CHAPTER 4

CENTRAL PROCESSING UNIT AND CONTROL UNIT

Rolisha Sthapit

CONTENTS

Introduction; Register Organization; Stack Organization; Instruction Format; Addressing Modes; Data Transfer and Manipulation; RISC and CISC.

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.
- 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.

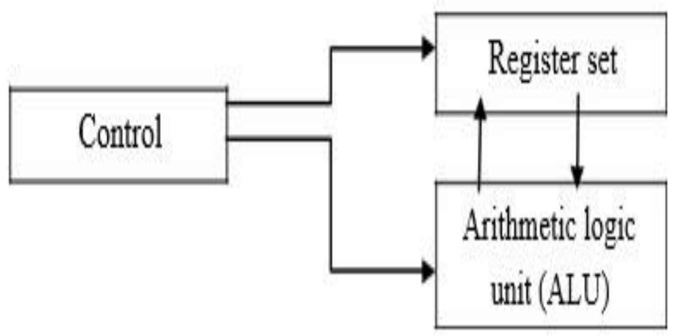
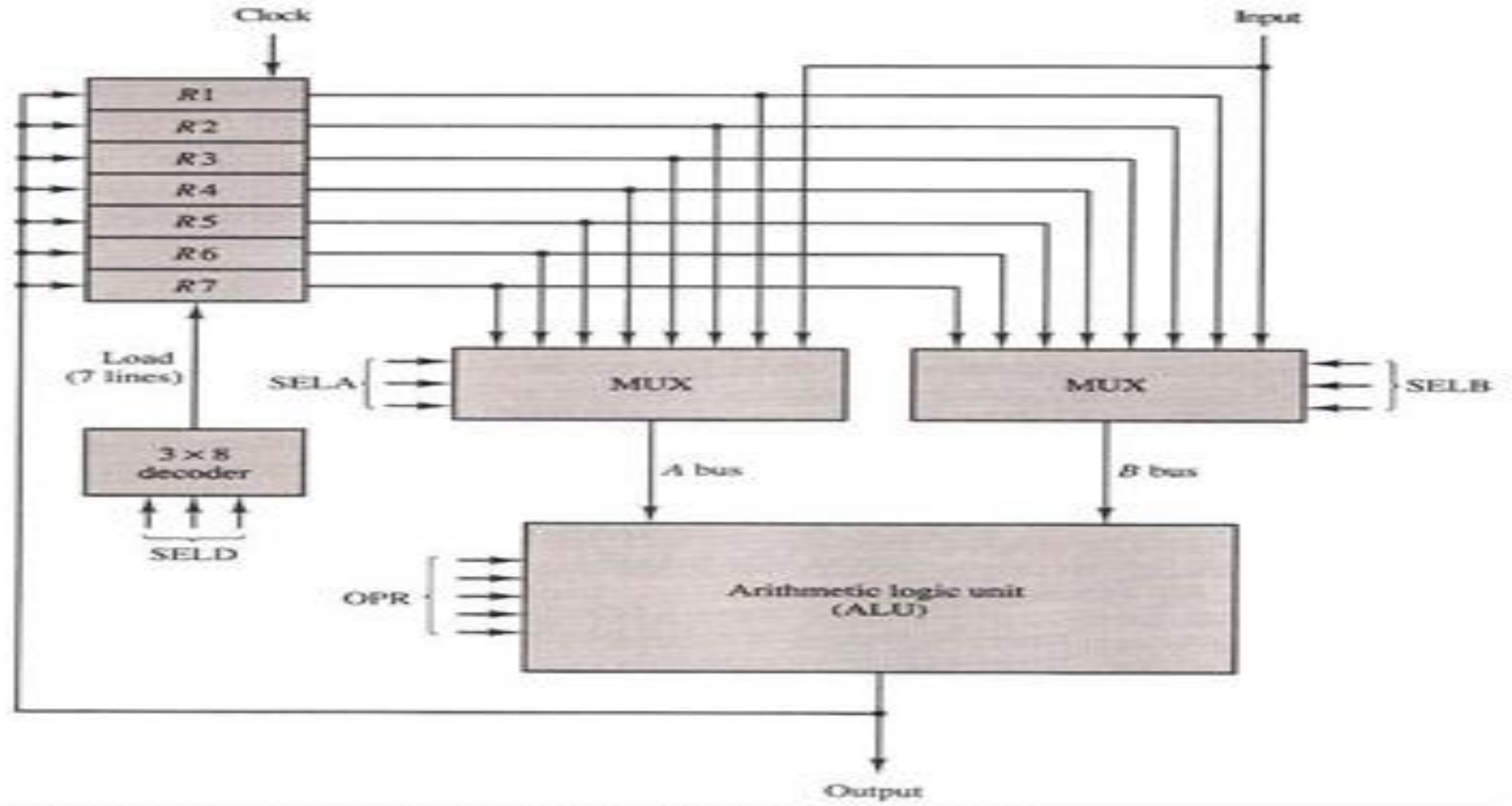


Fig: Major components of CPU

General Register Organization

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

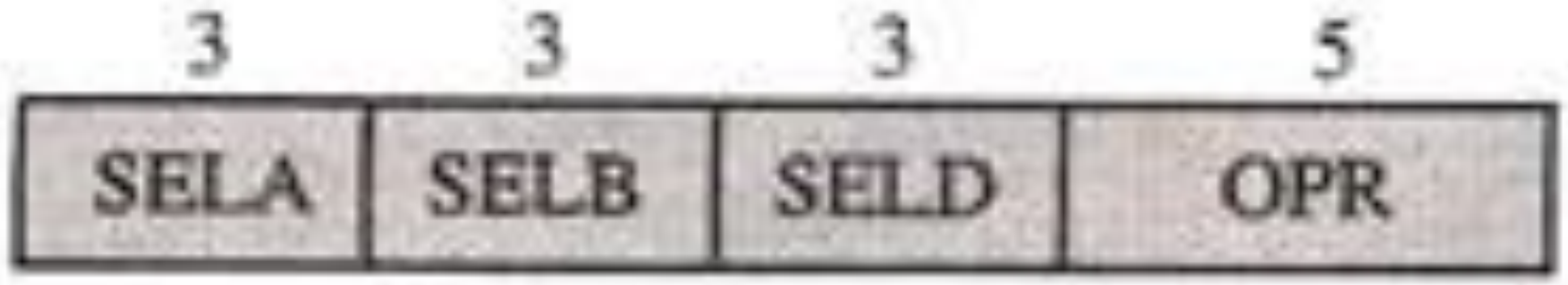


(a) Block diagram (register organization)

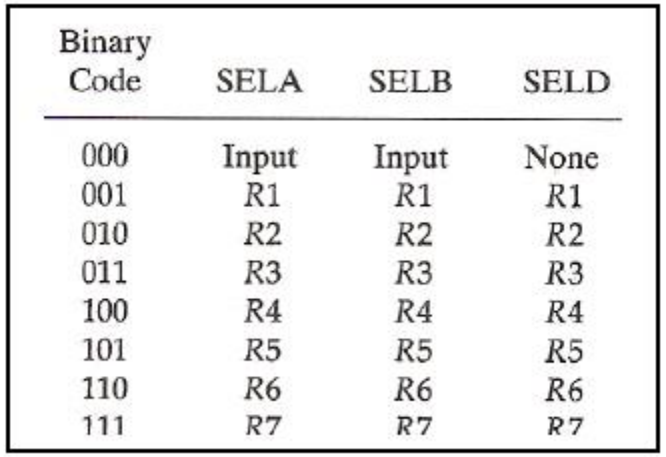
- All registers are connected to two multiplexers (MUX) that select the registers for bus A and bus B. Registers selected by multiplexers are sent to ALU. Another selector (OPR) connected to ALU selects the operation for the ALU. Output produced by ALU is stored in some register and this destination 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:

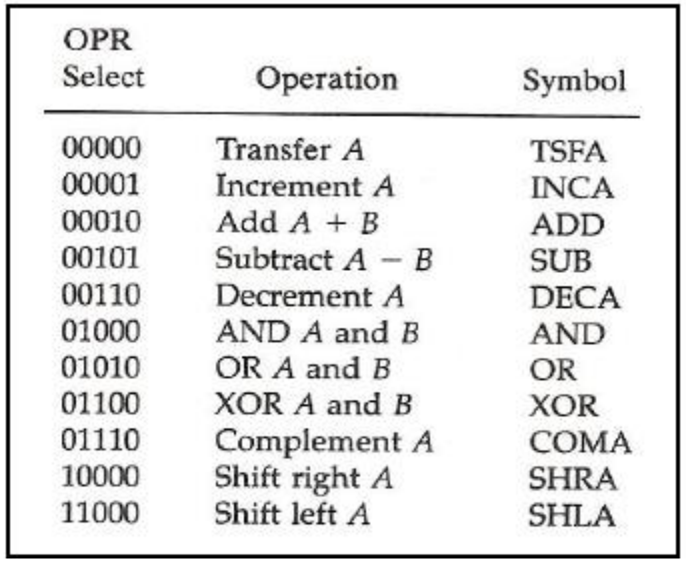
Combination of all selection bits of a processing unit is called control word. Control Word for above CPU is as below:



- 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	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7



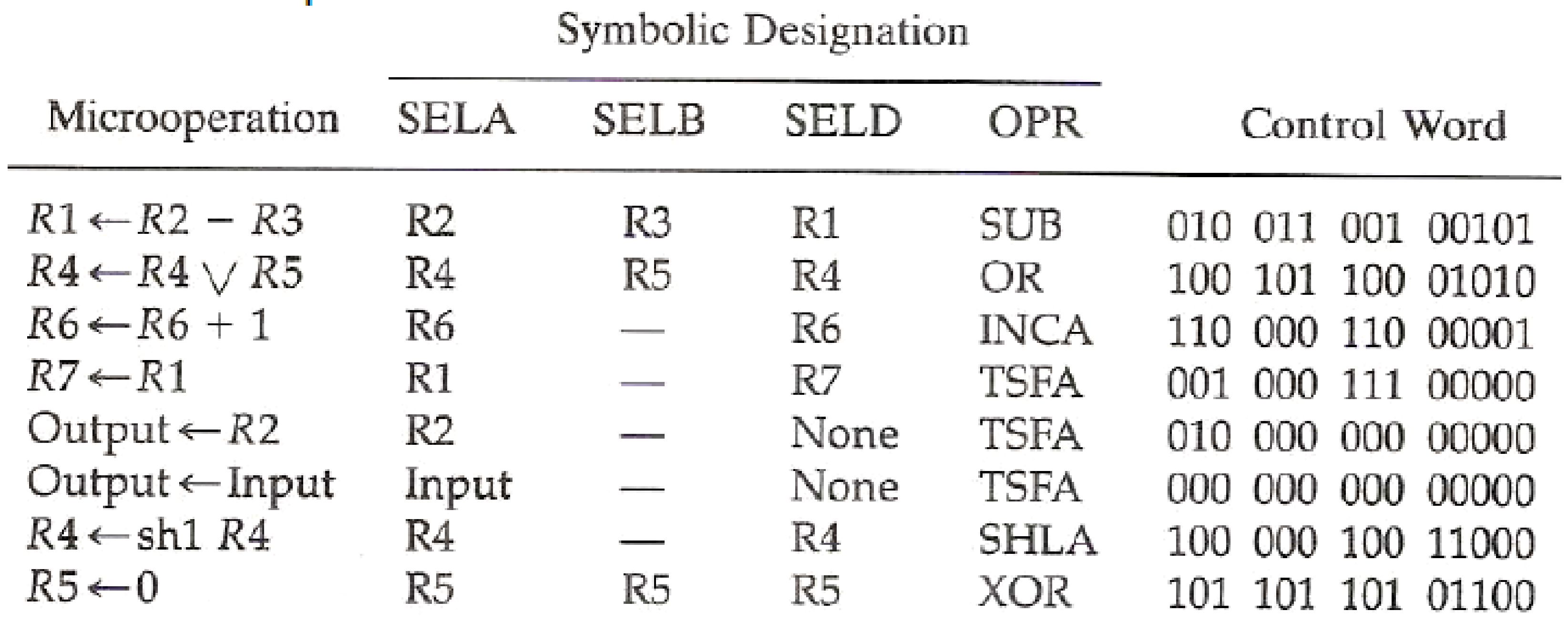
OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*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

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
$\text{Output} \leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
$\text{Output} \leftarrow \text{Input}$	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow \text{shl}\ R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Stack Organization

- This is useful *last-in, first-out* (LIFO) list (actually storage device) included in most CPU’s. Stack in digital computers is essentially a memory unit with a stack pointer (SP). SP is simply an address register that points stack top. Two operations of a stack are the insertion (push) and deletion (pop) of items. In a computer stack, nothing is pushed or popped; these operations are simulated by incrementing or decrementing the SP register.

Register stack

- It is the collection of finite number of registers. Stack pointer (SP) points to the register that is currently at the top of stack.

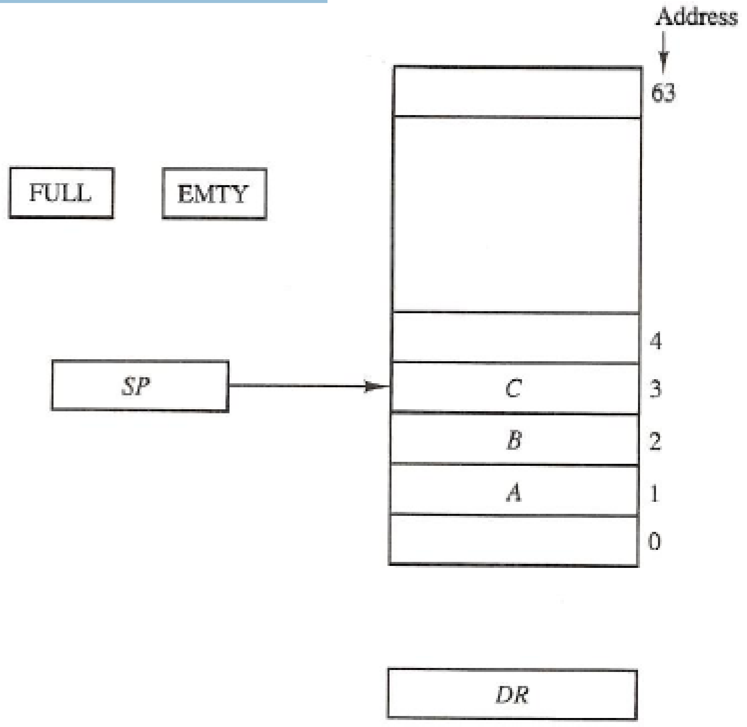


Fig: Block diagram of a 64-word stack

Diagram shows 64-word register stack. 6-bit address SP points stack top. Currently 3 items are placed in the stack: A, B and C do that content of SP is now 3 (actually 000011). 1-bit registers FULL and EMTY are set to 1 when the stack is full and empty respectively. DR is data register that holds the binary data to be written into or read out of the stack.

