## Instruction Codes

A computer instruction is a binary code that specifies a sequence of microoperations for the computer. Instruction codes together with data are stored in memory. 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 microoperations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general-purpose computer. An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

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 memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

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. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory. Fig(12) depicts this type of organization.

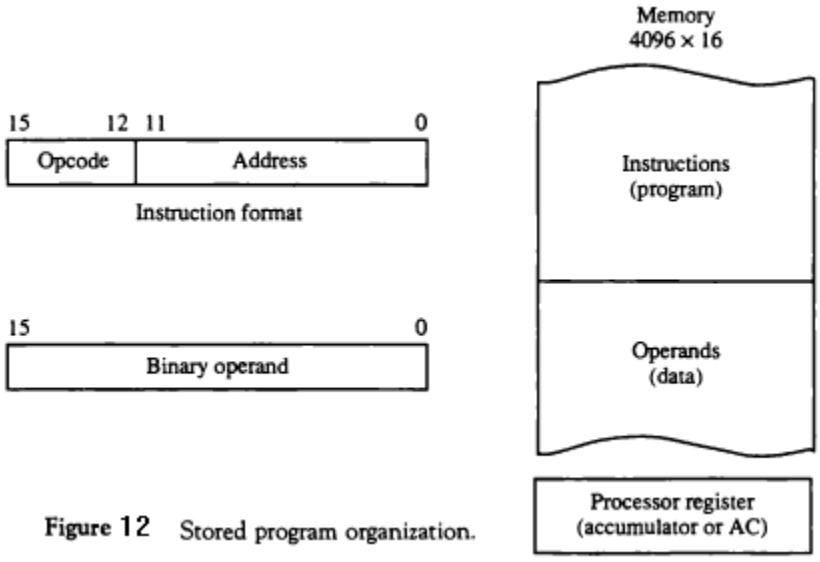



**Figure 12** Stored program organization.

Computers that have a single-processor register usually assign to it the name accumulator and label it AC.

For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register.

When the second part of an instruction immediate code specifies an operand, this type is called **immediate operand**. When the second part specifies the address of an operand, the instruction is said to have a **direct address**. The third possibility called **indirect address**, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found. One bit of the instruction code can be used to distinguish between a direct and an indirect address.

As an illustration of this configuration, consider the instruction code format shown in Fig(13).

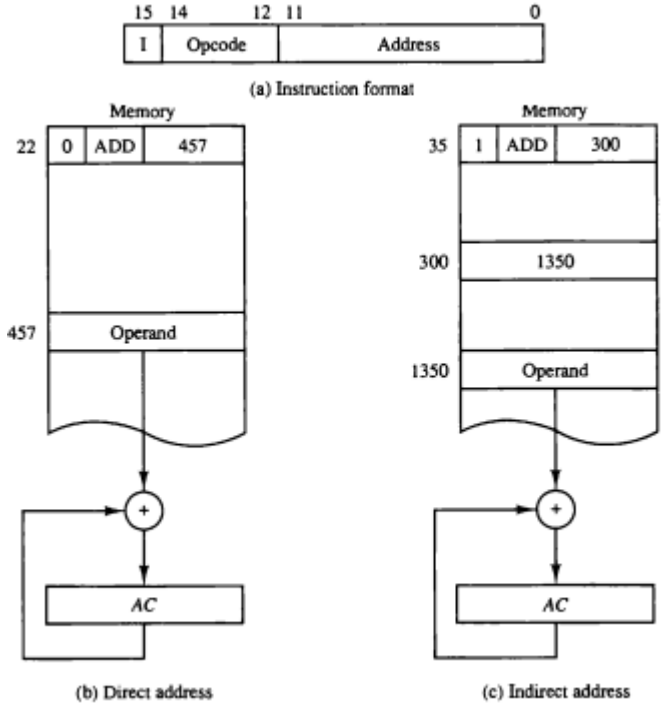



**Fig 13** Demonstration of direct and indirect address.

## Computer Registers

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on. This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed. The computer needs processor registers for manipulating data and a register for holding a memory address. These requirements dictate the register configuration are listed in Table(9) and shown in Fig(14).

