UNIT - I

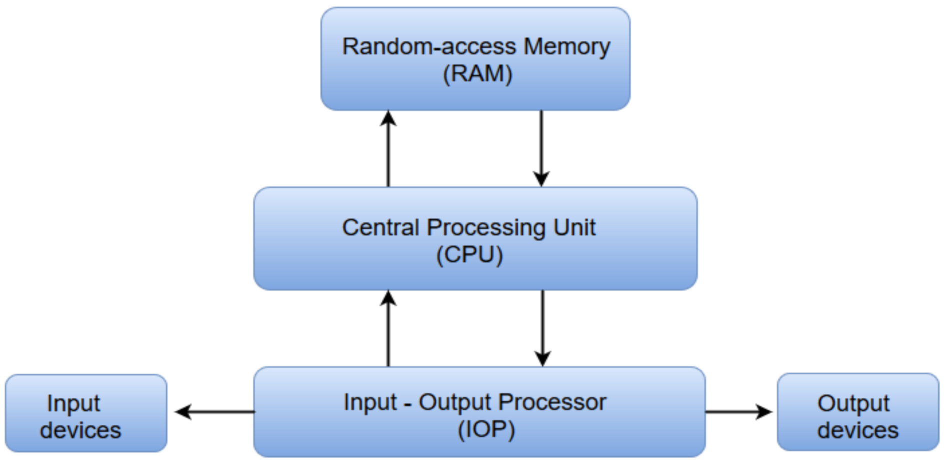
COMPUTER ORGANIZATION AND ARCHITECTURE

1.1 Digital Computers:

- A Digital computer can be considered as a digital system that performs various computational tasks.
- The first electronic digital computer was developed in the late 1940s and was used primarily for numerical computations.
- By convention, the digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.
- A computer system is subdivided into two functional entities: Hardware and Software.
- The hardware consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.
- The software of the computer consists of the instructions and data that the computer manipulates to perform various data-processing tasks.

BLOCK DIAGRAM OF DIGITAL COMPUTER:

Block diagram of a digital computer:


Random-access Memory (RAM)
Central Processing Unit (CPU)
Input devices
Input - Output Processor (IOP)
Output devices

- The Central Processing Unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and a control circuit for fetching and executing instructions.
- The memory unit of a digital computer contains storage for instructions and data.
- The Random-Access Memory (RAM) for real-time processing of the data.
- The Input-Output devices for generating inputs from the user and displaying the final results to the user.
- The Input-Output devices connected to the computer include the keyboard, mouse, terminals, magnetic disk drives, and other communication devices.

COMPUTER ORGANIZATION:

- **Computer organization** is concerned with the way the hardware components operate and the way they are connected together to form the computer system.
- The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate as intended.

COMPUTER DESIGN:

- Computer design is concerned with the hardware design of the computer.
- Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.
- Computer design is concerned with the determination of what hardware should be used and how the parts should be connected.
- This aspect of computer hardware is sometimes referred to as computer implementation.

COMPUTER ARCHITECTURE:

- **Computer architecture** is concerned with the structure and behavior of the computer as seen by the user.
- It includes the information formats, the instruction set, and techniques for addressing memory.
- The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system
- Computer Architecture is the field of study of selecting and interconnecting hardware components to create computers that satisfy functional performance and cost goals.
- It refers to those attributes of the computer system that are visible to a programmer and have a direct effect on the execution of a program.
- Computer Architecture concerns Machine Organization, interfaces, application, technology, measurement & simulation that Includes:
  - Instruction set
  - Data formats
  - Principle of Operation (formal description of every operation)
  - Features (organization of programmable storage, registers used, interrupts mechanism, etc.)
- It is the combination of Instruction Set Architecture, Machine Organization and the related hardware.

1.2 Register Transfer Language and Micro Operations:

- Register is a very fast computer memory, used to store data/instruction in-execution.
- A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information. An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.
- A register consists of a group of flip-flops and gates. The flip-flops hold the binary information and gates control when and how new information is transferred into a register. Various types of registers are available commercially. The simplest register is one that consists of only flip-flops with no external gates.

These days registers are also implemented as a register file.

Loading the Registers

The transfer of new information into a register is referred to as loading the register. If all the bits of register are loaded simultaneously with a common clock pulse than the loading is said to be done in parallel.

Register Transfer Language:

- The symbolic notation used to describe the micro-operation transfers amongst registers is called **Register transfer language**.
- The term **register transfer** means the availability of **hardware logic circuits** that can perform a stated micro-operation and transfer the result of the operation to the same or another register.
- The word **language** is borrowed from programmers who apply this term to programming languages. This programming language is a procedure for writing symbols to specify a given computational process.

Following are some commonly used registers:

1. **Accumulator**: This is the most common register, used to store data taken out from the memory.
2. **General Purpose Registers**: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.
3. **Special Purpose Registers**: Users do not access these registers. These registers are for Computer system,

- **MAR:** Memory Address Register are those registers that holds the address for memory unit.
- **MBR:** Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
- **PC:** Program Counter points to the next instruction to be executed.
- **IR:** Instruction Register holds the instruction to be executed.

Register Transfer

Information transferred from one register to another is designated in symbolic form by means of replacement operator.

R2 ← R1

It denotes the transfer of the data from register R1 into R2.

Normally we want the transfer to occur only in predetermined control condition. This can be shown by following **if-then** statement: if (P=1) then (R2 ← R1)

Here P is a control signal generated in the control section.

Control Function

A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:

P: R2 ← R1

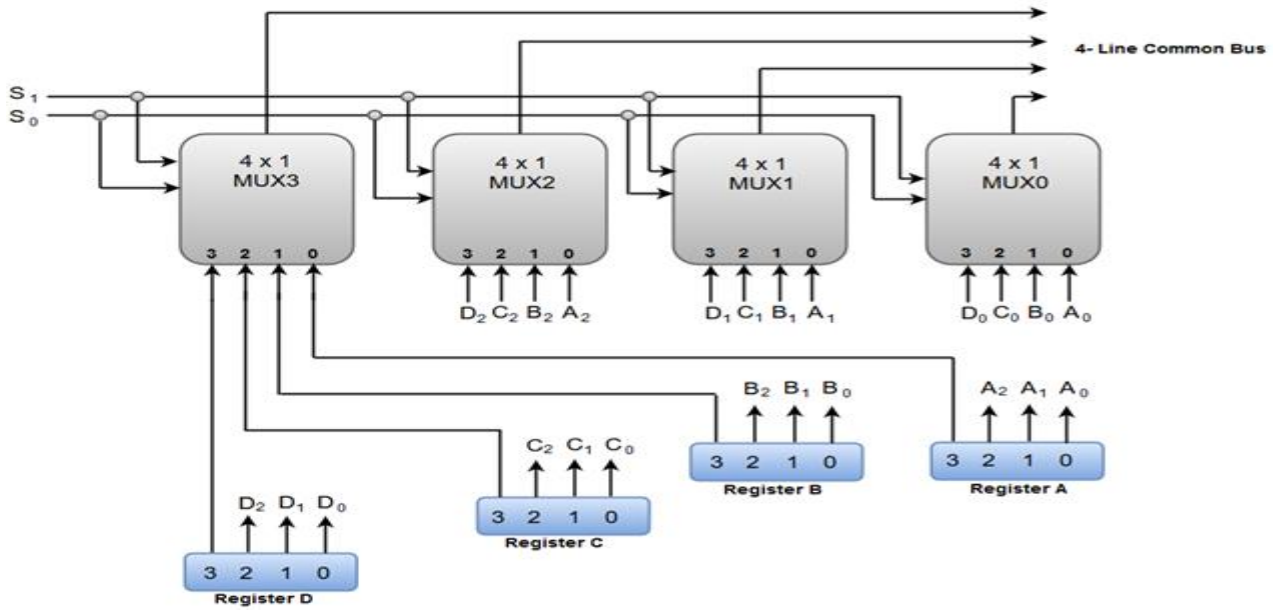
The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

Bus and Memory Transfers

- A digital system composed of many registers, and paths must be provided to transfer information from one register to another.
- The number of wires connecting all of the registers will be excessive if separate lines are used between each register and all other registers in the system.
- A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.
- A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during a particular register transfer.
- The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

We have used labels to make it more convenient for you to understand the input-output configuration of a Bus system for four registers. For instance, output 1 of register A is connected to input 0 of MUX1.

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers.

The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus.

This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

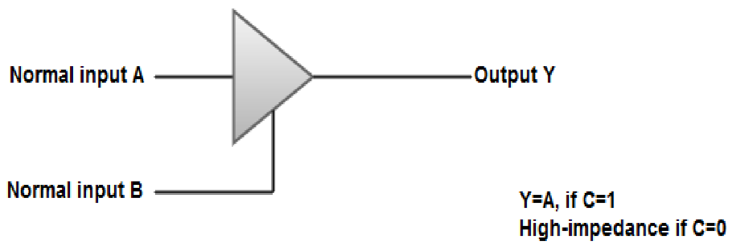
The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

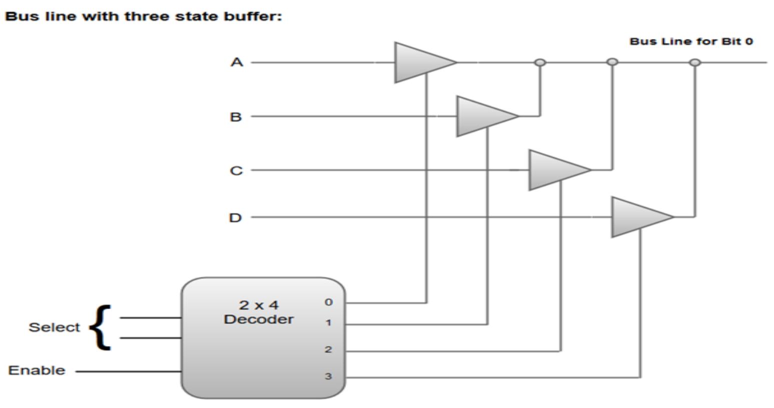
Note: The number of multiplexers needed to construct the bus is equal to the number of bits in each register. The size of each multiplexer must be 'k * 1' since it multiplexes 'k' data lines. For instance, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

- A bus system can also be constructed using **three-state gates** instead of multiplexers.
- The **three state gates** can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state.
- The most commonly used three state gates in case of the bus system is a **buffer gate**.

The graphical symbol of a three-state buffer gate can be represented as:



The following diagram demonstrates the construction of a bus system with three-state buffers.



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

Memory Transfer

Most of the standard notations used for specifying operations on memory transfer are stated below.

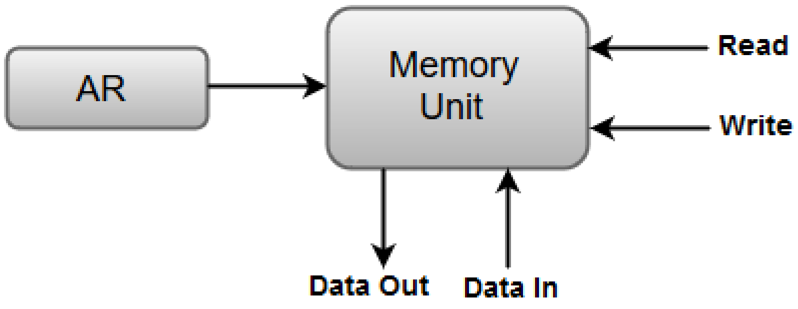
- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.
- Thus, a read operation can be stated as:

1. Read: DR ← M [AR]

- The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
- And the corresponding write operation can be stated as:

1. Write: M [AR] ← R1

- The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



Micro-Operations

The operations executed on data stored in registers are called micro-operations.

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

Example: Shift, count, clear and load.

Types of Micro-Operations

The micro-operations in digital computers are of 4 types:

1. Register transfer micro-operations transfer binary information from one register to another.
2. Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operations perform bit manipulation operation on non-numeric data stored in registers.
4. Shift micro-operations perform shift micro-operations performed on data.

Arithmetic Micro-Operations

Some of the basic micro-operations are addition, subtraction, increment and decrement.

***Add Micro-Operation*:**

It is defined by the following statement:

```
R3 → R1 + R2
```

The above statement instructs the data or contents of register R1 to be added to data or content of register R2 and the sum should be transferred to register R3.

Subtract Micro-Operation:

Let us again take an example:

```
R3 → R1 + R2' + 1
```

In subtract micro-operation, instead of using minus operator we take **1's compliment** and add 1 to the register which gets subtracted, i.e **R1 - R2** is equivalent to **R3 → R1 + R2' + 1**

Increment/Decrement Micro-Operation:

Increment and decrement micro-operations are generally performed by adding and subtracting 1 to and from the register respectively.

```
R1 → R1 + 1

R1 → R1 – 1
```

Symbolic Designation	**Description**
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 – 1	Decrement the contents of R1 by 1.

Logic Micro-Operations:

These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.

P: R1 ← R1 X-OR R2

In the above statement we have also included a Control Function.

Assume that each register has 3 bits. Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

$$\begin{array}{l} 010 \rightarrow \text{R1} \\ \underline{100} \rightarrow \text{R2} \\ \underline{110} \rightarrow \text{R1 after P=1} \end{array}$$

Shift Micro-Operations:

These are used for serial transfer of data. That means we can shift the contents of the register to the left or right. In the **shift left** operation the serial input transfers a bit to the right most position and in **shift right** operation the serial input transfers a bit to the left most position.

There are three types of shifts as follows:

a) Logical Shift

It transfers 0 through the serial input. The symbol **"shl"** is used for logical shift left and **"shr"** is used for logical shift right.

