UNIT- II

Micro Programmed Control AND Central Processing Unit

Microprogrammed Control: Control memory, Address sequencing, micro program example, design of control unit.

Central Processing Unit: General Register Organization, Instruction Formats, Addressing modes, Data Transfer and Manipulation, Program Control

---

Control memory:

- A computer that employs a micro programmed control unit will have two separate memories:
  - Main memory
  - Control memory.
- The main memory is available to the user for storing the program. The user's program in main memory consists of machine instructions and data.
- In contrast, the control memory holds a fixed micro program that cannot be altered by the occasional user.
- The micro program consists of microinstructions that specify control signals for execution of register micro operations.
- These micro instructions generate the micro operations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to **repeat the cycle** for the next instruction.
- The general configuration of a micro programmed control unit is demonstrated in the block diagram of Fig. 2-A. **The control memory is assumed to be a ROM**, within which all control information, is permanently stored.

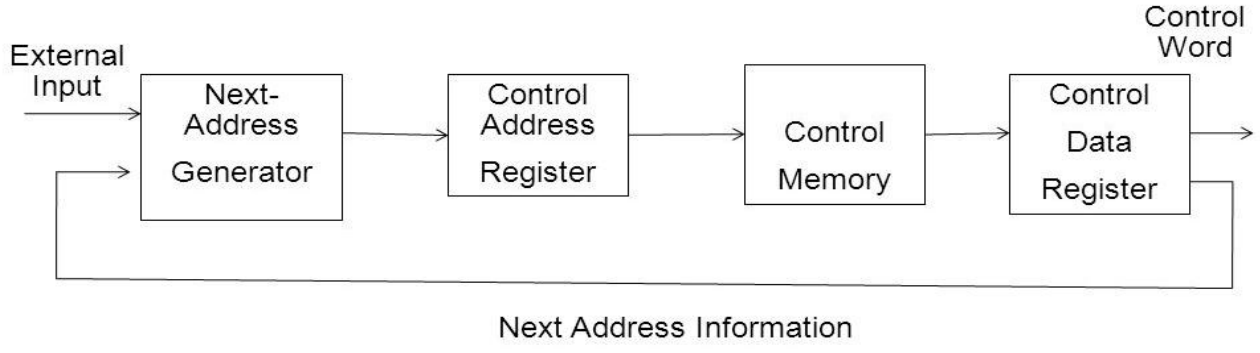

Fig. 2-A: Micro programmed control organization

- The control address register specifies the address of the microinstruction.

- The control data register (CDR) holds the microinstruction read from memory. The microinstruction contains a control word that specifies one or more micro operations. Once these operations are executed, the control must determine the nest 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.
- The next address generator is sometimes called a **micro program sequencer**, as it determines the address sequence that is read from control memory.
- 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**.

❖ **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.

❖ It should be mentioned that most computers based on the reduced instruction set computer (**RISC**) architecture concept use hardwired control rather than a control memory with a micro program.

Address sequencing:

❖ Microinstructions are stored in control memory in groups, with each group specifying a routine.

❖ Each computer instruction has its own micro program routine in control memory to generate the micro operations that execute the instruction.

❖ The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.

Let us enumerate the steps that the execution of a single computer instruction.

Step-1:

- An initial address is loaded into the control address register .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 micro operations that execute the instruction fetched from memory.
- The micro operation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction 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.

- Figure 2-B 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 micro operations 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.