TABLE 9 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 |

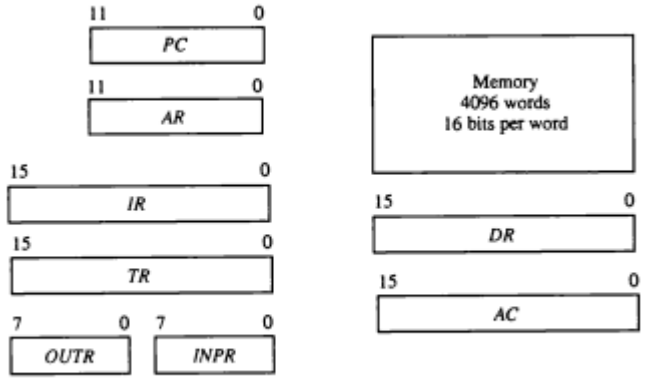



Figure 14 Basic computer registers and memory.

The basic computer has eight registers, a memory unit, and a control unit. The connection of the registers and memory of the basic computer to a common bus system is shown in Fig(15).

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 receive 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 and communicate with the eight least significant bits in the bus. The 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. There is no transfer from OUTR to any of the other registers.
Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).

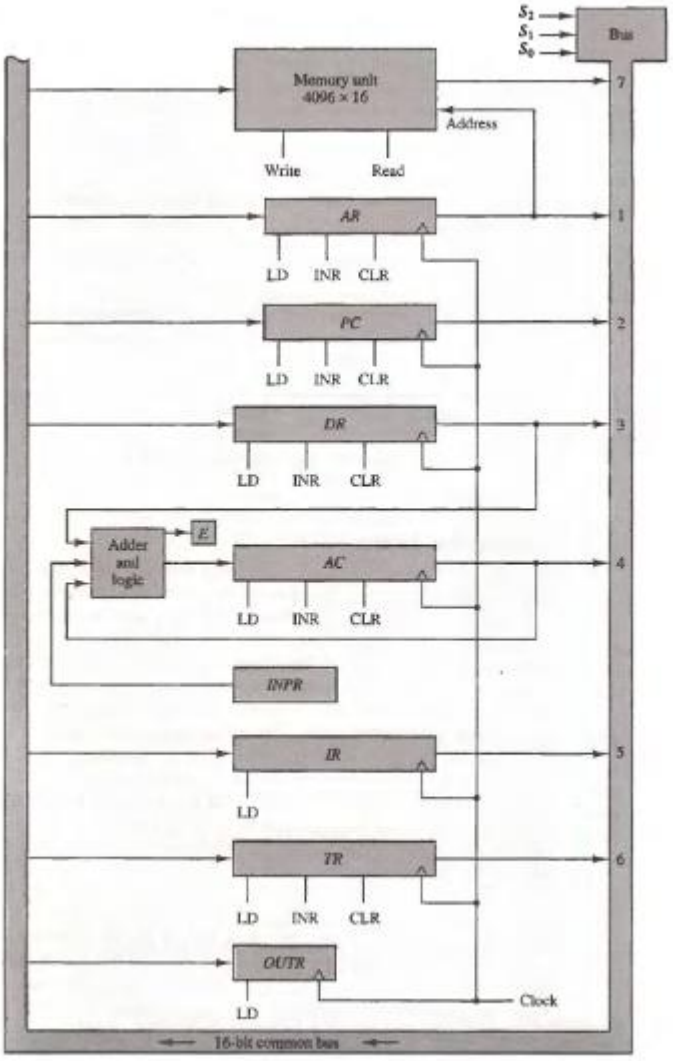



Figure 15 Basic computer registers connected to a common bus.

## Computer Instructions

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

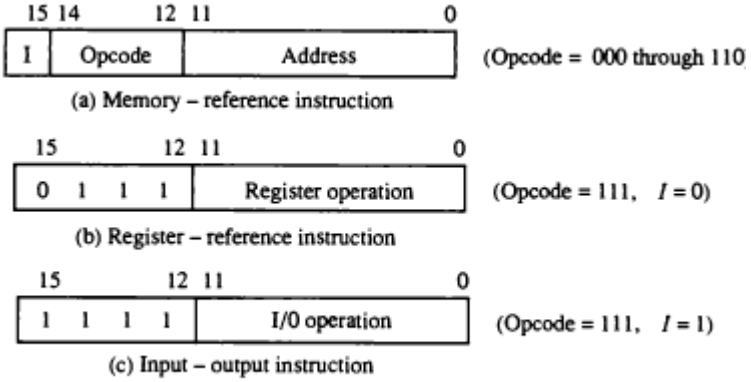



Figure 16 Basic computer instruction formats.