R1 ← she R1

R1 ← she R1

The register symbol must be same on both sides of arrows.

b) Circular Shift

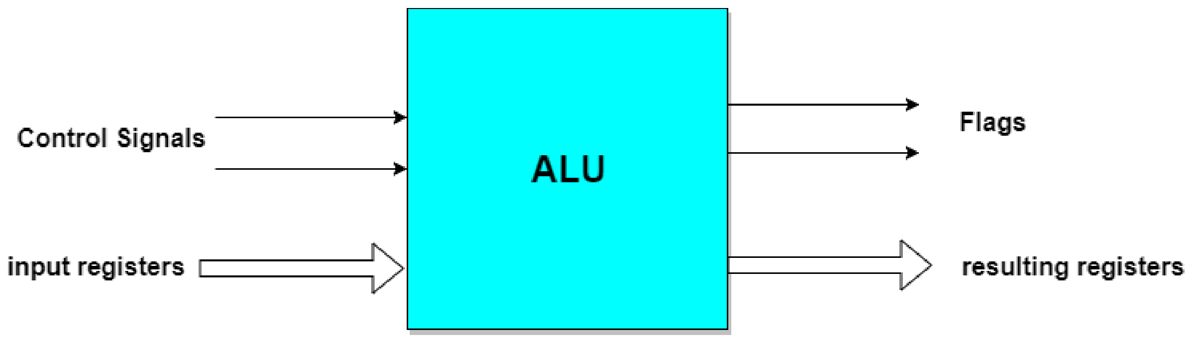
This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively.

c) Arithmetic Shift

This shifts a signed binary number to left or right. An **arithmetic shift left** multiplies a signed binary number by 2 and **shift left** divides the number by 2. Arithmetic shift micro-operation leaves the sign bit unchanged because the signed number remains same when it is multiplied or divided by 2.

Arithmetic Logical Unit :

- Instead of having individual registers performing the micro-operations, computer system provides a number of registers connected to a common unit called as Arithmetic Logical Unit (ALU).
- ALU is the main and one of the most important unit inisde CPU of computer.
- All the logical and mathematical operations of computer are performed here.
- The contents of specific register is placed in the in the input of ALU. ALU performs the given operation and then transfer it to the destination register.



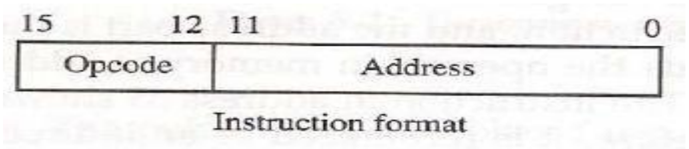
1.3 BASIC COMPUTER ORGANIZATION AND DESIGN

CONTENTS:

- ✓ Instruction Codes
- ✓ Computer Registers
- ✓ Computer Instructions
- ✓ Timing And Control
- ✓ Instruction Cycle
- ✓ Register – Reference Instructions
- ✓ Memory – Reference Instructions
- ✓ Input – Output And Interrupt

1. Instruction Codes:

- ➢ The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
- ➢ Internal organization of a computer is defined by the sequence of micro-operations it performs on data stored in its registers.
- ➢ Computer can be instructed about the specific sequence of operations it must perform.
- ➢ User controls this process by means of a Program.
- ➢ ***Program:*** set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- ➢ ***Instruction****:* a binary code that specifies a sequence of micro-operations for the computer.
- ➢ The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. – *Instruction Cycle*
- ➢ **Instruction Code:** group of bits that instruct the computer to perform specific operation.
- ➢ Instruction code is usually divided into two parts: Opcode and address(operand)



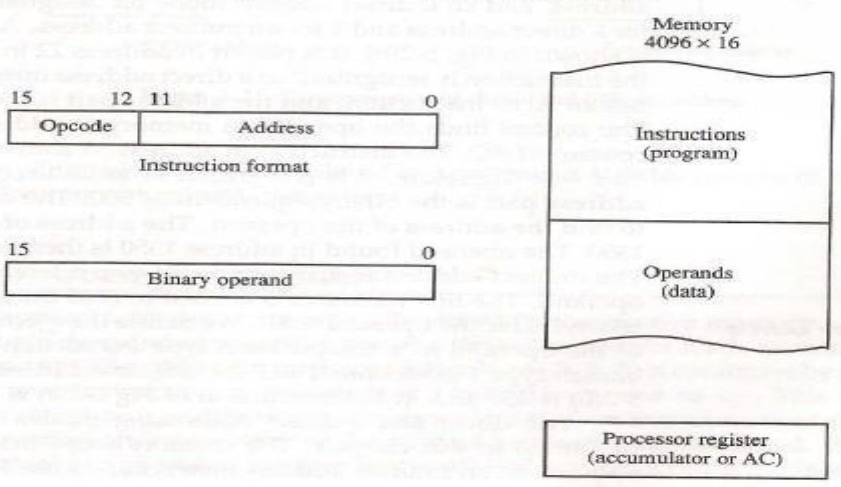
Instruction format

- ➢ *Operation Code (opcode):*
  - ✓ group of bits that define the operation
  - ✓ Eg: add, subtract, multiply, shift, complement.
  - ✓ No. of bits required for opcode depends on no. of operations available in computer.
  - ✓ n bit opcode >= 2^n (or less) operations
- ➢ Address (operand):
  - ✓ specifies the location of operands (registers or memory words)
  - ✓ Memory words are specified by their address
  - ✓ Registers are specified by their k-bit binary code
  - ✓ k-bit address >= 2^k registers

Stored Program Organization:

- The ability to store and execute instructions is the most important property of a general-purpose computer. That type of stored program concept is called stored program organization.
- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.
- The below figure shows the stored program organization

Figure 5-1 Stored program organization.



- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$.
- If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- ***Accumulator (AC):***
  - ✓ Computers that have a single-processor register usually assign to it the name accumulator and label it AC.
  - ✓ The operation is performed with the memory operand and the content of AC.

Addressing of Operand:

- Sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand.
- When the second part of an instruction code specifies an operand, the instruction is said to have an ***immediate operand*.**
- When the second part specifies the address of an operand, the instruction is said to have a ***direct address*.**
- When second part of the instruction designate an address of a memory word in which the address of the operand is found such instruction have ***indirect address***.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- The instruction code format shown in Fig. 5-2(a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirect address.

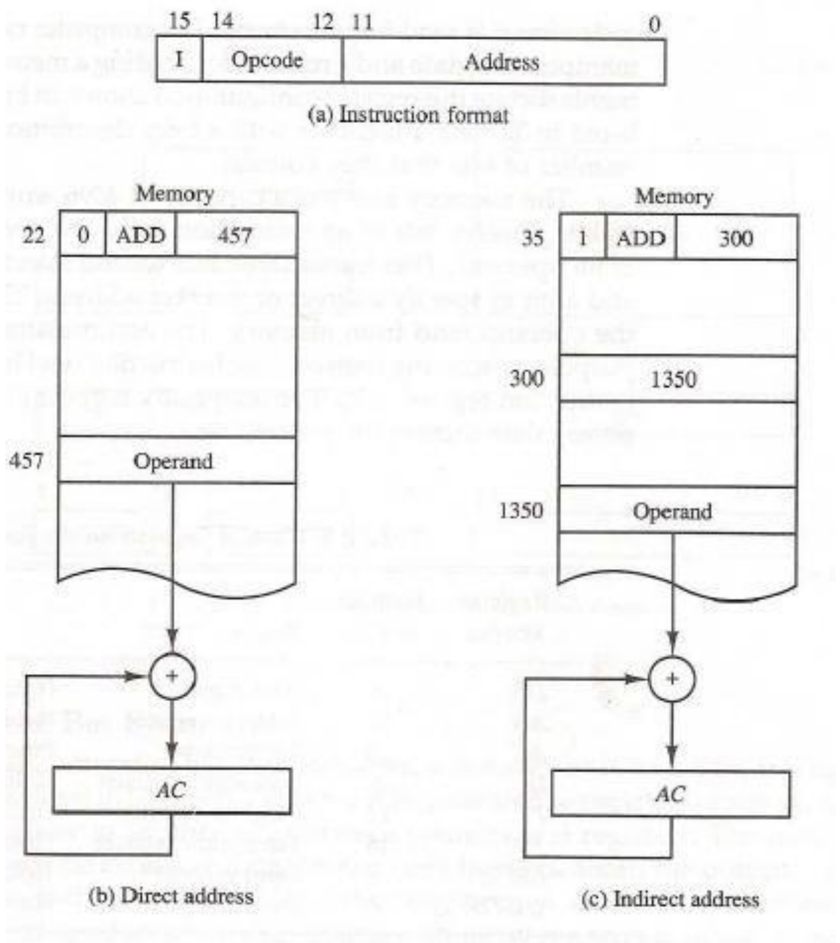


Figure 5-2 Demonstration of direct and indirect address.

- A direct address instruction is shown in Fig. 5-2(b).
- It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Fig. 5-2(c) has a mode bit I = 1.
- Therefore, it is recognized as an indirect address instruction.
- The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350.
- The operand found in address 1350 is then added to the content of *AC*.
- The ***effective address*** to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction.
- Thus the effective address in the instruction of Fig. 5-2(b) is 457 and in the instruction of Fig 5-2(c) is 1350.

2.Computer Registers:

- What is the need for computer registers?
  - ✓ The need of the registers in computer for
    - Instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed **(PC)**.
    - Necessary to provide a register in the control unit for storing the instruction code after it is read from memory **(IR)**.
    - Needs processor registers for manipulating data (AC and TR) and a register for holding a memory address **(AR)**.
- The above requirements dictate the register configuration shown in Fig. 5-3.

- The registers are also listed in Table 5.1 together with a brief description of their function and the number of bits that they contain.

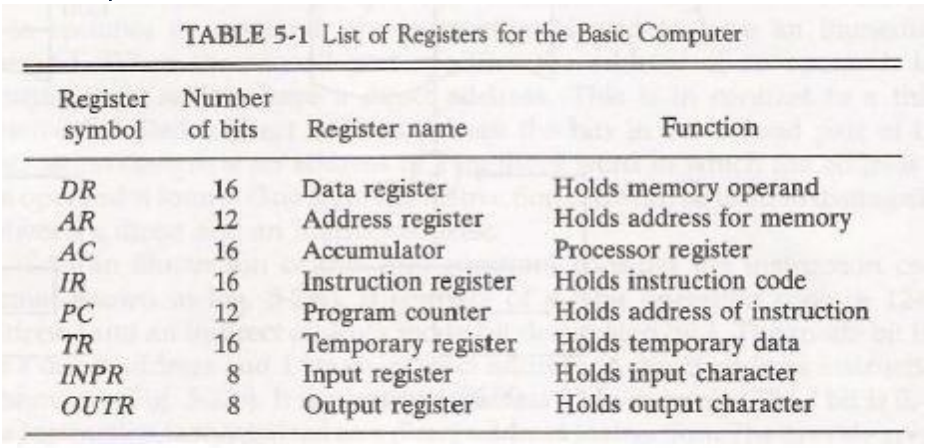


TABLE 5-1 List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

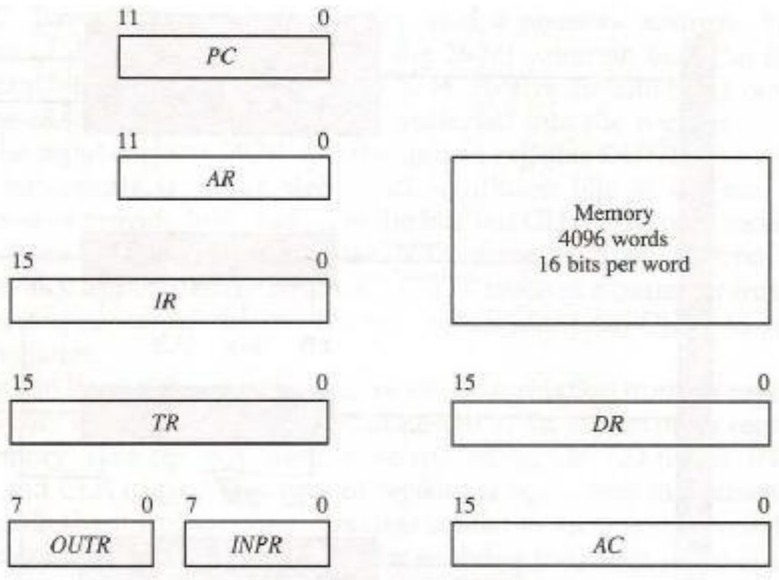


Figure 5-3 Basic computer registers and memory.

- The *data register (DR)* holds the operand read from memory.
- The *accumulator (AC)* register is a general purpose processing register.
- The instruction read from memory is placed in the *instruction register (IR).*
- The *temporary register (TR)* is used for holding temporary data during the processing.
- The *memory address register (AR)* has 12 bits since this is the width of a memory address.
- The *program counter (PC)* also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.
- Two registers are used for input and output.
    - The *input register (INPR)* receives an 8-bit character from an input device.
    - The *output register (OUTR)* holds an 8-bit character for an output device.

Common Bus System:

- The basic computer has eight registers, a memory unit, and a control unit
- Paths must be provided to transfer information from one register to another and between memory and registers.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- The connection of the registers and memory of the basic computer to a common bus system is shown in Fig. 5-4.
- The outputs of seven registers and memory are connected to the common bus.

- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the required binary selection.

- For example, the number along the output of *DR* is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0$ = 011.