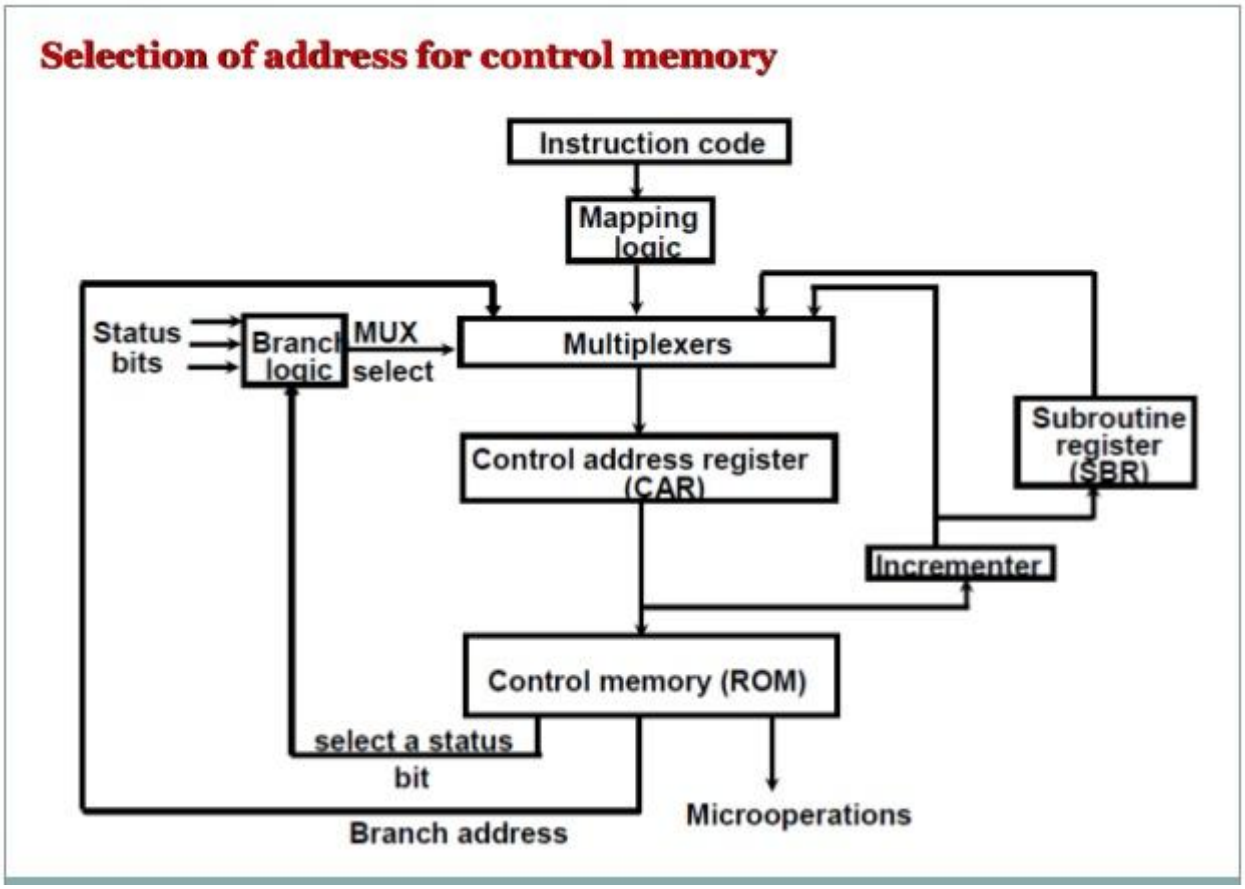

Figure 2-B: Selection of address for control memory

Conditional Branching:

- The branch logic of Fig. 2-B 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.
- Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0.
- The status bits, the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

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 micro program 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 2-C 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 2-C.
- 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 micro program 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.

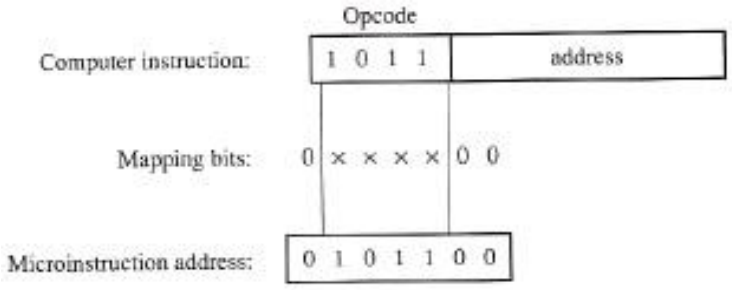

Figure 2-C: Mapping from instruction code to microinstruction address.

Micro program Example:

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

To appreciate this process, we present here a simple digital computer and show how it is micro programmed.

Computer Configuration:

- The block diagram of the computer is shown in Fig. 2-D.

