CHAPTER 4

CENTRAL PROCESSING UNIT

INTRODUCTION

- The part of the computer that performs the bulk of data processing operation is called central processing unit (CPU) which consists of ALU, control unit and register array.

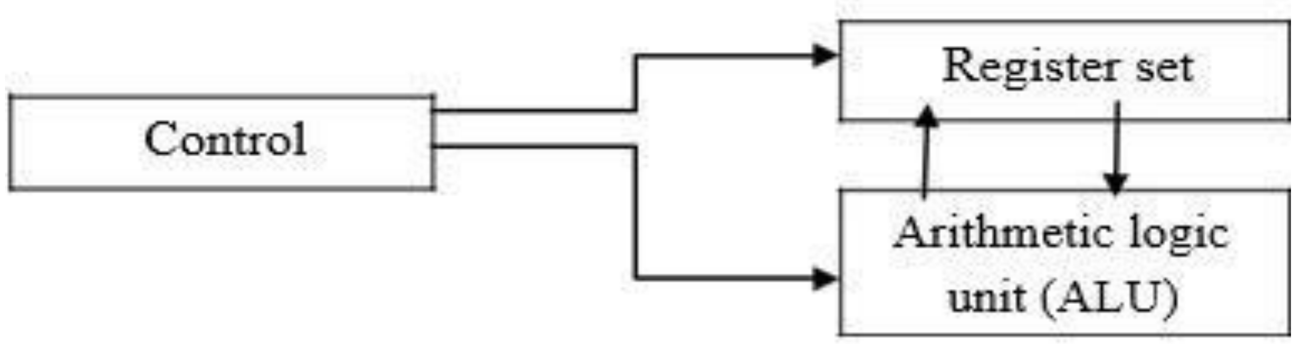


Fig: Major components of CPU

Cont.

- CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
- The register set stores intermediate data used during the execution of the instructions. The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions. The control (CU) unit supervises the transfer of information among the registers and instructs the ALU as to which operation to be performed.

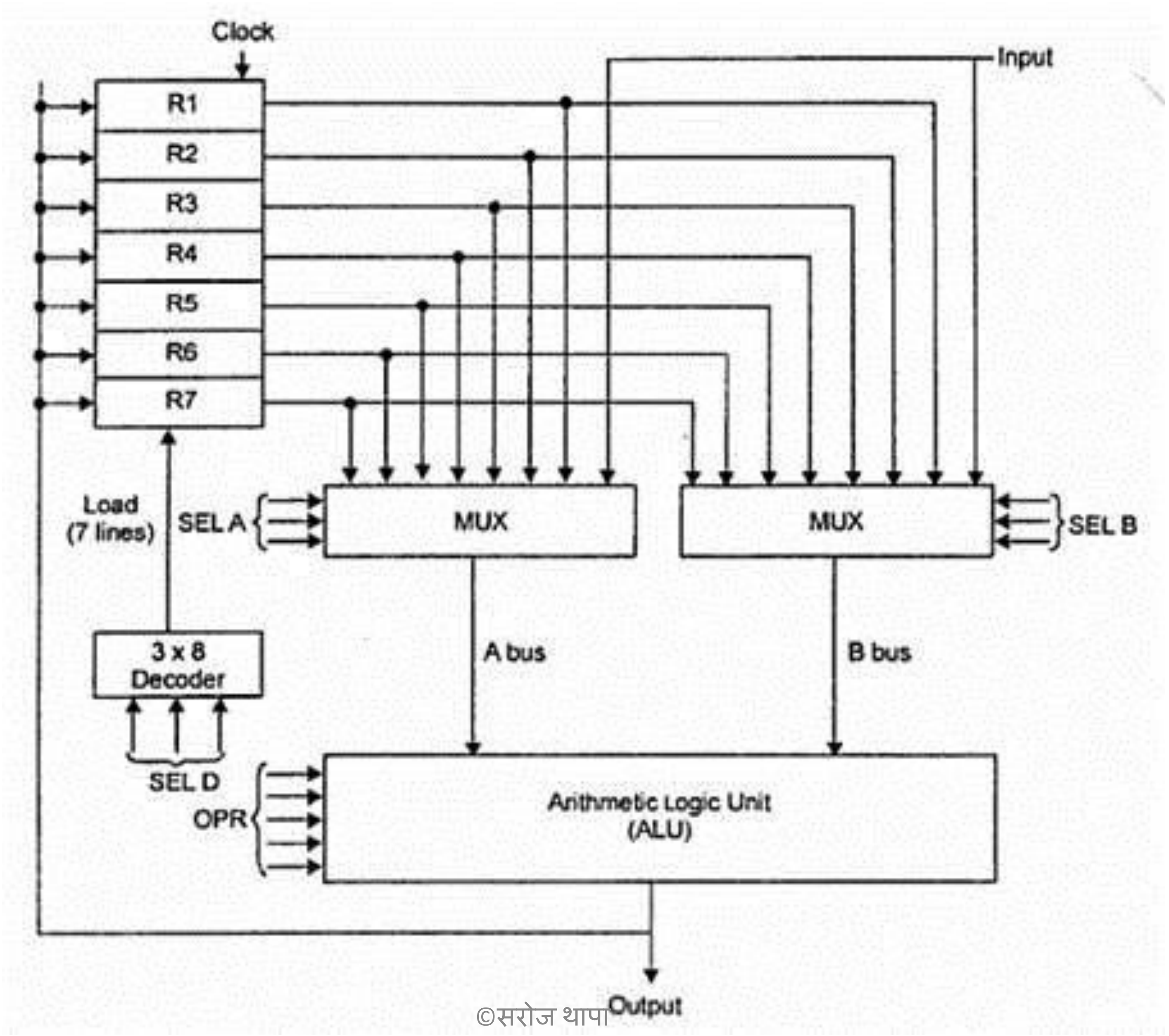
  Different organisation of CPU:

  1. *General Register Organisation*
  2. *Stack Organisation*
  3. *Single Accumulator based organisation*

General Register Organization

- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also while performing various micro operations.
- Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift micro operations in the processor.
- Generally CPU has seven general registers. Register organization show how registers are selected and how data flow between register and ALU.

- A bus organization of seven CPU registers is shown below:



Description

- A decoder is used to select a particular register.
- The output of each register is connected to two multiplexers to form the two buses A and B.
- The selection lines in each multiplexer select the input data for the particular bus.
- The A and B buses form the two inputs of an ALU. The operation select lines decide the micro operation to be performed by ALU.
- The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

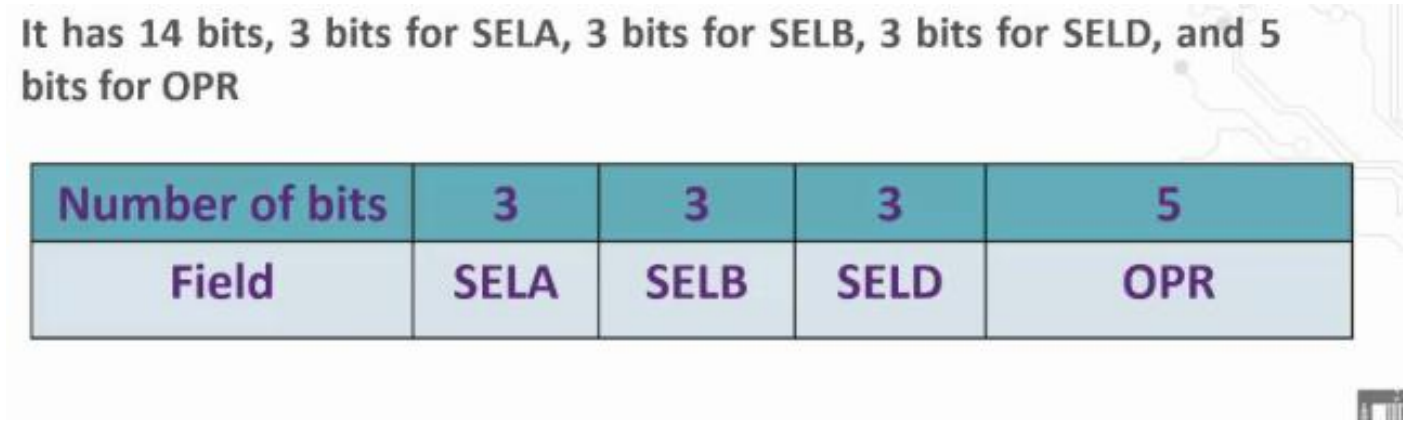
Cont..

- All registers are connected to two multiplexers (MUX) that select the registers for bus A and bus B. Registers selected by multiplexers are sent to ALU. Another selector (OPR) connected to ALU selects the operation for the ALU. Output produced by ALU is stored in some register and this destination register for storing the result is activated by the destination decoder (SELD).
- Example: $R1 \leftarrow R2 + R3$

  – MUX selector (SELA): BUS A $\leftarrow$ R2
  – MUX selector (SELB): BUS B $\leftarrow$ R3
  – ALU operation selector (OPR): ALU to ADD
  – Decoder destination selector (SELD): R1 $\leftarrow$ Out Bus

Control word:

Combination of all selection bits of a processing unit is called control word. Control Word for above CPU is as below:



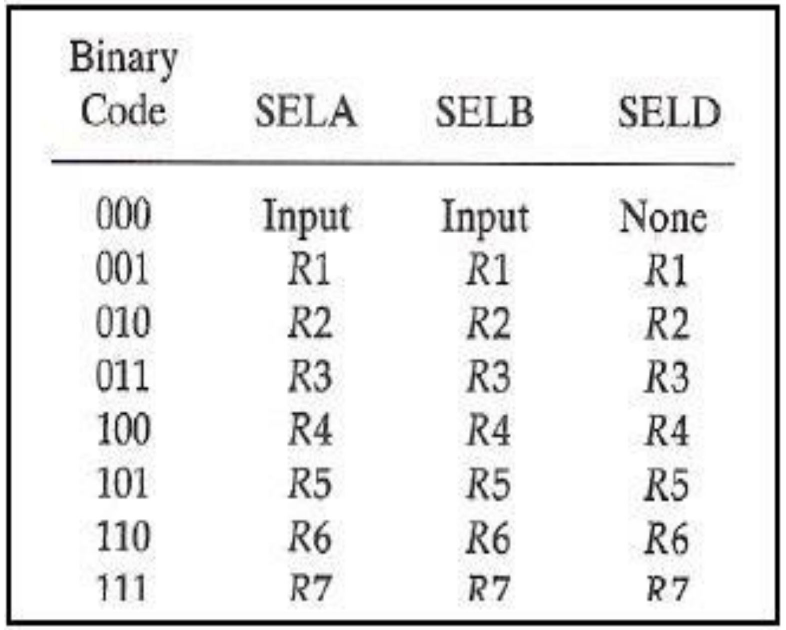
It has 14 bits, 3 bits for SELA, 3 bits for SELB, 3 bits for SELD, and 5 bits for OPR