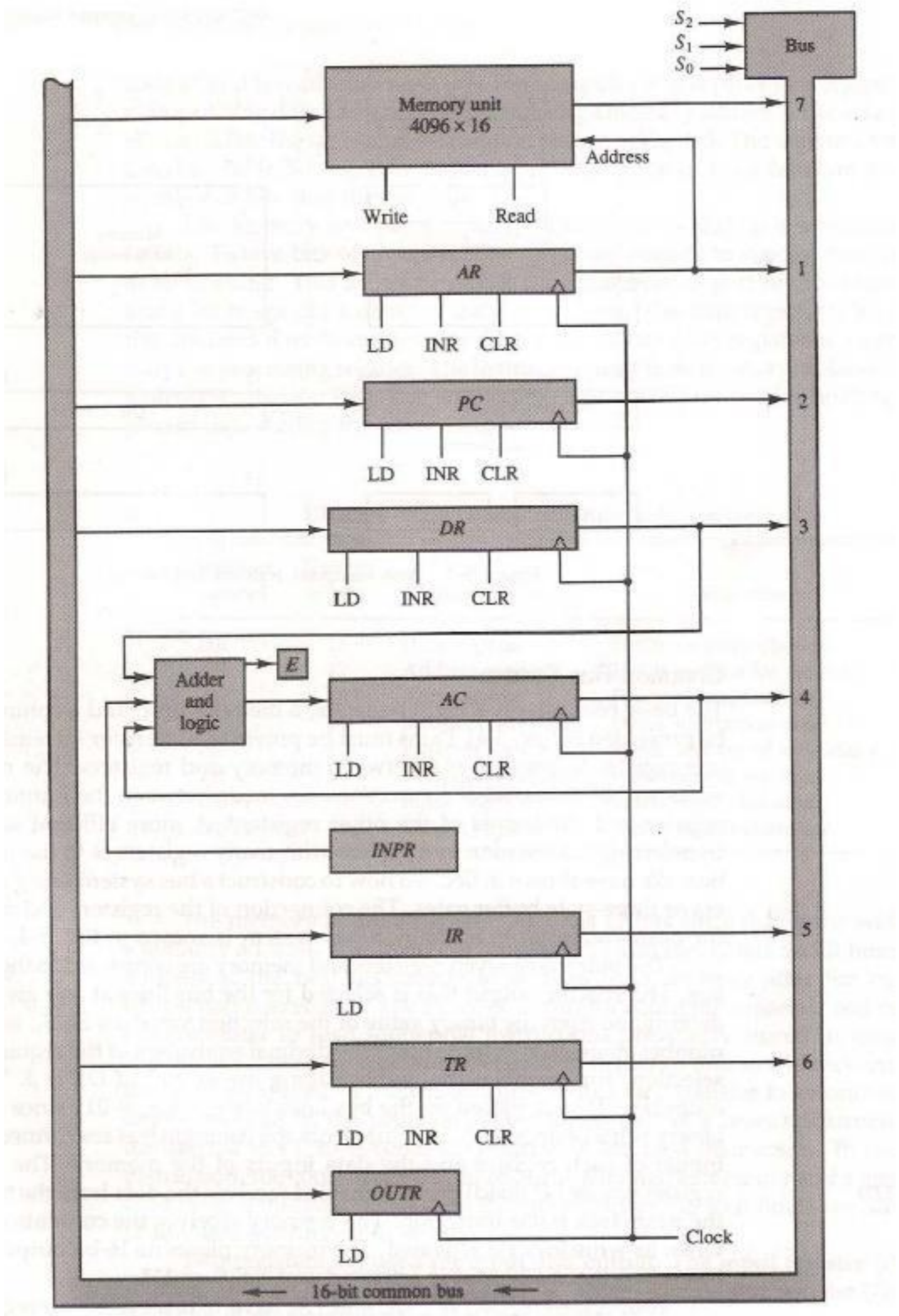


Figure 5-4 Basic computer registers connected to a common bus.

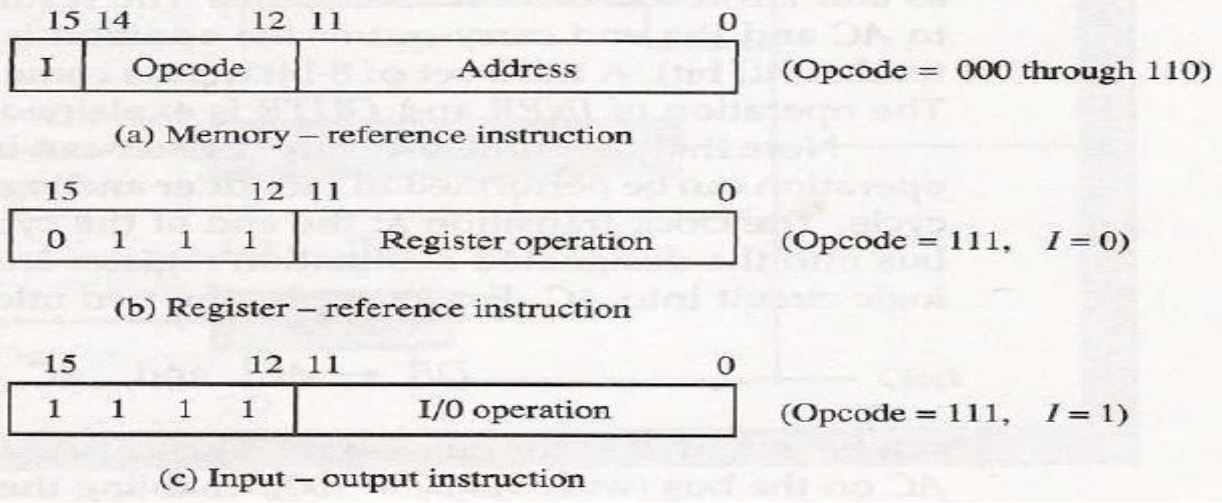
- The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.

- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0$ = 111.
- Two registers, AR and *PC,* have 12 bits each since they hold a memory address.
- When the contents of *AR* or *PC* are applied to the 16-bit common bus, the four most significant bits are set to 0's.
- When AR or *PC* receives information from the bus, only the 12 least significant bits are transferred into the register.
- The input register *INPR* and the output register OUTR have 8 bits each.
- They communicate with the eight least significant bits in the bus.
- INPR is connected to provide information to the bus but OUTR can only receive information from the bus.
- This is because INPR receives a character from an input device which is then transferred to *AC.*
- *OUTR* receives a character from AC and delivers it to an output device.
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).
- This type of register is equivalent to a binary counter with parallel load and synchronous clear.
- Two registers have only a LD input.
- The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR.
- Therefore, AR must always be used to specify a memory address.
- The 16 inputs of *AC* come from an adder and logic circuit. This circuit has three sets of inputs.
    - One set of 16-bit inputs come from the outputs of *AC.*
    - Another set of 16-bit inputs come from the data register *DR.*
    - The result of an addition is transferred to *AC* and the end carry-out of the addition is transferred to flip-flop E (extended *AC* bit).
    - A third set of 8-bit inputs come from the input register INPR.
- The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle.
- For example, the two microoperations DR←AC and AC ← DR can be executed at the same time.
- This can be done by placing the content of *AC* on the bus (with $S_2S_1S_0$ = 100), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of *AC,* all during the same clock cycle.

3. Computer Instructions:

- The basic computer has three instruction code formats, as shown in Fig. 5-5. Each format has 16 bits.

Figure 5-5 Basic computer instruction formats.



- The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
- A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.
- I is equal to 0 for direct address and to 1 for indirect address.
- The register-reference instructions are recognized by the operation code 1.11 with a 0 in the leftmost bit (bit 15) of the instruction.
- A register-reference instruction specifies an operation on the AC register. So an operand from memory is not needed. Therefore, the other 12 bits are used to specify the operation to be executed.
- An input—output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction.
- The remaining 12 bits are used to specify the type of input—output operation.
- The instructions for the computer are listed in Table 5-2.

- he symbol designation is a three-letter word and represents an abbreviation intended for programmers and users.
- The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

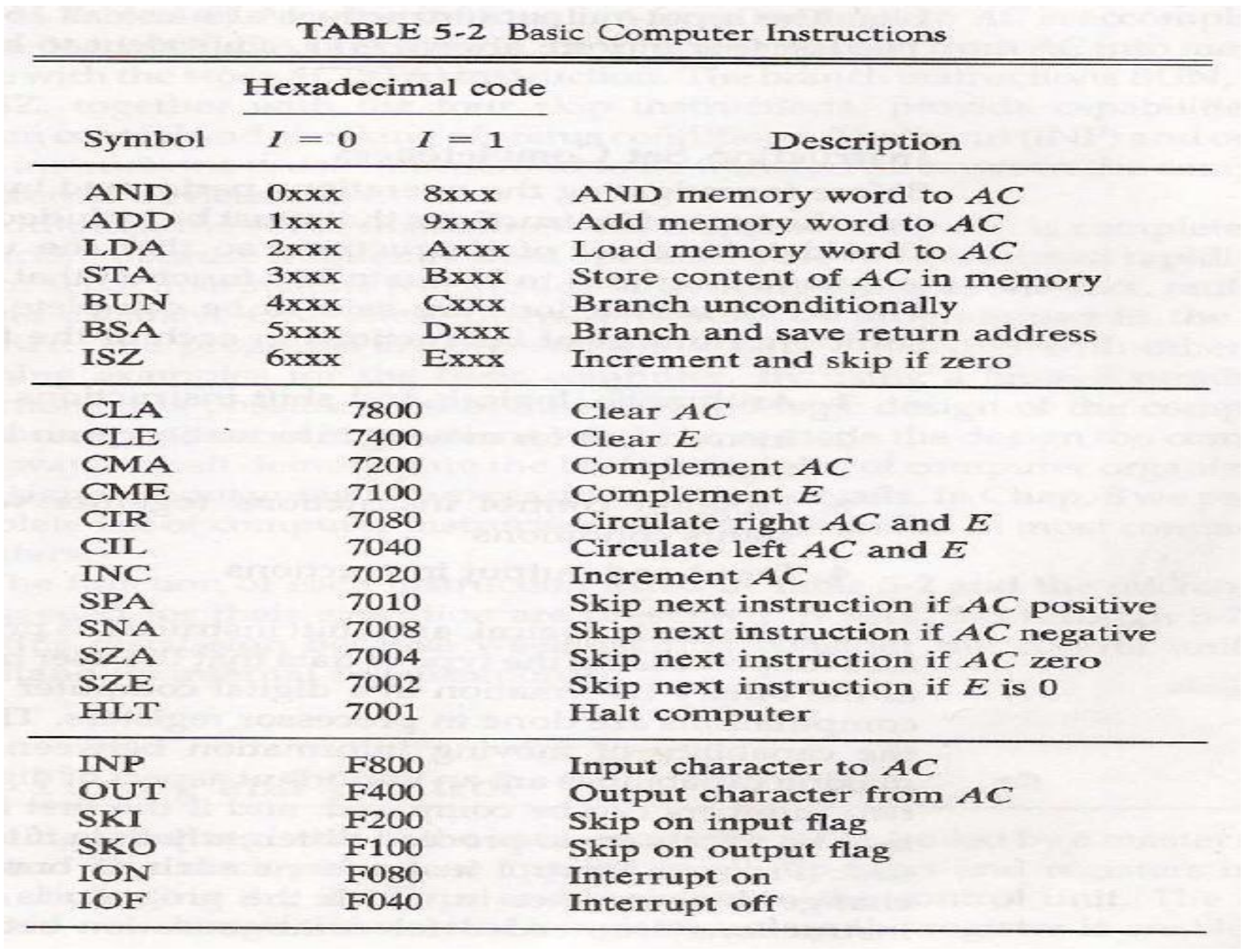


TABLE 5-2 Basic Computer Instructions

	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Instruction Set Completeness:

- A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
    - Arithmetic, logical, and shift instructions
    - Data Instructions (for moving information to and from memory and processor registers)
    - Program control or Brach
    - Input and output instructions
- There is one arithmetic instruction, ADD, and two related instructions, complement AC(CMA) and increment AC(INC). With these three instructions we can add and subtract binary numbers when negative numbers are in signed-2's complement representation.
- The circulate instructions, CIR and CIL; can be used for arithmetic shifts as well as any other type of shifts desired.
- There are three logic operations: AND, complement AC (CMA), and clear AC(CLA). The AND and complement provide a NAND operation.
- Moving information from memory to *AC* is accomplished with the load AC (LDA) instruction. Storing information from AC into memory is done with the store AC (STA) instruction.
- The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions.
- The input (INP} and output (OUT) instructions cause information to be transferred between the computer and external devices.

4. Timing and Control:

- The timing for all registers in the basic computer is controlled by a master clock generator.
- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.
- There are two major types of control organization:
    - *Hardwired control*
    - *Microprogrammed control*
- The differences between hardwired and microprogrammed control are

Hardwired control	**Microprogrammed control**
✓ The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.	✓ The control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
✓ The advantage that it can be optimized to produce a fast mode of operation.	✓ Compared with the hardwired control operation is slow.
✓ Requires changes in the wiring among the various components if the design has to be modified or changed.	✓ Required changes or modifications can be done by updating the microprogram in control memory.

- The block diagram of the hardwired control unit is shown in Fig. 5-6.
- It consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register (IR). It is divided into three parts: The I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit sequence counter can count in binary from 0 through 15.

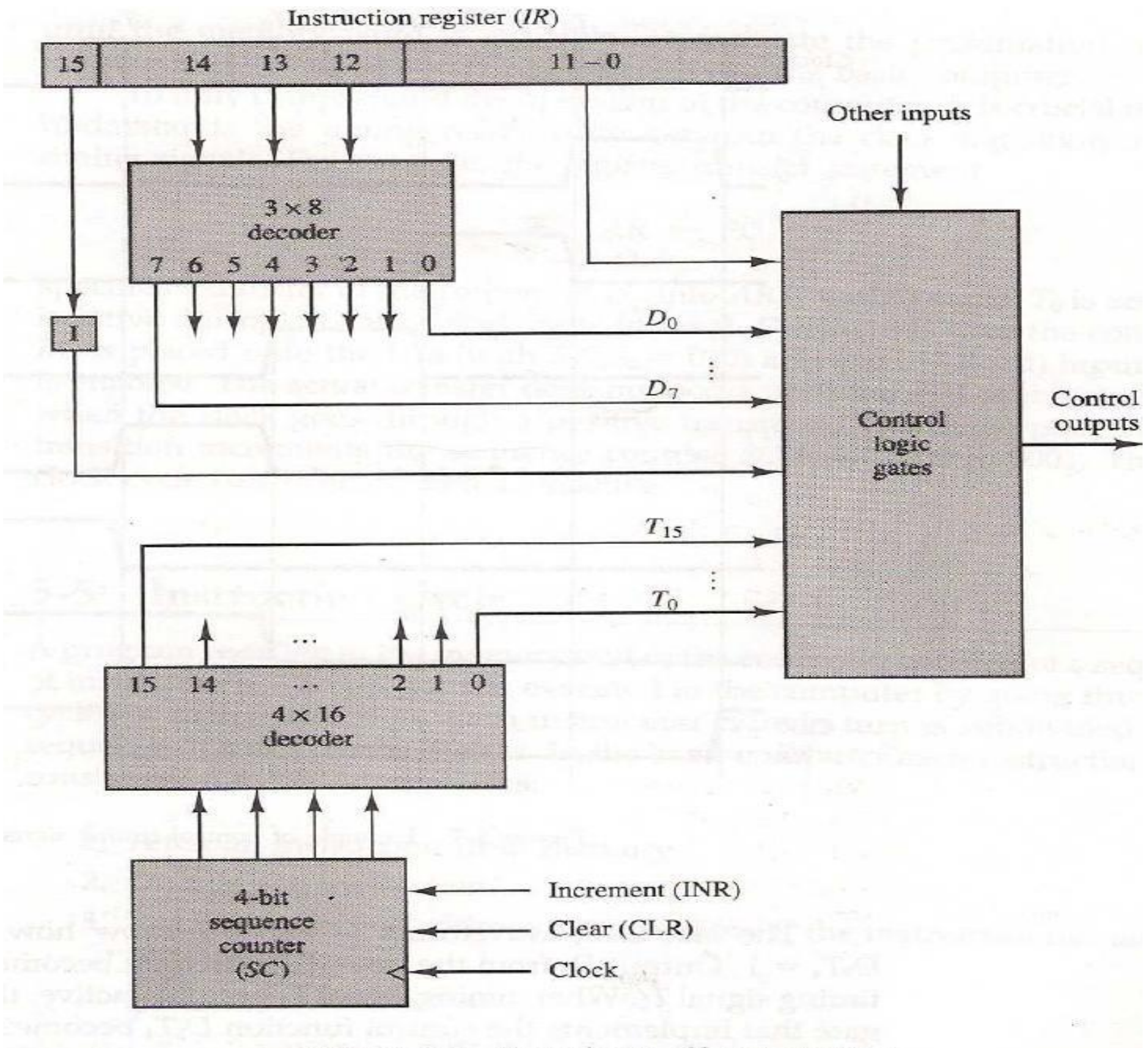


Figure 5-6 Control unit of basic computer.