- It consists of two memory units: a **main memory** for storing instructions and data, and a **control memory** for storing the micro program. Four registers are associated with the processor unit and two with the control unit. The processor registers are program counter PC, address register AR, data register DR, and accumulator register AC.
- The function of these registers is similar to the basic computer (Fig 1-CC). The control unit has a control address register CAR and a subroutine register SBR.
- The control memory and its registers are organized as a micro programmed control unit, as shown in Fig. 2-B.
- The transfer of information among the register in the processor is done through multiplexers rather than a common bus. DR can receive information from AC, PC, or memory. AR can receive information from PC or DR, PC can receive information only from AR.
- The arithmetic, logic, and shift unit performs micro operations with data from AC and DR and places the result in AC. Note that memory receive its address from AR. Input data written to memory come from DR, and data read from memory can go only to DR.

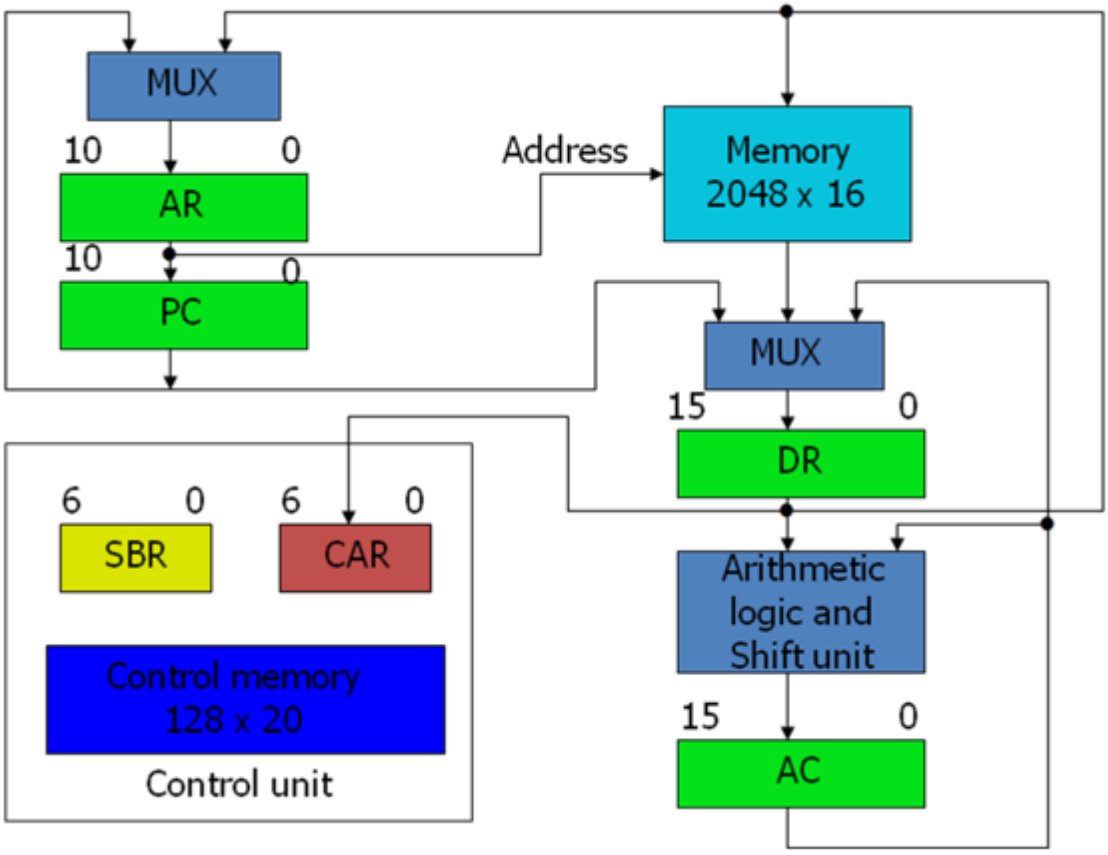

Figure 2-D: Computer hardware configuration

Microinstruction Format:

- The microinstruction format for the control memory is shown in Fig. 2-E. The 20 bits of the microinstruction are divided into four functional parts.
- The three fields F1, F2, and F3 specify micro operations for the computer. The CD field selects status bit conditions. The BR field specifies the type of branch to be used. The AD field contains a branch address.