Number of bits	3	3	3	5
Field	SELA	SELB	SELD	OPR

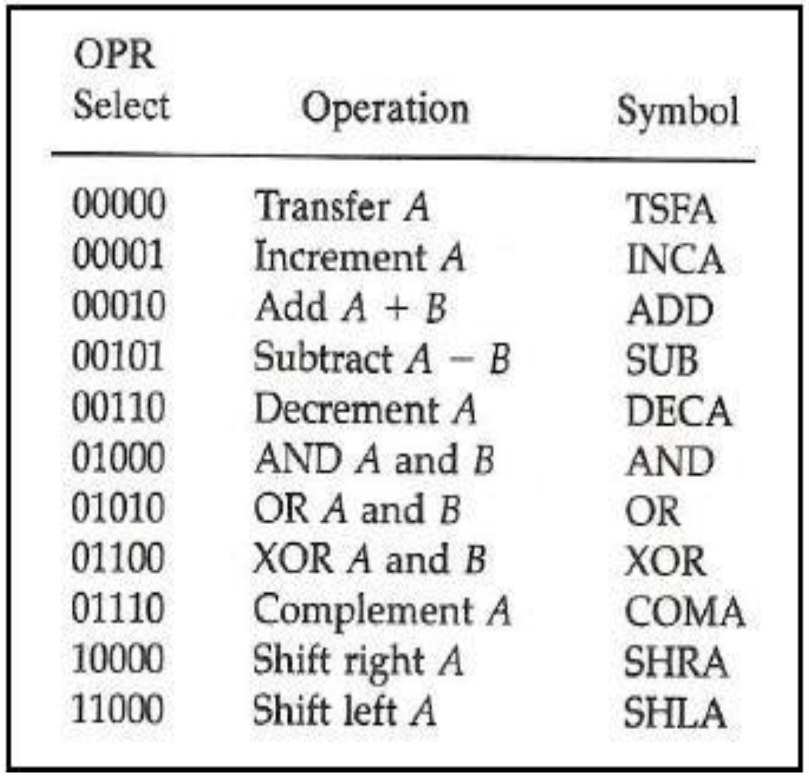
The three bit of SELA select a source registers of the **A** input of the ALU. The three bits of SELB select a source registers of the **b** input of the ALU. The three bits of SELED or SELREG select a destination register using the decoder. The five bits of SELOPR select the operation to be performed by ALU.

Cont..

- The 14 bit control word when applied to the selection inputs specify a particular microoperation. Encoding of the register selection fields and ALU operations is given below:

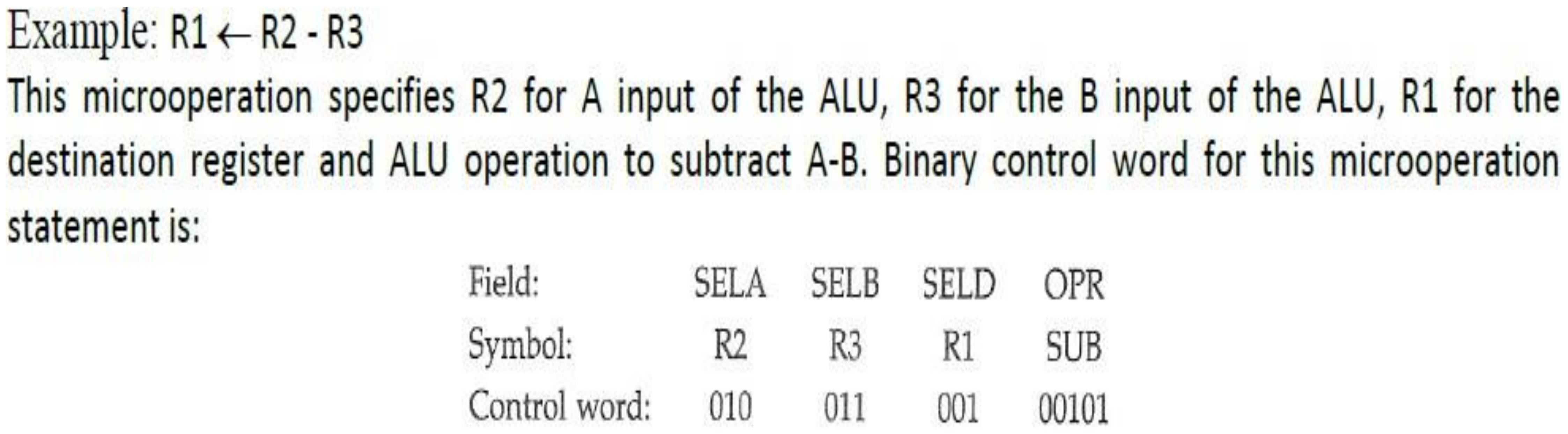


Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	$R1$	$R1$	$R1$
010	$R2$	$R2$	$R2$
011	$R3$	$R3$	$R3$
100	$R4$	$R4$	$R4$
101	$R5$	$R5$	$R5$
110	$R6$	$R6$	$R6$
111	$R7$	$R7$	$R7$



OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Cont..

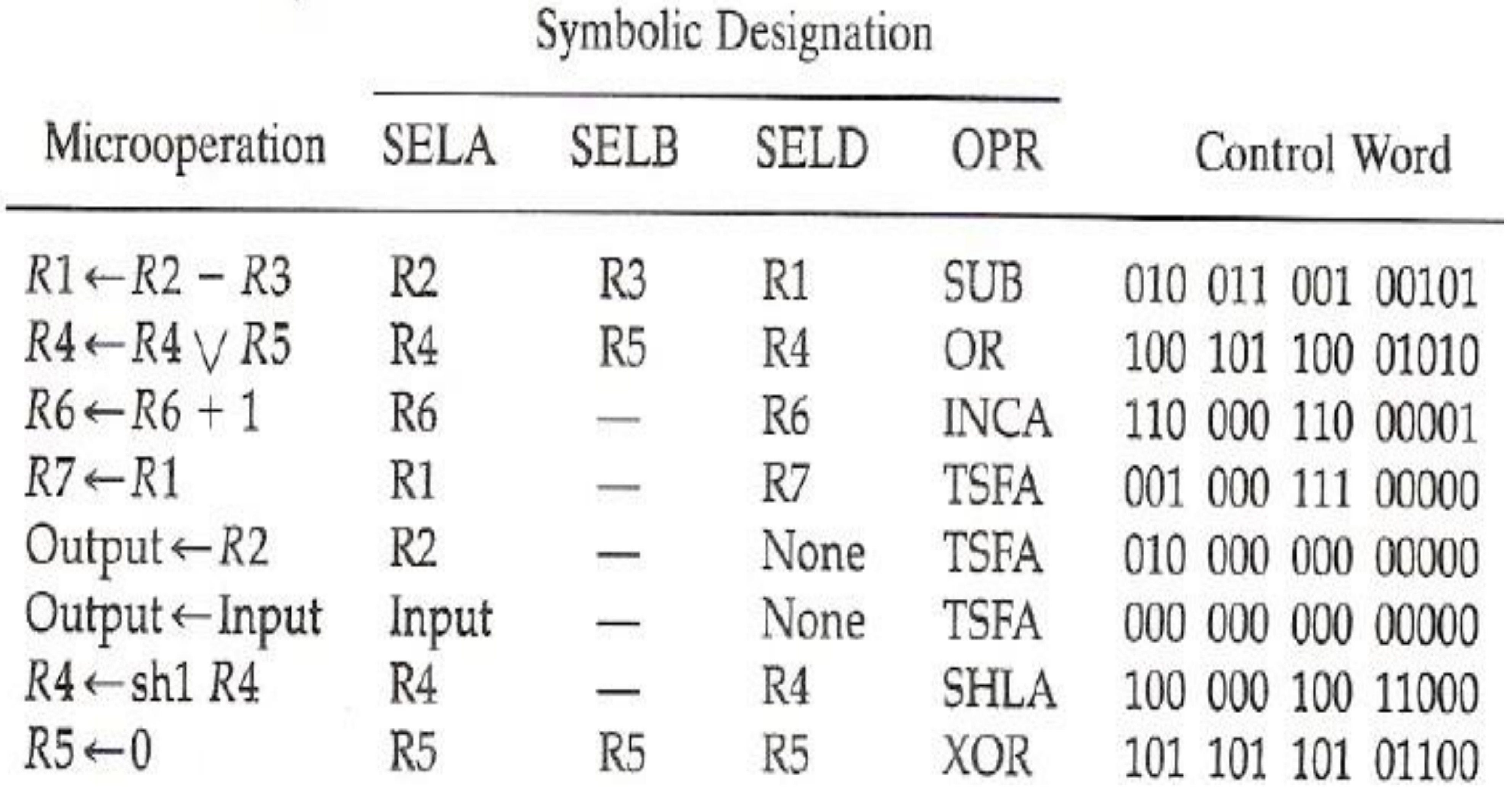


Example: R1 ← R2 - R3

This microoperation specifies R2 for A input of the ALU, R3 for the B input of the ALU, R1 for the destination register and ALU operation to subtract A-B. Binary control word for this microoperation statement is:

Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

Examples of different microoperations are shown below:



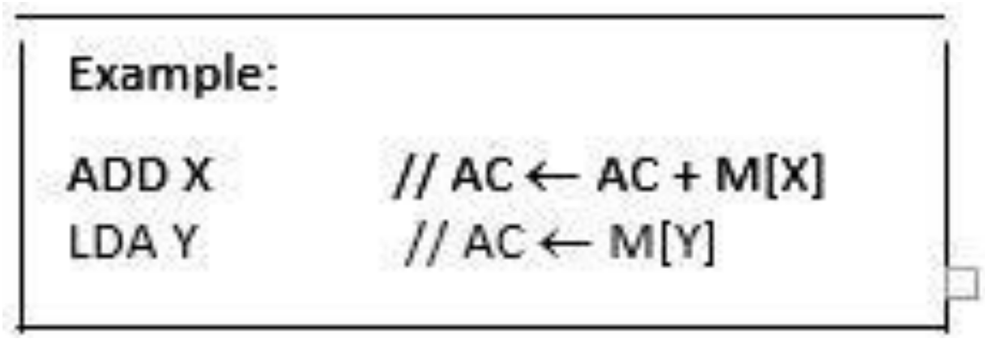
	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ sh1 $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

2. Stack Organization

- A useful feature that is included in the CPU of most computers is a stack or last-in, first-out (LIFO) list.
- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.
- The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
- The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push (or push-down) because it can be thought of as the result of pushing a new item on top.
- The operation of deletion is called pop (or pop-up) because it can be thought of as the result of removing one item so that the stack pops up. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

3. Single accumulator organization:

- In this type of organization all the operations are performed with an implied accumulator register.
- Basic computer is the good example of single accumulator organization.
- The instruction of this type of organization has an address field



Example:

```
ADD X        // AC ← AC + M[X]
LDA Y        // AC ← M[Y]
```

where X and Y is the address of the operand

Instruction Formats

Most common field found in register are:

a) Mode bit: It specifies the way the operand or the effective address is determined.

b) Op-code field: It specifies the operation to be performed.

c) Address field: It designates a memory address or a processor register.

The number of address fields in the instruction format depends on the internal organization of CPU. On the basis of no. of address field we can categorize the instruction as below:

1. Three-Address Instruction:

- Computer with three address instruction can use each address field to specify either processor register or memory operand.
- Advantage –it minimize the size of program
- Disadvantage –binary coded instruction requires too many bits to specify three address fields

  E.g. ADD R1, A, B : $R1 \leftarrow M[A]+M[B]$

Example

Program to evaluate the following arithmetic statement

X = (A+B) * (C+D) using three address fields instruction

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

2.Two-Address Instruction:

- Computer with two address instruction can use each address field to specify either processer register or memory operand
- Advantage –it minimize the size of instruction
- Disadvantage –the size of program is relatively larger

Example

Program to evaluate the following arithmetic statement

X = (A+B)*(C+D) using two address field instruction

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

3.One-Address Instruction:

- Execution of one address field instruction use an implied accumulator register for all data manipulation
- Advantage –relatively small instruction size
- Disadvantage –relatively large program size

Example

Program to evaluate the following arithmetic statement

X = (A+B)*(C+D) using one address field instruction

```
LOAD A      / AC ← M[A]
ADD B       / AC ← AC+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
```

4.Zero-Address Instruction:

- This type of instruction is used in stack organization computer. There is no address field in this type of instruction except PUSH and POP.
- Advantage –small instruction size
- Disadvantages –large the program size

Example

Program to evaluate the following arithmetic statement
X = (A+B)*(C+D) using zero address field instruction

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MUL
POP X
```

Exercise:

1. Y=A+B(CD+EF-G/H)
2. X=A-B+C+(D/E)

X=A-B+C+(D/E) solution

- Three address instruction format:

```
DIV R1,D,E
ADD R2,R1,C
ADD R3,R2,A
SUB X,R3,B
```

- Two address instruction format:

```
MOV R1,D
DIV   R1,E
ADD R1,C
ADD  R1,A
SUB   R1,B
MOV X,R1
```

- One address instruction:
  LOAD D
  DIV E
  ADD C
  ADD A
  SUB B
  STORE X

- Zero address instruction format:
  PUSH D
  PUSH E
  DIV
  PUSH C
  ADD
  PUSH A
  ADD
  PUSH B
  SUB
  POP X

1. Y=A+B(CD+EF-G/H) solution

Three address instruction format:

```
DIV R1,G,H
MUL R2,E,F
MUL R3,C,D
ADD R4,R2,R3
SUB Y,R4,R1
MUL Y,Y,B
ADD Y,Y,A
```

- Two Address Instruction Format:

```
MOV R1,G
DIV R1,H
MOV R2,E
MUL R2,F
MOV R3,C
MUL R3,D
ADD R2,R3
SUB R2,R1
MUL R2,B
ADD R2,A
MOV Y,R2
```

- One Address Instruction Format:

LOAD G

DIV H

STORE T

LOAD E

MUL F

SUB T

STORE T

LOAD C

MUL D

ADD T

MUL B

ADD A

STORE Y

- Zero Address instruction format:

PUSH G

PUSH H

DIV

PUSH E

PUSH F

MUL

SUB

PUSH C

PUSH D

MUL

ADD

PUSH B

MUL

PUSH A

ADD

POPY

Addressing Modes

The method of calculating or finding the effective address of the operand in the instruction is called addressing mode. The way operands (data) are chosen during program execution depends on the addressing mode of the instruction. So, *addressing mode* specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Why Addressing modes?

- To give programming versatility to the user (by providing facilities as: pointers to memory, counters for loop control, indexing of data and program relocation)
- To use the bits in the address field of the instruction efficiently

Types of Addressing Modes

The various addressing modes are:

i. Implied Mode
ii. Immediate Mode
iii. Register Mode
iv. Register Indirect Mode
v. Auto increment or Auto decrement Mode
vi. Direct Address Mode
vii. Indirect Address Mode
viii. Relative Address Mode
ix. Indexed Addressing Mode
x. Base Register Addressing Mode

i. Implied Mode:

- In this type of addressing mode, operands specified implicitly in the definition of instruction.
- All the register reference instructions that use an accumulator and zero-address instruction in a stack organized computer are implied mode instruction.
- No need to specify the address in the instruction.
- E.g. CMA (complement accumulator

ii. **Immediate Mode:**

- In this addressing mode, the operand is specified in the instruction itself i.e. there is no any address field to represent the operand
- Immediate mode instructions are useful for initializing register to a constant value.
- Instead of specifying the address of the operand, operand itself is specified in the instruction.

E.g. LDA data / $AC \leftarrow$ data

iii. Register Mode:

- In this type of addressing mode, the operands are in the register which is within the CPU .
- Faster to acquire an operand than the memory addressing

$$AC \longleftarrow R1$$

iv. Register Indirect Mode:

- In this addressing mode, the content of register present in the instruction specifies the effective address of operand.
- The advantage of this addressing mode is that the address field of the instruction uses fewer bits to select a register.
- EA = content of R

$$AC \leftarrow M[R1]$$

v. Auto Increment or Auto decrement mode:

- In auto increment mode, the content of CPU register is incremented by 1, which gives the effective address of the operand in memory.

  $AC \leftarrow M[R1],\ R1 \leftarrow R1 + 1$

- In auto decrement mode, the content of CPU register is decremented by 1, which gives the effective address of the operand in memory.