- The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.
- The sequence counter *SC* can be incremented or cleared synchronously.
- The counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.
- As an example, consider the case where *SC* is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, *SC* is cleared to 0 if decoder output D3 is active.
- This is expressed symbolically by the statement

$$D_3T_4: SC \leftarrow 0$$

- The timing diagram of Fig. 5-7 shows the time relationship of the control signals.
- The sequence counter *SC* responds to the positive transition of the clock.
- Initially, the CLR input of *SC* is active. The first positive transition of the clock clears *SC* to 0, which in turn activates the timing signal T_0 out of the decoder. T_0 is active during one clock cycle.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This produces the sequence of timing signals T_0, T_1, T_2, T_3, $T_4$and so on, as shown in the diagram.
- The last three waveforms in Fig.5-7 show how *SC* is cleared when $D_3T_4 = 1$.
- Output D_3 from the operation decoder becomes active at the end of timing signal T_2.
- When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active.
- This signal is applied to the CLR input of *SC.* On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0.
- This causes the timing signal T_0 to become active instead of T_5 that would have been active if *SC* were incremented instead of cleared.

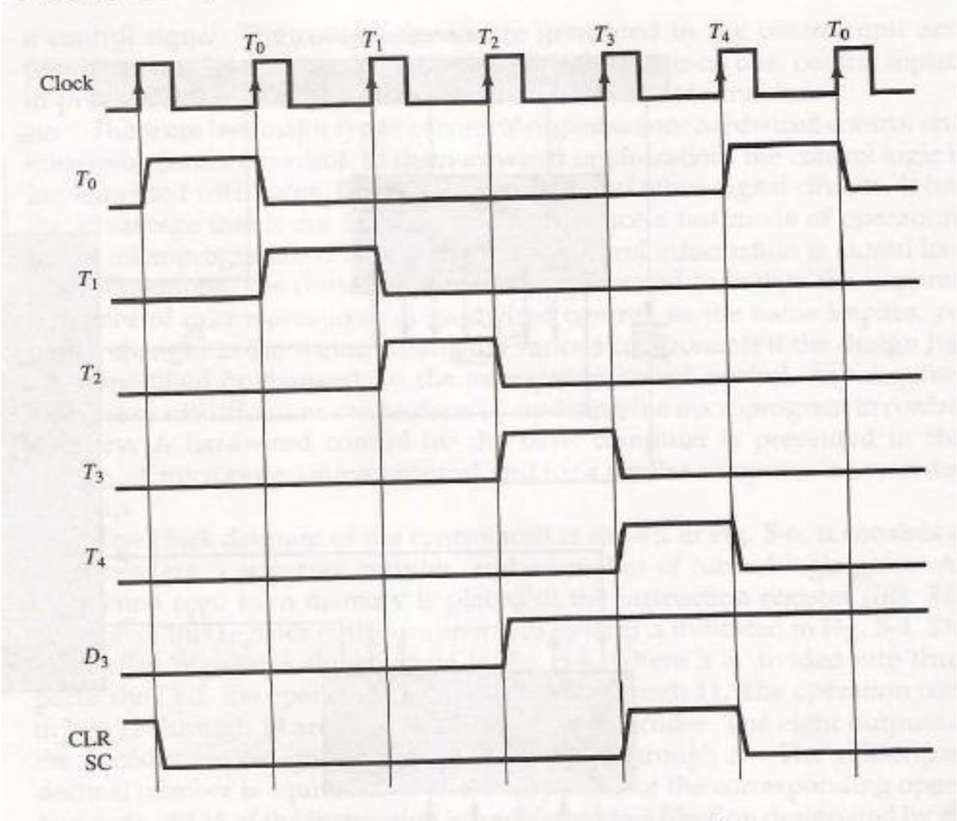


Figure 5-7 Example of control timing signals.

5. Instruction Cycle:

- A program residing in the memory unit of the computer consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.
- In the basic computer each instruction cycle consists of the following phases:
    1. Fetch an instruction from memory.
    2. Decode the instruction.
    3. Read the effective address from memory if the instruction has an indirect address.
    4. Execute the instruction.
- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.

Fetch and Decode:

- Initially, the program counter PC is loaded with the address of the first instruction in the program.
- The sequence counter *SC* is cleared to 0, providing a decoded timing signal T_0.
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

$$T_0: \quad AR \leftarrow PC$$
$$T_1: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$
$$T_2: \quad D_0, \ldots, D_7 \leftarrow \text{Decode } IR(12\text{–}14), \quad AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$$

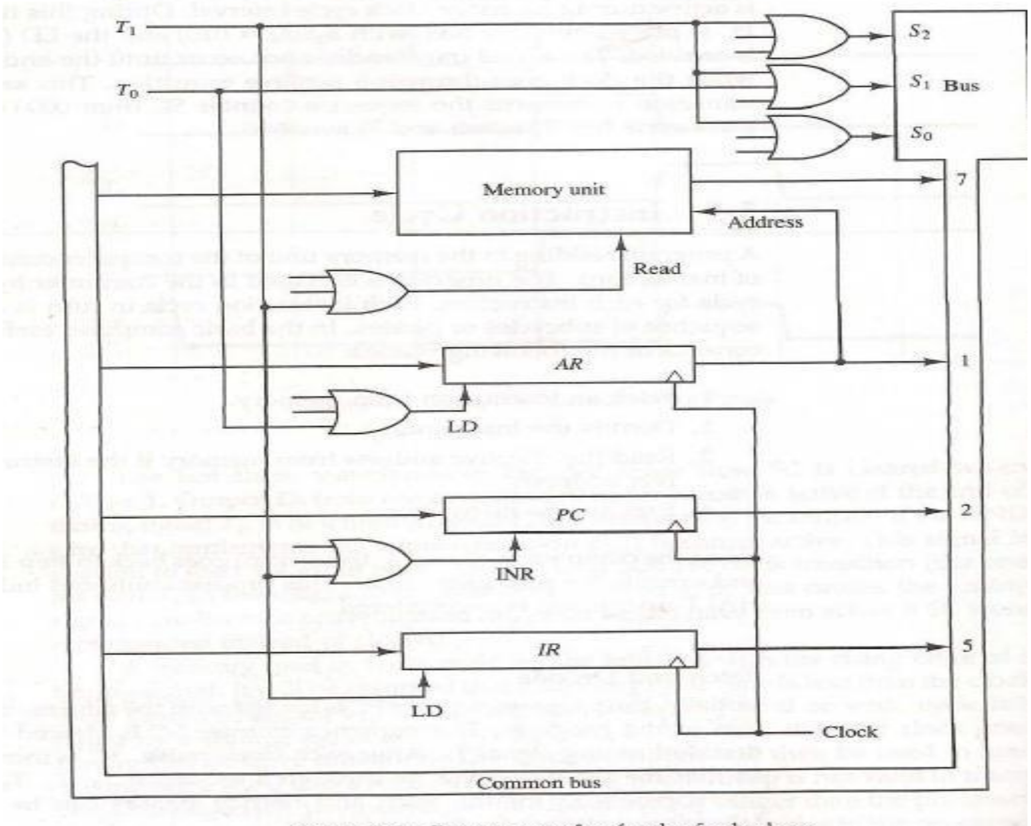


Figure 5-8 Register transfers for the fetch phase.

- Figure 5-8 shows how the first two register transfer statements are implemented in the bus system.
- To provide the data path for the transfer of *PC* to AR we must apply timing signal T_0 to achieve the following connection:
    - Place the content of *PC* onto the bus by making the bus selection inputs S_2, S_1, S_0 equal to 010.
    - Transfer the content of the bus to AR by enabling the LD input of AR.

- In order to implement the second statement it is necessary to use timing signal T_1 to provide the following connections in the bus system.
    - Enable the read input of memory.
    - Place the content of memory onto the bus by making $S_2S_1S_0$=111.
    - Transfer the content of the bus to IR by enabling the LD input of IR.
    - Increment PC by enabling the INR input of PC.
- Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction:

- The timing signal that is active after the decoding is T_3.
- During time T_3, the control unit determine the type of instruction that was read from the memory.
- The flowchart of fig.5-9 shows the initial configurations for the instruction cycle and also how the control determines the instruction cycle type after the decoding.
- Decoder output D_7 is equal to 1 if the operation code is equal to binary 111.
- If D_7=1, the instruction must be a register-reference or input-output type.
- If *D7* = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I.
- If *D7* = 0 and I = 1, indicates a memory-reference instruction with an indirect address. So it is then necessary to read the effective address from memory.
- If *D7* = 0 and I = 0, indicates a memory-reference instruction with a direct address.
- If *D7* = 1 and I = 0, indicates a register-reference instruction.
- If *D7* = 01and I = 1, indicates an input-output instruction.
- The three instruction types are subdivided into four separate paths.
- The selected operation is activated with the clock transition associated with timing signal T_3.
- This can be symbolized as follows:

$D_7'IT_3$: $AR \leftarrow M[AR]$
$D_7'I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instruction
D_7IT_3: Execute an input–output instruction

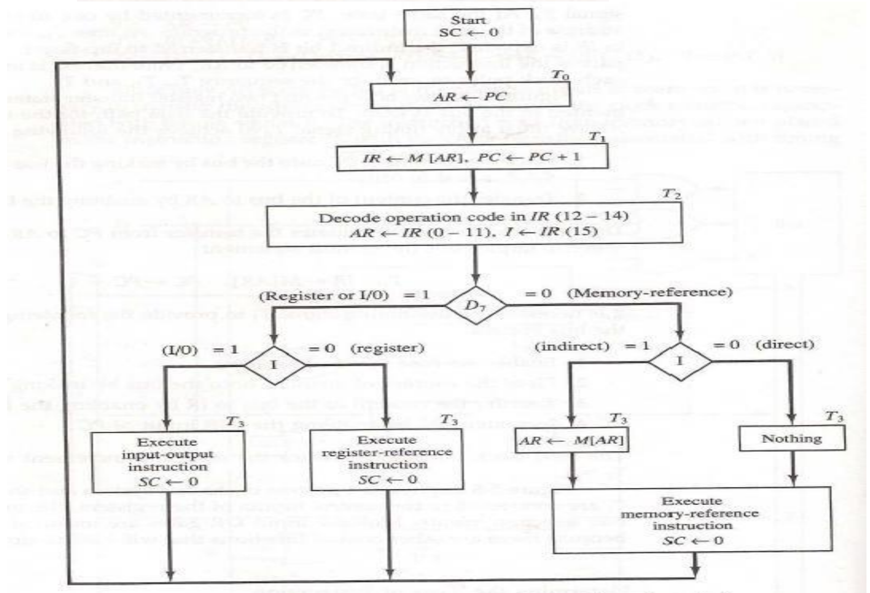

Start
SC ← 0
T0
AR ← PC
T1
IR ← M [AR], PC ← PC + 1
T2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M[AR]
T3
Nothing
Execute memory-reference instruction SC ← 0

Figure 5-9 Flowchart for instruction cycle (initial configuration).

Register-Reference Instructions:

- Register-reference instructions are recognized by the control when D7 = 1 and I=0.
- These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions.
- These 12 bits are available in IR (0-11).
- The control functions and microoperations for the register-reference instructions are listed in Table 5-3.
- These instructions are executed with the clock transition associated with timing variable T_3.
- Control function needs the Boolean relation $D_7I'T_3$, which we designate for convenience by the symbol r.
- By assigning the symbol *B_i* to bit i of *IR,* all control functions can be simply denoted by *rB_i.*

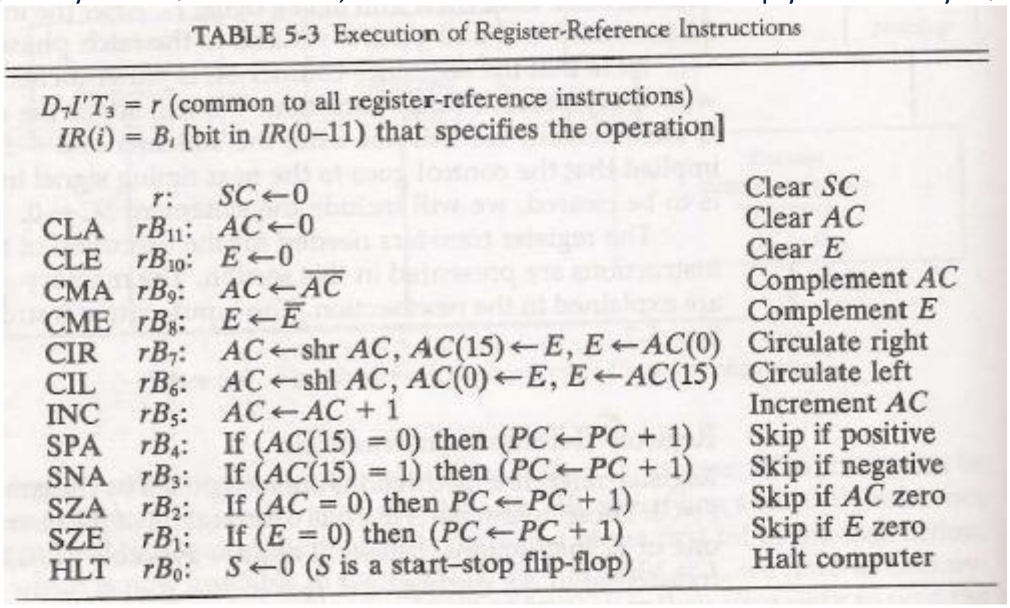


TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

- For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent to I′.
- The next three bits constitute the operation code and are recognized from decoder output D_7.
- Bit 11 in IR is 1 and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is *D_7*$I'T_3$ B_{11} = rB_{11}.
- The execution of a register-reference instruction is completed at time T_3.
- The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T_0.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers.
- The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing *PC* once again.
- The condition control statements must be recognized as part of the control conditions.
- The *AC* is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of *AC* is zero *(AC* = 0) if all the flip-flops of the register are zero.
- The HLT instruction clears a start-stop flip-flop *S* and stops the sequence counter from counting.