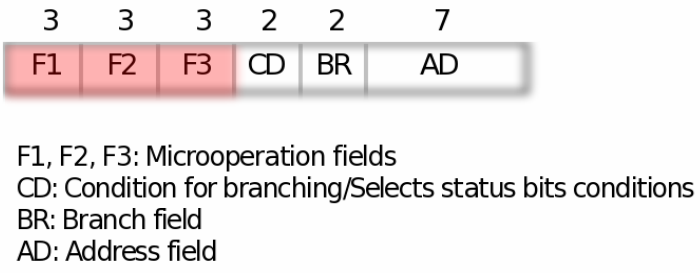

Figure 2-E: Microinstruction code format (20 bits).

- The address field is seven bits wide, since the control memory has $128 = 2^7$ words.
- The micro operations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct micro operations as listed in Table 2-1.
- This gives a total of 21 micro operations. No more than three micro operations can be chosen for a micro instruction, one from each field.
- If fewer than three micro operations are used, one or more of the fields will use the binary code 000 for no operation.

  As an illustration, a microinstruction can specify two simultaneous micro operation from F2 and F3 and none from F1.

$$DR \leftarrow M[AR] \text{ with } F2 = 100$$

$$\text{and } PC \leftarrow PC + 1 \text{ with } F3 = 101$$

The nine bits of the micro operation fields will then be 000 100 101.

- It is important to realize that two or more conflicting micro operations cannot be specified simultaneously.

  For example, a micro operation field 010 001 000 has no meaning because it specifies the operations to clear AC to 0 and subtract DR from AC at the same time.
- Each micro operation in Table 2-1 is defined with a register transfer statement and is assigned a **symbol** for use in a symbolic micro program.
- All transfer-type **micro operations symbols use five letters**. The first two letters designate the source register, the third letters is always a T, and the last two letters designate the destination register.

For example, the micro operation that specifies the transfer AC ← DR (F1 = 100) has the symbol DRTAC, which stands for a transfer from DR to AC.

Table 2-1 Symbols and Binary Code for Microinstruction Fields

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC V DR	OR
011	AC ← AC ∧DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

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

- **The CD (condition)** field consists of two bits which are encoded to specify four status bit conditions as listed below.
  - The first condition is always a 1, so that a reference to CD = 00 (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation.
  - The indirect bit **I** is available from bit 15 of DR after an instruction is read from memory.
  - The sign bit of AC provides the next status bit.
  - The zero value, symbolized by Z, is a binary variable whose value is equal to 1 if all the bits in AC are equal to zero.

- We will use the symbols U, I, S, and Z for the four status bits when we write micro programs in symbolic form.

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

❖ **The BR (branch**) field consists of two bits. It is used, in conjunction with address field AD, to choose the address of the next microinstruction.

- As shown below, when BR = 00, the control performs a jump (JMP) operation (which is similar to a branch), and when BR = 01, it performs a call to subroutine (CALL) operation.
- The two operations are identical except that a call microinstruction stores the return address in the **subroutine register SBR**.
- The jump and call operations depend on the value of the CD field. If the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to the control address register CAR. Otherwise, CAR is incremented by 1.
- The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR.
- The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11.
- The bits of the operation code are in DR after an instruction is read from memory. Note that the last two conditions in the BR field are independent of the values in the CD and AD fields.

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

DESIGN OF CONTROL UNIT:

- The control memory output of each subfield must be decoded to provide the distinct micro operations. The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- Figure 2-F shows the three decoders and some of the connections that must be made from their outputs.
- Each of the three fields of the microinstruction presently available in control memory are decoded with a 3 × 8 decoder to provide eight outputs. Each of these outputs must be connected to the proper circuit to initiate the corresponding micro operation as specified in Table 2-1.