/* Initially, SP = 0, EMPTY = 1(true), FULL = 0(false) */

Push operation

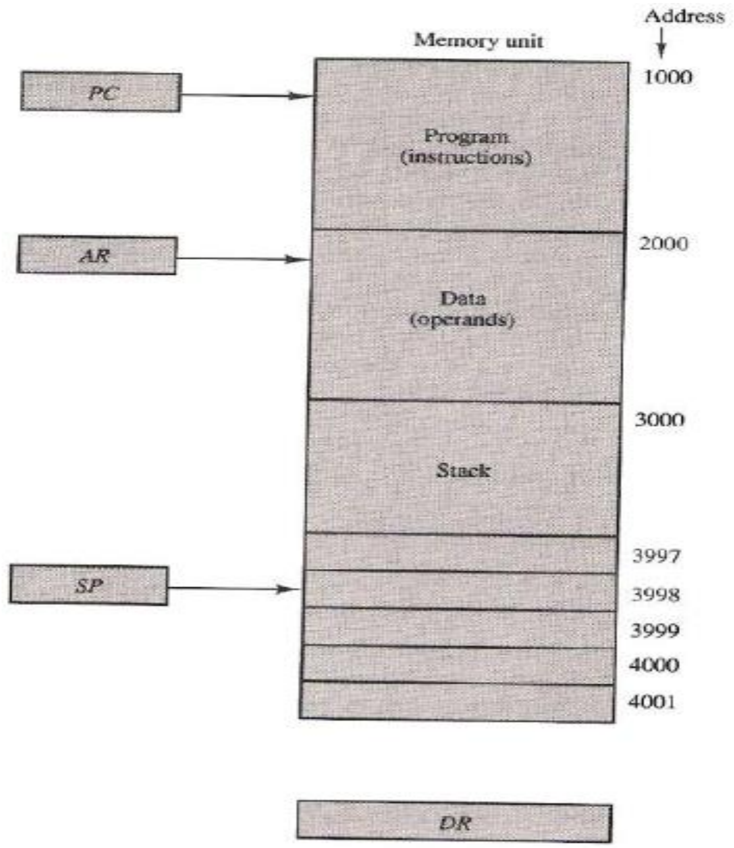
```
SP ← SP + 1
M [SP] ← DR
If (SP = 0) then (FULL ← 1)
EMPTY ← 0
```

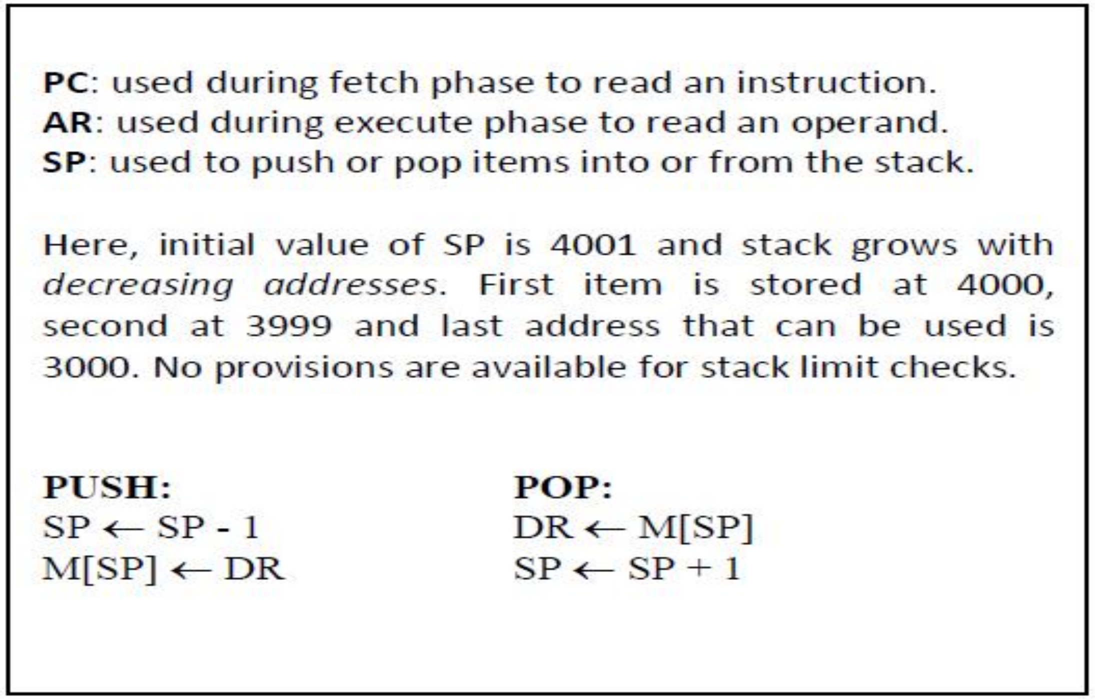
Pop operation

```
DR ← M [SP]
SP ← SP – 1
If (SP = 0) then (EMPTY ← 1)
FULL ← 0
```

Memory stack

- A portion of memory can be used as a stack with a processor register as a SP. Figure below shows a portion of memory partitioned into 3 parts: program, data and stack.





PC: used during fetch phase to read an instruction.
AR: used during execute phase to read an operand.
SP: used to push or pop items into or from the stack.

Here, initial value of SP is 4001 and stack grows with *decreasing addresses*. First item is stored at 4000, second at 3999 and last address that can be used is 3000. No provisions are available for stack limit checks.

PUSH:	**POP:**
$SP \leftarrow SP - 1$	$DR \leftarrow M[SP]$
$M[SP] \leftarrow DR$	$SP \leftarrow SP + 1$

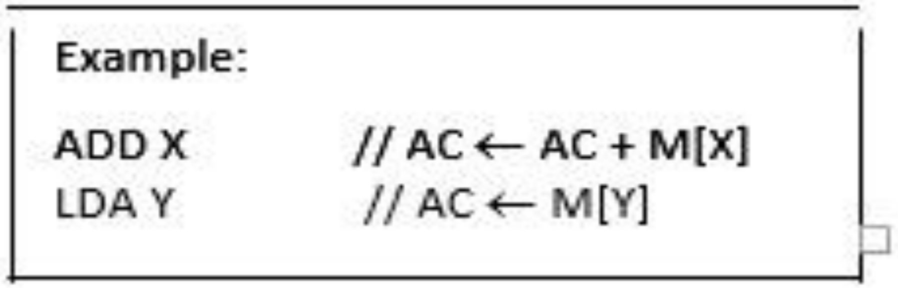
CPU Organizations/Processor Organization

There are three types of CPU organization based on the instruction format:

1. Single accumulator organization
2. General register organization
3. Stack Organization

1.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



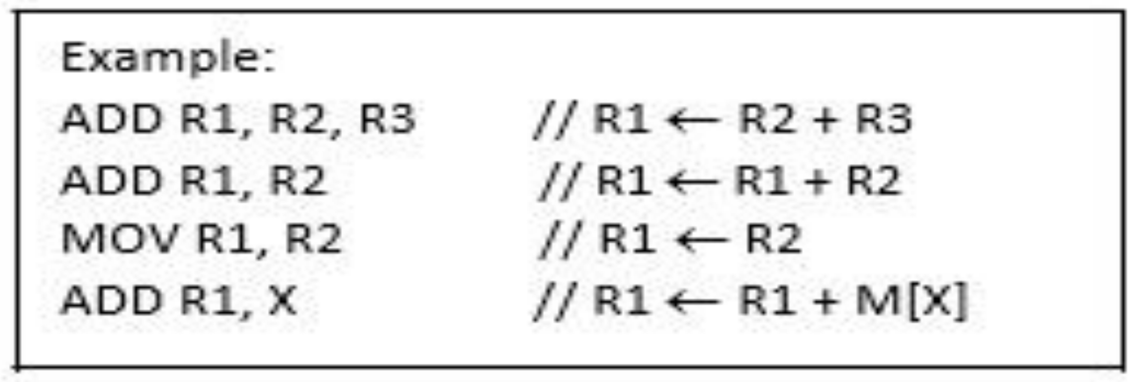
Example:

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

where X and Y is the address of the operand

2. General register organization:

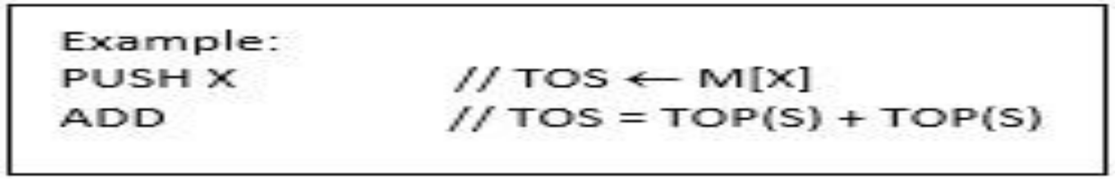
- When a large number of processor registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfer, but also while performing various microoperations. Hence, it is necessary to provide a common unit that can perform all the arithmetic, logic and shift microoperations in the processor.
- In this type of organization the instruction has two or three address field



```
Example:
ADD R1, R2, R3        // R1 ← R2 + R3
ADD R1, R2            // R1 ← R1 + R2
MOV R1, R2            // R1 ← R2
ADD R1, X             // R1 ← R1 + M[X]
```

3. Stack organization:

- Last-in, first-out (LIFO) mechanism.
- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- In this type of organization of CPU, all the operations are performed with stack.
- The PUSH and POP instruction only need address field. The operation-type instructions do not need address field.



```
Example:
PUSH X          // TOS ← M[X]
ADD             // TOS = TOP(S) + TOP(S)
```

This ADD instruction in the stack organization performs addition of two top of the stack element and stores the result in the top of the stack. First pops two operands from the top of the stack; adds them and stores the result in the top of the stack.