The type of instruction is recognized by the computer control from the four bits in positions 12through 15 of the instruction. If the three opcode bits in positions 12 through 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode J. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, the instruction is a register-reference type. If the bit is 1, the instruction is an input-output type. Note that the bit in position 15 of the instruction code is designated by the symbol J but is not used as a mode bit when the operation code is equal to 111.

The instructions for the computer are listed in Table(10):

**TABLE 10** Basic Computer Instructions

| Symbol | Hexadecimal code $I = 0$ | Hexadecimal code $I = 1$ | 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 |

The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

1. Arithmetic, logical, and shift instructions.
2. Instructions for moving information to and from memory and processor registers.
3. Instructions that check status information to provide decision making capabilities.
4. Input and output instructions.
5. The capability of stopping the computer.

## 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 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:

1- **hardwired control** : the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.

2- **microprogrammed control:** the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.

The block diagram of the control unit is shown in Fig(17):

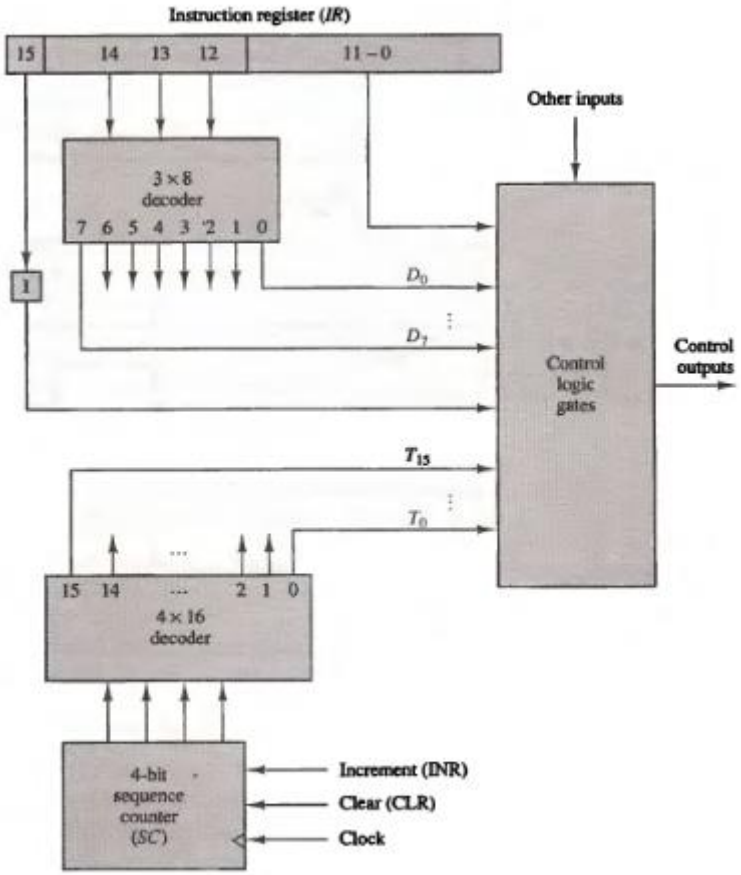


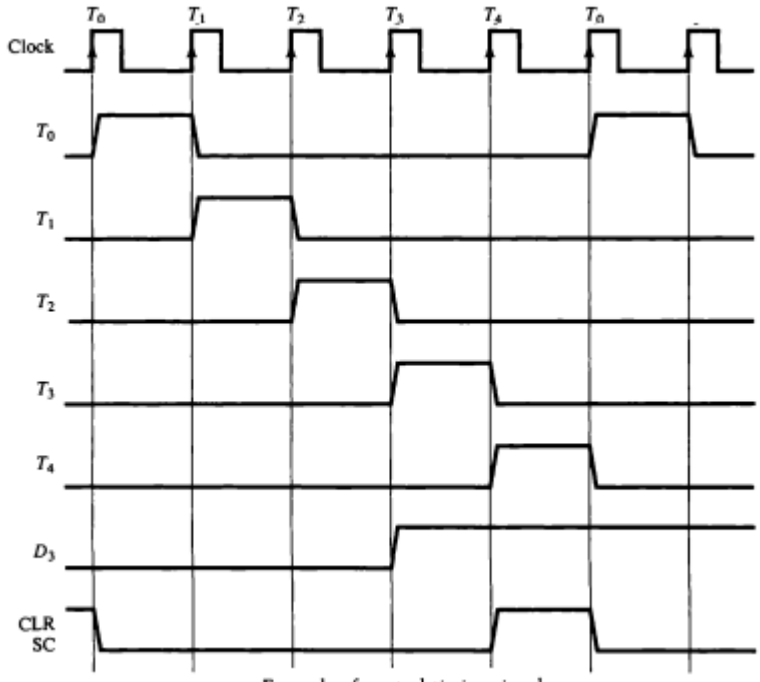

Figure 17 Control unit of basic computer.

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. (Note the relationship between the timing signal and its corresponding positive

clock transition.) If SC is not cleared, the timing signals will continue with $T_5$, $T_6$, up to $T_{15}$ and back to $T_0$.

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 $D_3$ is active. This is expressed symbolically by the statement

$$D_3T_4: SC \leftarrow 0$$




Example of control timing signals.