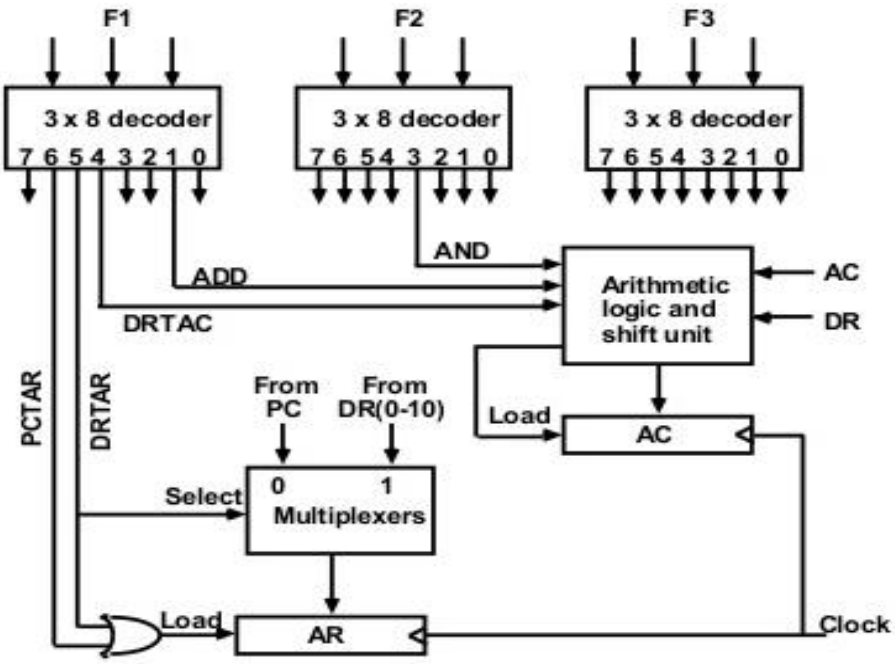

Figure 2-F: Decoding of microoperation fields

- For example, when FI = 101 (binary 5), the next clock pulse transition transfers the content of DR (0-10) to AR (symbolized by DRTAR in Table 2-1).
- Similarly, when F1 = 101 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR). As shown in Fig. 2-F, outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.
- The multiplexers select the information from DR when output 5 is active and from PC when output 5 in inactive.
- The transfer into AR occurs with a clock pulse transition only when output 5 or output 6 of the decoder are active. The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion.

- The arithmetic logic shift unit can be designed, instead of using gates to generate the control signals marked by the symbols AND, ADD, and DR in Fig 2-F, these inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively as shown in Fig. 2-F.
- The other output of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

INSTRUCTION FORMATS

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:
    1. An operation code field that specifies the operation to be performed.
    2. An address field that designates a memory address or a processor registers.
    3. A mode field that specifies the way the operand or the effective address is determined.
- The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.
- The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.
- The various addressing modes that have been formulated for digital computers are discussed in Addressing Modes concept.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

Most computers fall into one of three types of CPU organizations:

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

❖ **An example of an accumulator-type organization**:

All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.

For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as **ADD X**

- ✓ Where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC + M[X].
- ✓ AC is the accumulator register and M[X] symbolizes the memory word located at address X.

❖ **An example of a general register type of organization**:

The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

ADD R1, R2, R3

- ✓ To denote the operation R1 ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
- ✓ Thus the instruction **ADD R1, R2** Would denote the operation R1 ← R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.
- ✓ Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction.

  Thus the instruction **MOV R1, R2**
- ✓ Denotes the transfer R1 ← R2 (or R2 ← R1, depending on the particular computer).
- ✓ Thus transfer-type instructions need two address fields to specify the source and the destination.

➢ General register-type computers employ two or three address fields in their instruction format.

➢ Each address field may specify a processor register or a memory word.

➢ An instruction symbolized by ADD R1, X Would specify the operation R1 ← R1 + M [X]. It has two address fields, one for register R1 and the other for the memory address X

❖ **The stack-organized:** The Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction **PUSH X** Will push the word at address X to the top of the stack.

- ✓ The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack.
- ✓ The instruction ADD In a stack computer consists of an operation code only with no address field.
- ✓ This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack.
- ✓ There is no need to specify operands with an address field since all operands are implied to be in the stack.