6. Memory-Reference Instructions:

- Table 5-4 lists the seven memory-reference instructions.
- The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.
- The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I= 0, or during timing signal T_3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T_4.

- The symbolic description of each instruction is specified in the table in terms of register transfer notation.

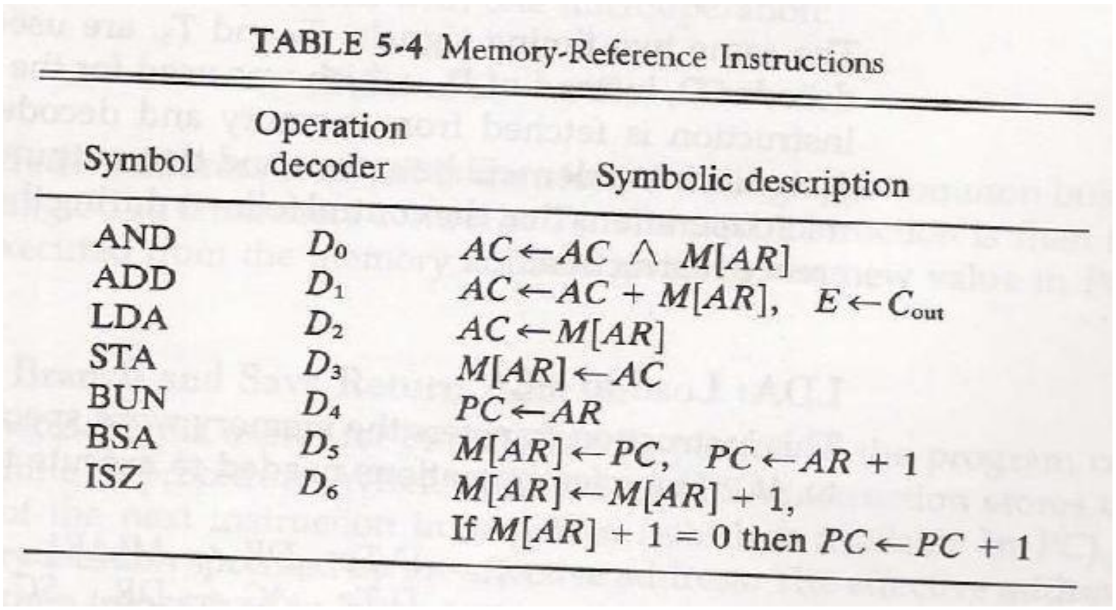


TABLE 5-4 Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC:

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.
- The result of the operation is transferred to AC.
- The microoperations that execute this instruction are:

D_0T_4: $DR \leftarrow M[AR]$

D_0T_5: $AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$

ADD to AC:

- This instruction adds the content of the memory word specified by the effective address to the value of *AC.*
- The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop.
- The microoperations needed to execute this instruction are

D_1T_4: $DR \leftarrow M[AR]$

D_1T_5: $AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC.
- The microoperations needed to execute this instruction are

$$D_2T_4: \; DR \leftarrow M[AR]$$
$$D_2T_5: \; AC \leftarrow DR, \quad SC \leftarrow 0$$

<u>STA:</u> Store AC

- This instruction stores the content of *AC* into the memory word specified by the effective address.
- Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation.

$$D_3T_4: \; M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

<u>BUN:</u> Branch Unconditionally

- This instruction transfers the program to the instruction specified by the effective address.
- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.
- The instruction is executed with one microoperation:

$$D_4T_4: \; PC \leftarrow AR, \quad SC \leftarrow 0$$

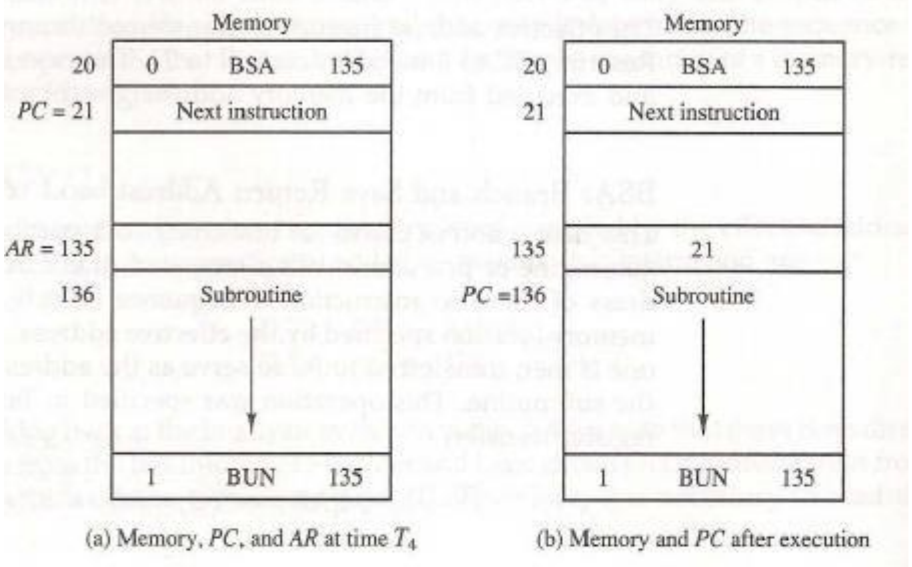
<u>BSA:</u> Branch and Save Return Address

- This instruction is useful for branching to a portion of the program called a subroutine or procedure.
- When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.
- The effective address plus one is then transferred to *PC* to serve as the address of the first instruction in the subroutine.
- This operation was specified with the following register transfer:

$$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$$

- A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10.

Figure 5-10 Example of BSA instruction execution.



- The BSA instruction is assumed to be in memory at address 20.
- The I bit is 0 and the address part of the instruction has the binary equivalent of 135.
- After the fetch and decode phases, *PC* contains 21, which is the address of the next instruction in the program (referred to as the return *address).* AR holds the effective address 135.
- This is shown in part (a) of the figure.
- The BSA instruction performs the following numerical operation:

$$M[135] \leftarrow 21, \quad PC \leftarrow 135 + 1 = 136$$

- The result of this operation is shown in part (b) of the figure.
- The return address21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.
- The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

- When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.
- When the BUN instruction is executed, the effective address 21 is transferred to PC.
- The next instruction cycle finds *PC with* the value 21, so control continues to execute the instruction at the return address.
- The BSA instruction must be executed with a sequence of two microoperations:

$$\begin{aligned} D_5T_4&: \quad M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1 \\ D_5T_5&: \quad PC \leftarrow AR, \quad SC \leftarrow 0 \end{aligned}$$

ISZ: Increment and Skip if Zero

- This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1 to skip the next instruction in the program.
- Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.
- This is done with the following sequence of microoperations:

$$\begin{aligned} D_6T_4&: \quad DR \leftarrow M[AR] \\ D_6T_5&: \quad DR \leftarrow DR + 1 \\ D_6T_6&: \quad M[AR] \leftarrow DR, \quad \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), \quad SC \leftarrow 0 \end{aligned}$$

Control Flowchart:

- A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. 5.11.

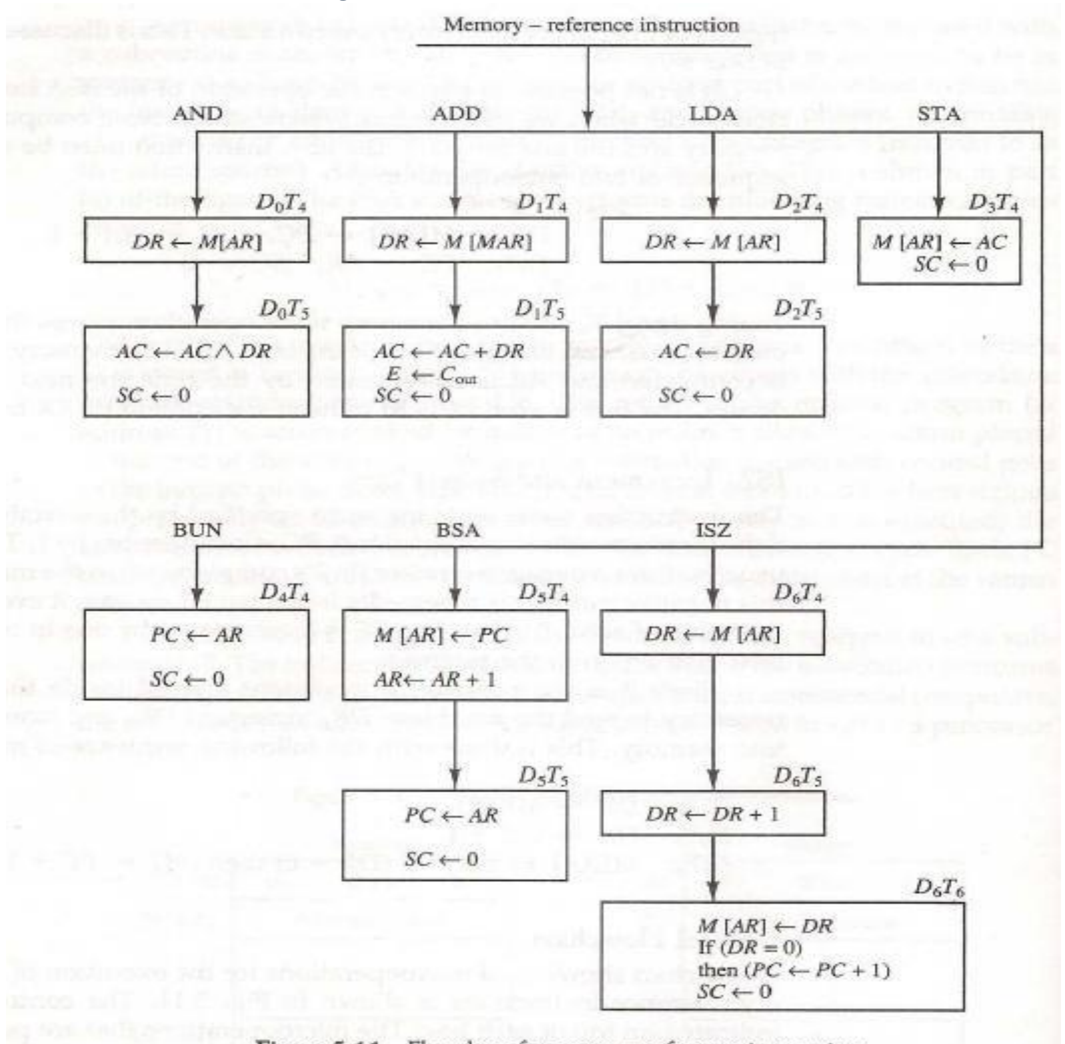


Figure 5-11 Flowchart for memory-reference instructions.

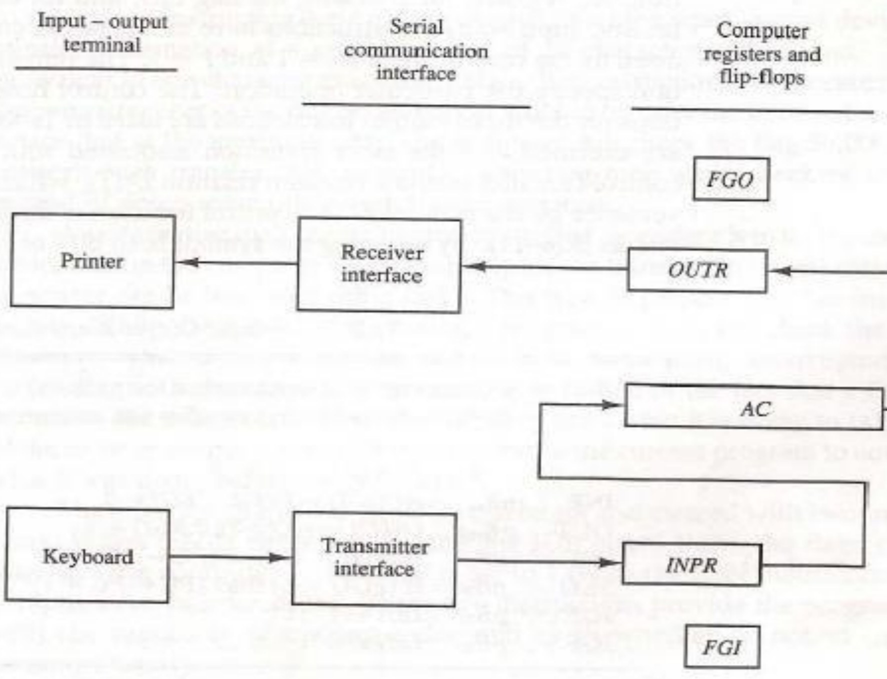
7. Input-Output and Interrupt:

- Instructions and data stored in memory must come from some input device.
- Computational results must be transmitted to the user through some output device.
- To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

Input-Output Configuration:

- The terminal sends and receives serial information.
- Each quantity of information has eight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register *INPR.*
- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with the AC in parallel.
- The input—output configuration is shown in Fig. 5-12.

Figure 5-12 Input-output configuration.



- The input register INPR consists of eight bits and holds alphanumeric input information.
- The 1-bit input flag *FGI* is a control flip-flop.
- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The output register OUTR works similarly but the direction of information flow is reversed.
- Initially, the output flag *FGO* is set to 1.
- The computer checks the flag bit; if it is 1, the information from *AC* is transferred in parallel to OUTR and FGO is cleared to 0.
- The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

Input-Output Instructions:

- Input and output instructions are needed for transferring information to and from *AC* register, for checking the flag bits, and for controlling the interrupt facility.
- Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1.
- The remaining bits of the instruction specify the particular operation.

- The control functions and microoperations for the input-output instructions are listed in Table 5-5.