## 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. 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.

This process continues indefinitely unless a HALT instruction is encountered. 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 $T_0$. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence $T_0$, $T_1$, $T_2$, and so on. The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

$$T_0: AR \leftarrow PC$$

$$T_1: IR \leftarrow M[AR], PC = PC + 1$$

$$T_2: D_0,,,,D_7 \leftarrow Decode\ IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$$

It is necessary to use timing signal $T_1$ to provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0$ =111.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

## Memory - Reference Instructions

The table(11) lists the seven memory-reference instructions. The decoded output D, for i = 0,1, 2, 3, 4, 5, and 6 from the operation decoder that belongs effective address 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 symbolic description of each instruction is specified in the table in terms of register transfer notation. The actual execution of the instruction in the bus system will require a sequence of microoperations.

**TABLE 11** Memory-Reference Instructions

| Symbol | Operation decoder | Symbolic description | Means |
|---|---|---|---|
| AND | $D_0$ | $AC \leftarrow AC \wedge M[AR]$ | AND to AC |
| ADD | $D_1$ | $AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$ | ADD to AC |
| LDA | $D_2$ | $AC \leftarrow M[AR]$ | LOAD to AC |
| STA | $D_3$ | $M[AR] \leftarrow AC$ | STORE AC |
| BUN | $D_4$ | $PC \leftarrow AR$ | Branch Unconditional |
| BSA | $D_5$ | $M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$ | Branch and save address |
| ISZ | $D_6$ | $M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$ | Increment and skip if 0 |

## Input - Output and Interrupt

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. The input-output configuration is shown in Fig(18). The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially. 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.

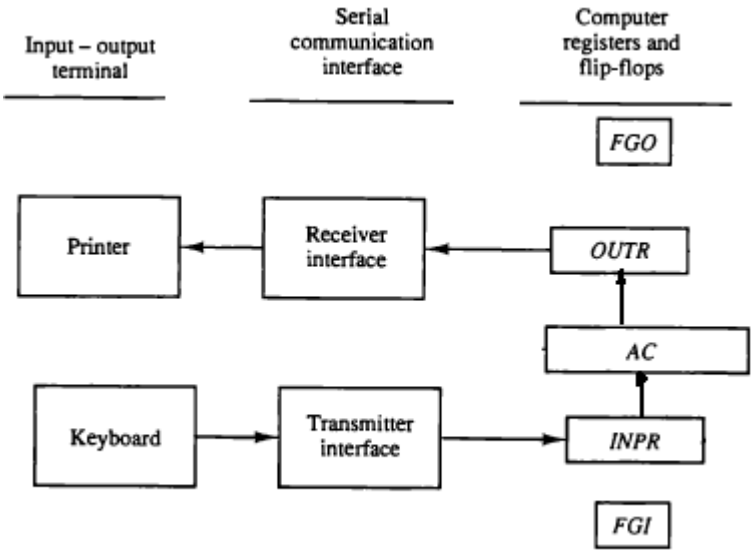



**Figure 18** Input-output configuration.

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(12). These instructions are executed with the clock transition associated with timing signal $T_3$.