  $AC \leftarrow M[R1 - 1]$

vi. Direct Address Mode

In this addressing mode, the address field of an instruction gives the effective address of operand.

$AC \leftarrow M[ADR]$

vii. Indirect Address Mode

In this addressing mode, the address field of the instruction gives the address of effective address.

$AC \leftarrow M[M[ADR]]$

viii. Relative Address Mode:

In this addressing mode, the content of program counter is added to the address part of the instruction which gives the effective address of the operand.

$AC \leftarrow M[PC + ADR]$

ix. Indexed Addressing Mode:

In this addressing mode, the content of index register is added to the address field of the instruction which gives the effective address of operand.

$AC \leftarrow M[ADR + XR]$

x. Base Register Addressing Mode:

In this addressing mode, the content of the base register is added to the address part of the instruction which gives the effective address of the operand.

$$AC \leftarrow M[ADR + BR]$$

Numerical Example

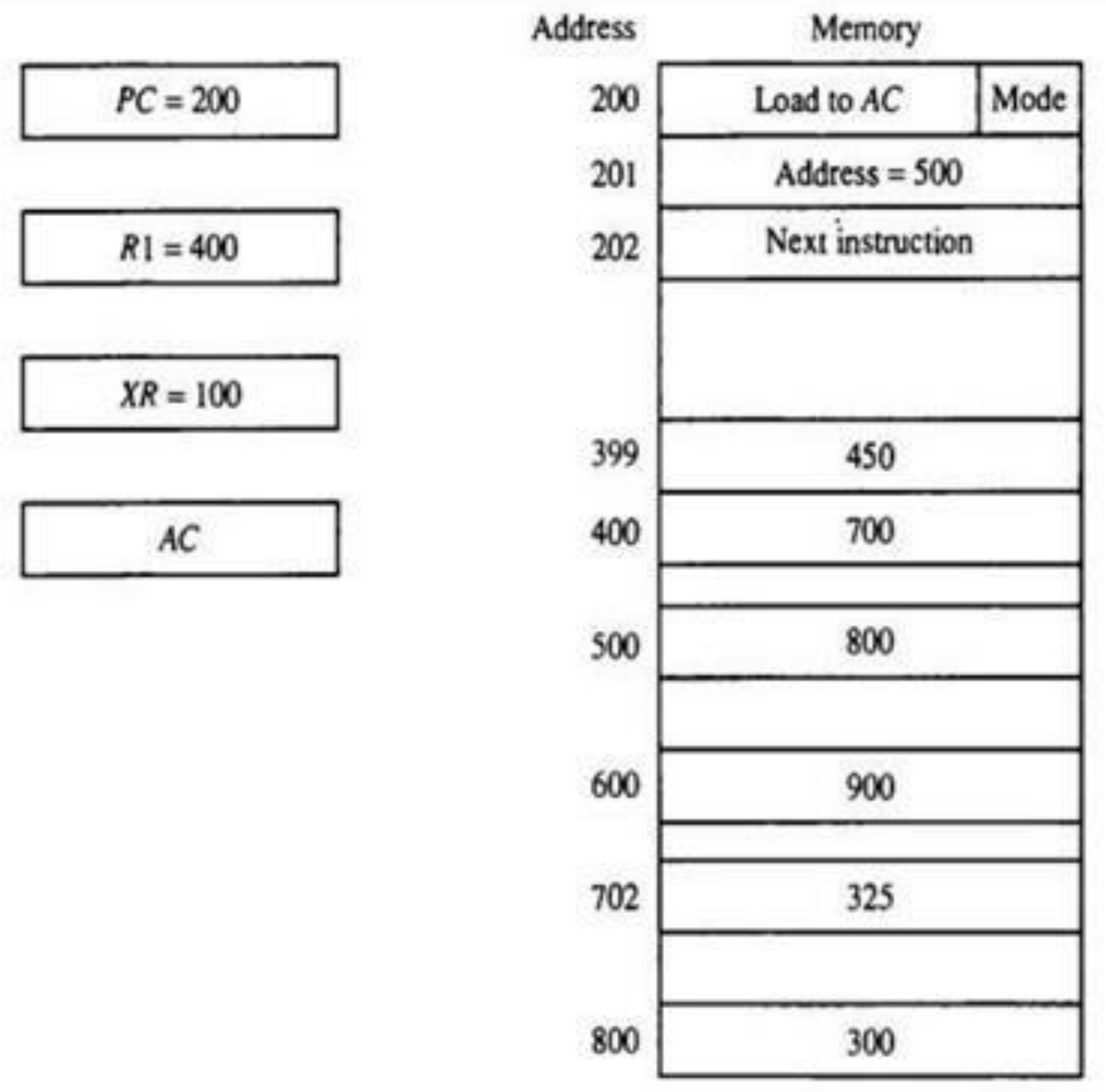
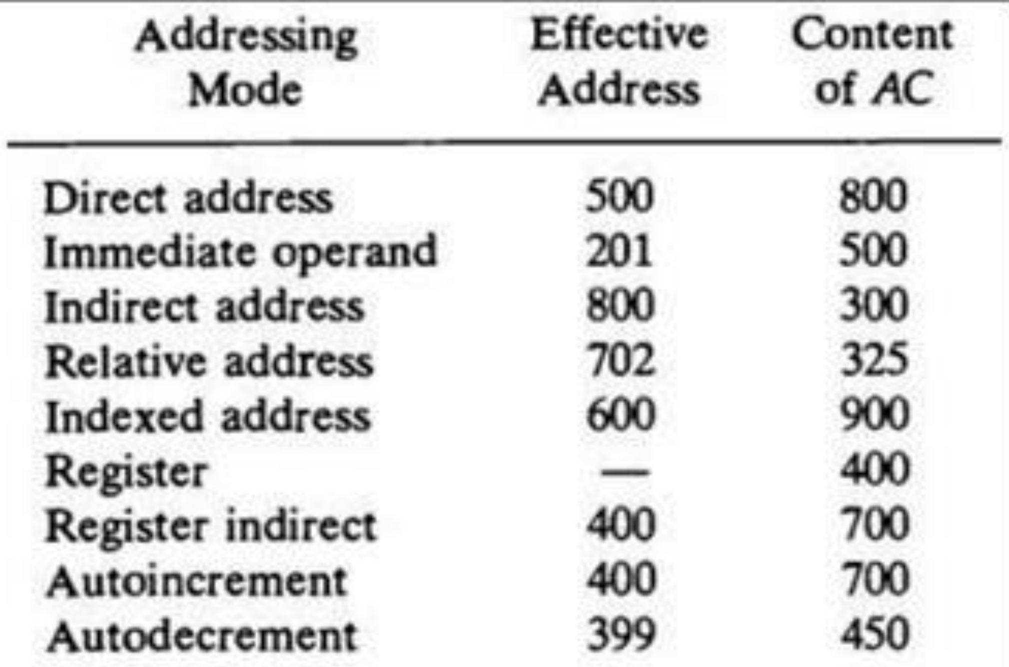


Fig: Numerical example for addressing modes



Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Fig: Content of AC after each addressing modes

Data Transfer and Manipulation

- Computers give extensive set of instructions to give the user the flexibility to carryout various computational tasks. The actual operations in the instruction set are not very different from one computer to another although binary encodings and symbol name (operation) may vary. So, most computer instructions can be classified into 3 categories:

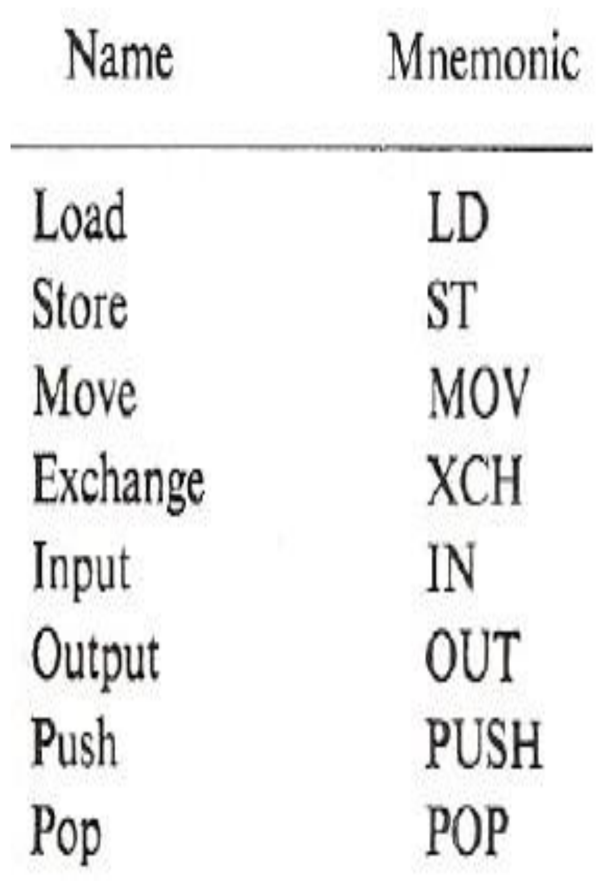
1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

1. Data Transfer Instructions:

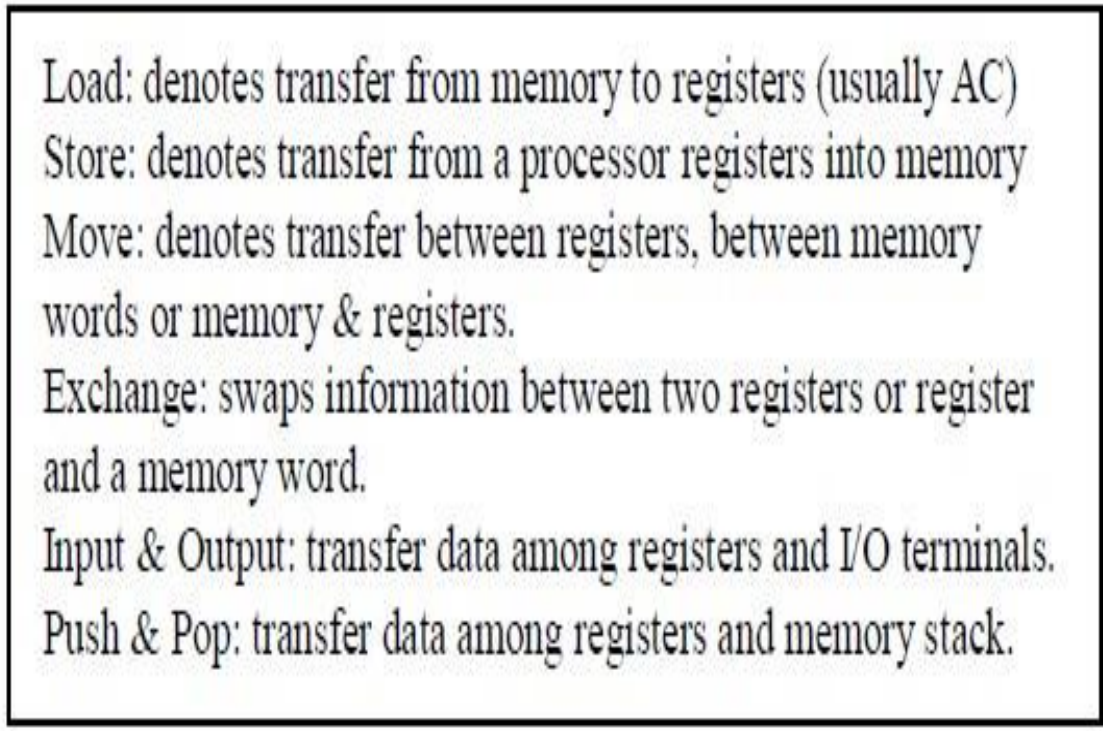
Data transfer instructions causes transfer of data from one location to another without modifying the binary information content. The most common transfers are:

- between memory and processor registers
- between processor registers and I/O
- between processor register themselves

Example: Load, store, exchange, move, push, pop, etc



Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

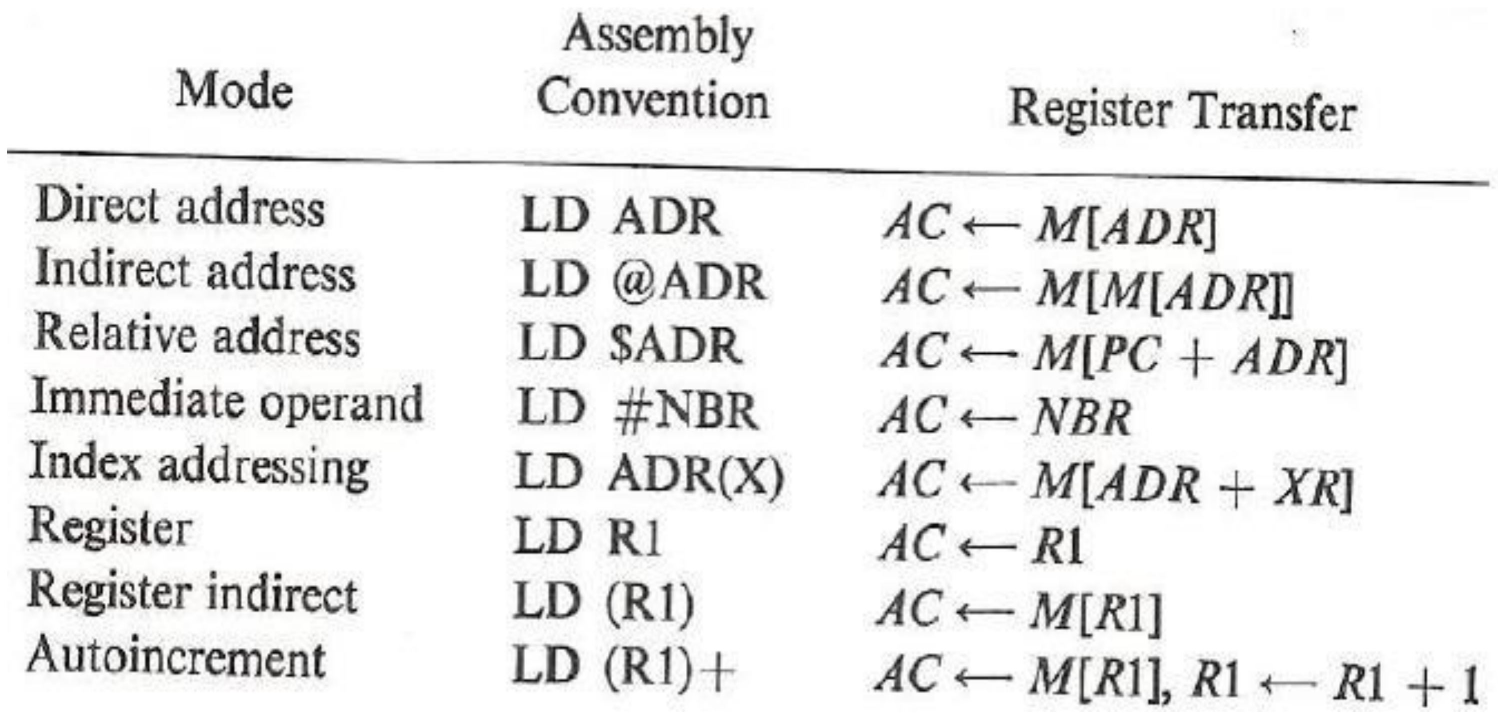


Load: denotes transfer from memory to registers (usually AC)
Store: denotes transfer from a processor registers into memory
Move: denotes transfer between registers, between memory words or memory & registers.
Exchange: swaps information between two registers or register and a memory word.
Input & Output: transfer data among registers and I/O terminals.
Push & Pop: transfer data among registers and memory stack.

Cont..

- Instructions described above are often associated with the variety of addressing modes.