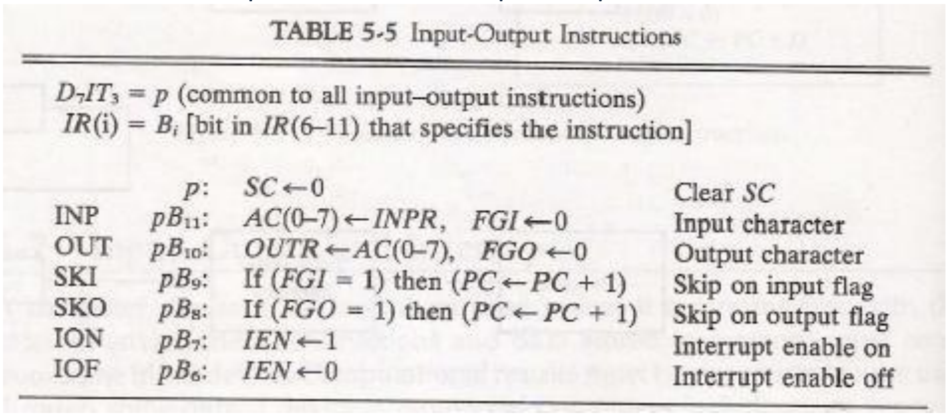


TABLE 5-5 Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6\text{–}11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear *SC*
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR, \quad FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7), \quad FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

- These instructions are executed with the clock transition associated with timing signal T_3.
- Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol p.
- The control function is distinguished by one of the bits in IR (6-11).
- By assigning the symbol B_i to bit *i* of IR, all control functions can be denoted by pB_i for i = 6 though 11.
- The sequence counter *SC* is cleared to 0 when p = D_7IT_3 = 1.
- The last two instructions set and clear an interrupt enable flip-flop IEN.

Program Interrupt:

- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- The difference of information flow rate between the computer and that of the input—output device makes this type of transfer inefficient.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check the flags.
- When a flag is set, the computer is momentarily interrupted from the current program.
- The computer deviates momentarily from what it is doing to perform of the input or output transfer.
- It then returns to the current program to continue what it was doing before the interrupt.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions.
    - When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer.
    - When *IEN* is set to (with the ION instruction), the computer can be interrupted.

- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. 5-13.
- An interrupt flip-flop R is included in the computer. When *R* = 0, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control.
- If it is 0, it indicates that the programmer does not want to use the interrupt,so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle.
- If either flag is set to 1 while *1EN* = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

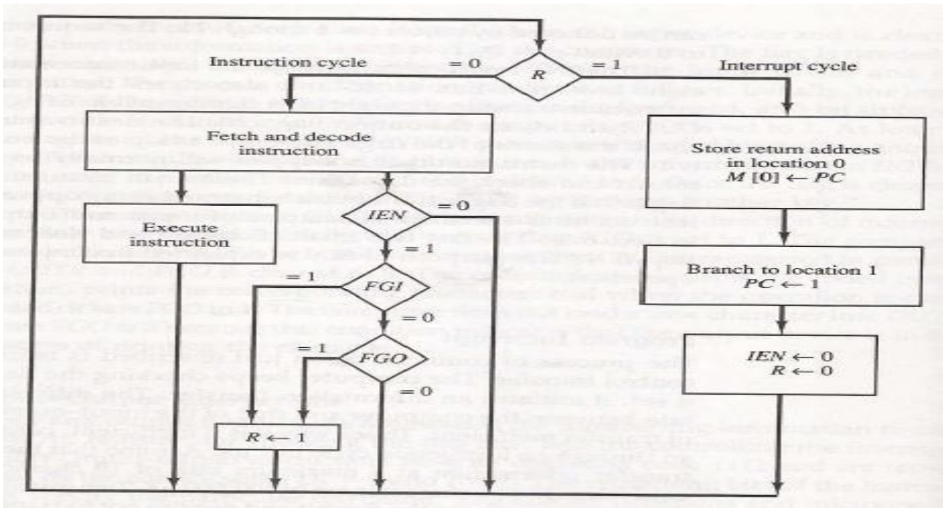
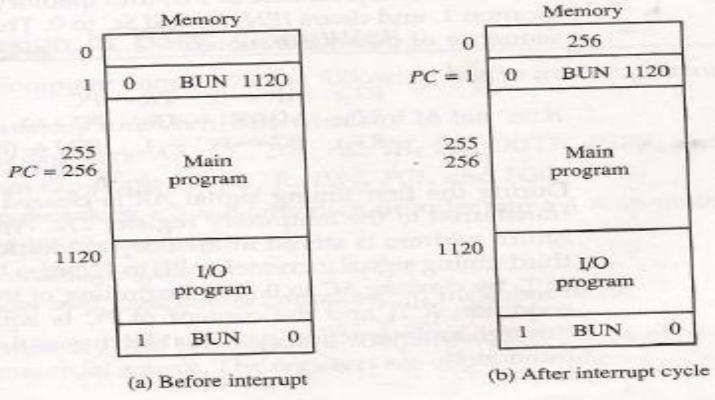


Figure 5-13 Flowchart for interrupt cycle.

Interrupt cycle:

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location.
- This location may be a processor register, a memory stack, or a specific memory location.
- An example that shows what happens during the interrupt cycle is shown in Fig. 5-14.

Figure 5-14 Demonstration of the interrupt cycle.



- When an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.
- At this time, the returns address 256 is in *PC.*
- The programmer has previously placed an input—output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig. 5.14(a).
- When control reaches timing signal $T_0$and finds that *R = 1,* it proceeds with the interrupt cycle.
- The content of *PC* (256) is stored in memory location 0, *PC* is set to 1, and R is cleared to 0.
- The branch instruction at address 1 causes the program to transfer to the input—output service program at address 1120.

- This program checks the flags, determines which flag is set, and then transfers the required input or output information.
- Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.
- This is shown in Fig. 5-14

UNIT II

1.1 **Micro programmed control:**

Hardwired Control Unit:

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

Micro programmed control unit:

A control unit whose binary control variable are stored in memory is called a micro programmed control unit.

Dynamic microprogramming:

A more advanced development known as dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic microprogramming employ

a writable control memory. This type of memory can be used for writing.

Control Memory:

Control Memory is the storage in the microprogrammed control unit to store the microprogram.

Writeable Control Memory:

Control Storage whose contents can be modified, allow the change in microprogram and Instruction set can be changed or modified is referred as Writeable Control Memory.

Control Word:

The control variables at any given time can be represented by a control word string of 1 's and 0's called a control word.

Microoperation, Microinstruction, Micro program, Microcode.

Microoperations:

- In computer central processing units, micro-operations (also known as a micro-ops or μops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in this context).

Micro instruction:

- A symbolic microprogram can be translated into its binary equivalent by means of an assembler.
- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD,BR, and AD.

Micro program:

- A sequence of microinstructions constitutes a microprogram.
- Since alterations of the microprogram are not needed once the control unit is in operation,the control memory can be a read-only memory (ROM).
- ROM words are made permanent during the hardware production of the unit.
- The use of a micro program involves placing all control variables in words of ROM for use by the control unit through successive read operations.
- The content of the word in ROM at a given address specifies a microinstruction.

Microcode:

- Microinstructions can be saved by employing subroutines that use common sections of microcode.

- For example, the sequence of micro operations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions.

- This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Organization of micro programmed control unit

- The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure 4.1.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.

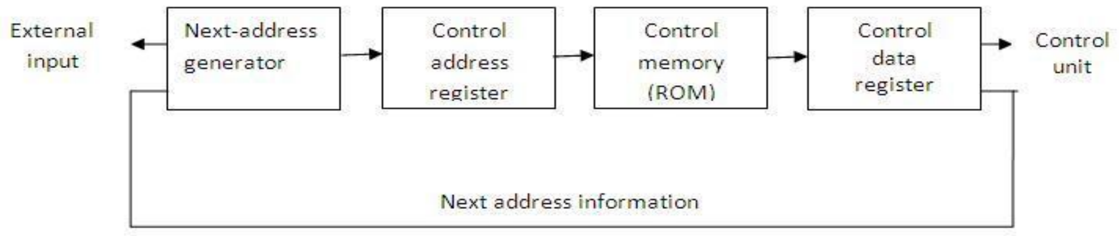


figure 4.1: Micro-programmed control organization

- The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.

- 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.
- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.
- The next address generator is sometimes called a micro-program sequencer, as it

  determines the address sequence that is read from control memory.

- Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.
- The control data register holds the present microinstruction while the next address is

computed and read from memory.

- 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.
- The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

Address Sequencing:

- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1:

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.

- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is available in the memory address register.

Step-3:

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction has its own micro-program routine stored in a given location of control memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4:

- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing the return address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.

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

4. A facility for subroutine call and return.

selection of address for control memory

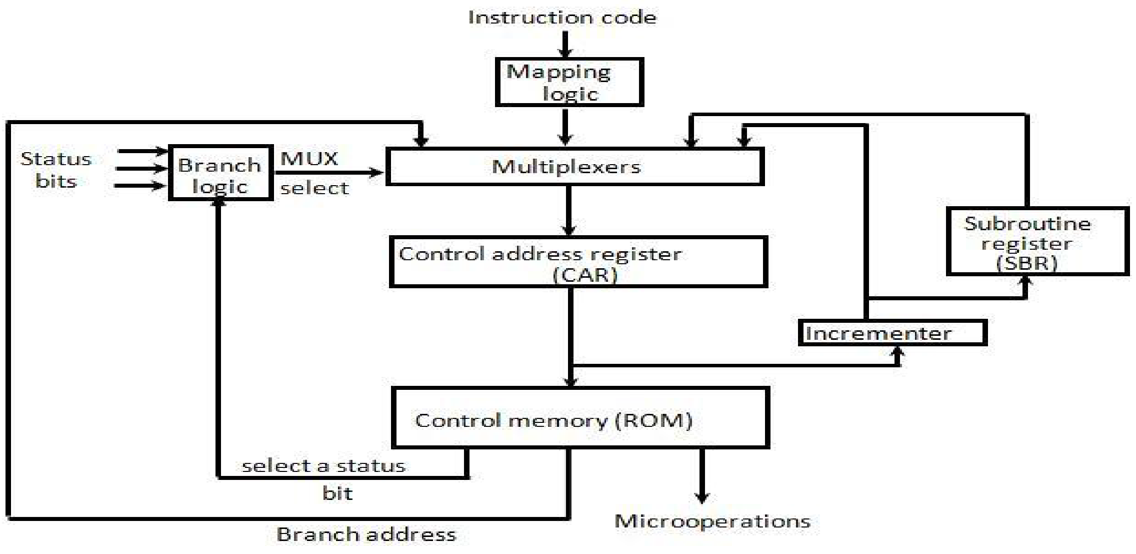


Figure 4.2: Selection of address for control memory

- Above figure 4.2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.
- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- 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 micro-program wishes to return from the subroutine.
- The branch logic of figure 4.2 provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.

- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.
- A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.
- A 0 output in the multiplexer causes the address register to be incremented.

Micro programme Example:

Mapping of an Instruction

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction.For example, a computer with a simple instruction format as shown in figure 4.3 has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure 4.3.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.

- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

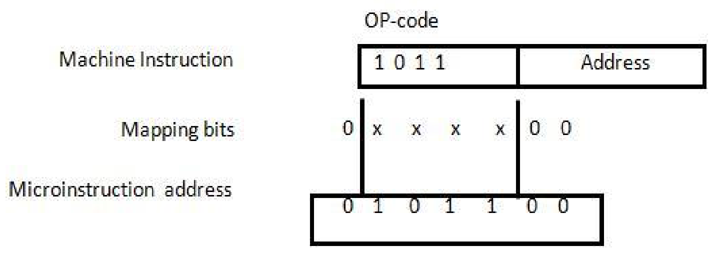


Figure 4.3: Mapping from instruction code to microinstruction address

- One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function.
- The contents of the mapping ROM give the bits for the control address register.
- In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.
- The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Computer Hardware Configuration

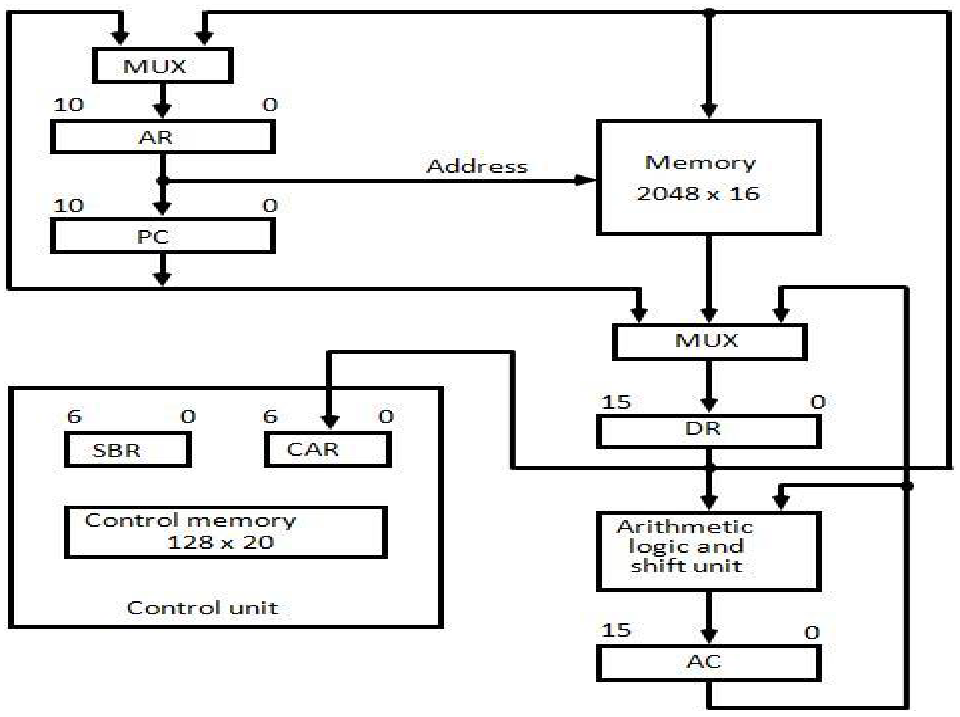


Figure 4.4: Computer hardware configuration

The block diagram of the computer is shown in Figure 4.4. It consists of

1. Two memory units:
   Main memory -> for storing instructions and data, and
   Control memory -> for storing the microprogram.
2. Six Registers:
   Processor unit register: AC(accumulator),PC(Program Counter), AR(Address Register),DR(Data Register)
   Control unit register: CAR (Control Address Register), SBR(Subroutine Register)
3. Multiplexers:
   The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.
4. ALU:
   The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.
   - DR can receive information from AC, PC, or memory.
   - AR can receive information from PC or DR.
   - PC can receive information only from AR.
   - Input data written to memory come from DR, and data read from memory can go only to DR.