**TABLE 12** Input-Output Instructions

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

| | | | |
|---|---|---|---|
| | $p$: | $SC \leftarrow 0$ | Clear $SC$ |
| INP | $pB_{11}$: | $AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$ | Input character |
| OUT | $pB_{10}$: | $OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$ | 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 |

Consider a computer that can go through an instruction cycle in 1μs. Assume that the input-output device can transfer information at a maximum rate of 10 characters per second. This is equivalent to one character every 100,000μs. Two instructions are executed when the computer checks the flag bit and decides not to transfer the information. This means that at the maximum rate, the computer will check the flag 50,000 times between each transfer. The computer is wasting time while checking the flag instead of doing some other useful processing task.

The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig(19). An interrupt flip-flop R is included in the computer.

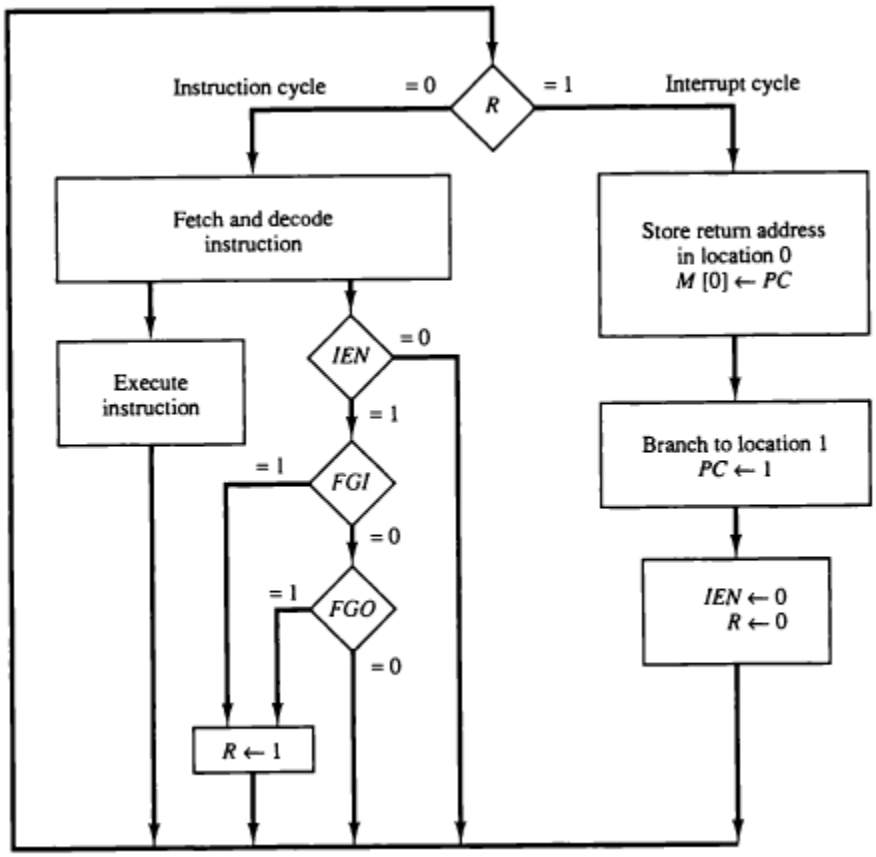



**Figure 19** Flowchart for interrupt cycle.

## Design of Basic Computer

The basic computer consists of the following hardware components:

1. A memory unit with 4096 words of 16 bits each
2. Nine registers: AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
3. Seven flip-flops: I, S, E, R, IEN, FGI, and FGO
4. Two decoders: a 3 x 8 operation decoder and a 4 x 16 timing decoder
5. A 16-bit common bus
6. Control logic gates
7. Adder and logic circuit connected to the input of AC

The outputs of the control logic circuit are:

1. Signals to control the inputs of the nine registers
2. Signals to control the read and write inputs of memory
3. Signals to set, clear, or complement the flip-flops
4. Signals for $S_2$, $S_1$, and $S_0$ to select a register for the bus
5. Signals to control the AC adder and logic circuit.