- Assembly language uses special character to designate the addressing mode. E.g. # sign placed before the operand to recognize the immediate mode. (Some other assembly languages modify the mnemonics symbol to denote various addressing modes,
  e.g. for load immediate: LDI).
  - Example: consider load to accumulator instruction when used with 8 different addressing modes:

Cont..



Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

Table: Recommended assembly language conventions for load instruction in different addressing modes

Data manipulation Instructions:

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

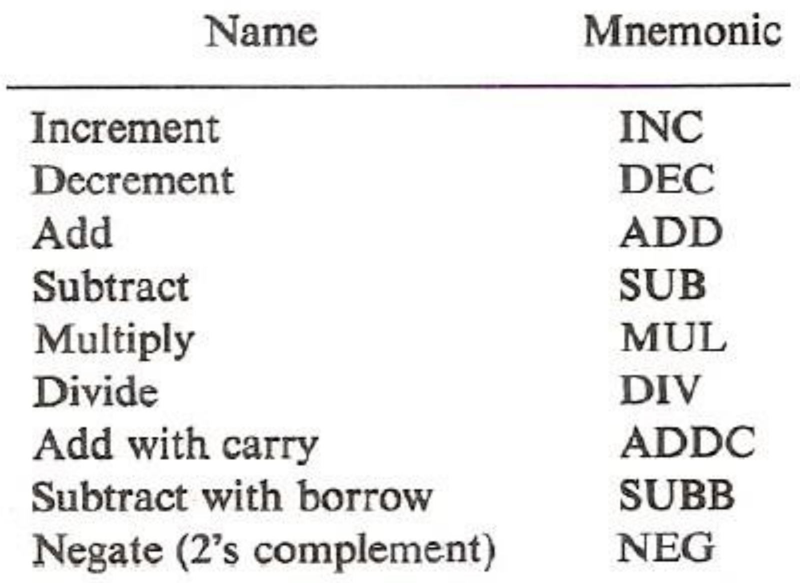
1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Example: increment, decrement, add, subtract, add with carry, subtract with borrow, 2’s complement.

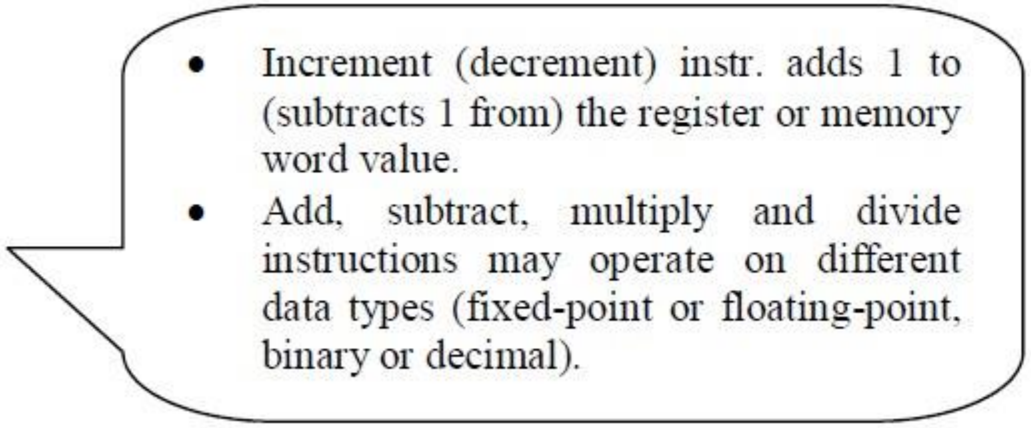
Arithmetic instructions:

The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions.

- Typical arithmetic instructions are listed below:



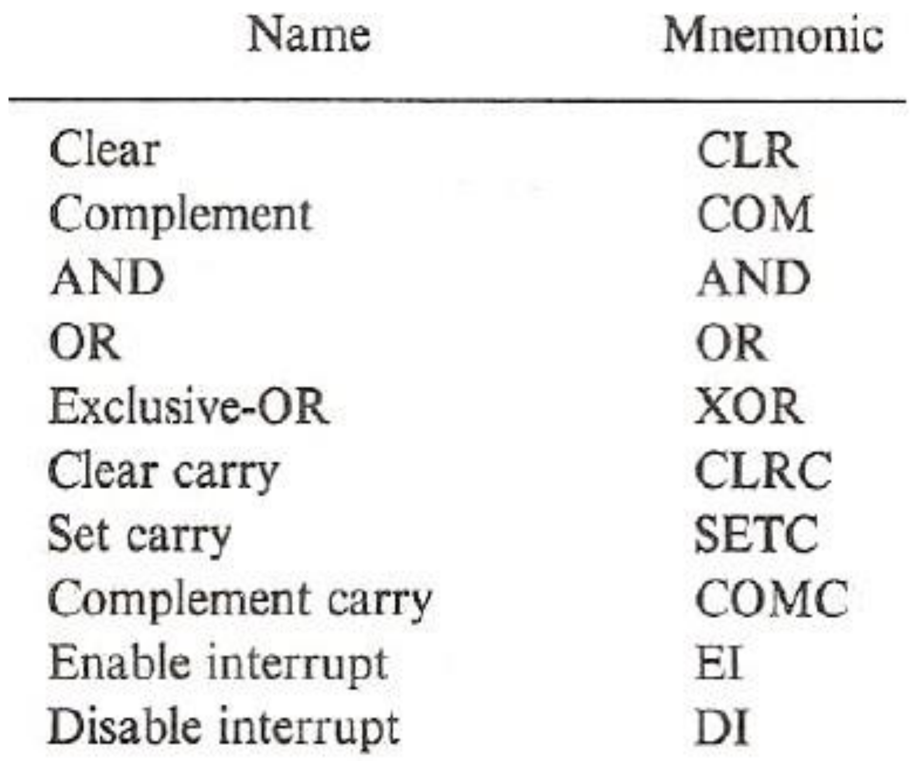
Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG



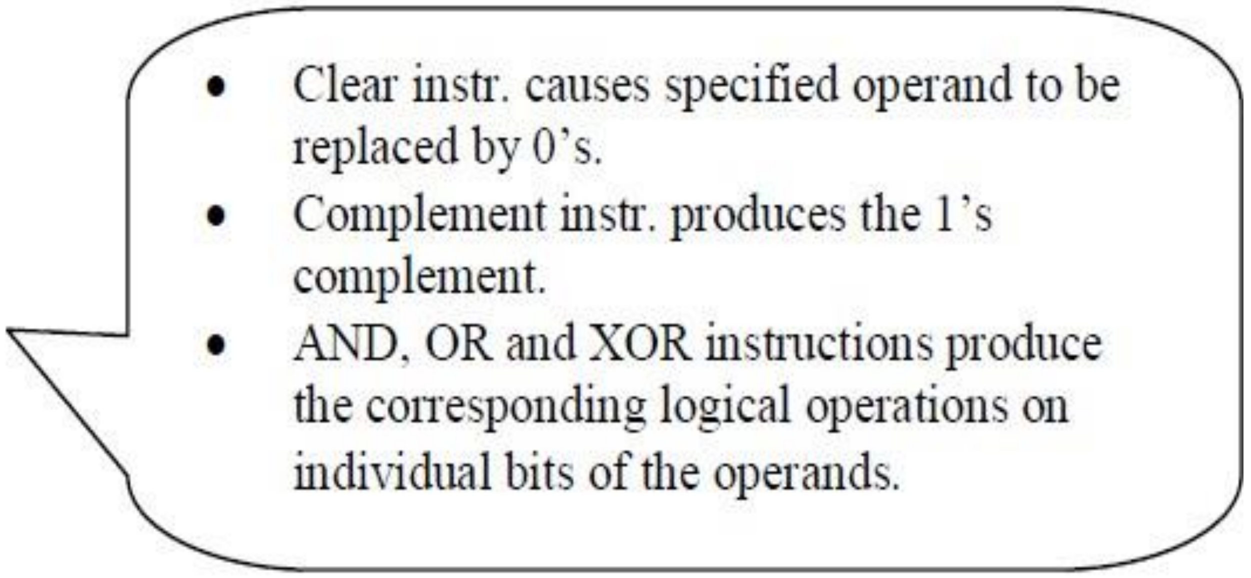
- Increment (decrement) instr. adds 1 to (subtracts 1 from) the register or memory word value.
- Add, subtract, multiply and divide instructions may operate on different data types (fixed-point or floating-point, binary or decimal).

Logical and bit manipulation instructions :

- Logical instructions perform binary operations on strings of bits stored in registers and are useful for manipulating individual or group of bits representing binary coded information. Logical instructions each bit of the operand separately and treat it as a Boolean variable.



Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI



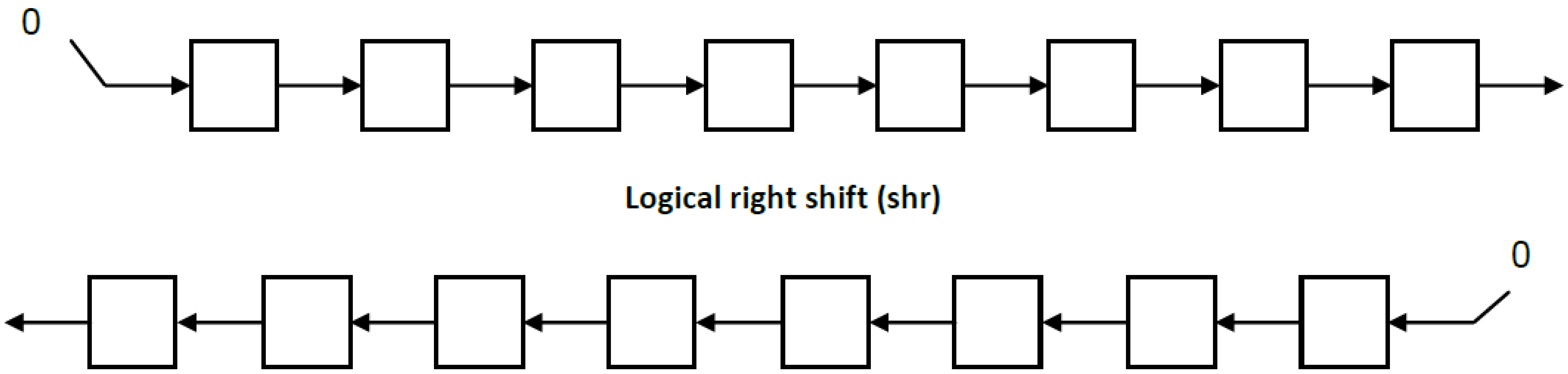
- Clear instr. causes specified operand to be replaced by 0's.
- Complement instr. produces the 1's complement.
- AND, OR and XOR instructions produce the corresponding logical operations on individual bits of the operands.

- The clear instruction causes the specified operand to be replaced by 0's.
- The complement instruction produces the 1's complement by inverting all the bits of the operand.
- The AND, OR, and XOR instructions produce the corresponding logical operations on individual bits of the operands.
- Although they perform Boolean operations, when used in computer instructions, the logical instructions should be considered as performing bit manipulation operations.
- Individual bits such as a carry can be cleared, set, or complemented with appropriate instructions. Another example is a flip-flop that controls the interrupt facility and is either enabled or disabled by means of bit manipulation instructions.

Shift instructions

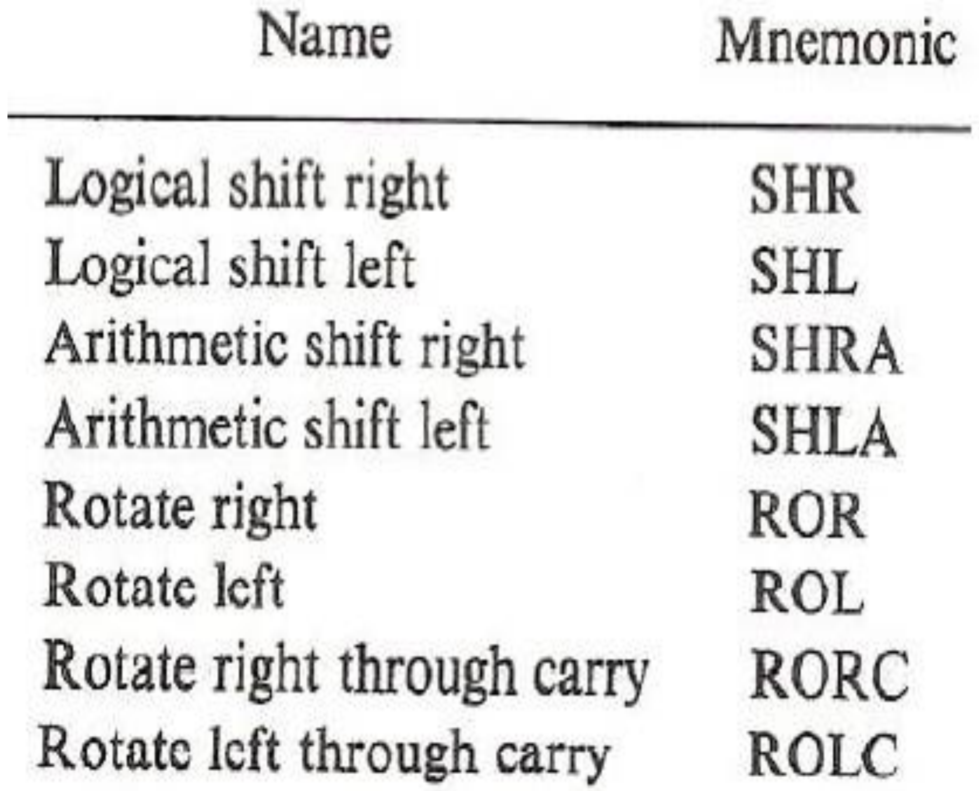
- Instructions to shift the content of an operand are quite useful and are often provided in several variations (bit shifted at the end of word determine the variation of shift). Shift instructions may specify 3 different shifts:
    - Logical shifts
    - Arithmetic shifts
    - Rotate-type operations

Logical Shift



Logical right shift (shr)

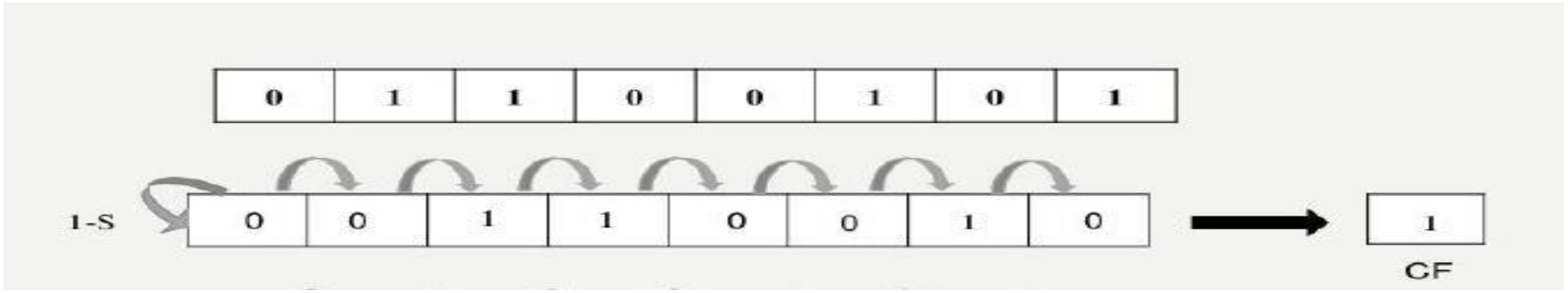
Logical left shift (shl)



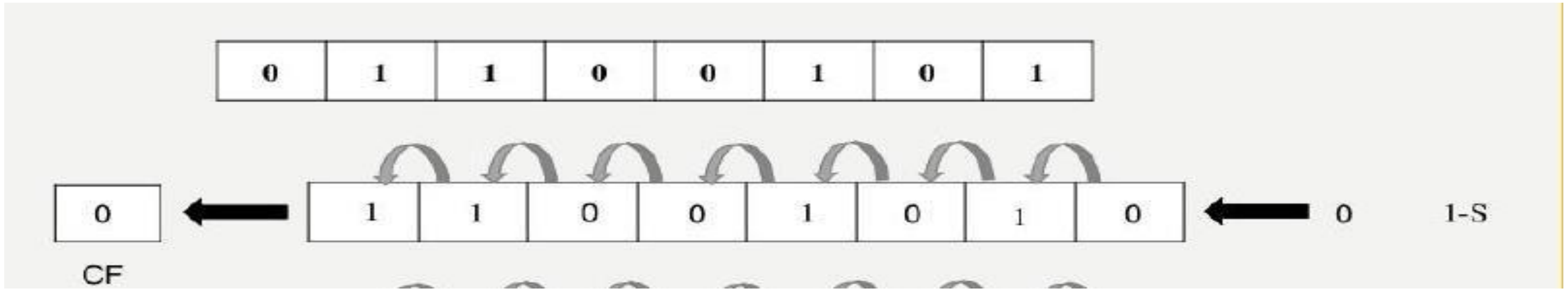
Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- Table lists 4 types of shift instructions.
- Logical shift inserts 0 at the end position
- Arithmetic shift left inserts 0 at the end (identical to logical left shift) and arithmetic shift right leave the sign bit unchanged (should preserve the sign).
- Rotate instructions produce a circular shift.
- Rotate left through carry instruction transfers carry bit to right and so is for rotate shift right.

Arithmetic Shift

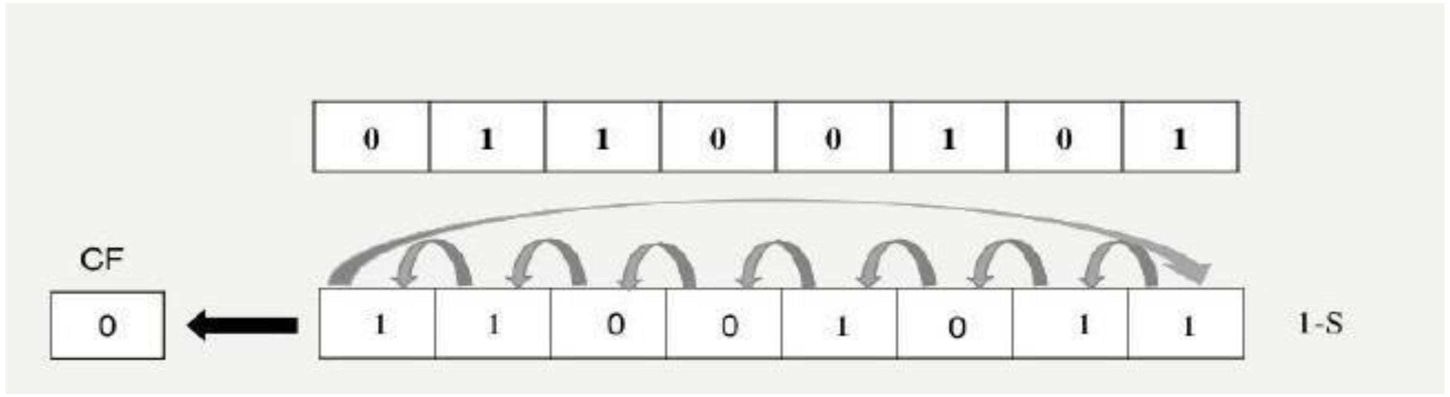


Arithmetic Shift Right

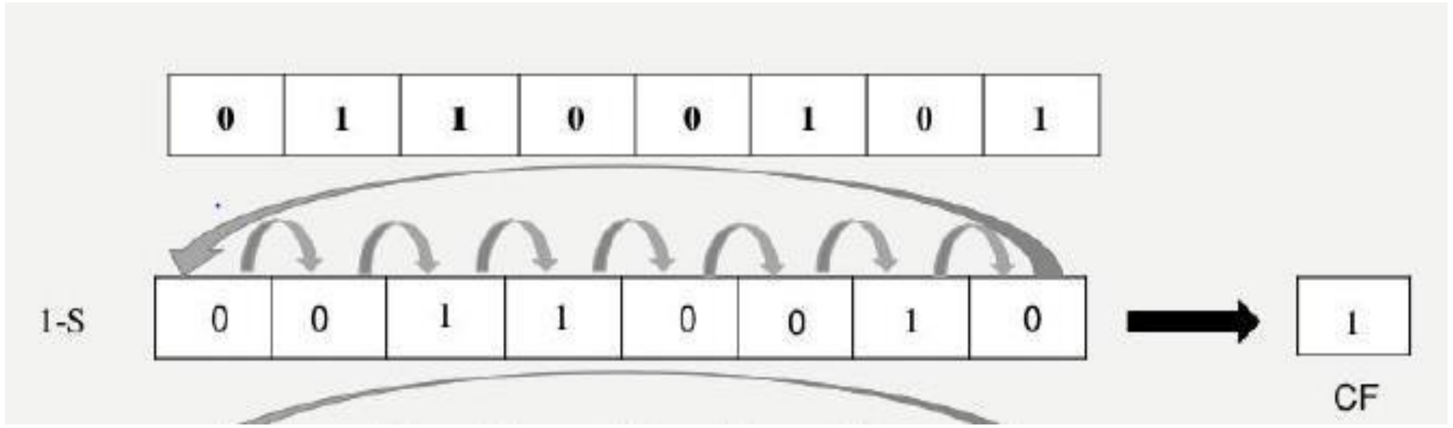


Arithmetic Shift Left

Rotate Instruction



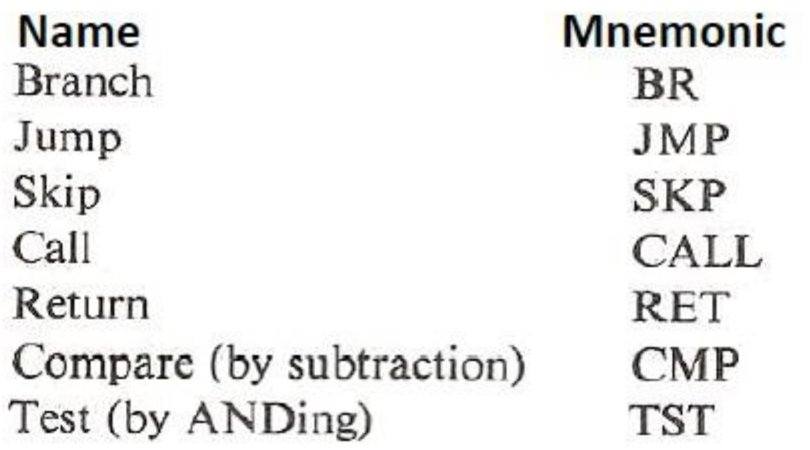
Rotate Left



Rotate Right

Program control instructions

- Instructions are always stored in successive memory locations and are executed accordingly. But sometimes it is necessary to condition the data processing instructions which change the PC value accidently causing a break in the instruction execution and branching to different program segments.



Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

- Branch (usually one address instruction) and jump instructions can be changed interchangeably.
- Skip is zero address instruction and may be conditional & unconditional.
- Call and return instructions are used in conjunction with subroutine calls.

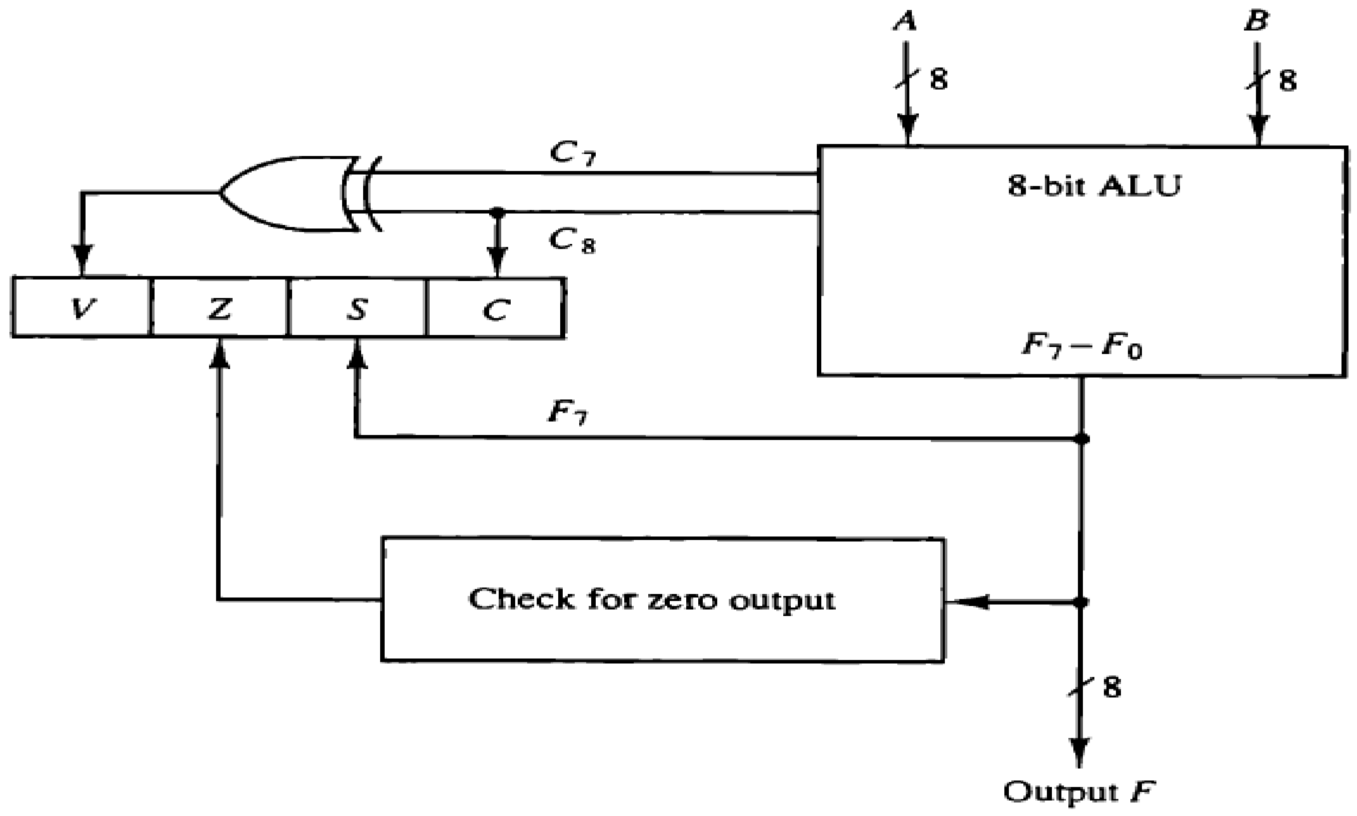


Figure 8-8 Status register bits.

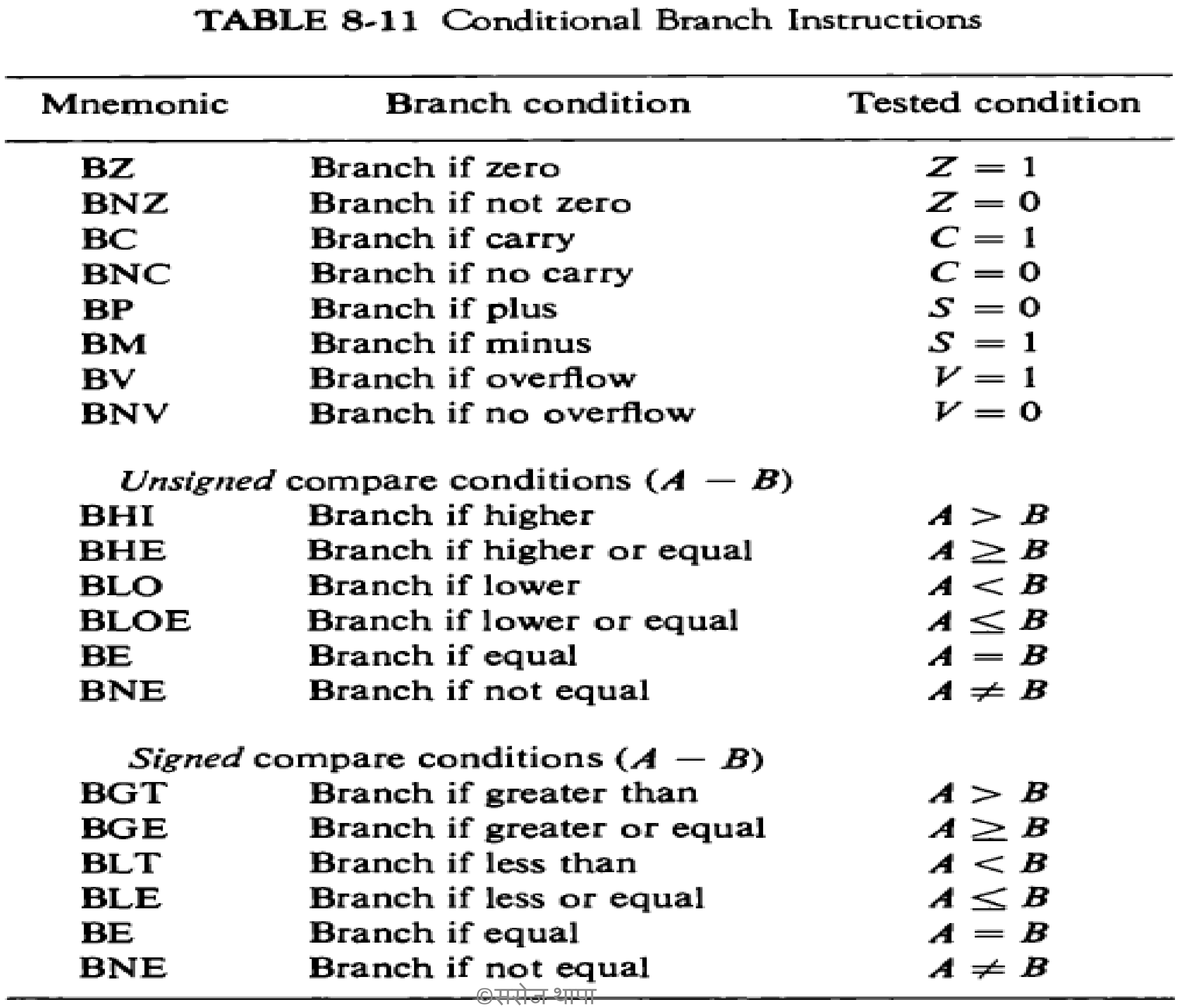


TABLE 8-11 Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions ($A - B$)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions ($A - B$)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

NOTE:
Also include Subroutine Call and Return in Program control

Program Interrupt

- The concept of program interrupt is to handle a variety of problems that arise out of normal program sequence.
- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.
- After a program has been interrupted and the service routine has been executed, the CPU must return to exactly the same state that it was when the interrupt occurred.

Types of Interrupts:

External interrupts

External Interrupts initiated from the outside of CPU and Memory

- I/O Device -> Data transfer request or Data transfer complete
- Timing Device -> Timeout
- Power Failure

Internal interrupts (traps)

Internal Interrupts are caused by the currently running program

- Register, Stack Overflow
- Divide by zero
- OP-code Violation
- Protection Violation

Software Interrupts

- Both External and Internal Interrupts are initiated by the computer Hardware.
- Software Interrupts are initiated by executing an instruction.

Complex Instruction Set Computer (CISC):

Computers with many instructions and addressing modes came to be known as Complex Instruction Set Computers (CISC).
Characteristics of CISC computers are:

- The large number of instructions and addressing modes led CISC machines to have variable length instruction formats
- Multiple operand instructions could specify different addressing modes for each operand
- Variable length instructions greatly complicate the fetch and decode problem for a processor
- They have instructions that act directly on memory addresses due to which multiple memory cycle are needed for executing instructions.
- Microprogrammed control is used rather than hardwired control.

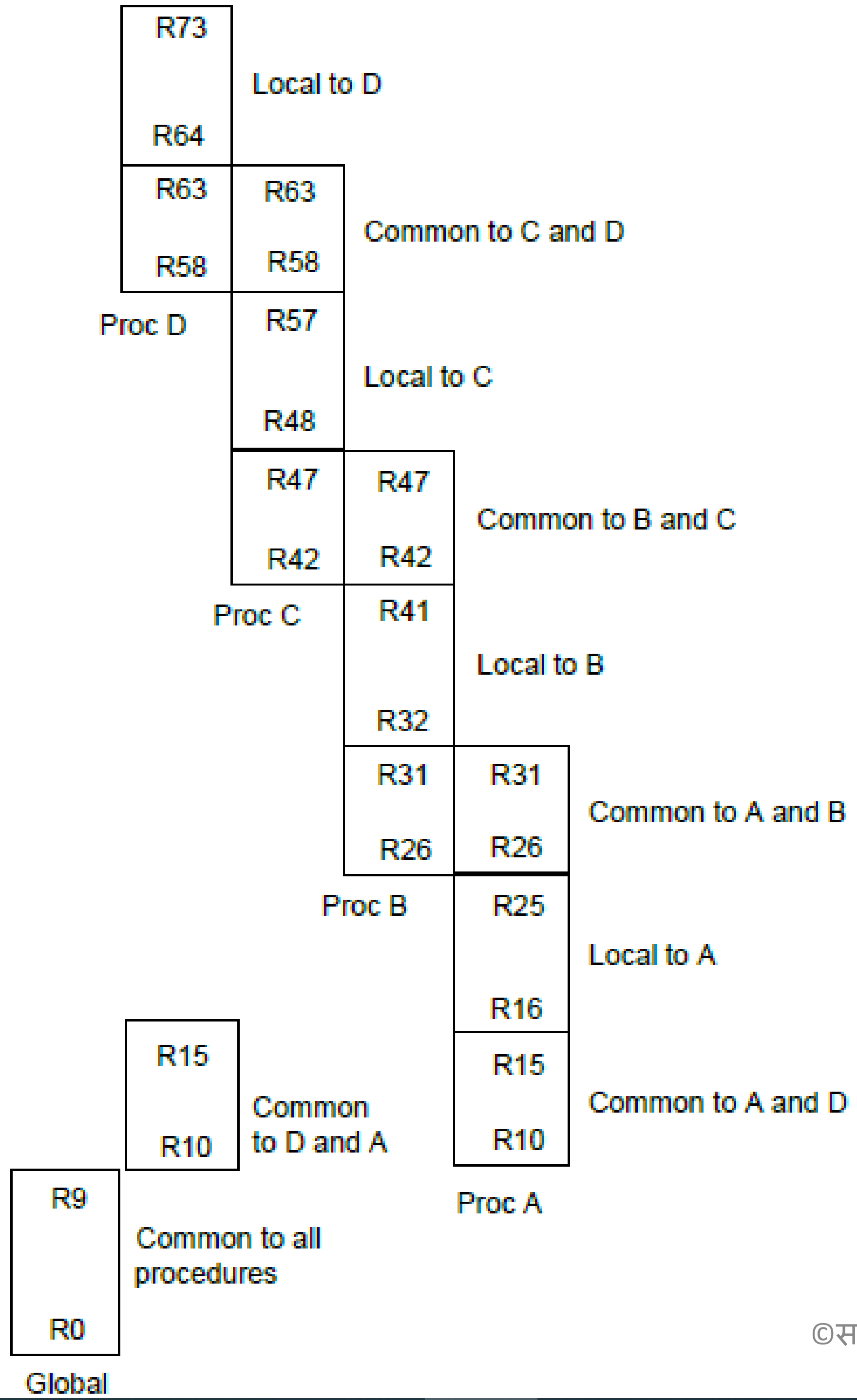
Reduced Instruction Set Computer (RISC)

Reduced Instruction Set Computers (RISC) were proposed as an alternative The underlying idea behind RISC processors is to simplify the instruction set and reduce instruction execution time. Characteristics of RISC are:

- Few instructions
- Few addressing modes
- Only load and store instructions access memory
- All other operations are done using on-processor registers
- Fixed length instructions
- Single cycle execution of instructions
- The control unit is hardwired, not microprogrammed

Register Overlapped Windows:

- The procedure (function) call/return is the most time-consuming operations in typical HLL programs.
- The depth of procedure activation is within a relatively narrow range.
- If we use multiple small sets of registers (windows), each assigned to a different procedure, a procedure call automatically switches the CPU to use a different window of registers, rather than saving registers in memory.
- Windows for adjacent procedures are overlapped to allow parameter passing.



There are three classes of registers:

– Global Registers

» Available to all functions

– Window local registers

» Variables local to the function

– Window shared registers

» Permit data to be shared without actually needing to copy it

Cont..

- Only one register window is active at a time.
- The active register window is indicated by a pointer. When a function is called, a new register window is activated.
- This is done by incrementing the pointer. When a function calls a new function, the high numbered registers of the calling function window are shared with the called function as the low numbered registers in its register window.
- This way the caller's high and the called function's low registers overlap and can be used to pass parameters.
- The advantage of overlapped register windows is that the processor does not have to push registers on a stack to save values and to pass parameters when there is a function call

END OF CHAPTER 4