Microinstruction Format

The microinstruction format for the control memory is shown in figure 4.5. The 20 bits of the microinstruction are divided into four functional parts as follows:

1. The three fields F1, F2, and F3 specify microoperations for the computer. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations.
2. The CD field selects status bit conditions.
3. The BR field specifies the type of branch to be used.
4. The AD field contains a branch address. The address field is seven bits wide, since the control memory has 128 = 27 words.

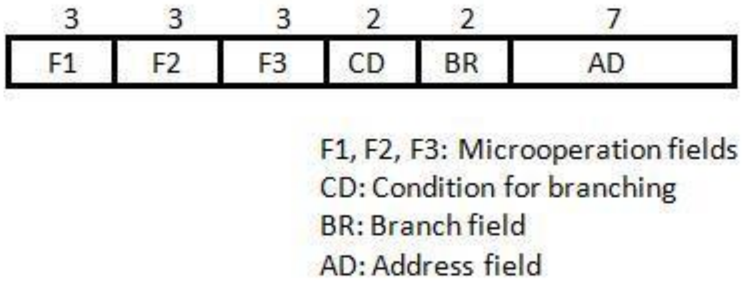

3
3
3
2
2
7
F1
F2
F3
CD
BR
AD
F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

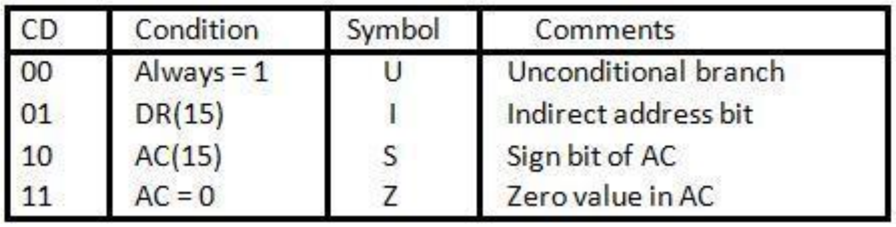
Figure 4.5: Microinstruction Format

- ➢ As an example, a microinstruction can specify two simultaneous microoperations fromF2 and F3 and none from F1.

DR --- M[AR] with F2 = 100

PC --- PC + 1 with F3 = 101

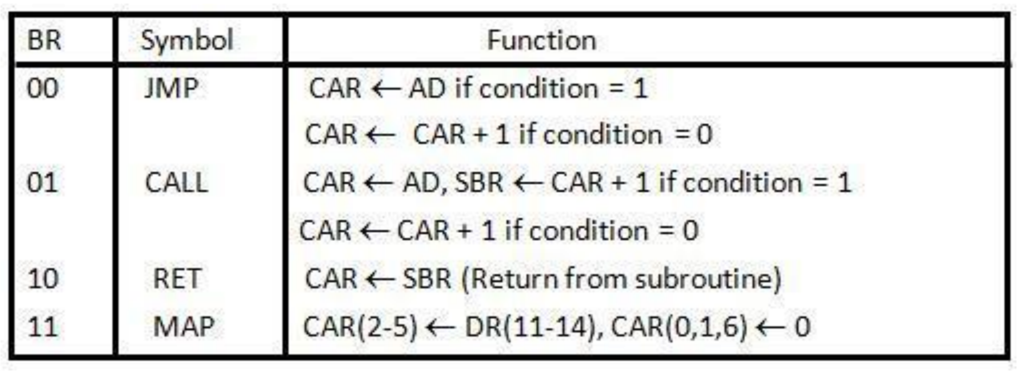
- ➢ The nine bits of the microoperation fields will then be 000 100 101.
- ➢ The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 4.1.



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

Table 4.1: Condition Field

- ➢ The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction shown in Table 4.2.



BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Table 4.2: Branch Field

Symbolic Microinstruction:

- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table 4.3.

1. Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2. Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.
3. CD	The CD field has one of the letters U, I, S, or Z.
4. BR	The BR field contains one of the four symbols defined in Table 5.2.
5. AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear as a label. ii. With the symbol NEXT to designate the next address in sequence. iii. When the BR field contains a RET or MAP symbol,the AD field is left empty and is converted to seven zeros by the assembler

Table 4.3: Symbolic Microinstruction

Design of Control Unit:

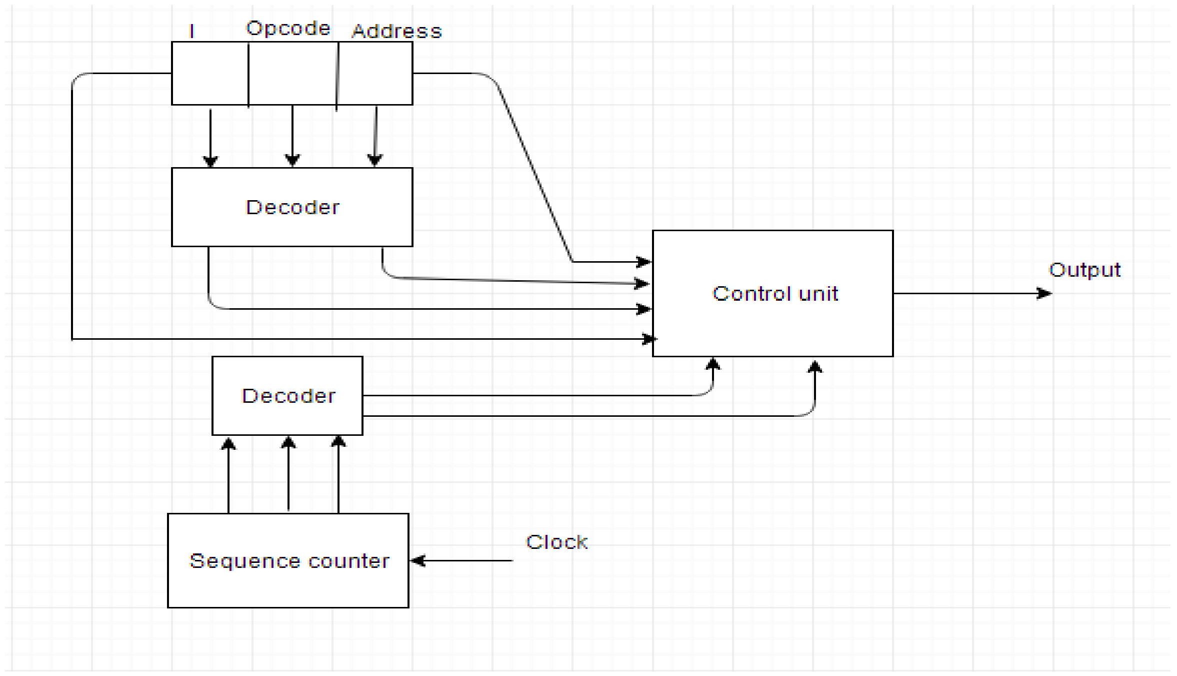
- Control unit generates timing and control signals for the operations of the computer.
- The control unit communicates with ALU and main memory.
- It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.

Control unit can be designed by two methods which are given below:

Hardwired Control Unit:

- It is implemented with the help of gates, flip flops, decoders etc. in the hardware.
- The inputs to control unit are the instruction register, flags, timing signals etc.
- This organization can be very complicated if we have to make the control unit large.

  If the design has to be modified or changed, all the combinational circuits have to be modified which is a very difficult task.



Microprogrammed Control Unit:

- It is implemented by using programming approach.
- A sequence of micro operations is carried out by executing a program consisting of micro-instructions.
- In this organization any modifications or changes can be done by updating the micro program in the control memory by the programmer.

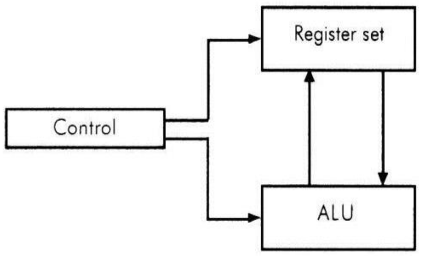
. **Difference between Hardwired Control and Microprogrammed Control:**

Hardwired Control	**Microprogrammed Control**
Technology is circuit based.	Technology is software based.
It is implemented through flip-flops, gates, decoders etc.	Microinstructions generate signals to control the execution of instructions.
Fixed instruction format.	Variable instruction format (16-64 bits per instruction).
Instructions are register based.	Instructions are not register based.
ROM is not used.	ROM is used.
It is used in RISC.	It is used in CISC.
Faster decoding.	Slower decoding.
Difficult to modify.	Easily modified.
Chip area is less.	Chip area is large.

2.2 Central Processing Unit

General Register Organization:

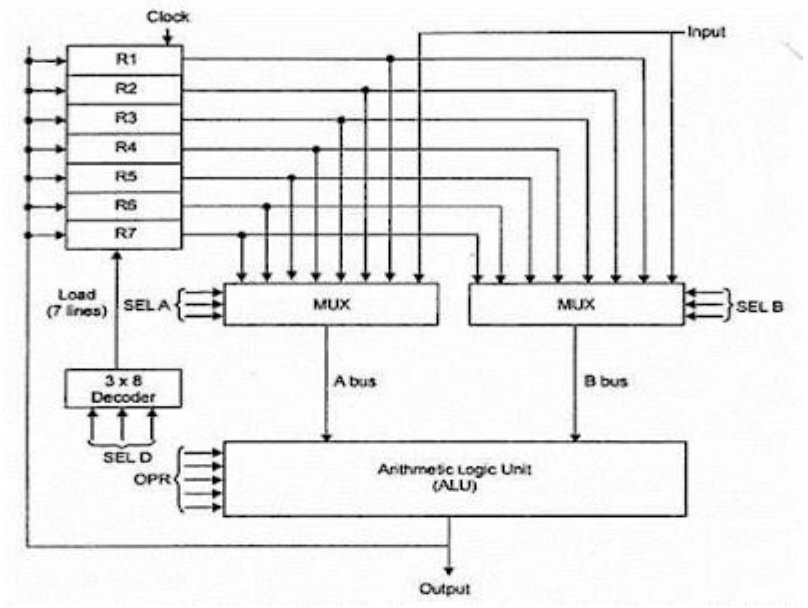
The Central Processing Unit (CPU) is called the brain of the computer that performs data processing operations. Figure 3.1 shows the three major parts of CPU.



- Intermediate data is stored in the register set during the execution of the instructions.
- The microoperations required for executing the instructions are performed by the arithmetic logic unit whereas the control unit takes care of transfer of information among the registers and guides the ALU.

- The control unit services the transfer of information among the registers and instructs the ALU about which operation is to be performed.
- The computer instruction set is meant for providing the specifications for the design of the CPU. The design of the CPU largely, involves choosing the hardware for implementing the machine instructions.
- The need for memory locations arises for storing pointers, counters, return address, temporary results and partial products.
- Memory access consumes the most of the time off an operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers.
- A common bus system is employed to contact registers that are included in the CPU in a

  large number.
- Communications between registers is not only for direct data transfer but also for performing various micro-operations.
- A bus organization for such CPU register shown in Figure 3.2, is connected to two multiplexers (MUX) to form two buses A and B. The selected lines in each multiplexers select one register of the input data for the particular bus.



EXAMPLE:

- To perform the operation ***R3 = R1+R2*** We have to provide following binary selection variable to the select inputs.

1. **SEL A** : **001** -To place the contents of R1 into bus A.
2. **SEL B** : **010** - to place the contents of R2 into bus B
3. **SEL OPR** : **10010** – to perform the arithmetic addition A+B
4. **SEL REG or SEL D : 011** – to place the result available on output bus in R3

Register and multiplexer input selection code

Binary code	SELA	SELB	SELD or SELREG
000	Input	Input	---
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

Operation with symbol

Operation selection code	Operation	Symbol
0000	Transfer A	TSFA
0001	Increment A	INC A
0010	A+B	ADD
0011	A-B	SUB
0100	Decrement A	DEC
0101	A AND B	AND
0110	A OR B	OR
0111	A XOR B	XOR
1000	Complement A	COMA
1001	Shift right A	SHR
1010	Shift left A	SHL

Instruction Formats

- Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields.
- These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform.
  The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

- A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

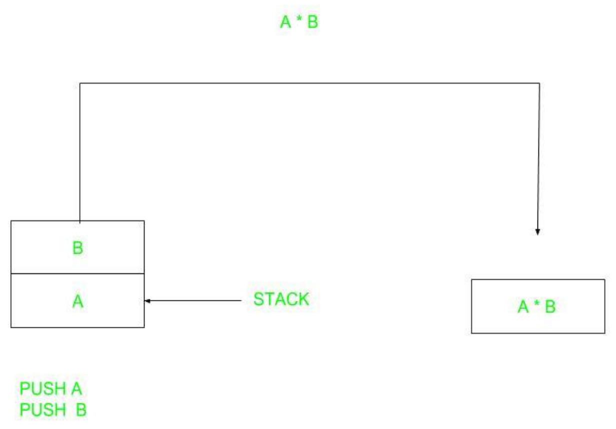
1. Single Accumulator organization
2. General register organization
3. Stack organization

- In first organization operation is done involving a special register called accumulator.
- In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field.
- It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**



- A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2.One Address Instructions –

- This use a implied ACCUMULATOR register for data manipulation.
- One operand is in accumulator and other is in register or memory location.
- Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

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

3.Two Address Instructions –

- This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

opcode	Destination address	Source address	mode

Here destination address can also contain operand.