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 ← R1*R2
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          / TOS←M[A]
PUSH B          / TOS←M[B]
ADD             / TOS← (A+B)
PUSH C          / TOS←M[C]
PUSH D          / TOS←M[D]
ADD             / TOS←(C+D)
MUL             / TOS← (A+B)*(C+D)
POP X           / M[X] ←TOS
```

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

Three Address	Two Address	One Address	Zero Address
DIV R1,D,E	MOV R1,D	LOAD D	PUSH D
ADD R2,R1,C	DIV R1,E	DIV E	PUSH E
ADD R3,R2,A	ADD R1,C	ADD C	DIV
SUB X,R3,B	ADD R1,A	ADD A	PUSH C
	SUB R1,B	SUB B	ADD
	MOV X,R1	STORE X	PUSH A
			ADD
			PUSH B
			SUB
			POP X

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

Three address	Two address	One address	Zero address
DIV R1,G,H	MOV R1,G	LOAD G	PUSH G
MUL R2,E,F	DIV R1,H	DIV H	PUSH H
MUL R3,C,D	MOV R2,E	STORE T	DIV
ADD R4,R2,R3	MUL R2,F	LOAD E	PUSH E
SUB Y,R4,R1	MOV R3,C	MUL F	PUSH F
MUL Y,Y,B	MUL R3,D	SUB T	MUL
ADD Y,Y,A	ADD R2,R3	STORE T	SUB
	SUB R2,R1	LOAD C	PUSH C
	MUL R2,B	MUL D	PUSH D
	ADD R2,A	ADD T	MUL
	MOV Y,R2	MUL B	ADD
		ADD A	PUSH B
		STORE Y	MUL
			PUSH A
			ADD
			POP Y

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), CLA, CME, etc.

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 #NBR / $AC \leftarrow NBR$

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

$$AC \leftarrow R1$$

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

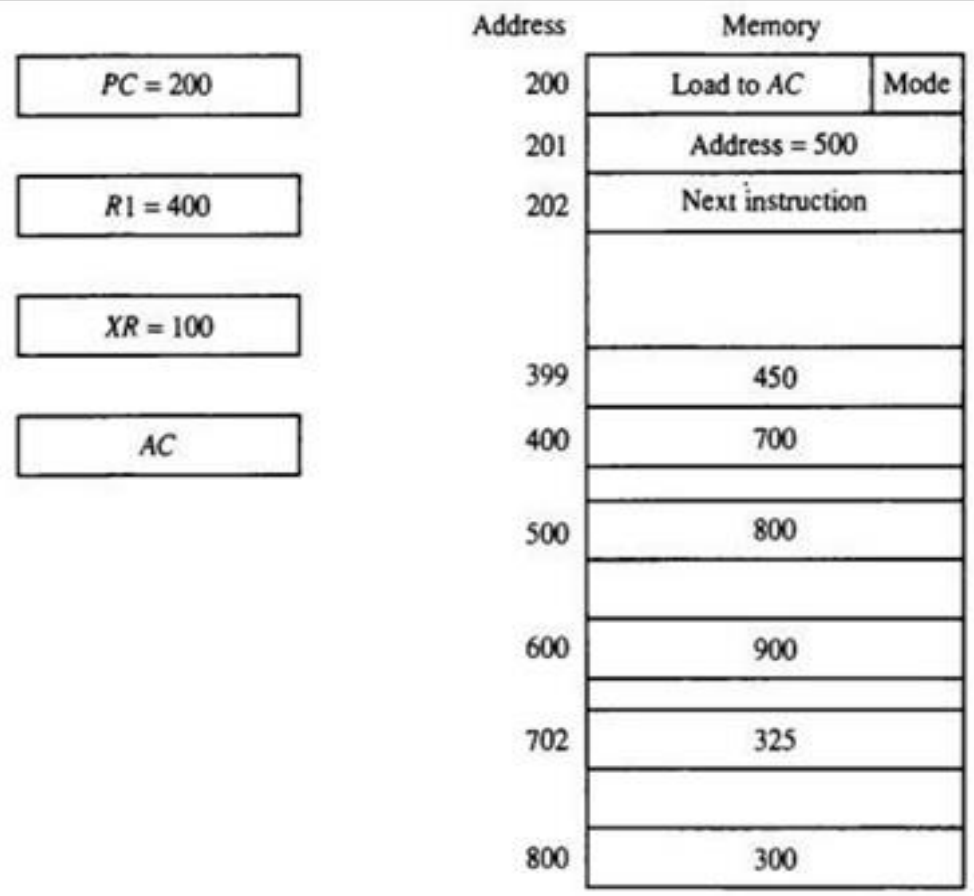
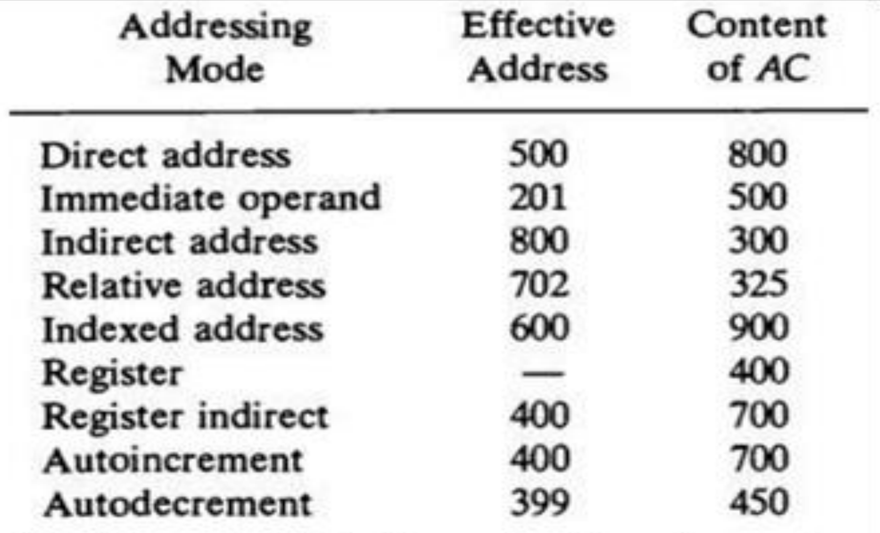


Fig: Numerical example for addressing modes



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

Calculate the value of EA and AC using all addressing modes.

PC = 150

R1 = 500

XR=107

BR=106

Address	Memory
400	Load to AC
401	Address= 600
407	Next Instruction
501	101
550	400
600	800
706	900
708	350
1007	850

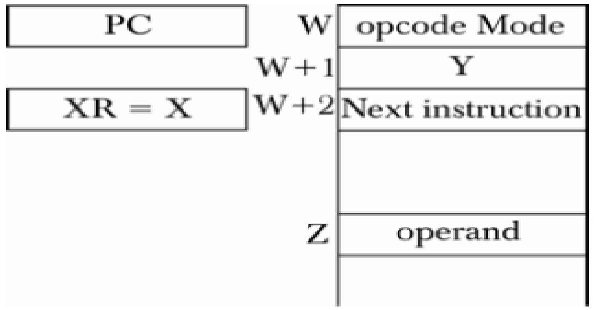
Addressing Modes	Effective Address	Content of AC
Immediate	401	600
Register	-	500
Register Indirect	500	-
Auto Increment	500	- ,500+1=501
Auto Decrement	499	-
Direct	600	800
Indirect	800	-
Relative Addressing	1007	850
Indexed Register	707	-
Base Register	706	900

8-14. A two-word instruction is stored in memory at an address designated by the symbol W. The address field of the instruction (stored at W + 1) is designated by the symbol Y. The operand used during the execution of the instruction is stored at an address symbolized by Z. An index register contains the value X. State how Z is calculated from the other addresses if the addressing mode of the instruction is

a. direct
b. indirect
c. relative
d. indexed

Z = Effective address

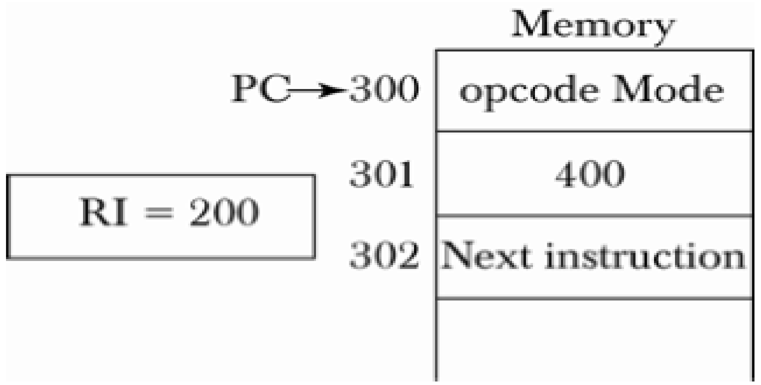
(a) Direct:	Z = Y
(b) Indirect:	Z = M[Y]
(c) Relative:	Z = Y + W + 2
(d) Indexed:	Z = Y + X



8-18. An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register *R*1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is (a) direct; (b) immediate; (c) relative; (d) register indirect; (e) index with *R*1 as the index register.

<u>Effective address</u>

(a) Direct: 400
(b) Immediate: 301
(c) Relative: 302 + 400 = 702
(d) Reg. Indirect: 200
(e) Indexed: 200 + 400 = 600



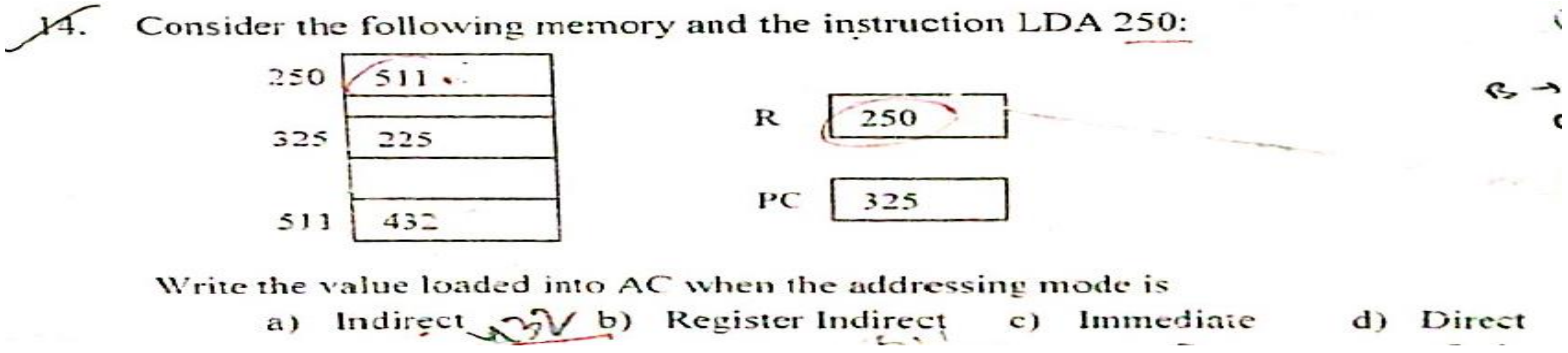
14. Consider the following figure :

PC =300
R1= 500

What is the value in AC if the instruction is
LDA 300 if the modes are:

i. Direct addressing 450
ii. immediate 300
iii. Indirect addressing 200
iv. Register indirect if LDA (R1) is used 100

Address	Memory
300	450
⋮	⋮
450	200
500	100



14. Consider the following memory and the instruction LDA 250:

250	511
325	225
511	432

R	250
PC	325

Write the value loaded into AC when the addressing mode is

a) Indirect b) Register Indirect c) Immediate d) Direct

Solution:

a) Indirect : 432

b)Register Indirect: 511

c) Immediate : 250

d) Direct : 511

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:

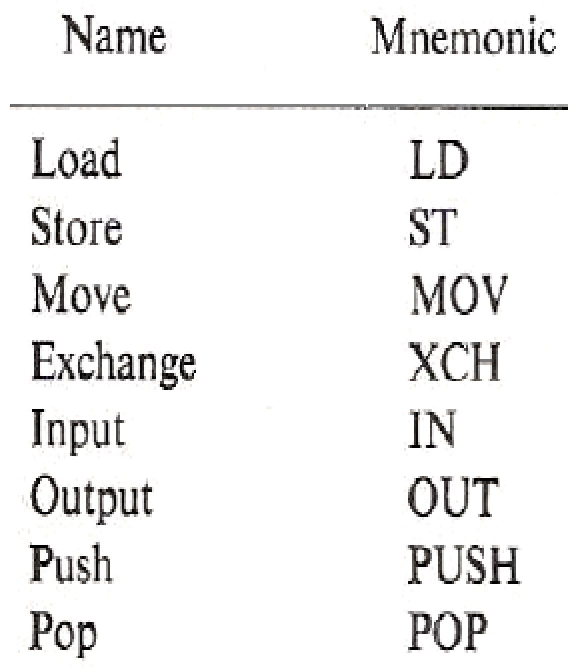
1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

Data Transfer Instructions:

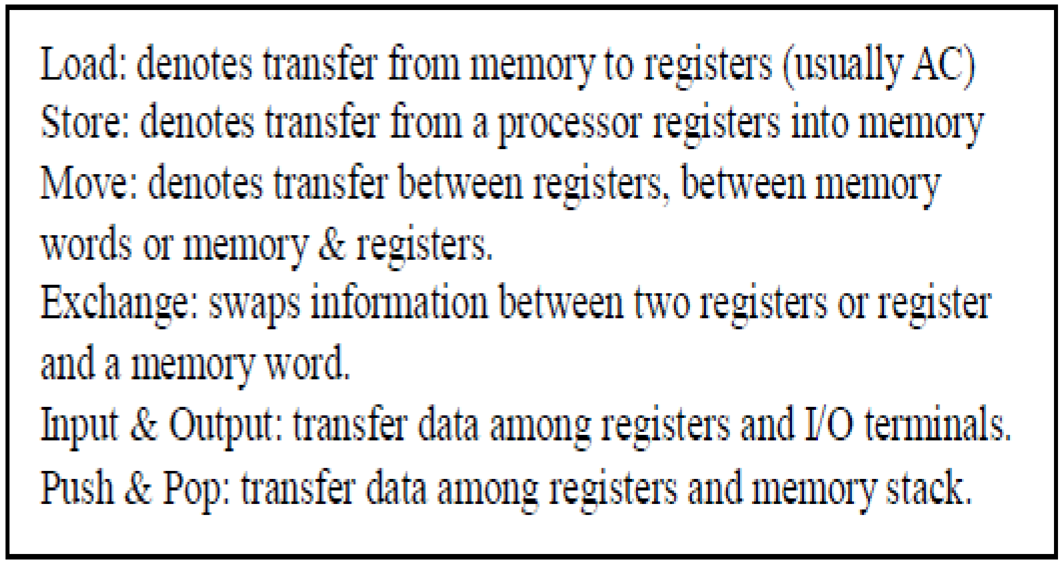
Data transfer instructions causes transfer of data from one location to another without modifying the binary information content. The most common transfers are:

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

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

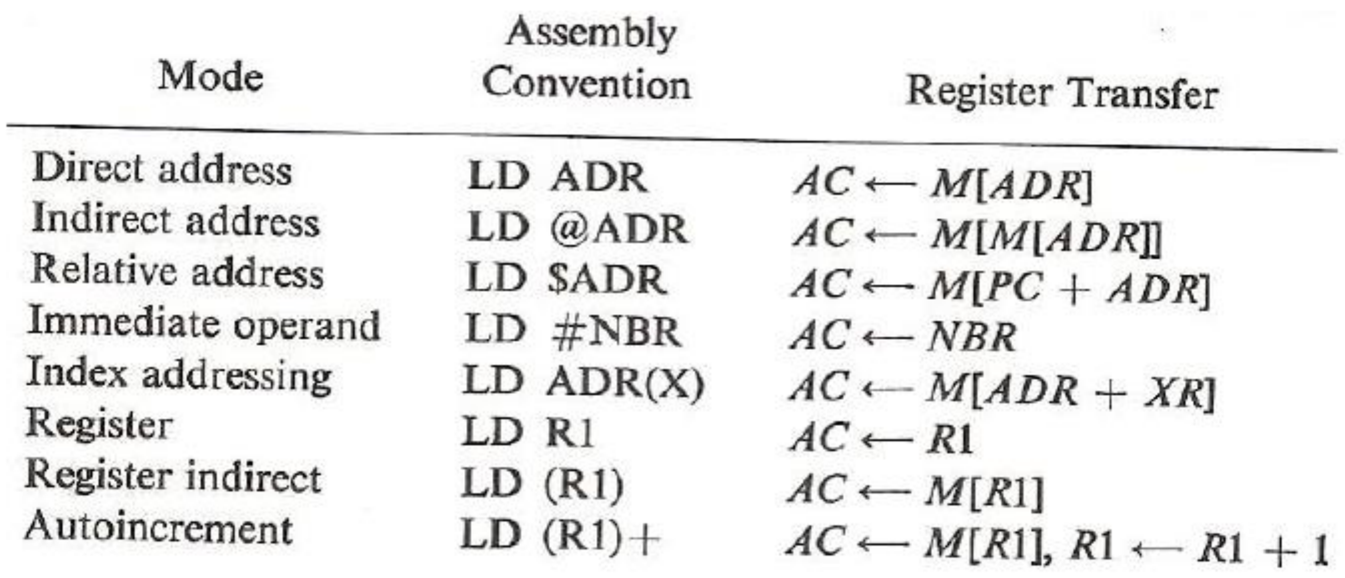


Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP



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.

- 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:



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

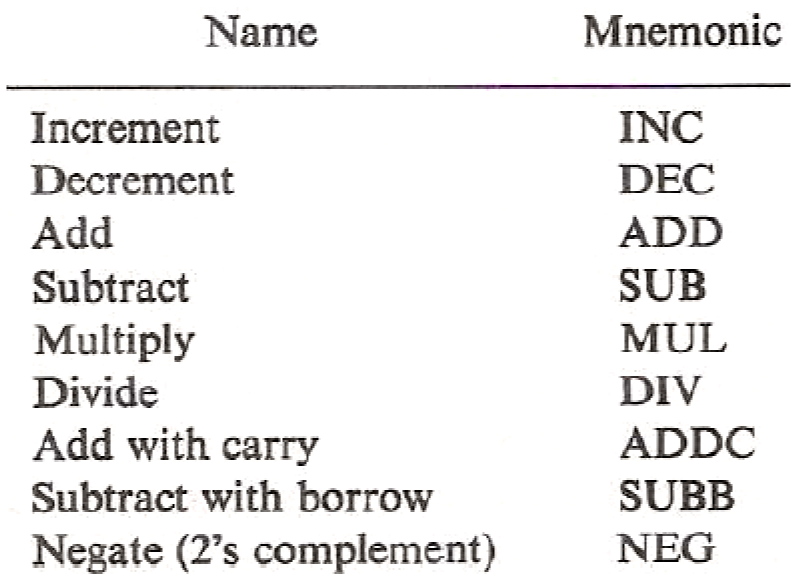
Data manipulation Instructions:

Data manipulation Instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.

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

Arithmetic instructions:

- Typical arithmetic instructions are listed below:

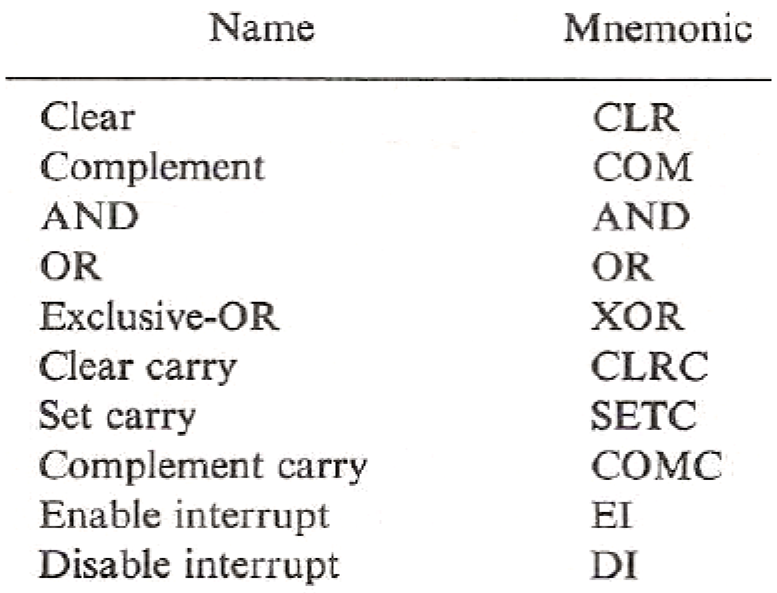


Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

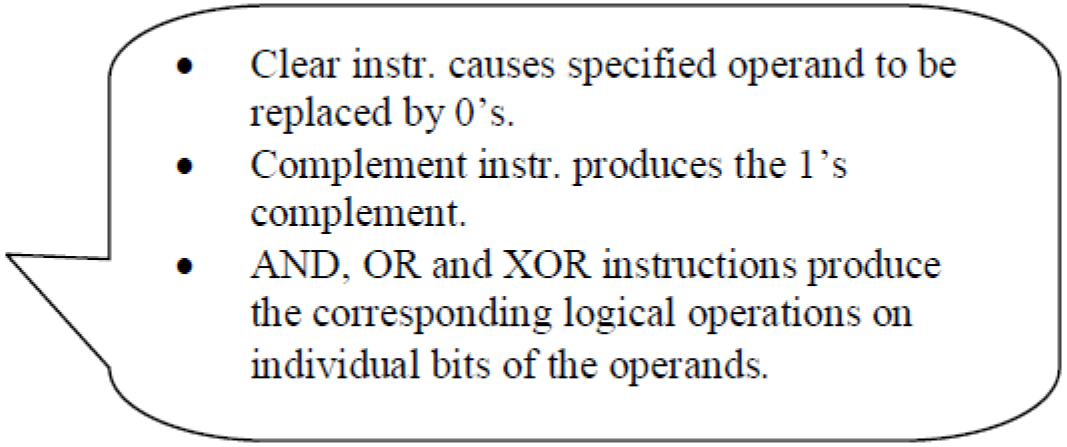
- Increment (decrement) instr. adds 1 to (subtracts 1 from) the register or memory word value.
- Add, subtract, multiply and divide instructions may operate on different data types (fixed-point or floating-point, binary or decimal).

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.

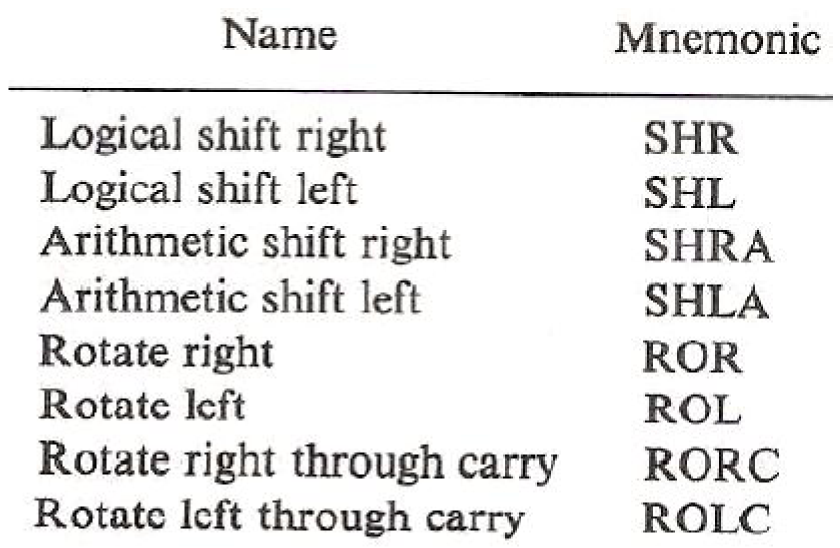
Shift instructions

- 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



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.

5.3 RISC and CISC characteristics

RISC (reduced instruction set computer) characteristics

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- The control unit is hardwired rather than micro programmed
- Relatively large number of registers in the processor unit
- Efficient instruction pipeline

The main concept of RISC is to reduce execution time by simplifying the instruction set of the computer. Example: MIPS(**Microprocessor without Interlocked Pipeline Stages**), ARM (Advanced RISC Machine)

- **CISC (complex instruction set computer) characteristics**

- A large number of instructions - typically from 100 to 250 instructions
- Some instructions that perform specialized tasks and are used infrequently
- A large variety of addressing modes – typically from 5 to 20 different modes
- Variable-length instruction format
- Uses memory to load and store instruction and operand as well
- Instructions that manipulate operands in memory

- Principle of CISC:

To provide single machine instruction for each statement that is written in a higher level language. One reason to provide a complex instruction set is to simplify the compilation and improve the overall computer performance.

Example: IBM computer, desktop computer, Digital equipment corporation.

Assignment: Write any 8 differences between RISC and CISC.

Reverse Polish Notation

A+B – Infix Notation

+AB – Prefix or Polish Notation

AB+ - Postfix or Reverse Polish Notation

Infix Notation:

the notation where the operator is in between the operands.

Prefix Notation:

the notation where the operator lies before the operand.

Postfix Notation:

the notation where the operator lies after the operand.

- Example:

A*B+C*D

RPN- AB*CD*+

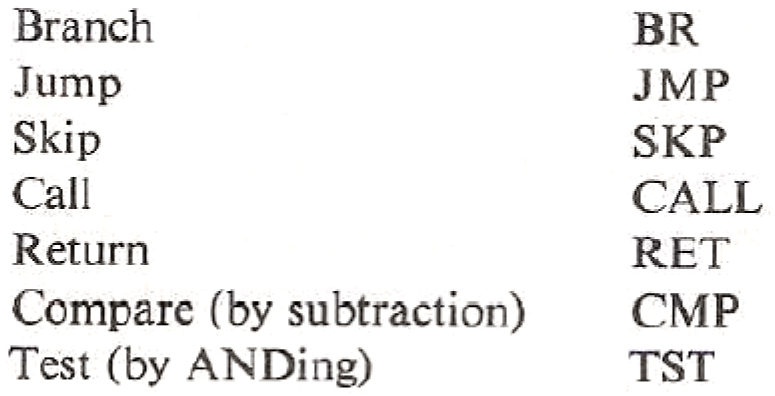
PN- +*AB*CD

(A+B)*[C*(D+E)+F]

RPN- AB+CDE+*F+*

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.

It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions can be stored for further analysis. Status bits are also called *condition-code* bits or *flag* bits. Figure 8-8 shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit C (carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.
2. Bit S (sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, $Z = 1$ if the output is zero and $Z = 0$ if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement
For the 8-bit ALU, $V = 1$ if the output is greater than $+127$ or less than -128.

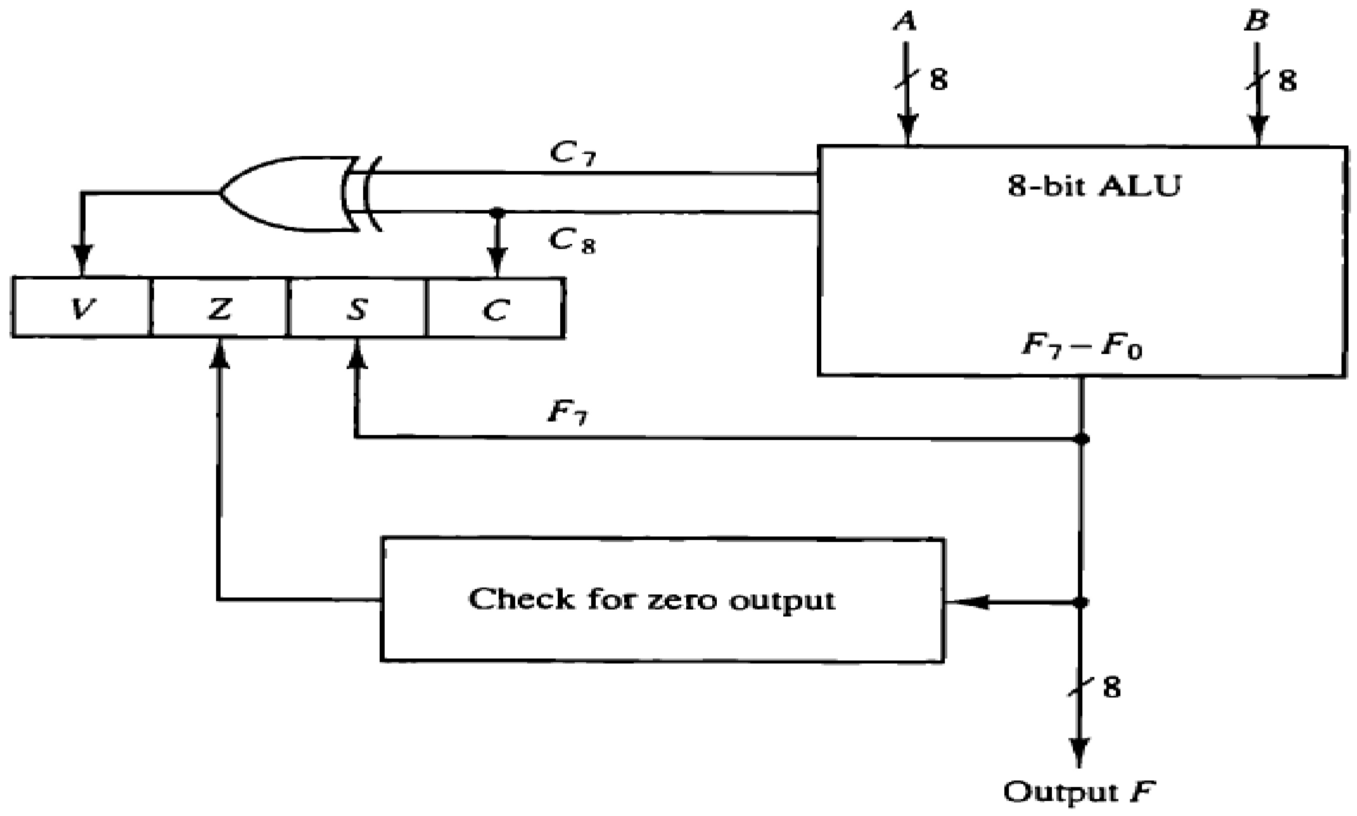


Figure 8-8 Status register bits.

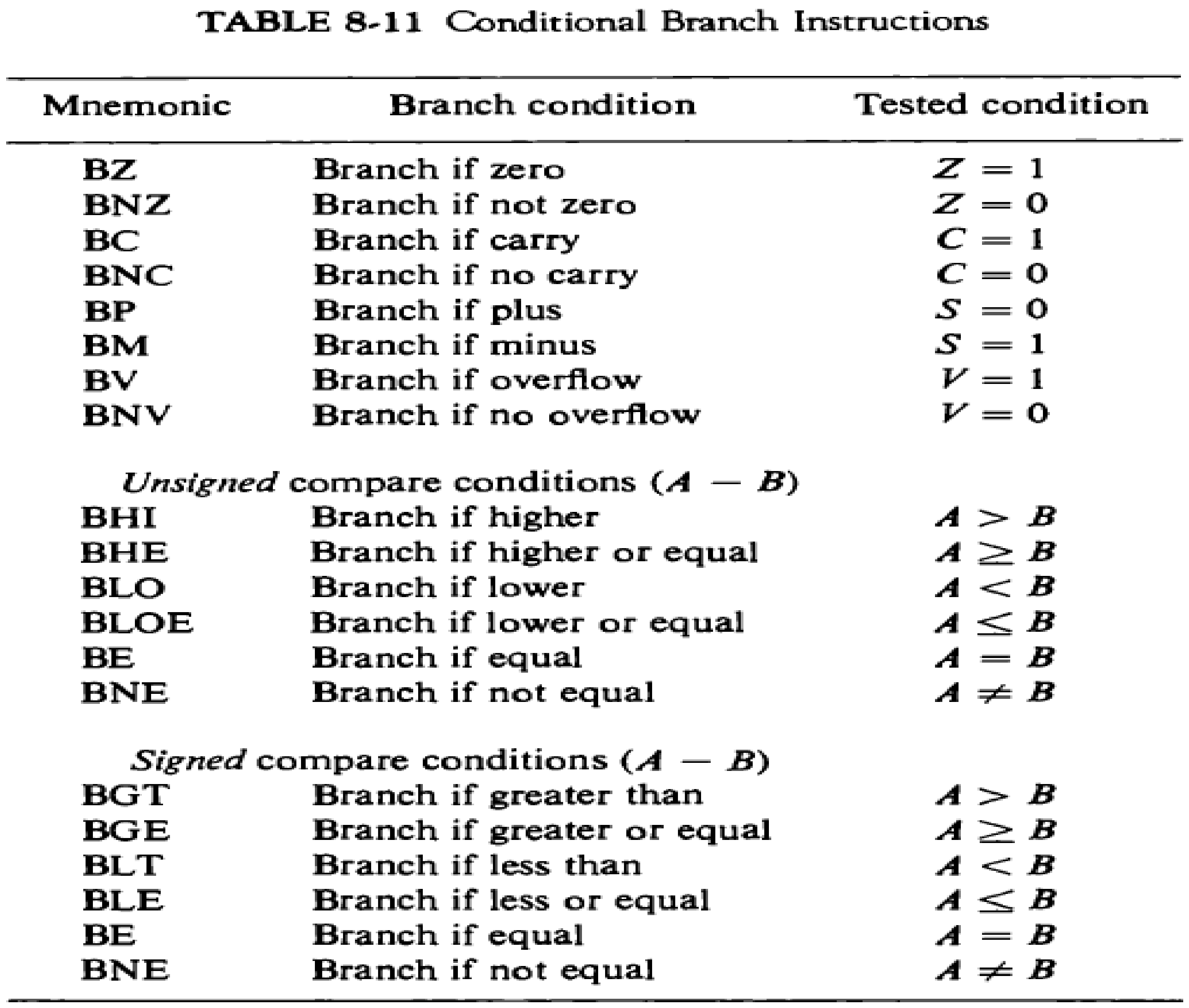
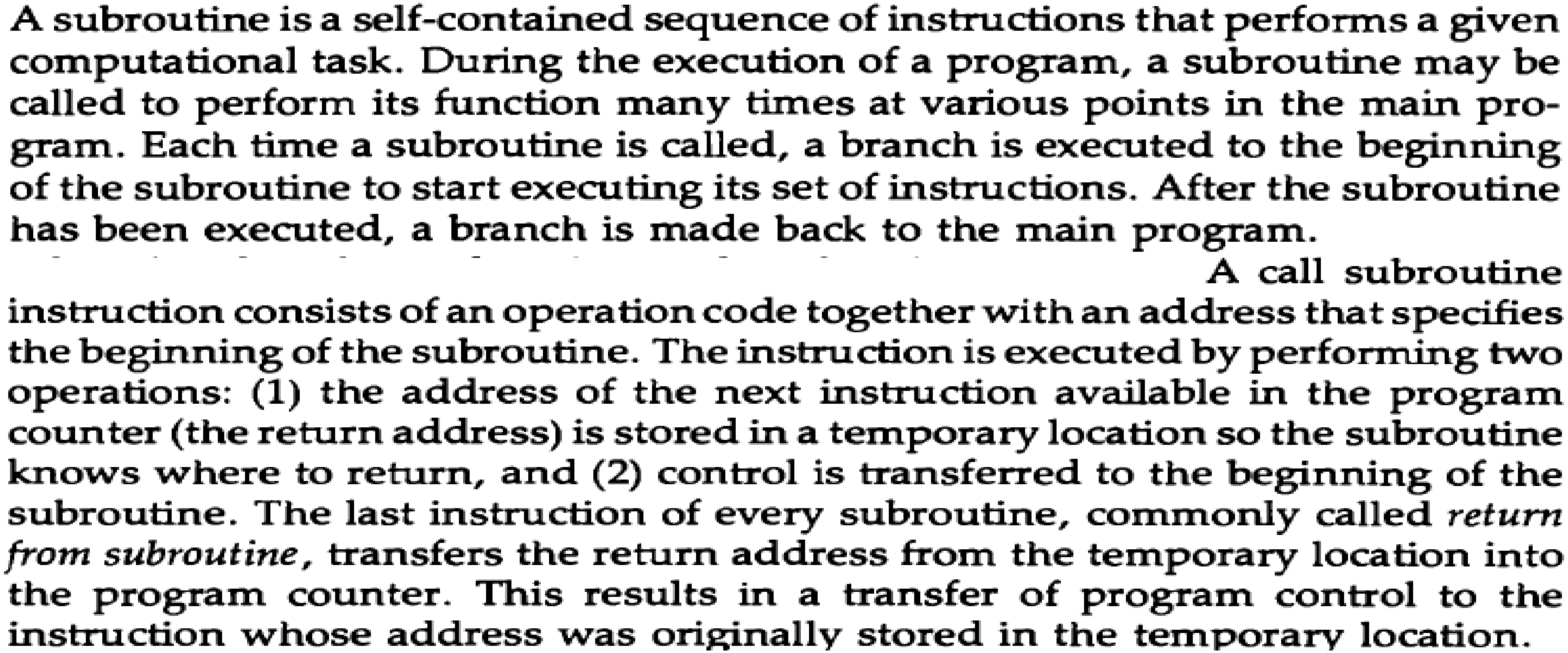


TABLE 8-11 Conditional Branch Instructions

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	$A \neq B$

Subroutine Call and Return

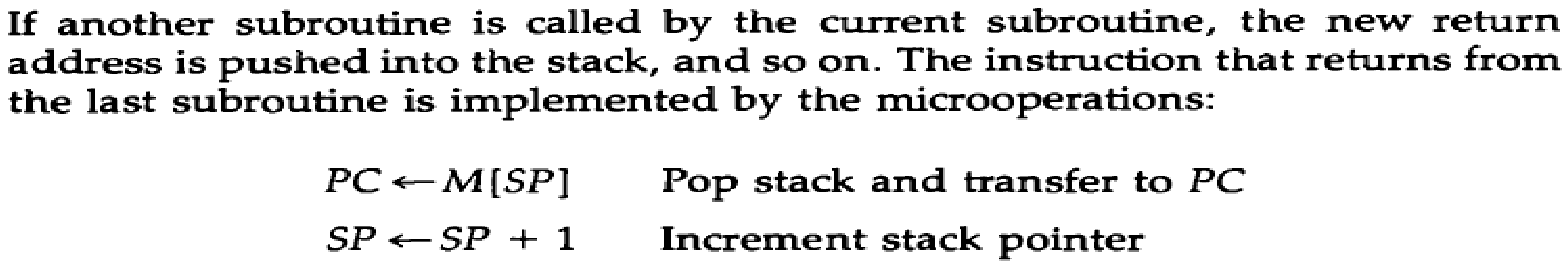


A subroutine is a self-contained sequence of instructions that performs a given computational task. During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program. Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a branch is made back to the main program.

A call subroutine instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two operations: (1) the address of the next instruction available in the program counter (the return address) is stored in a temporary location so the subroutine knows where to return, and (2) control is transferred to the beginning of the subroutine. The last instruction of every subroutine, commonly called *return from subroutine*, transfers the return address from the temporary location into the program counter. This results in a transfer of program control to the instruction whose address was originally stored in the temporary location.

Different computers use a different temporary location for storing the return address. Some store the return address in the first memory location of the subroutine, some store it in a fixed location in memory, some store it in a processor register, and some store it in a memory stack. The most efficient way is to store the return address in a memory stack. The advantage of using a stack for the return address is that when a succession of subroutines is called, the sequential return addresses can be pushed into the stack. The return from subroutine instruction causes the stack to pop and the contents of the top of the stack are transferred to the program counter. In this way, the return is always to the program that last called a subroutine. A subroutine call is implemented with the following microoperations:

$SP \leftarrow SP - 1$	Decrement stack pointer
$M[SP] \leftarrow PC$	Push content of PC onto the stack
$PC \leftarrow$ effective address	Transfer control to the subroutine



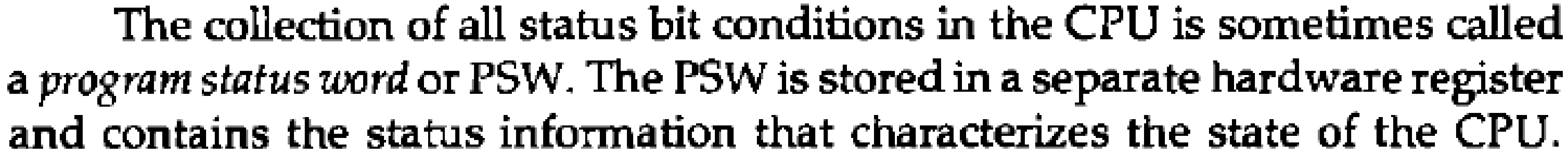
If another subroutine is called by the current subroutine, the new return address is pushed into the stack, and so on. The instruction that returns from the last subroutine is implemented by the microoperations:

$PC \leftarrow M[SP]$	Pop stack and transfer to PC
$SP \leftarrow SP + 1$	Increment stack pointer

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.

- The state of the CPU at the end of the execute cycle (when the interrupt is recognised) is determined from:
    1. The content of program counter
    2. The content of all processor registers
    3. The content of certain status conditions



The collection of all status bit conditions in the CPU is sometimes called a *program status word* or PSW. The PSW is stored in a separate hardware register and contains the status information that characterizes the state of the CPU.

Types of Interrupt

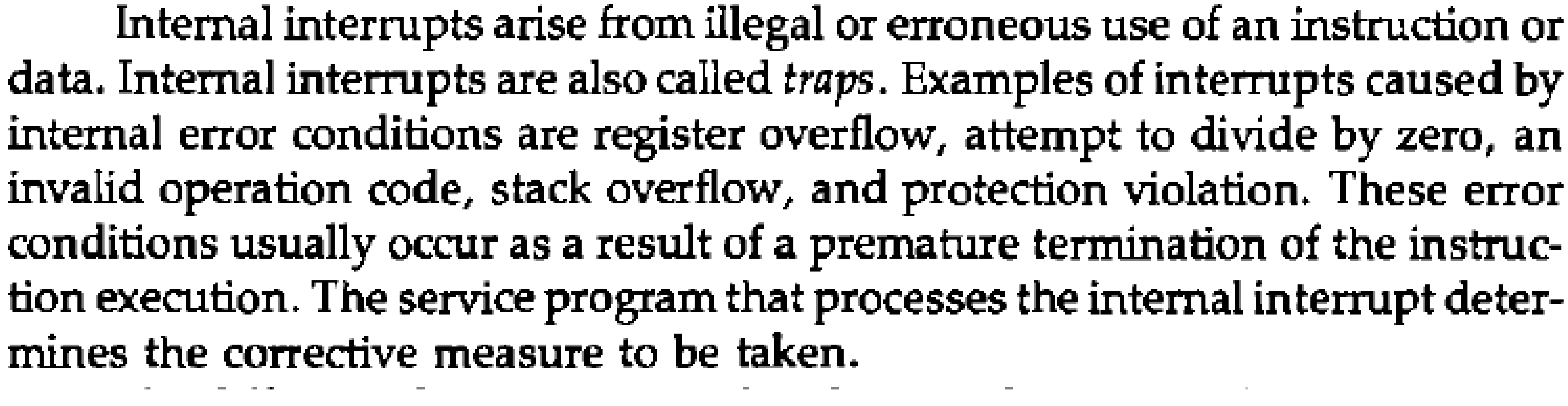
There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

1. External Interrupt

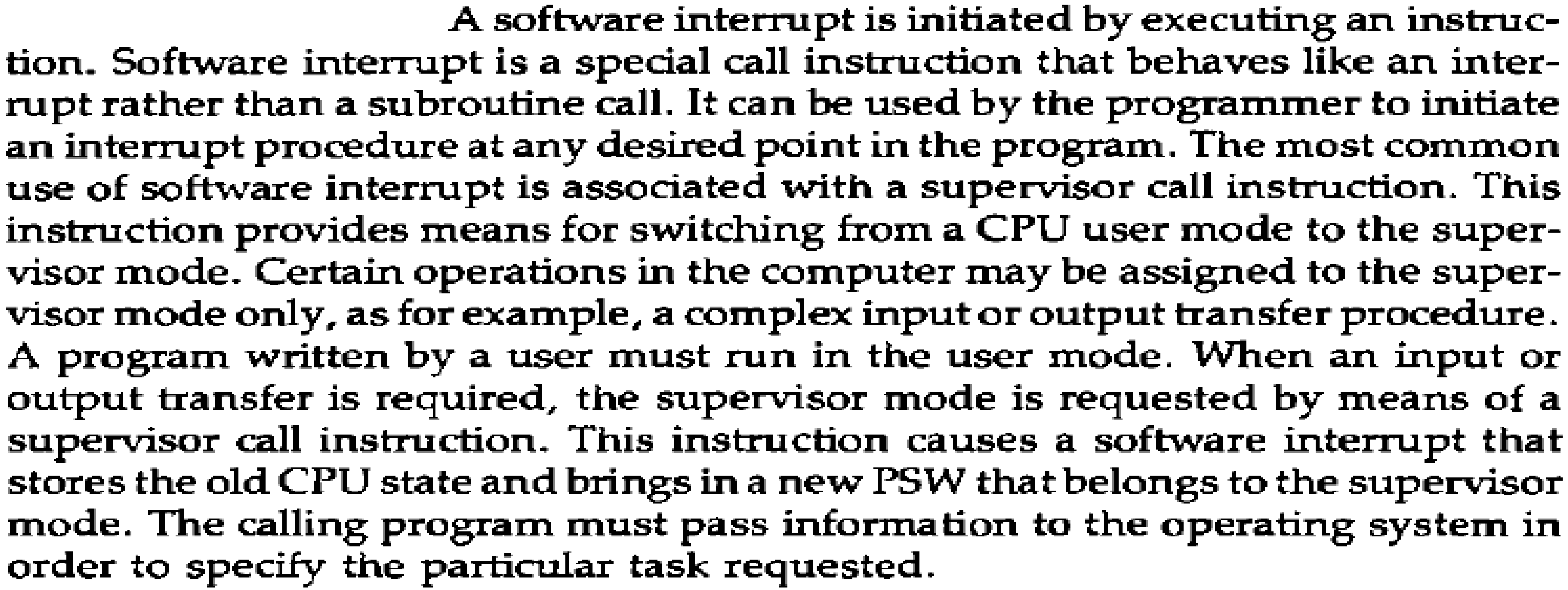
External interrupts come from input–output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its time allocation. Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a nondestructive memory in the few milliseconds before power ceases.

2. Internal Interrupt

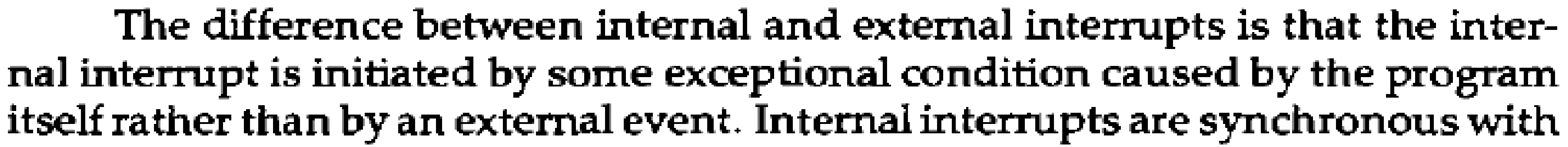


Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called *traps*. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.

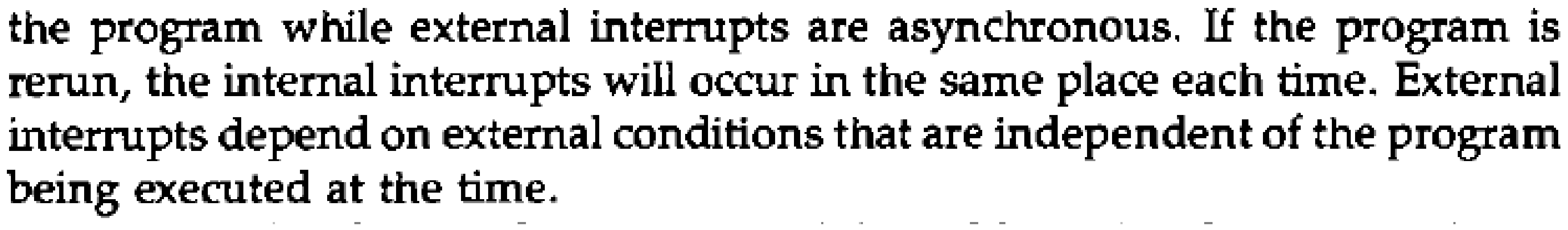
3. Software Interrupt



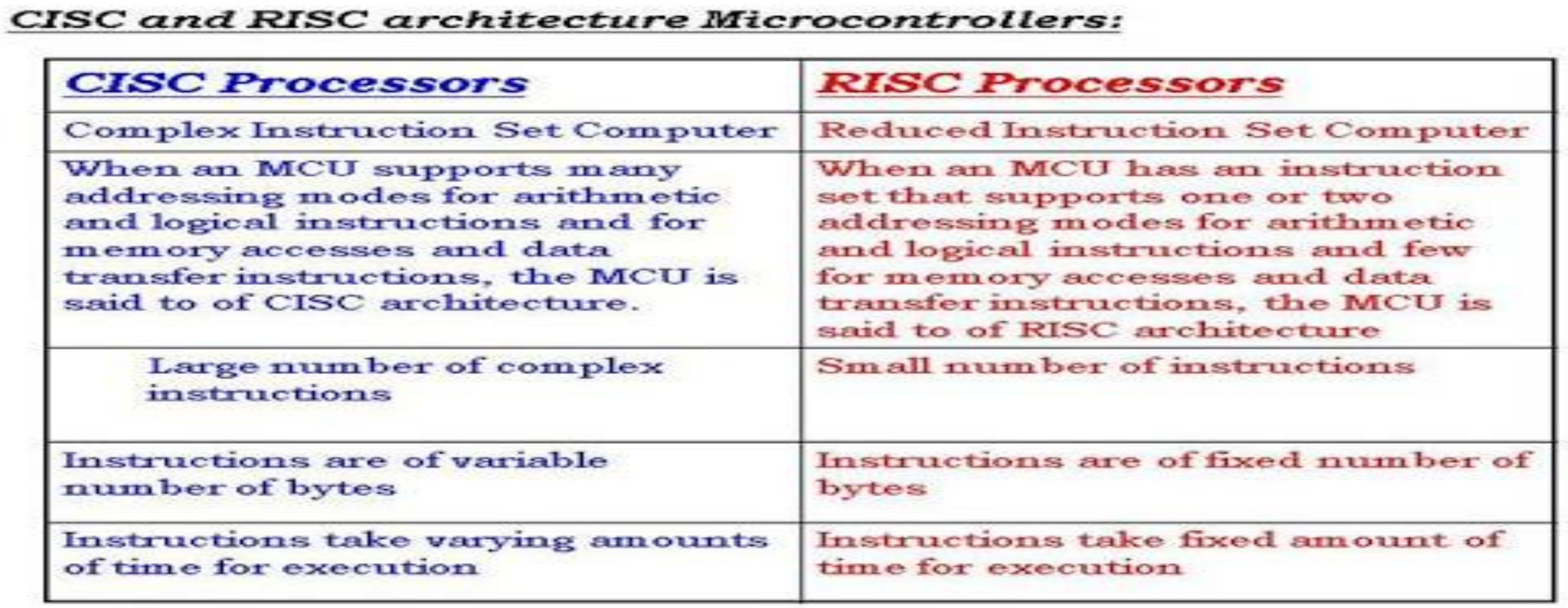
A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode. The calling program must pass information to the operating system in order to specify the particular task requested.



The difference between internal and external interrupts is that the internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event. Internal interrupts are synchronous with



the program while external interrupts are asynchronous. If the program is rerun, the internal interrupts will occur in the same place each time. External interrupts depend on external conditions that are independent of the program being executed at the time.



CISC and RISC architecture Microcontrollers:

CISC Processors	***RISC Processors***
Complex Instruction Set Computer	Reduced Instruction Set Computer
When an MCU supports many addressing modes for arithmetic and logical instructions and for memory accesses and data transfer instructions, the MCU is said to of CISC architecture.	When an MCU has an instruction set that supports one or two addressing modes for arithmetic and logical instructions and few for memory accesses and data transfer instructions, the MCU is said to of RISC architecture
Large number of complex instructions	Small number of instructions
Instructions are of variable number of bytes	Instructions are of fixed number of bytes
Instructions take varying amounts of time for execution	Instructions take fixed amount of time for execution

Timing and Control Unit

Control Unit

Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them. There are two types of control organization:

a) Hardwired Control

b) Microprogrammed Control

a) Hardwired Control

- ➢CU is made up of sequential and combinational circuits to generate the control signals.
- ➢If logic is changed, we need to change the whole circuit.
- ➢Expensive
- ➢Fast

b) Microprogrammed Control

- A control memory on the processor contains microprograms that activate the necessary control signals.
- If logic is changed, we only need to change the microprogram.
- Cheap
- Slow

Control Unit of a Basic Computer (Hardwired Control)

The block diagram of a hardwired control unit is shown below. It consists of two decoders, a sequence counter, and a number of control logic gates.

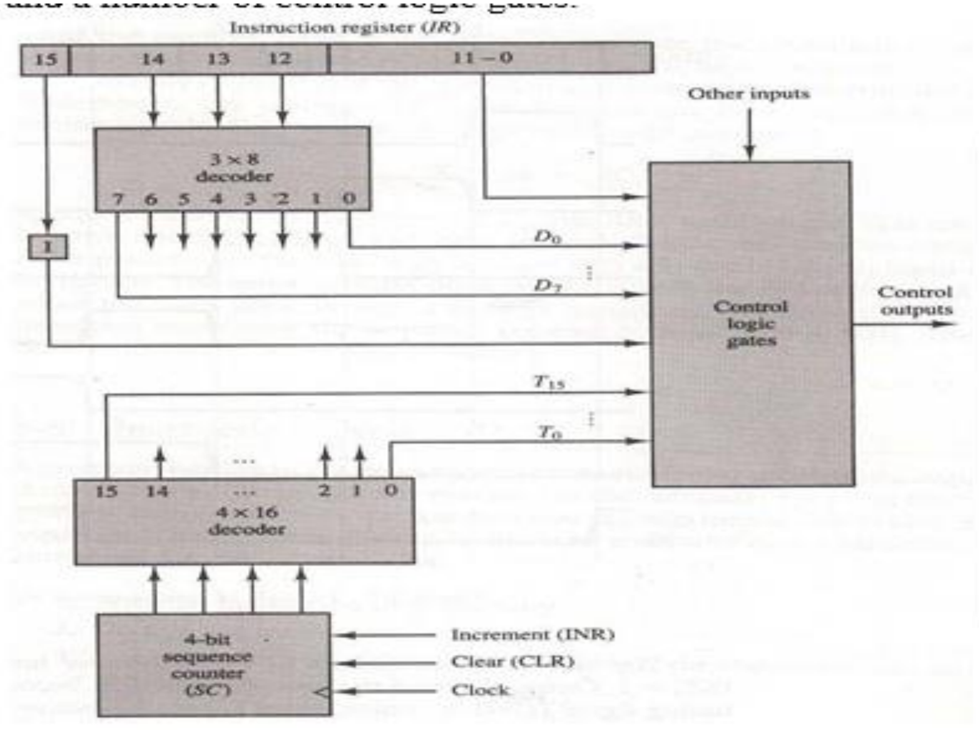


Fig: Control unit of a basic computer

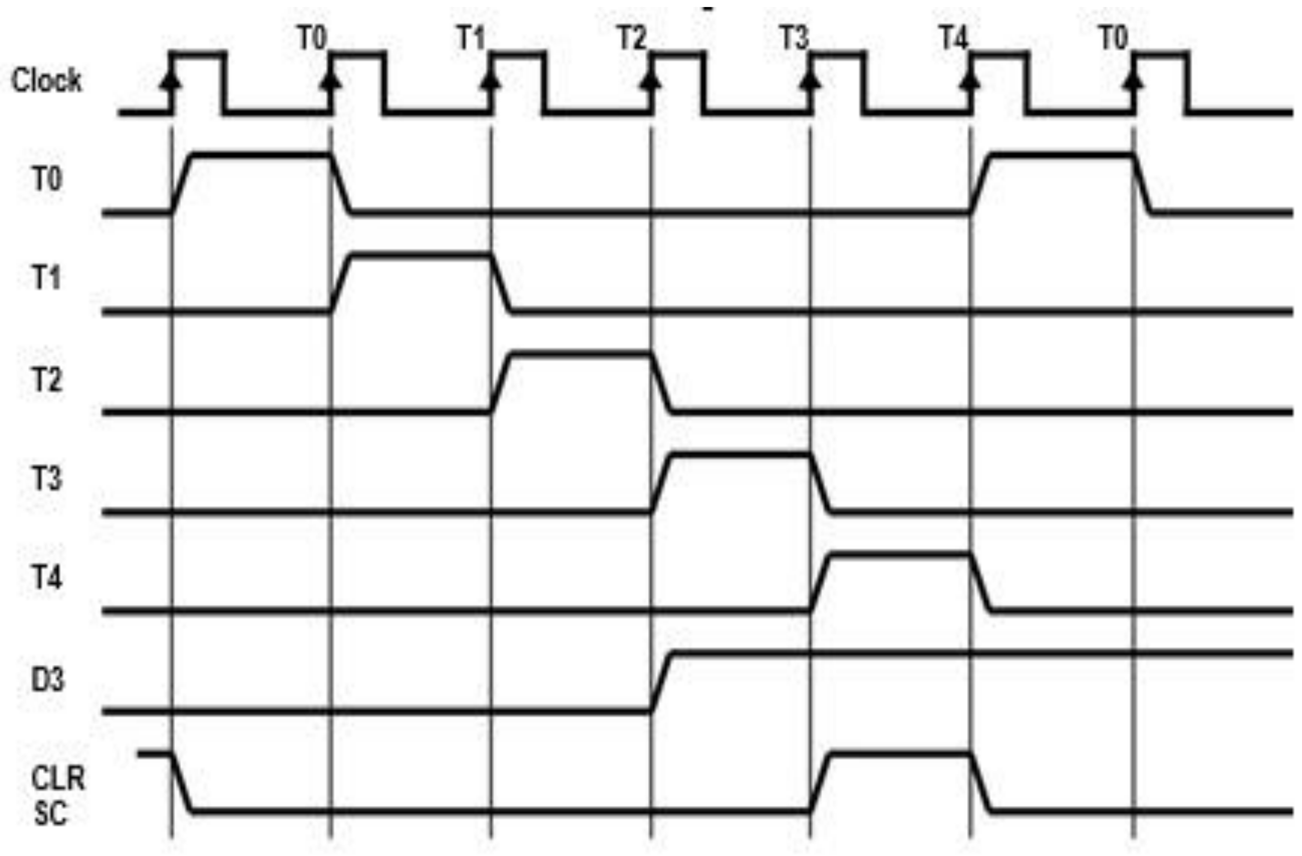
Mechanism:

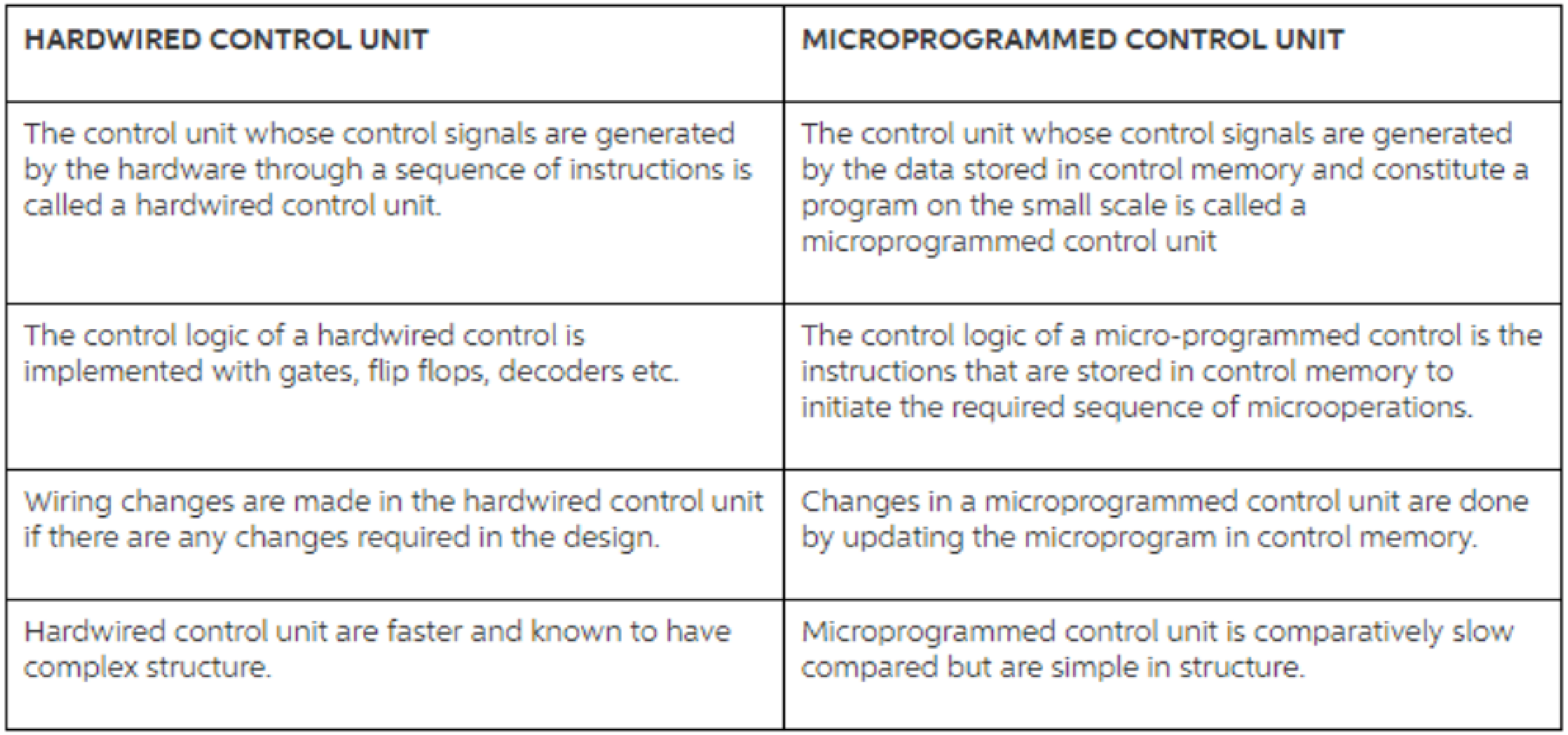
- An instruction read from memory is placed in the instruction resister (IR) where it is decoded into three parts: **I** bit, **operation code** and bits **0 through 11**.
- The operation code bit is decoded with 3 x 8 decoder producing 8 outputs D_0 through D_7.
- Bit 15 of the instruction is transferred to a flip-flop I.
- And operand bits are applied to control logic gates.
- The 16 outputs of 4-bit sequence counter (SC) are decoded into 16 timing signals T_0 through T_{15}.

This means instruction cycle of basic computer cannot take more than 16.

Timing Signal

- Generated by 4-bit sequence counter and 4x16 decoder.
- The SC can be incremented or cleared.
- Example: T_0, T_1, T_2, T_3, T_4, T_0, T_1 . . .
- Assume: At time T4, SC is cleared to 0 if decoder output D3 is active:
- D_3T_4: SC $\leftarrow$ 0





HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
The control unit whose control signals are generated by the hardware through a sequence of instructions is called a hardwired control unit.	The control unit whose control signals are generated by the data stored in control memory and constitute a program on the small scale is called a microprogrammed control unit
The control logic of a hardwired control is implemented with gates, flip flops, decoders etc.	The control logic of a micro-programmed control is the instructions that are stored in control memory to initiate the required sequence of microoperations.
Wiring changes are made in the hardwired control unit if there are any changes required in the design.	Changes in a microprogrammed control unit are done by updating the microprogram in control memory.
Hardwired control unit are faster and known to have complex structure.	Microprogrammed control unit is comparatively slow compared but are simple in structure.

- The function of the control unit in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite. The complexity of the digital system is derived from the number of sequences of microoperations that are performed. Two techniques used for implementing control unit are hardwired and microprogrammed.
- **Hardwired Control:**

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

- **Microprogrammed Control:**

Microprogramming is a second alternative for designing the control unit of a digital computer which uses microoperations sequences.

A computer that employs a microprogrammed control unit will have two separate memories: a main memory and a control memory.

Control Memory

- **Control Memory (Control Storage: CS):** Storage in the microprogrammed control unit to store the microprogram.
- **Control word:** It is a string of control variables (0's and 1's) occupying a word in control memory.
- **Microprogram:**

➢Program stored in control memory that generates all the control signals required to execute the instruction set correctly

➢Consists of microinstructions

- **Microinstruction:**

➢Contains a control word and a sequencing word

➢Control Word – contains all the control information required for one clock cycle

➢Sequencing Word - Contains information needed to decide the next microinstruction address

- **Microoperation:**

➢A microinstruction contains one or more microoperations to be completed.

- **Writable Control Memory (Writable Control Storage: WCS):**

CS whose contents can be modified:

➢Microprogram can be changed

➢Instruction set can be changed or modified

A computer that employs a microprogrammed control unit will have two separate memories: main memory and a control memory. The user’s program in main memory consists of machine instructions and data whereas control memory holds a fixed microprogram that cannot be altered by the user. Each machine instruction initiates a series of microinstructions in control memory.

The general configuration of a microprogrammed control unit is demonstrated in the following block diagram:

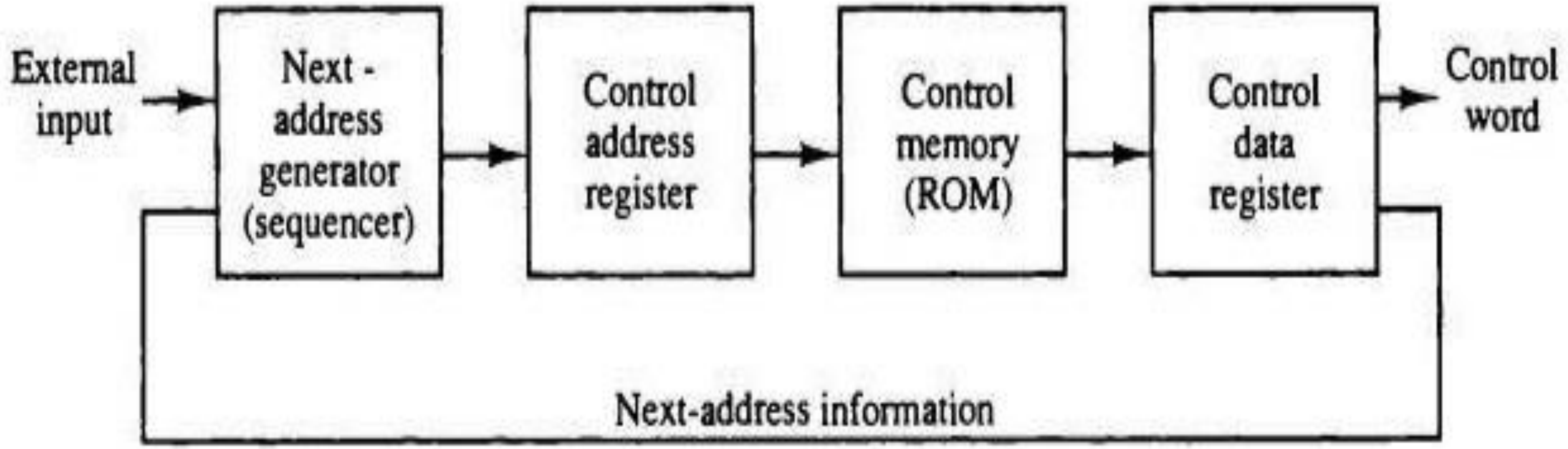


Fig: Microprogrammed control organization

Dynamic Microprogramming

- ➢A more advanced development that permits a microprogram to be loaded initially from an auxiliary memory such as magnetic disk.
- ➢Computer system whose control unit is implemented with a microprogram in WCS.
- ➢Microprogram can be changed by a systems programmer or a user.

- **Sequencer:** The device or program that generates address of next microinstruction to be executed is called sequencer. While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

- **Control Address Register:** CAR contains address of microinstruction.
- **Control Data Register:** CDR contains microinstruction read from memory. The microinstruction contains a control word that specifies one or more microoperations. The data register is sometimes called a *pipeline register.*

It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

Address Sequencing

Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction. Process of finding address of next microinstruction to be executed is called ***address sequencing***. The address sequencing capabilities required in a control memory are:

a) Incrementing of the control address register.

b) Unconditional branch or conditional branch, depending on status bit conditions.

c) A mapping process from the bits of the instruction to an address for control memory.

d) A facility for subroutine call and return.

Following is the block diagram for control memory and the associated hardware needed for selecting the next microinstruction address.

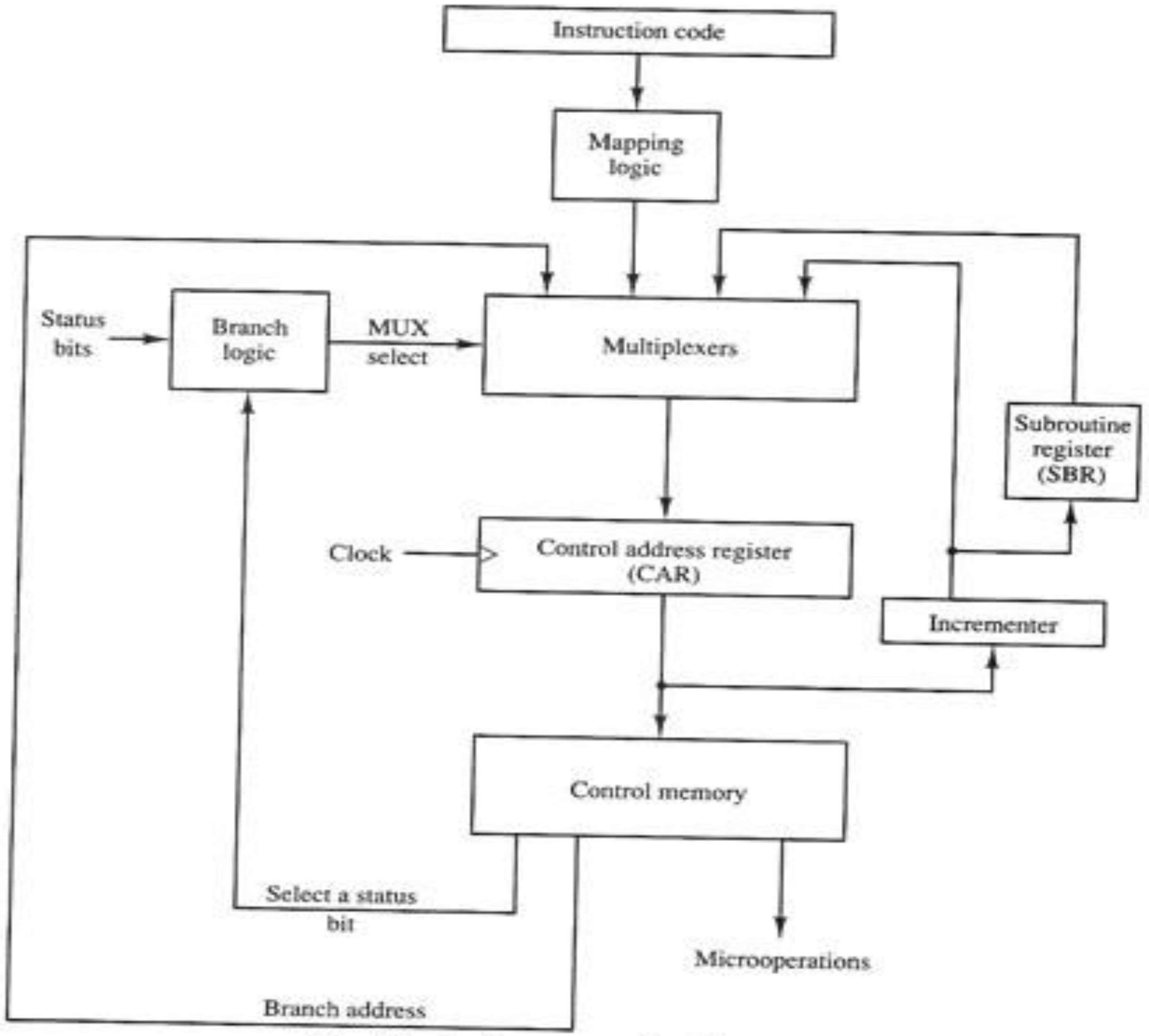


Fig: Block diagram of address sequencer.

The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence. Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine.

Control address register receives address of next microinstruction from different sources.

a) Incrementer simply increments the address by one
b) In case of branching, branch address is specified in one of the field of microinstruction.
c) In case of subroutine call, return address is stored in the register SBR which is used when returning from called subroutine.
d) A mapping process from the bits of the instruction to an address for control memory.

- **Conditional Branch**:

Simplest way of implementing branch logic hardware is to test the specified condition and branch to the indicated address if condition is met otherwise address register is simply incremented. If Condition is true, hardware set the appropriate field of status register to 1. Conditions are tested for O (overflow), N (negative), Z (zero), C (carry), etc.

- **Unconditional Branch:**

Fix the value of one status bit at the input of the multiplexer to 1. So that, branching can always be done.

- **Mapping**

Assuming operation code of 4-bits which can specify 16 (2^4) distinct instructions. Assume further and control memory has 128 words, requiring an address of 7-bits. Now we have to map 4-bit operation code into 7-bit control memory address. Thus, we have to map Op-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its subroutine in memory.

- **Approach of direct mapping:**

Transfer Op-code bits to use it as an address of control memory. In this mapping, one 0 is placed in the MSB and two 0s in the LSB as shown in figure:

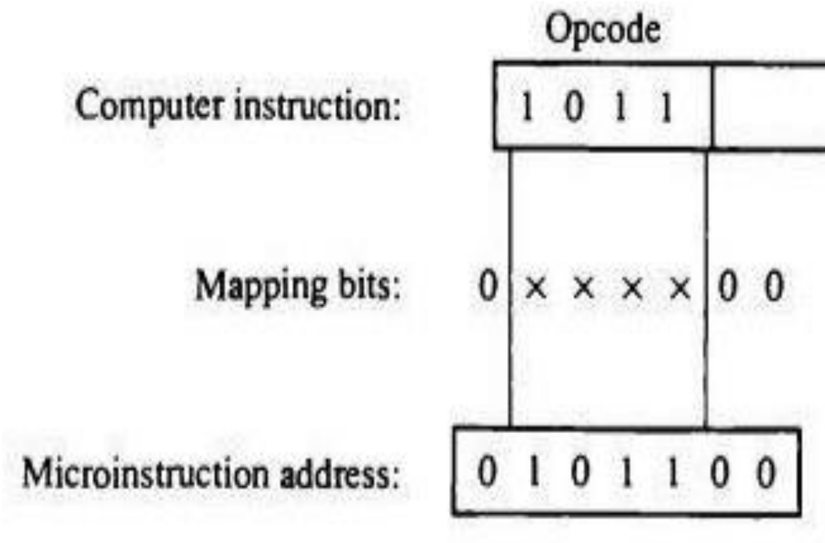


Fig: mapping from instruction code to microinstruction address

- **Subroutines**:

Subroutines are programs that are used by another program to accomplish a particular task. Microinstructions can be saved by employing subroutines that use common sections of micro code. Example: the sequence of microoperations needed to generate the effective address is common to all memory reference instructions. Thus, this sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Subroutine register is used to save a return address during a subroutine call which is organized in LIFO (last in, first out) stack.

Microprogram Example

Once the configuration of a computer and its microprogrammed control unit is established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming and is a process similar to conventional machine language programming.

Computer Configuration:

- It consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the microprogram
- Four registers are associated with the processor unit and two with the control unit. The processor registers are PC, AR, DR and AC.
- The control unit has control address register CAR and subroutine register SBR.
- The transfer of information among the registers in processor is done through multiplexer rather than a common bus. DR can receive information from AC, PC or memory. AR can receive information from PC or DR. PC can receive information only from AR.
- The arithmetic, logic and shift unit performs microoperations with data from AC and DR and places the result in AC. Note that memory receives its address from AR. Input data written to memory come from DR, and data read from memory can go only to DR.
- The computer instruction format has three fields: a 1-bit field for indirect addressing symbolized by I, a 4-bit operation code (op-code), and an 11-bit address field. The figure below lists four of the 16 possible memory reference instructions.

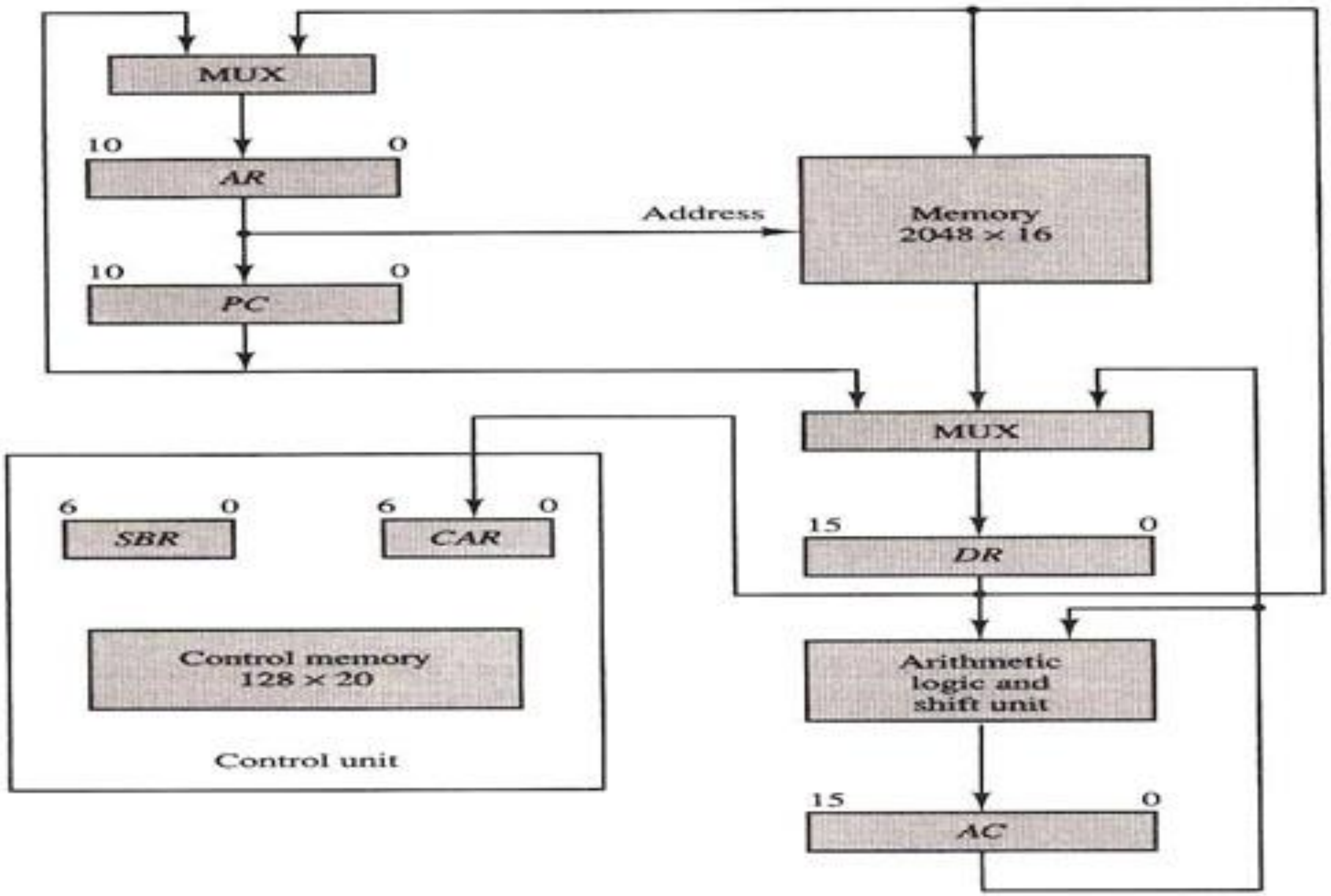
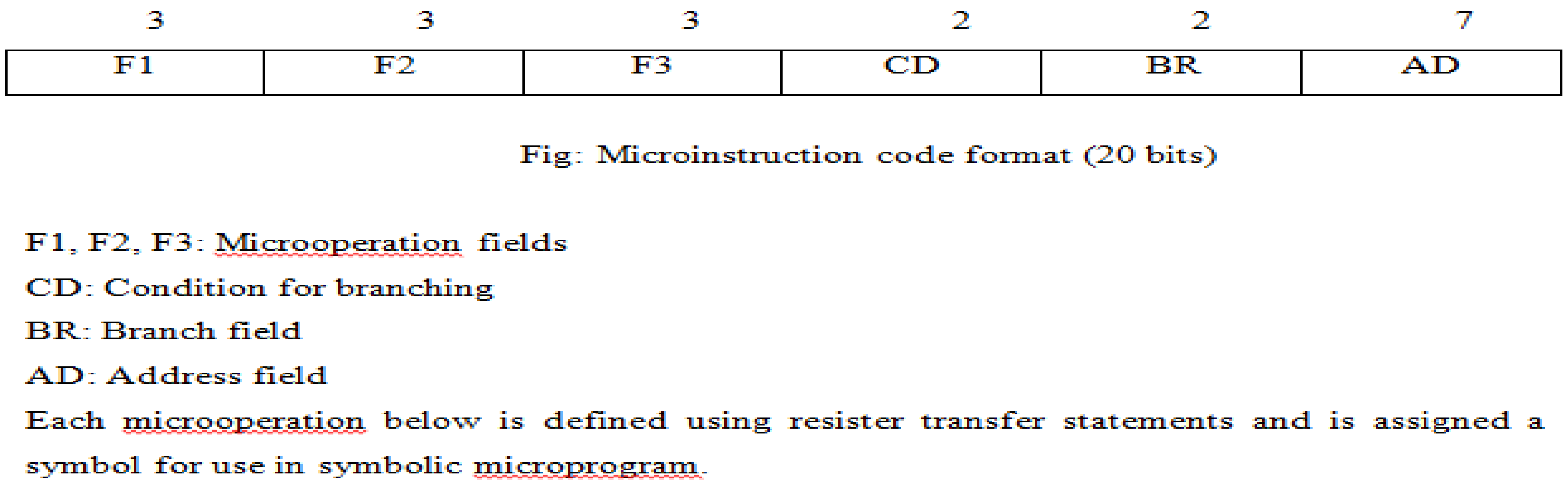


Fig: Computer hardware configuration

Microinstruction Format and Description

We know the computer instruction format (explained in previous chapter) for different set of instruction in main memory. Similarly, microinstruction in control memory has 20-bit format divided into 4 functional parts as shown below.



3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

Fig: Microinstruction code format (20 bits)

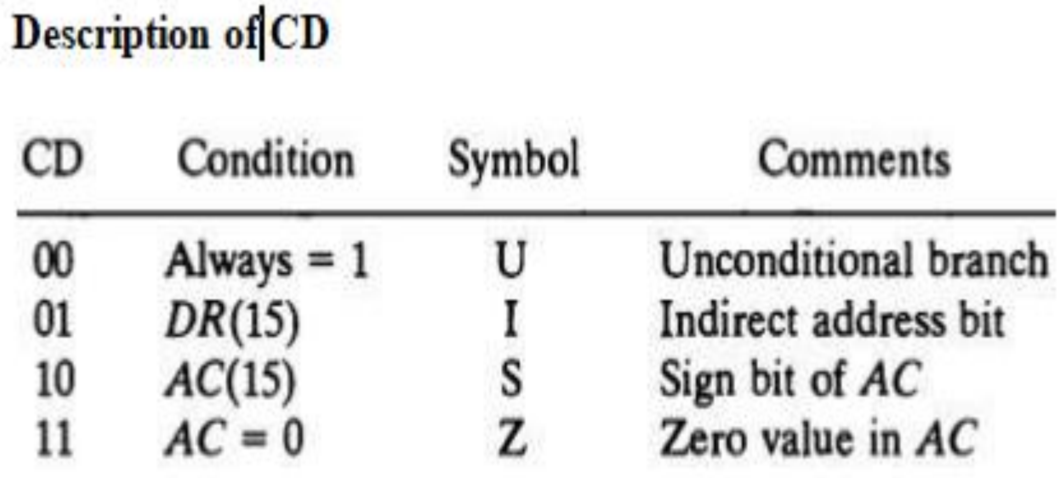
F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

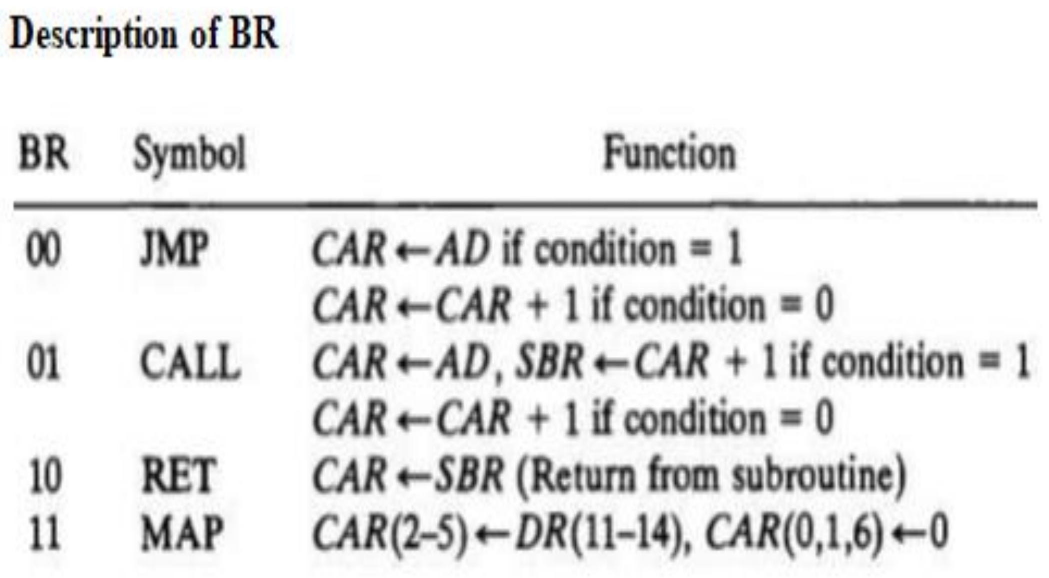
AD: Address field

Each microoperation below is defined using resister transfer statements and is assigned a symbol for use in symbolic microprogram.



Description of CD

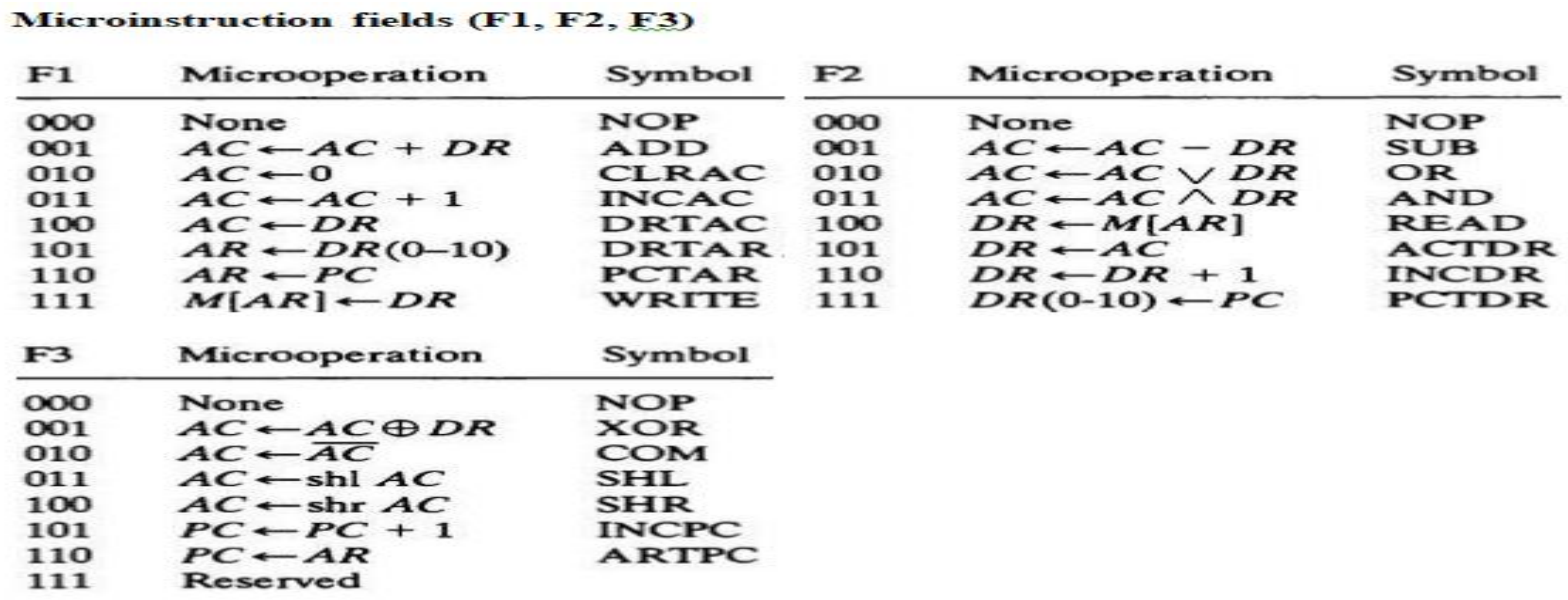
CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	*DR*(15)	I	Indirect address bit
10	*AC*(15)	S	Sign bit of *AC*
11	*AC* = 0	Z	Zero value in *AC*



Description of BR

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

CD (condition) field consists of two bits representing 4 status bits and BR (branch) field (2-bits) used together with address field AD, to choose the address of the next microinstruction.



Microinstruction fields (F1, F2, F3)

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Here, microoperations are subdivided into three fields of 3-bits each. These 3 bits are used to encode 7 different microoperations. No more than 3 microoperations can be chosen for a microinstruction, one for each field. If fewer than 3 microoperations are used, one or more fields will contain 000 for no operation.

Symbolic Microinstructions

Symbols are used in microinstructions as in assembly language. A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.

Format of Microinstruction:

Contains five fields: label; micro-ops; CD; BR; AD

Label: may be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z},

Where

U: Unconditional Branch

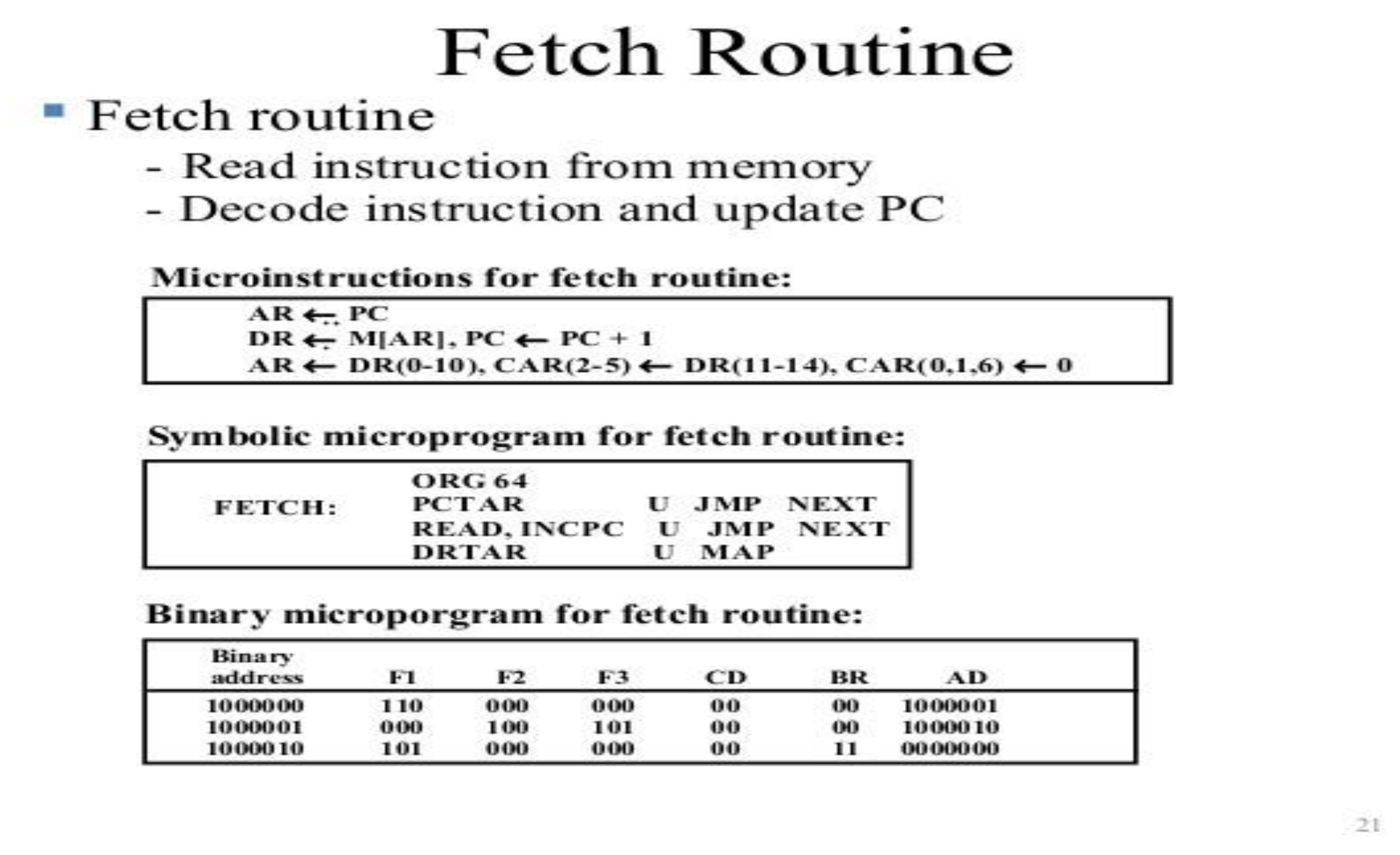
I: Indirect address bit

S: Sign of AC

Z: Zero value in AC

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty (in case of MAP and RET)}



Fetch Routine

- Fetch routine
  - Read instruction from memory
  - Decode instruction and update PC

Microinstructions for fetch routine:

$AR \leftarrow PC$
$DR \leftarrow M[AR], PC \leftarrow PC + 1$
$AR \leftarrow DR(0\text{-}10), CAR(2\text{-}5) \leftarrow DR(11\text{-}14), CAR(0,1,6) \leftarrow 0$

Symbolic microprogram for fetch routine:

```
            ORG 64
FETCH:      PCTAR            U   JMP   NEXT
            READ, INCPC      U   JMP   NEXT
            DRTAR            U   MAP
```

Binary microporgram for fetch routine:

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000