces the $\overset{out}{-}$put 1101 if R0 contains 1011.

Rotate operations

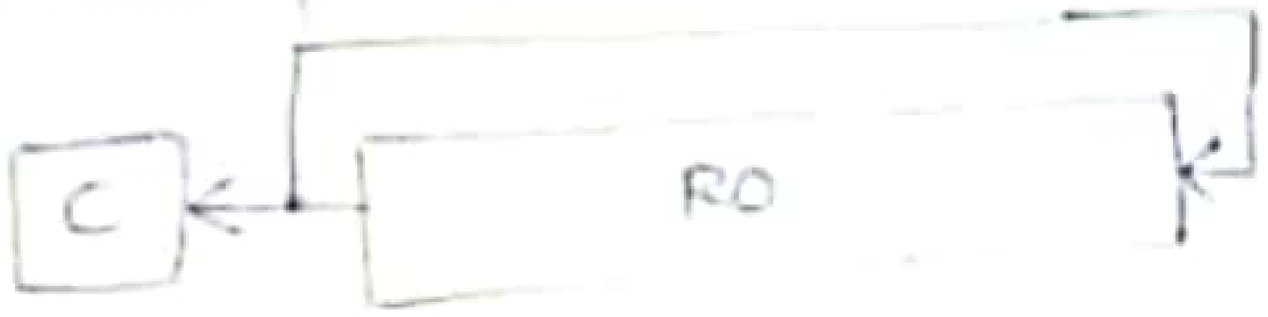
There are 4 types of rotate operations. they are as follows.

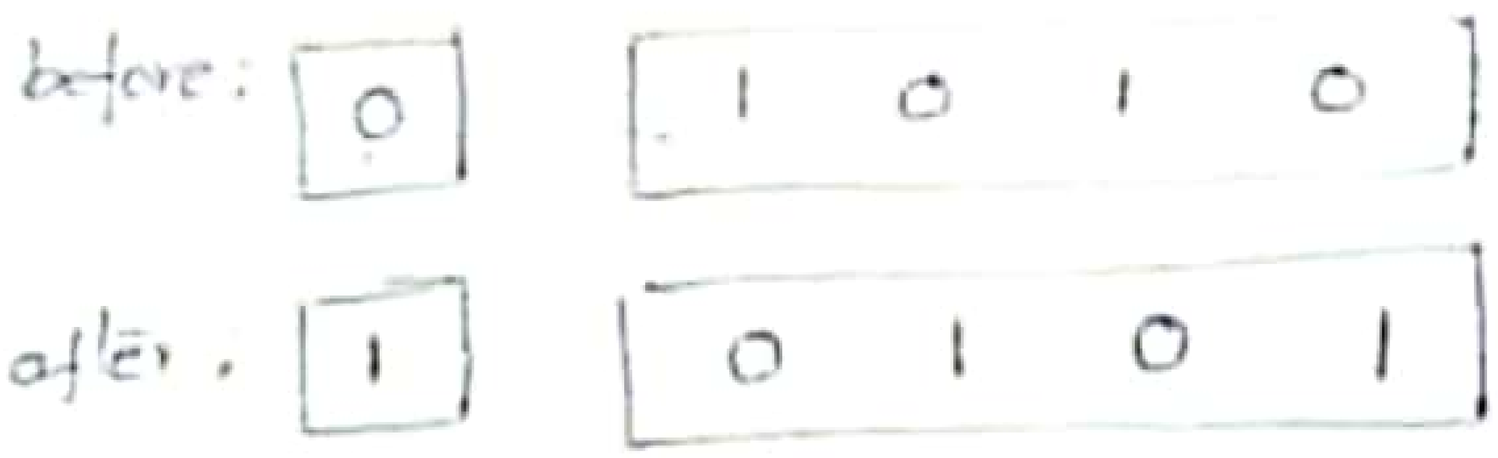
(i) RotateL (ii) RotateR (iii) RotateLC (iv) RotateRC

In all these operations, we use a special bit called carry bit. It is used to retain the value of bit which is lost when shifting

RotateL (Rotate Left without Carry)

Example Rotate #1, R0

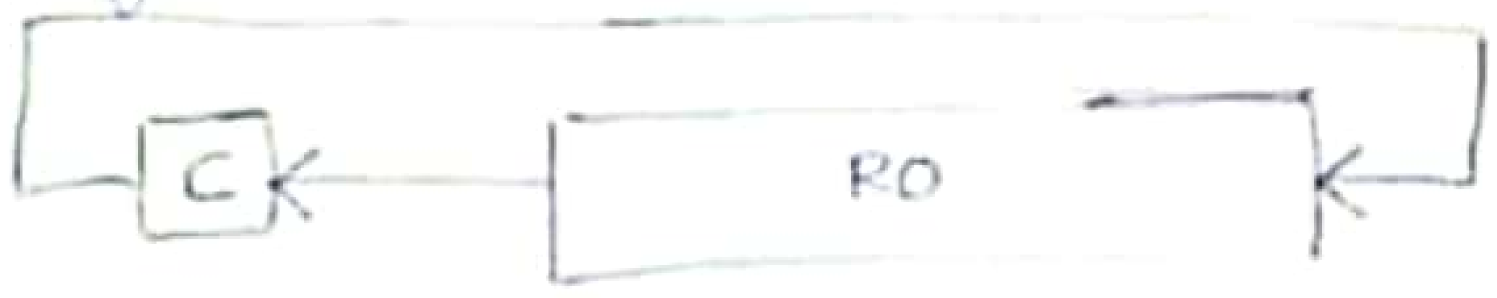




In this, when the bits are shifting in ~~right~~ left direction the rightmost vacant bit is filled with previous leftmost bit and it is retained in Carry bit.

RotateLC (Rotate left with Carry)

In this, when the bits are shifting in left direction, the rightmost ~~a~~ vacant bit is filled with Carry bit and then the carry bit now contains the leftmost bit.



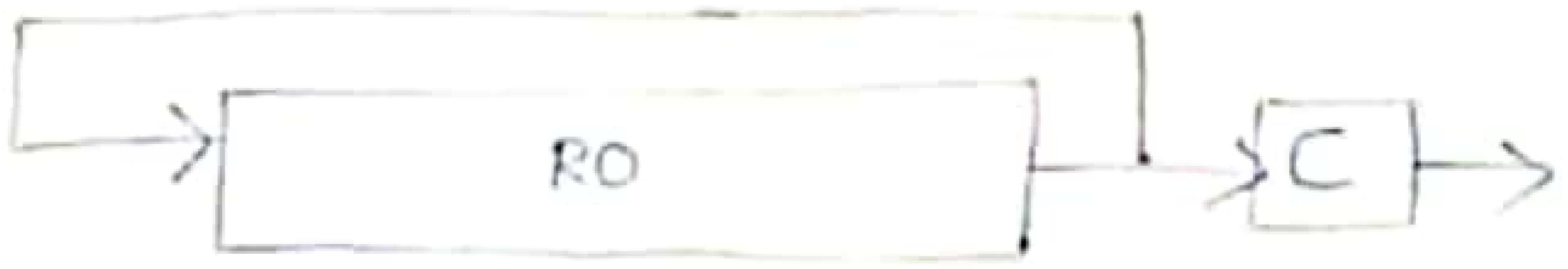
Example

RotateLC #1, R0

	C	R0
before:	0	1 0 1 0
after:	1	0 1 0 0

RotateR (Rotate Right without Carry)

In this, when the bits are shifting in right direction, the leftmost vacant bit ~~of~~ is filled with previous rightmost bit.



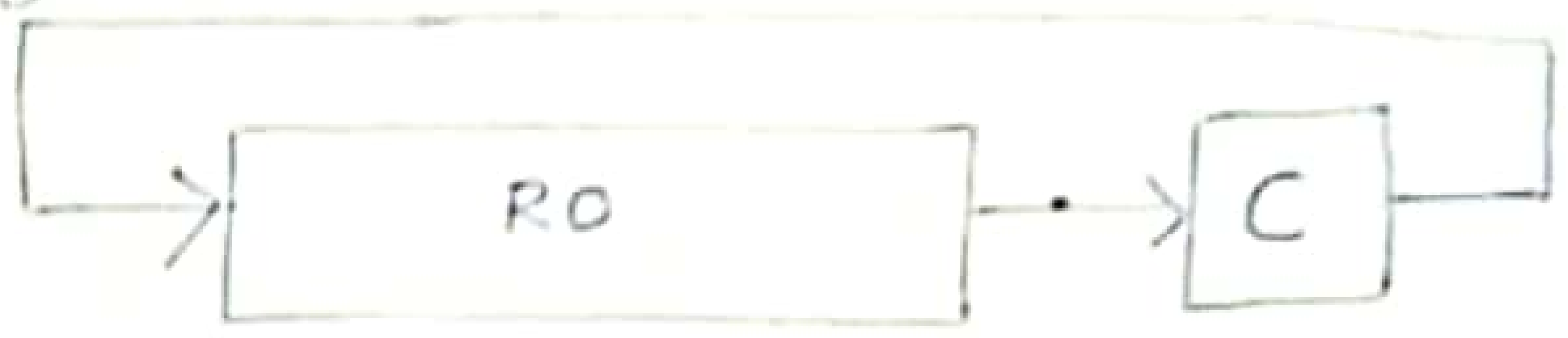
Example

RotateR #1, R0

	R0	C
before:	1 0 1 0	0
after:	0 1 0 1	0

RotateRC (Rotate Right With Carry)

In this, when the bits are shifting in right direction, the leftmos vacant bit is filled with Carry bit and then the carry bit now contains rightmost bit.



Example

RotateRC #1, R0

before : $\boxed{1\ \ 0\ \ 1\ \ 0}$ $\boxed{0}$

after : $\boxed{0\ \ 1\ \ 0\ \ 1}$ $\boxed{0}$

UNIT -3
TYPES OF INSTRUCTIONS

Arithmetic instructions:

There are so many arithmetic Instructions, in ARM. The basic Instruction format is as follows.

OP code Rd, R_n, R_m

Examples for Arithmetic instructions.

(i) ADD R0, R2, R4 performs the operation R0 ← [R2] + [R4]

(ii) SUB R0, R6, R5 performs the operation R0 ← [R6] + [R5]

(iii) ADD R0, R3, #17 performs the operation R0 ← [R3] + 17

(iv) ADD R0, R1, R5, LSL #4 performs the operations as follows:

$R0 \leftarrow [R1] + [R5] \times 2^4$

(iv) MUL R0, R1, R2 performs the following operation R0 ← [R1] * [R2]

(v) MLA (Multiply - Accumulative operation)

MLA R0, R1, R2, R3 performs the following operation as

R0 ← [R1] * [R2] + R3

Logical Instructions:

The logic operations AND, OR, XOR, and Bit-Clear are implemented by instructions with the OP codes AND, ORR, EOR, and BIC. They have the same format as the arithmetic instructions. The instruction

AND R*d*,R*n*,R*m*

performs the operation

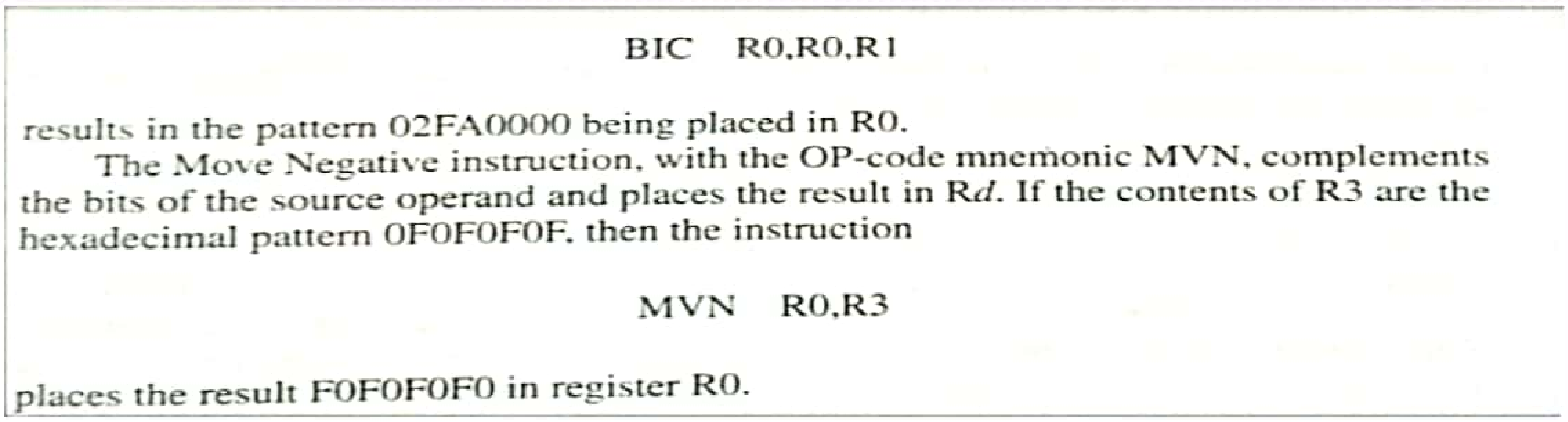
$$Rd \leftarrow [Rn] \wedge [Rm]$$

which is a bitwise logical AND between the operands in registers R*n* and R*m*. For example, if register R0 contains the hexadecimal pattern 02FA62CA and R1 contains the pattern 0000FFFF, then the instruction

AND R0,R0,R1

will result in the pattern 000062CA being placed in register R0.

The Bit-Clear instruction (BIC) is closely related to the AND instruction. It complements each bit in operand R*m* before ANDing them with the bits in register R*n*. Using the same R0 and R1 bit patterns as in the above example, the instruction



BIC R0,R0,R1

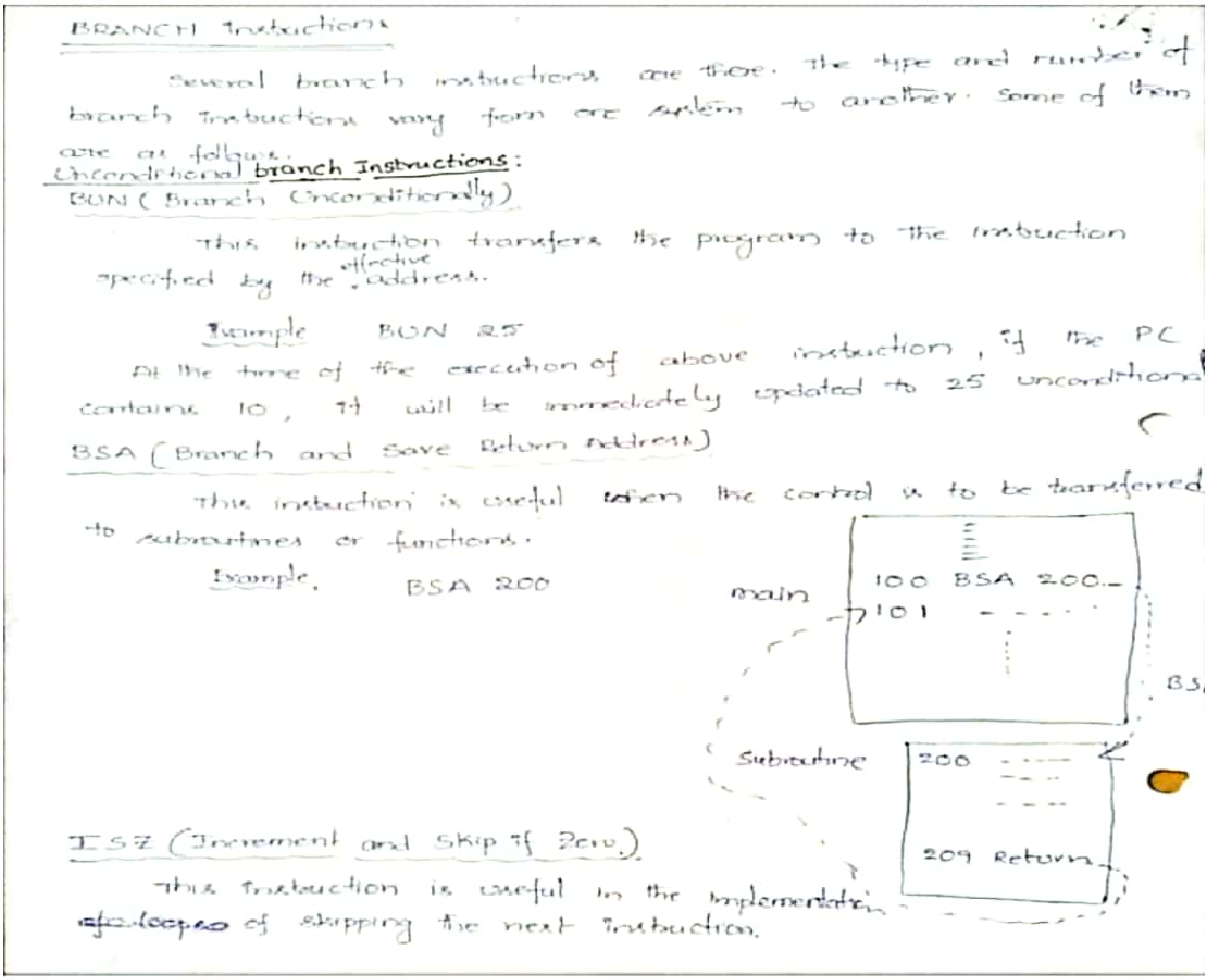
results in the pattern 02FA0000 being placed in R0.

The Move Negative instruction, with the OP-code mnemonic MVN, complements the bits of the source operand and places the result in R*d*. If the contents of R3 are the hexadecimal pattern 0F0F0F0F, then the instruction

MVN R0,R3

places the result F0F0F0F0 in register R0.

Branch instructions:



BRANCH Instructions

Several branch instructions are there. The type and number of branch Instructions vary from one system to another. Some of them are as follows.

Unconditional **branch Instructions**:

BUN (Branch Unconditionally)

This instruction transfers the program to the instruction specified by the effective address.

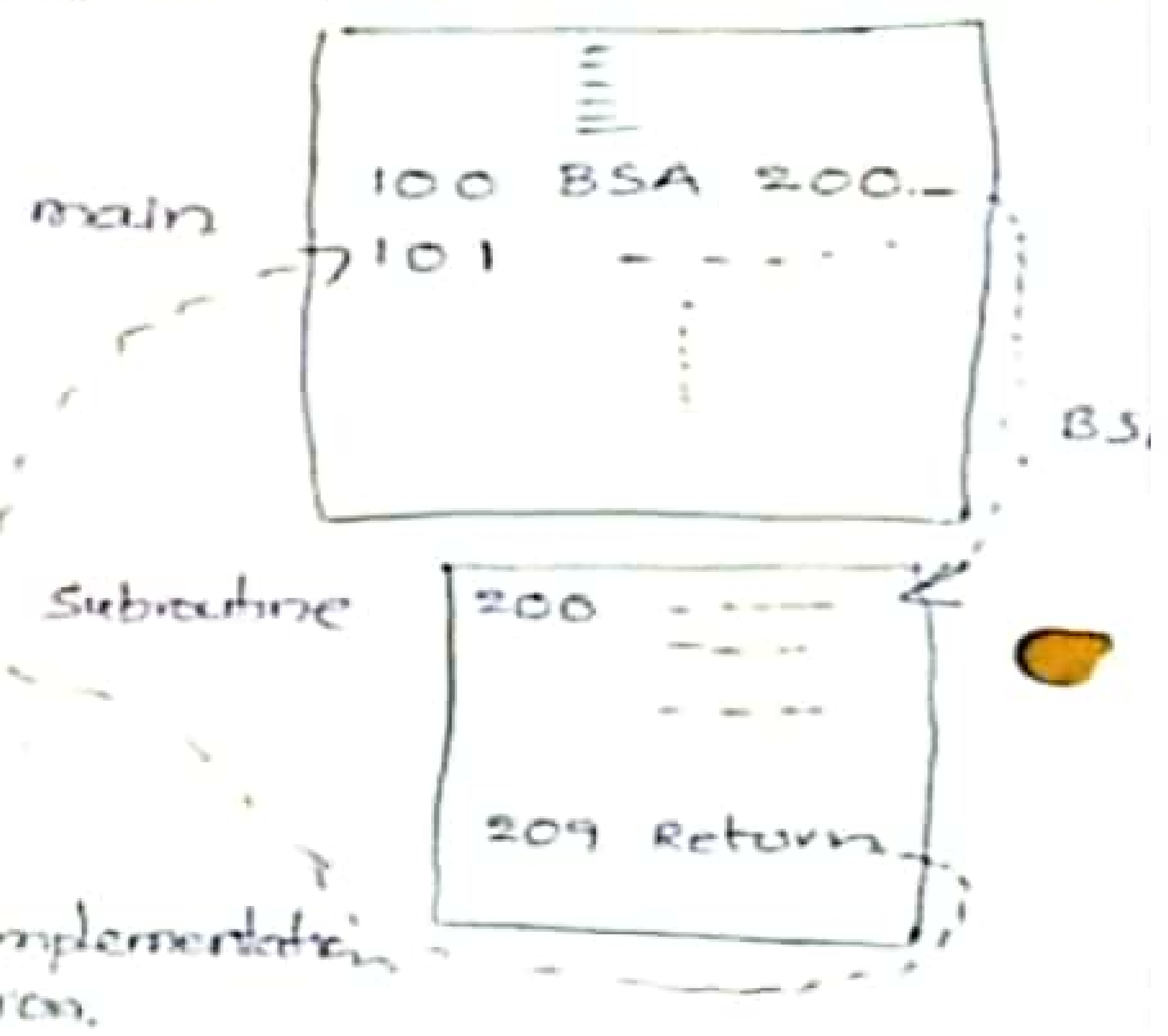
Example BUN 25

At the time of the execution of above instruction, if the PC contains 10, it will be immediately updated to 25 unconditional

BSA (Branch and Save Return Address)

This instruction is useful when the control is to be transferred to subroutines or functions.

Example. BSA 200



ISZ (Increment and Skip if Zero)

This Instruction is useful in the implementation ~~of loops~~ of skipping the next Instruction.

Conditional branch instructions

Conditional branch instructions contain a value in the instruction itself. It is used to generate the target address. whether branching should take place or not is dependent on the Condition Codes.

Condition Codes

The Condition Code flags are used to keep track of information about the results of various operations. they are used by subsequent branch instructions. these flags are usually grouped together in a special purpose register called "Condition Code Register" or "Status Register";

Individual condition code flags are set to '1' or cleared to '0' depending on results of previous operations.

there are 4 flags. They are as follows.

N (Negative) : sets to 1 if the result is negative, otherwise cleared to '0'

Z (Zero) : sets to 1 if the result is zero, otherwise cleared '0'

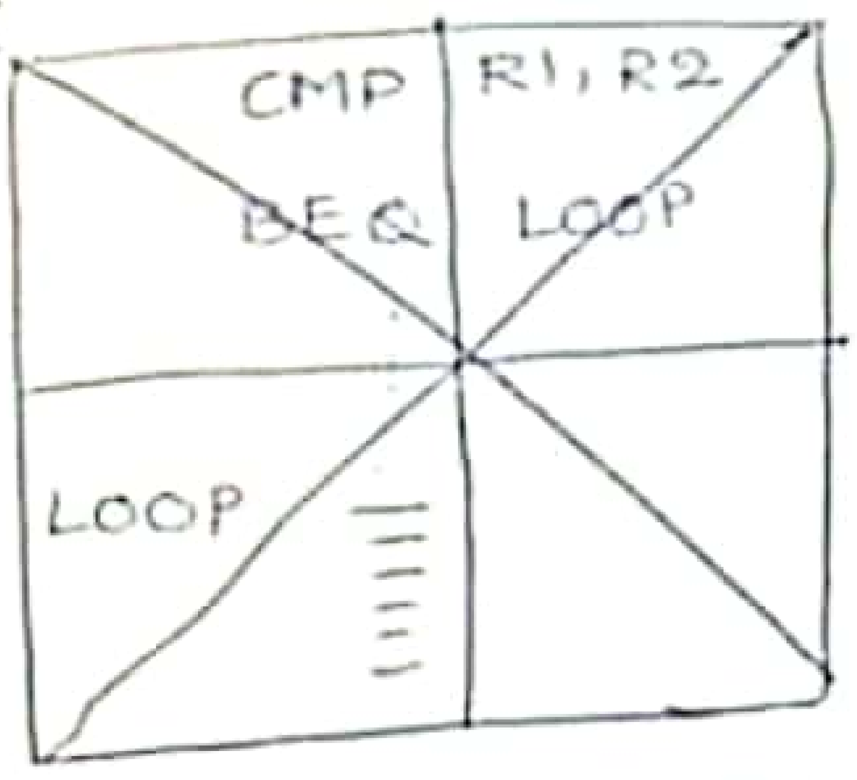
V (overflow) : sets to 1 if the arithmetic overflow occurs, otherwise '0'

C (Carry) : sets to 1 if the carry-out is '1' otherwise '0'.

BEQ (Branch if Equals to 0)

The BEQ instruction causes a branch if Z flag is set to 1.

Example



```
      SUB R1, R2
      BEQ LOOP
          :
LOOP:     :
```

If the result is '0', the control branches to LOOP.

BNZ, BPOS, BNEG, BCC, BCS, BVC, BVS
(!Z) (!N) (N) (!C) (C) (!V) (V)

Branching Examples

1)

```
if ( a > b )
        c = a;
else
        c = b;
```

~~#define a 97~~

```
    CMP a, b
    BLE L1
    MOV a, c
L1: MOV ~~b~~, ~~c~~ b, c
```

2)

```
for(i = 0; i < n; i++)
{
    :
    :
}
```

```
    CLR i
L1: CMP i, n
    BGE L2
    INC i
    BA L1
L2: :
```

Scanned with CamScanner