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4.Three Address Instructions –

- This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase.
- These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only

opcode	Destination address	Source address	Source address	mode

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

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

Adressing Modes:

- The operation field of an instruction specifies the operation to be performed.
- This operation will be executed on some data which is stored in computer registers or the main memory.
- The way any operand is selected during the program execution is dependent on the addressing mode of the instruction.
- The purpose of using addressing modes is as follows:

1. To give the programming versatility to the user.
2. To reduce the number of bits in addressing field of instruction.

Types of Addressing Modes

Below we have discussed different types of addressing modes one by one:

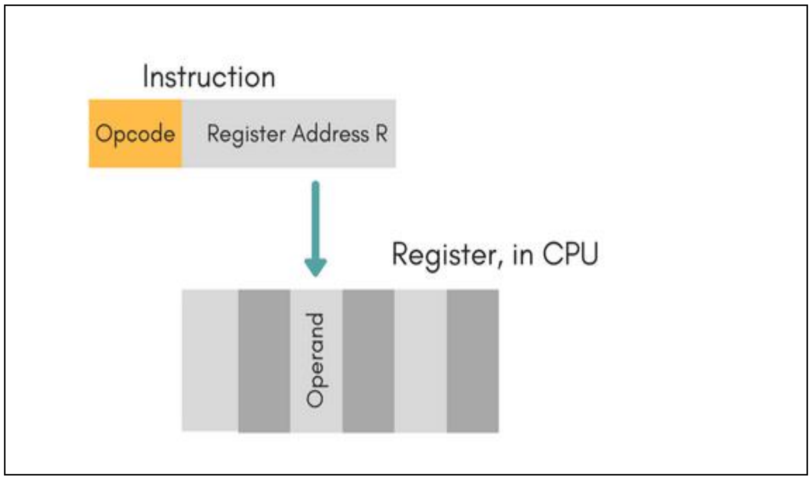
Immediate Mode:

In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

For example: `ADD 7`, which says Add 7 to contents of accumulator. 7 is the operand here.

Register Mode:

In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.



Advantages

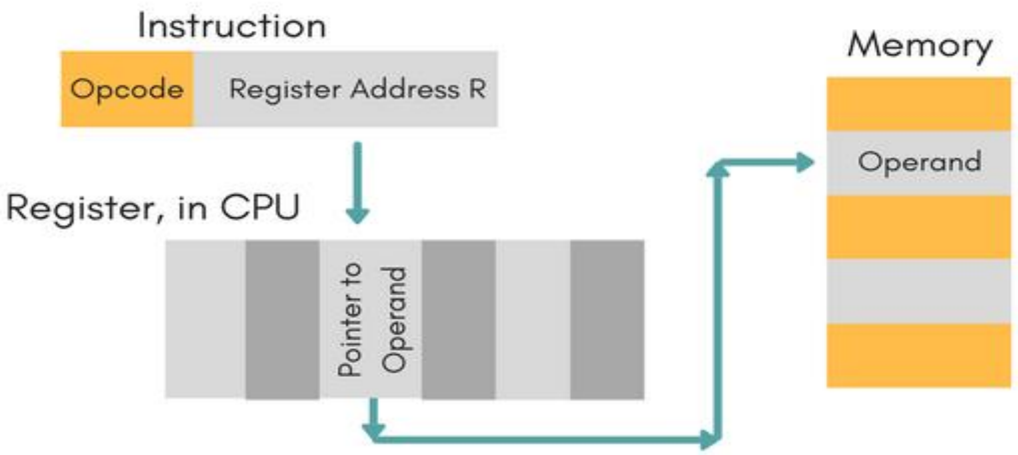
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

Register Indirect Mode:

In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



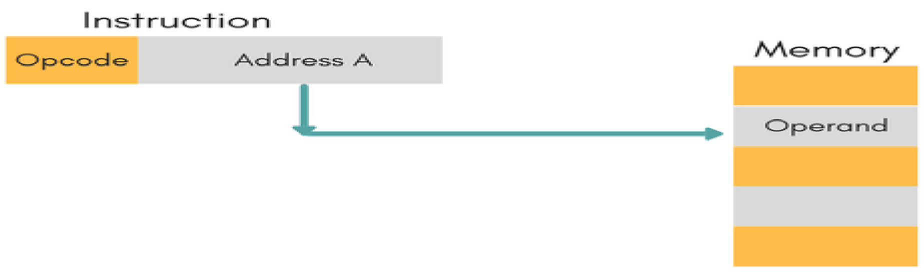
Auto Increment/Decrement Mode:

In this the register is incremented or decremented after or before its value is used.

Direct Addressing Mode:

In this mode, effective address of operand is present in instruction itself.

- Single memory reference to access data.
- No additional calculations to find the effective address of the operand.

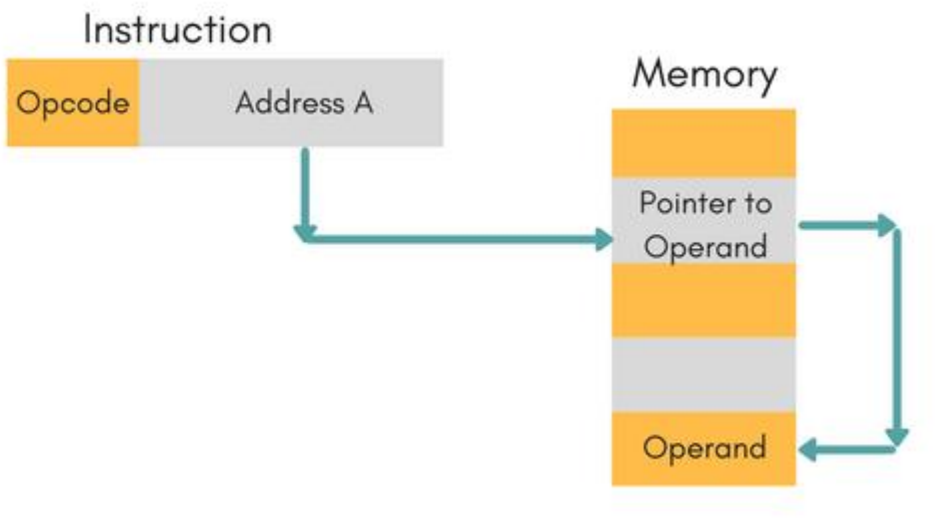


For Example: ADD R1, 4000 - In this the 4000 is effective address of operand.

NOTE: Effective Address is the location where operand is present.

Indirect Addressing Mode:

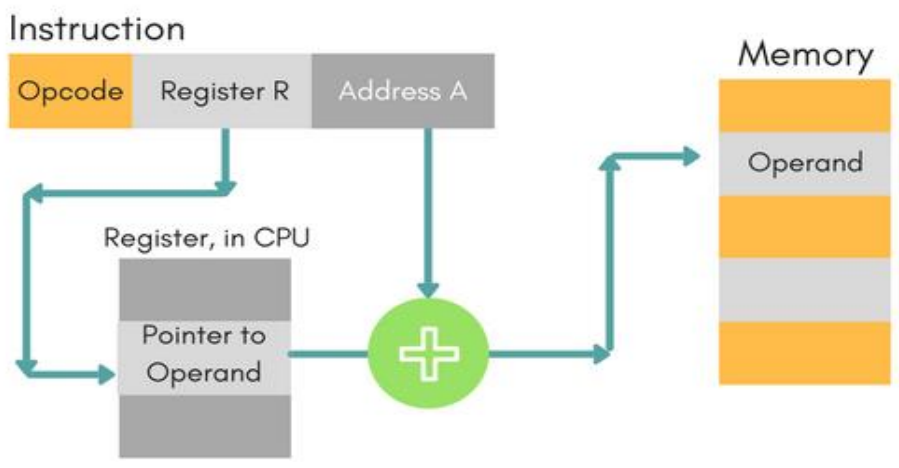
In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.



Displacement Addressing Mode:

In this the contents of the indexed register is added to the Address part of the instruction, to obtain the effective address of operand.

EA = A + (R), In this the address field holds two values, A(which is the base value) and R(that holds the displacement), or vice versa.



Relative Addressing Mode

It is a version of Displacement addressing mode.

In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.

EA = A + (PC), where EA is effective address and PC is program counter.

The operand is A cells away from the current cell(the one pointed to by PC)

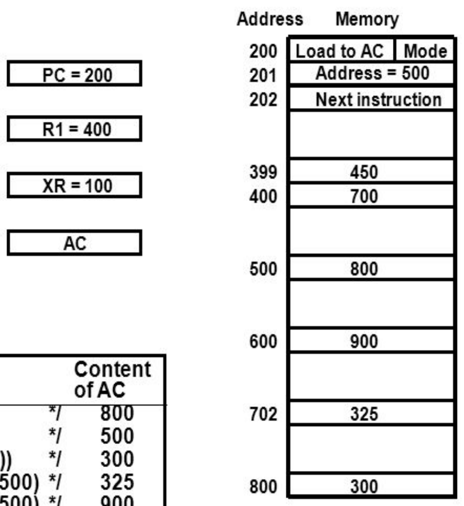
Base Register Addressing Mode

It is again a version of Displacement addressing mode. This can be defined as EA = A + (R), where A is displacement and R holds pointer to base address.

Stack Addressing Mode

In this mode, operand is at the top of the stack. For example: ADD, this instruction will *POP* top two items from the stack, add them, and will then *PUSH* the result to the top of the stack.

ADDRESSING MODES - EXAMPLES



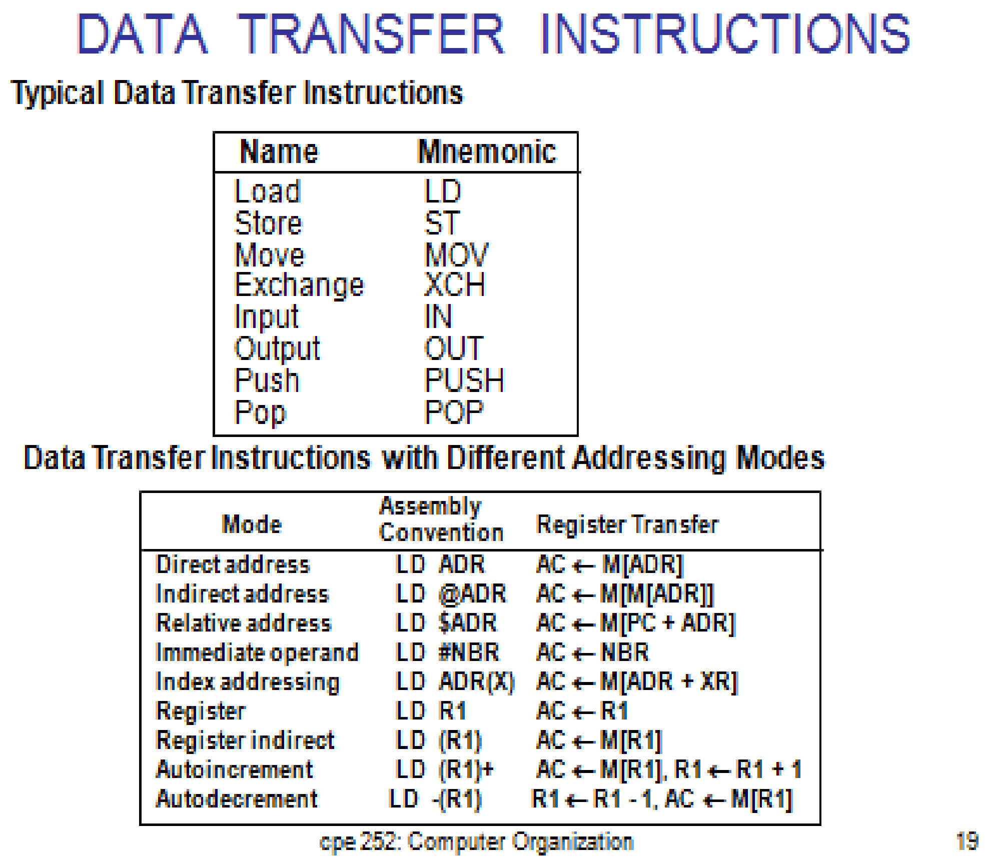
Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R1) */	450

6

Data Transfer Instructions:

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- The load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The store instruction designates a transfer from a processor register into memory.
- The move instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.
- The exchange instruction swaps information between two registers or a register and a memory word.
- The input and output instructions transfer data among processor registers and input or output terminals.

- The push and pop instructions transfer data between processor registers and a memory stack.



DATA TRANSFER INSTRUCTIONS

Typical Data Transfer Instructions

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

Data Transfer Instructions with Different Addressing Modes

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

cpe 252: Computer Organization 19

DATA MANIPULATION INSTRUCTIONS:

DATA MANIPULATION INSTRUCTIONS

Three Basic Types: **Arithmetic instructions** **Logical and bit manipulation instructions**
Shift instructions

Arithmetic Instructions

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

Logical and Bit Manipulation Instructions

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

Shift Instructions

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 thru carry	RORC
Rotate left thru carry	ROLC

PROGRAM CONTROL INSTRUCTIONS:

- It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions be stored for further analysis. Status bits are also called condition-code bits or flag bits.
- Figure 5.3 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.

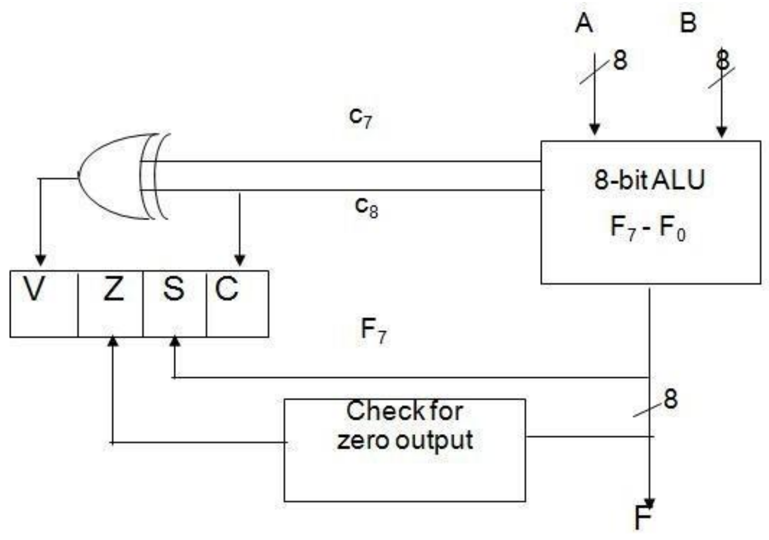


Figure 5.3: Status Register Bits

1. Bit C (carry) is set to 1 if the end carry C8 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 F7 is 1. It is set to 0 if 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 exclusives-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.

- The status bits can be checked after an ALU operation to determine certain relationships that exist between the vales of A and B.
- If bit V is set after the addition of two signed numbers, it indicates an overflow condition.
- If Z is set after an exclusive-OR operation, it indicates that A = B.
- A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST

CMP and TST instructions do not retain their results of operations (- and AND, respectively). They only set or clear certain Flags.