Example: To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement **X = (A + B) * (C + D).**

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

❖ Three-Address Instructions:

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

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

- It is assumed that the computer has two processor registers, R1 and R2. The symbol M [A] denotes the operand at memory address symbolized by A.
- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

❖ Two-Address Instructions:

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

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

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

One-Address Instructions:

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

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

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

❖ Zero-Address Instructions:

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

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

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

ADDRESSING MODES:

- The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution in dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
  1. To give programming versatility to the user by providing such facilities as pointers to Memory, counters for loop control, indexing of data, and program relocation
  2. To reduce the number of bits in the addressing field of the instruction.

- The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.
- To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the computer. The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:
  1. Fetch the instruction from memory
  2. Decode the instruction.
  3. Execute the instruction

There is one register in the computer called the program counter of PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory.

The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction. Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.

An example of an instruction format with a distinct addressing mode field is shown in Fig. 1. The operation code specified the operation to be performed. The mode field is sued to locate the operands needed for the operation. There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register. Moreover, as discussed in the preceding section, the instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.

Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

1. Implied Mode:

- In this mode the operands are specified implicitly in the definition of the instruction.
- For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions.

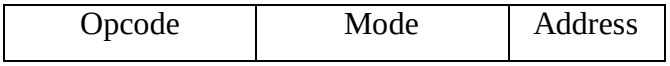

Figure: Instruction format with mode field

- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2. Immediate Mode:

- In this mode the operand is specified in the instruction itself. In other words, an immediate mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate-mode instructions are useful for initializing registers to a constant value.
- It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register.
- When the address field specifies a processor register, the instruction is said to be in the register mode.

3. **Register Mode:**
   - In this mode the operands are in registers that reside within the CPU.
   - The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.
4. **Register Indirect Mode:**
   - In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself.
   - Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
   - A reference to the register is then equivalent to specifying a memory address.
   - The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
5. **Auto increment or Auto decrement Mode:**
   - This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
   - When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
   - This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access.
   - The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory. Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.
   - To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction.
   - The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
   - The effective address is the address of the operand in a computational-type instruction. It is the address where control branches in response to a branch-type instruction.

6. **Direct Address Mode:**
   - In this mode the effective address is equal to the address part of the instruction.
   - The operand resides in memory and its address is given directly by the address field of the instruction.
   - In a branch-type instruction the address field specifies the actual branch address.
7. **Indirect Address Mode:**
   - In this mode the address field of the instruction gives the address where the effective address is stored in memory.
   - Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
8. **Relative Address Mode:**
   - In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
   - The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative.
   - When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
   - **To clarify with an example**, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24. The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction.
   - Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.
9. **Indexed Addressing Mode:**
   - In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
   - The index register is a special CPU register that contains an index value.
   - The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the beginning address.
   - The distance between the beginning address and the address of the operand is the index value stores in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value.

- The index register can be incremented to facilitate access to consecutive operands. Note that if an index-type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation.
- Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used.
- In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

10. Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- The difference between the two modes is in the way they are used rather
- than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.
- When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of the base register requires updating to reflect the beginning of a new memory segment.

Numerical Example:

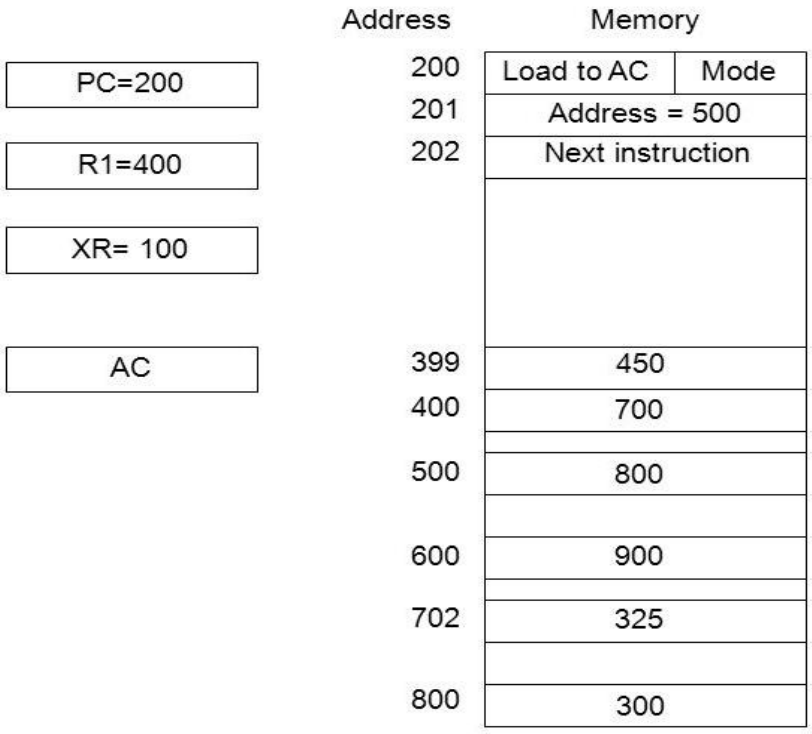

Figure 2-G : Numerical example for addressing modes.

The mode field of the instruction can specify any one of a number of modes. For each possible mode we calculate the effective address and the operand that must be loaded into AC .

- In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.
- In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC . (The effective address in this case is 201 .)
- In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.
- In the relative mode the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202.)
- In the index mode the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900.
- In the register mode the operand is in R 1 and 400 is loaded into AC . (There is no effective address in this case.)
- In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700.

- The auto increment mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction.
- The auto decrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.

Table lists the values of the effective address and the operand loaded into AC for the nine addressing modes.

Addressing Mode	Effective Address	Content of AC
Direct Address	500	800
Immediate operand	201	500
Indirect Address	800	300
Relative Address	702	325
Indexed Address	600	900
Register	-	400
Register Indirect	400	700
Auto increment	400	700
Auto decrement	399	450

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.
- Table 2-H gives a list of eight data transfer instructions used in many computers.
- Accompanying each instruction is a mnemonic symbol.
- **NOTE**: It must be realized that different computers use different mnemonics for the same instruction name.

❖ 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.

TABLE 2-H: Typical Data Transfer Instructions

Name	Mnemonic
LOAD	LD
STORE	ST
MOVE	MOV
EXCHANGE	XCH
INPUT	IN
OUTPUT	OUT
PUSH	PUSH
POP	POP

Data Manipulation Instructions:

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:
    1. Arithmetic instructions
    2. Logical and bit manipulation instructions
    3. Shift instructions

1. Arithmetic instructions:

A list of typical arithmetic instructions is given in Table 2-I.

- The increment instruction adds 1 to the value stored in a register or memory word.
- One common characteristic of the increment operations when executed in processor registers is that a binary number of all 1' s when incremented produces a result of all 0' s.
- The decrement instruction subtracts 1 from a value stored in a register or memory word. A number with all D's, when decremented, produces a number with all 1's.
- The add, subtract, multiply, and divide instructions may be available for different types of data.
- The data type assumed ·to be in processor registers during the execution of these arithmetic operations is included in the definition of the operation code.
- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision or double-precision data.

TABLE 2-I: Typical Arithmetic Instructions

Name	Mnemonic
INCREMENT	INC
DECREMENT	DEC
ADD	ADD
SUBTRACT	SUB
MULTIPLY	MUL
DIVIDE	DIV
ADD WITH CARRY	ADDC
SUBTRACT WITH BORROW	SUBB
NEGATIVE	NEG

2. Logical and bit manipulation instructions:

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary-coded information.

Some typical logical and bit manipulation instructions are listed in Table 2-J .

- The clear instruction causes the specified operand to be replaced by 0's.
- The complement instruction produces the 1's complement by inverting all the bits of the operand.
- The AND, OR, and XOR instructions produce the corresponding logical operations on individual bits of the operands.
- Although they perform Boolean operations, when used in computer instructions, the logical instructions should be considered as performing bit manipulation operations.
- There are three bit manipulation operations possible: a selected bit can be cleared to 0, or can be set to 1, or can be complemented.

TABLE 2-J: Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

3. Shift instructions:

- Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify either logical shifts, arithmetic shifts, or rotate-type operations. In either case the shift may be to the right or to the left.

Table 2--K lists four types of shift instructions.

- The logical shift inserts 0 to the end bit position. The end position is the leftmost bit for shift right and the rightmost bit position for the shift left.
- The arithmetic shift-right instruction must preserve the sign bit in the leftmost position. The sign bit is shifted to the right together with the rest of the number, but the sign bit itself remains unchanged. This is a shift-right operation with the end bit remaining the same.
- The arithmetic shift-left instruction inserts 0 to the end position and is identical to the logical shift-left instruction.
- The rotate instructions produce a circular shift. Bits shifted out at one end of the word are not lost as in a logical shift but are circulated back into the other end.

TABLE 2-K: Typical Shift Instructions

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

Program Control:

- Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed.

Some typical program control instructions are listed in Table 2-L.

Name	Mnemonic
BRANCH	BR
JUMP	JMP
SKIP	SKP
CALL	CALL
RETURN	RET
COMPARE	CMP
TEST	TST

- **Branch and jump** instructions may be conditional or unconditional. An unconditional branch instruction causes a branch to the specified address without any conditions.
  - ✓ The conditional branch instruction specifies a condition such as branch if positive or branch if zero.
  - ✓ If the condition is met, the program counter is loaded with the branch address and the next instruction is taken from this address.
  - ✓ If the condition is not met, the program counter is not changed and the next instruction is taken from the next location in sequence
- The **skip** instruction does not need an address field and is therefore a zero-address instruction.
  - ✓ A conditional skip instruction will skip the next instruction if the condition is met. This is accomplished by incrementing the program counter during the execute phase in addition to its being incremented during the fetch phase.
  - ✓ If the condition is not met, control proceeds with the next instruction in sequence where the programmer inserts an unconditional branch instruction.

Status Bit Conditions:

- Status bits are also called condition-code bits or flag bits. Figure 2-M 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**.**

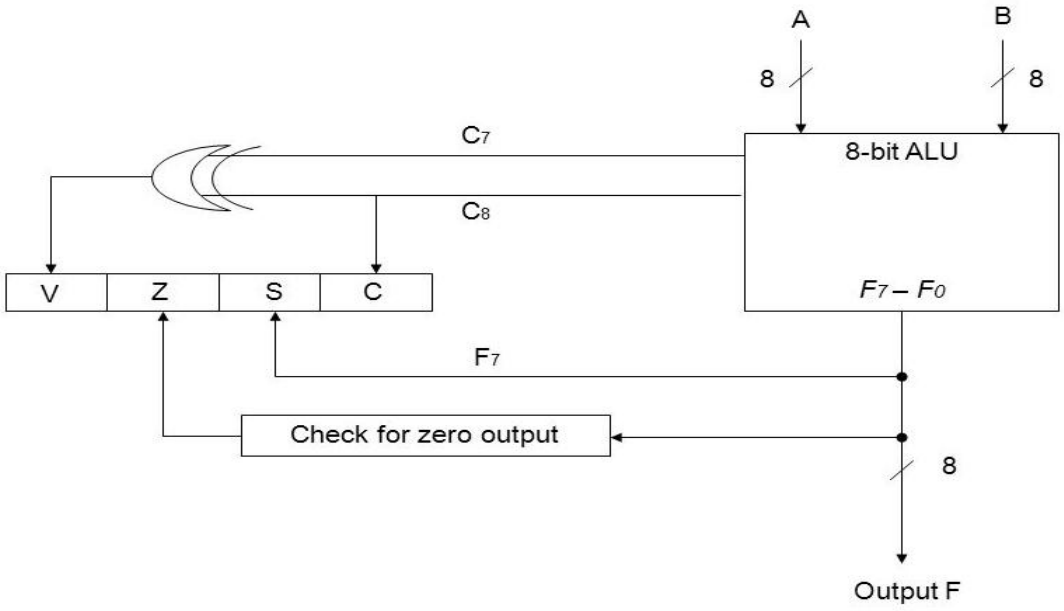

Figure 2-M: Status registers bits.

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 F_7 is 1. It is set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. !t is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement