Chapter 2

Computer Organization and Design

2.1 Instruction Codes

The internal organization of a digital system is defined by the sequence of micro-operations it performs on data stored in its registers. A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. Instruction codes together with data are stored in memory. 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.

2.1.1 Operation Code (Op-Code)

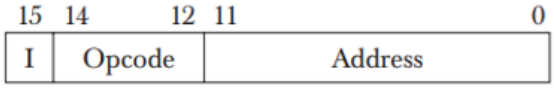
The operation part of an instruction code specifies the operation to be performed. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.

Consider a computer with 5 distinct operations, as given above. To identify 5 instructions the operation code consists of 3 bits. The bit configuration 000 may be assigned to the ADD operation. When this operation code is decoded in the control unit, the computer issues control signals to read operand add the operand to a processor register like an accumulator.

2.1.2 Address

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.

As an illustration, consider the instruction code format shown. 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.



(a) Instruction format

A direct address instruction is placed in address 22 in memory. The ‘I’ bit is 0, so the instruction is recognized as a direct address instruction. The op-code 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 2 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 indirect address instruction needs two references

to memory to fetch an operand. The first reference is needed to read the address of the operand; the second is for the operand itself.

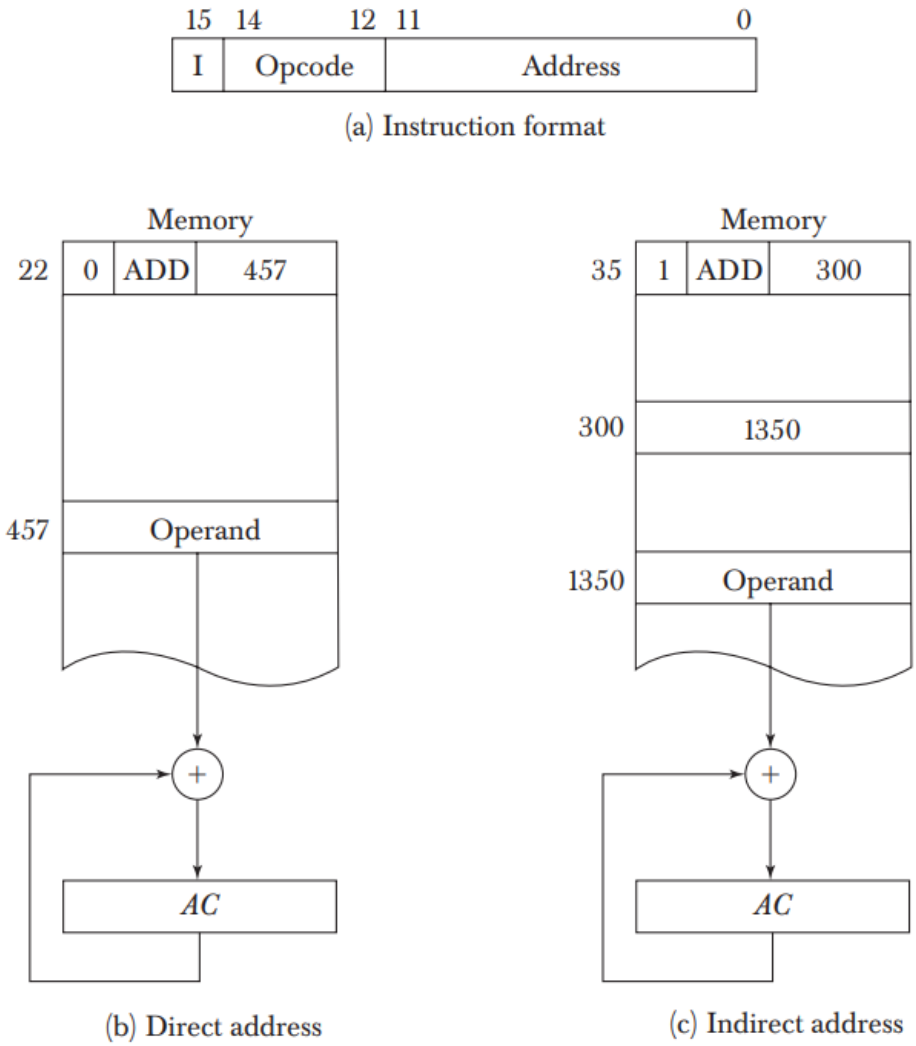


Figure 5-2 Demonstration of direct and indirect address.

2.2 Computer Registers

The computer needs processor registers for manipulating data and a register for holding a memory address. It is necessary to provide a register in the control unit for storing the instruction read from memory. It also needs a counter to calculate the address of the next instruction after execution of the current instruction is completed.

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 (MAR/AR) is the CPU register that either stores the memory address from which data

will be fetched to the DR, or the address to which data will be sent and stored. The program counter (PC) 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.

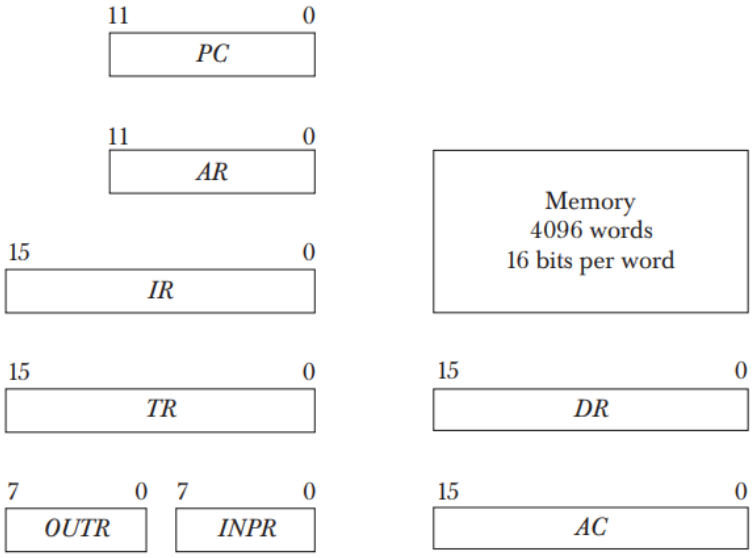
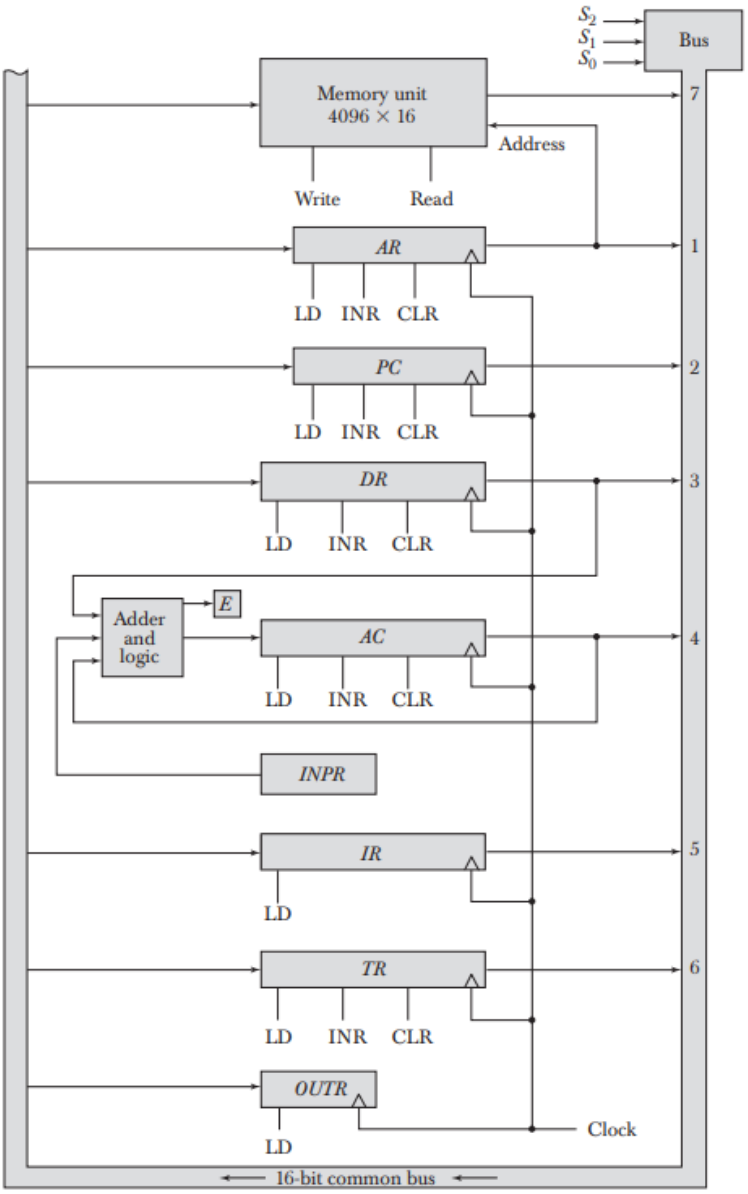


Figure 5-3 Basic computer registers and memory.

Common Bus System

The Common Bus System provides paths to transfer information from one register to another and between memory and registers. An 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 figure below.



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 S2, S1, and S0. 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 S2S1S0 = 011 since this is the binary value of decimal 3. 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 S2S1S0 = 111.

The memory address is connected to AR. Therefore, AR must always be used to specify a memory address. The content of any register can be specified for the memory data input during a write operation. Similarly, any register can receive the data from memory after a read operation except AC.

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**. They are used to implement register micro-operations such as complement AC and shift AC.

Another set of 16-bit inputs come from the data register **DR**. The inputs from DR and AC are used for arithmetic and logic micro-operations, such as add DR to AC or AND DR to AC. 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 Input Register.

At the clock transition the content of one register is transferred to the bus and then into the designated destination register. For example, the micro-operations

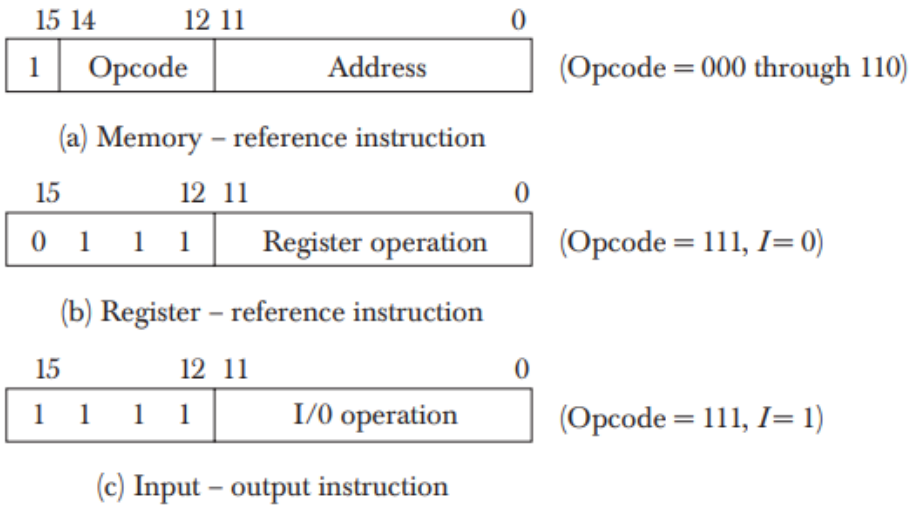
DR ← AC can be executed in the following manner.

First the contents of AC is transferred on the bus (with S2S1S0 = 100) and then the LD (load) input of DR is enabled.

2.3 Computer Instructions

The basic computer has three instruction code formats, as shown in figure below.

Figure 5-5 Basic computer instruction formats.



A **memory-reference instruction** uses 12 bits to specify an address and one bit to specify the addressing mode I. Only three bits of the instruction are used for the operation code. The address part is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address.

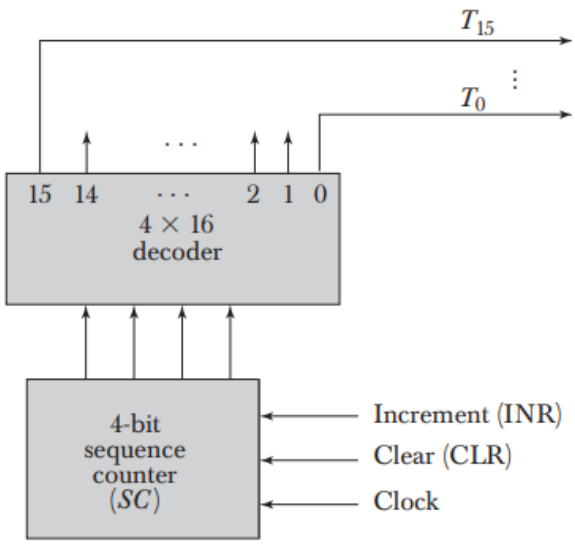
Symbol	Hexadecimal code $I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	lxxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero

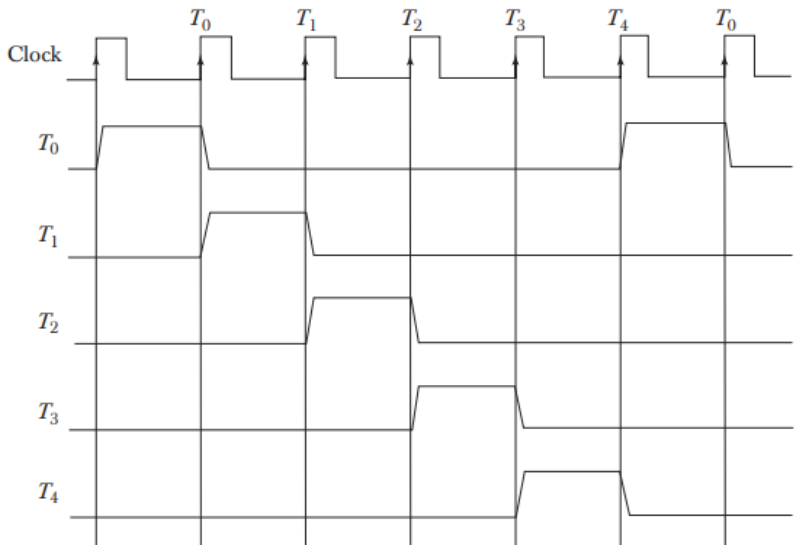
The register reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation to be executed.

Similarly, an **input–output instruction** does not need a reference to memory and is recognized by the operation code 111 with a 1 in the left-most bit of the instruction. The remaining 12 bits are used to specify the type of input–output operation.

2.4 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 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.





To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals.

For example, the register transfer statement

T0: AR ← PC specifies a transfer of the content of PC into AR if timing signal T0 is active. During this time the content of PC is placed onto the bus (with S2S1S0 = 010) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T1 active and T0 inactive.

2.5 Instruction Cycle

The program executed in the computer goes through a sequence of cycle called Instruction 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

2.5.1 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 T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The micro-operations for the fetch and decode phases can be specified by the following register transfer statements.

T0: AR ← PC

T1: IR ← M [AR], PC ← PC + 1

T2: D0 . . . D7 ← Decode IR (12–14), AR ← IR (0 - 11), I ← IR (15)

Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program. At time T2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR. Note that SC is incremented after each clock pulse to produce the sequence T0, T1, and T2.

To provide the data path for the transfer of PC to AR, **T0: AR ← PC** following task is performed:

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 equal to 010.

2. Transfer the content of the bus to AR by enabling the LD input of AR. The next clock transition initiates the transfer from PC to AR since T0 = 1.

In order to implement the second statement **T1: IR ← M [AR], PC ← PC + 1** it is necessary to provide the following connections in the bus system and enable T1.

1. Enable the read input of memory.

2. Place the content of memory onto the bus by making S2S1S0 = 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.

Determine the Type of Instruction

During time T3, the control unit determines the type of instruction that was just read from memory. The three possible instruction types available in the basic computer are specified in Fig below.

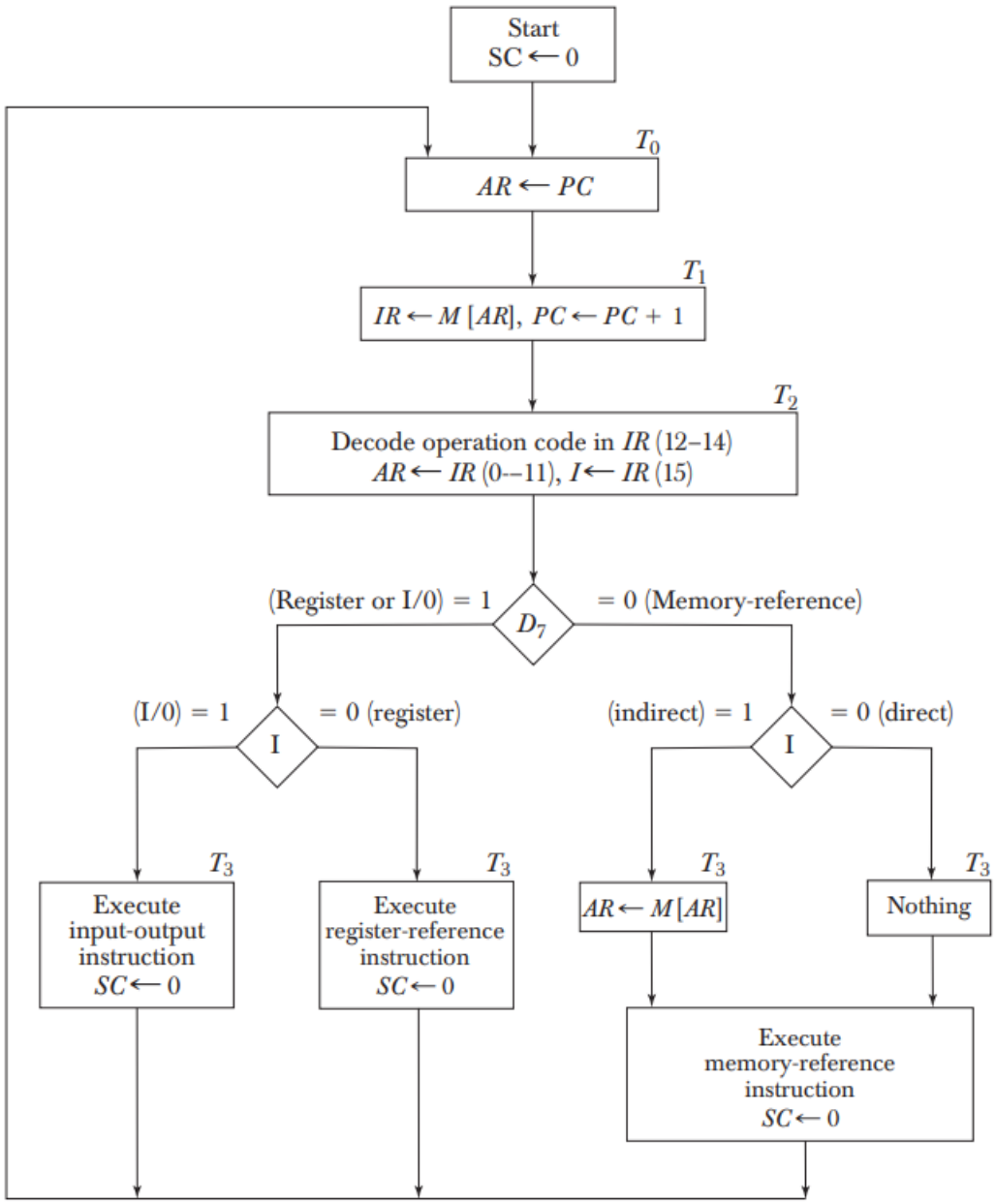


Figure 5-9 Flowchart for instruction cycle (initial configuration).

Decoder output D7 is equal to 1 if the operation code is equal to binary 111. From Fig. 5-5 we determine that if D7 = 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, we have a memory reference instruction with an indirect address.

The micro-operation for the indirect address condition can be symbolized by the register transfer statement **AR ← M [AR].**

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

D7’IT3 : AR ← M[AR]

D7’I’T3 : Nothing

D7I’T3: Execute a register-reference instruction

D7IT3 : Execute an input–output instruction

2.5 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). They were also transferred to AR during time T2.

The Register Reference Instructions are given below:

CLA	Clear AC
CLE	Clear E
CMA	Complement AC
CME	Complement E
CIR	Circulate right AC and E
CIL	Circulate left AC and E
INC	Increment AC
SPA	Skip next instruction if AC positive
SNA	Skip next instruction if AC negative
SZA	Skip next instruction if AC zero
SZE	Skip next instruction if E is 0
HLT	Halt computer

2.6 Memory Reference Instructions

Table below lists the seven memory-reference instructions. The decoded output Di 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 T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4. 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 micro-operations.

TABLE 5-4 Memory-Reference Instructions

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

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 micro-operations that execute this instruction are:

D0T4: DR ← M [AR]

D0T5: AC ← AC ∧ DR, SC ← 0

The control function for this instruction uses the operation decoder D0 since this output of the decoder is active when the instruction has an AND operation whose binary code value is 000. Two timing signals are needed to execute the instruction. The clock transition associated with timing signal T4 transfers the operand from memory into DR. The clock transition associated with the next timing signal T5 transfers to AC the result of the AND logic operation between the contents of DR and AC. The same clock transition clears SC to 0, transferring control to timing signal T0 to start a new instruction cycle.

Similarly, the micro-operations for the rest of the instructions are shown below.

2. **ADD to AC**

D1T4: DR ← M [AR]

D1T5: AC ← AC + DR, E ← Cout, SC ← 0

Note that, it is necessary to read the memory word into DR first and then transfer the content of DR into AC.

3. **LDA: Load to AC**

D2T4: DR ← M [AR] D2T5: AC ← DR, SC ← 0

4. **STA: Store AC**

D3T4: M [AR] ← AC, SC ← 0

5. **BUN: Branch Unconditionally**

D4T4: PC ← AR, SC ← 0

6. **ISZ: Increment and Skip if Zero**

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

This is done with the following sequence of micro-operations:

D6T4: DR ← M [AR]

D6T5: DR ← DR + 1

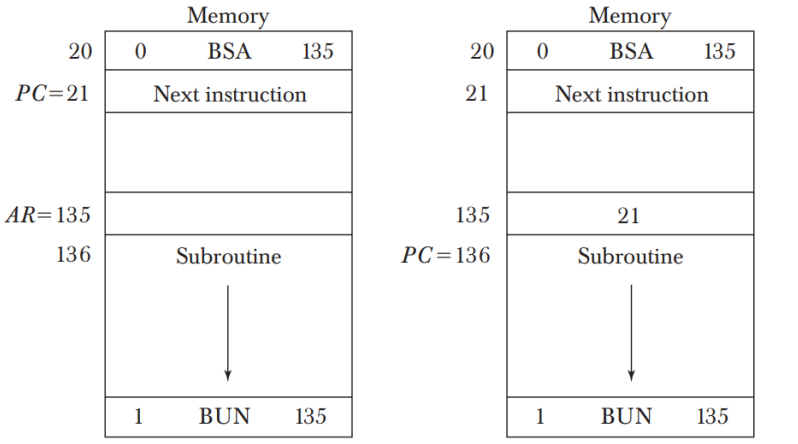
D6T6: M [AR] ← DR, if (DR = 0) then (PC ← PC = 1), SC ← 0

7. BSA: Branch and Save Return Address

M [AR] ← PC, PC ← AR + 1

Example of BSA Instruction Execution

Figure 5-10 Example of BSA instruction execution.



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after 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.

The BSA instruction performs the following numerical operation:

M [135] ← 21, PC ← 135 + 1 = 136

The return address 21 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 performs the function usually referred to as a subroutine call.

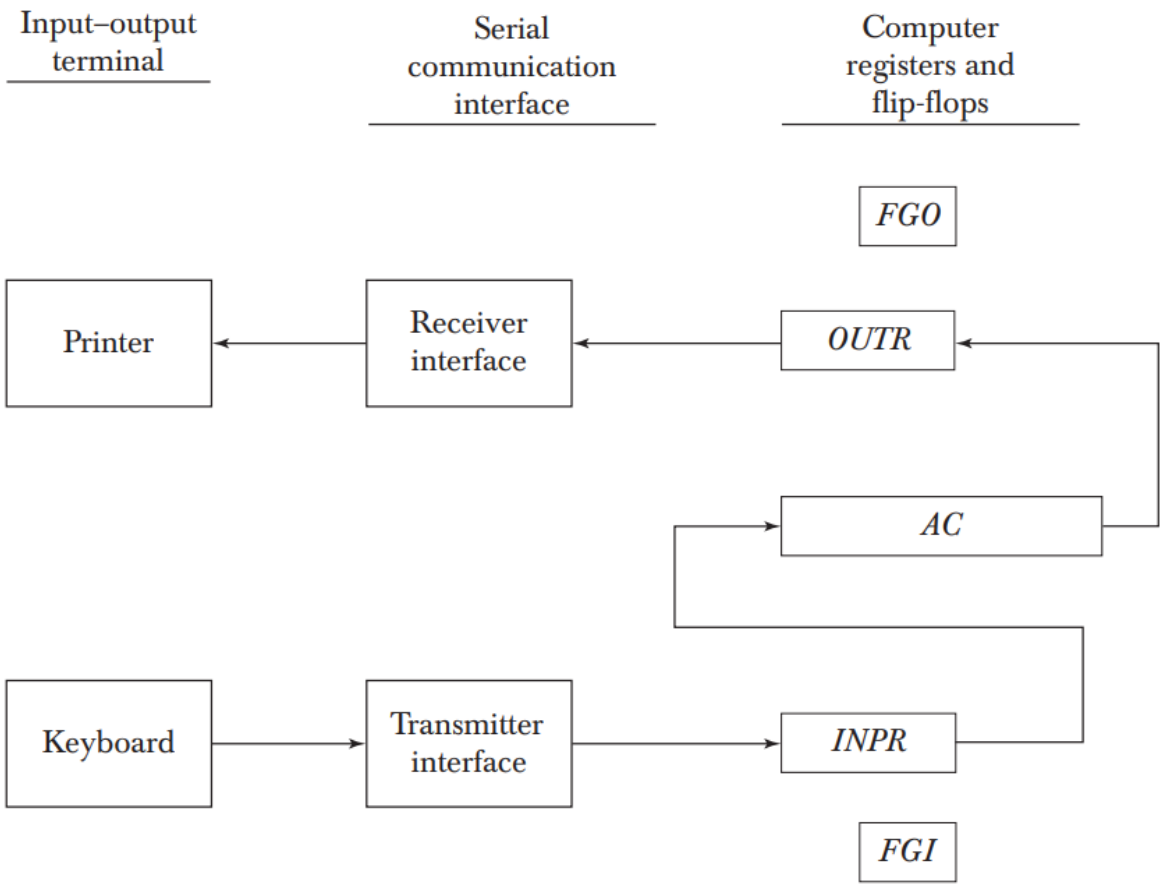
2.7 Input–Output and Interrupt

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. 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. below.

Figure 5-12 Input–output configuration.



The operation of the serial communication interface is explained in Sec. 11-3. The input register INPR consists of eight bits and holds an 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 process of information transfer is as follows.

1. Initially, the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.
2. As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.
3. Once the flag is cleared, new information can be shifted into INPR by striking another key.

The output register OUTR works similarly but the direction of information flow is reversed.

1. 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.
2. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
3. The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

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 micro-operations for the input–output instructions are listed in Table. These instructions are executed with the clock transition associated with timing signal T3.

TABLE 5-5 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]

	$SC \leftarrow 0$	Clear SC
INP	$AC(0–7) \leftarrow INPR, \quad FGI \leftarrow 0$	Input character
OUT	$OUTR \leftarrow AC\,(0–7), \quad FGO \leftarrow 0$	Output character
SKI	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	$IEN \leftarrow 1$	Interrupt enable on
IOF	$IEN \leftarrow 0$	Interrupt enable off

Program Interrupt

The process of communication just described is referred to as programmed control transfer. 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. The computer is wasting time while checking the flag instead of doing some other useful processing task. 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. The computer deviates momentarily from what it is doing to take care 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 1 (with the

ION instruction), the computer can be interrupted. These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility. The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. below.

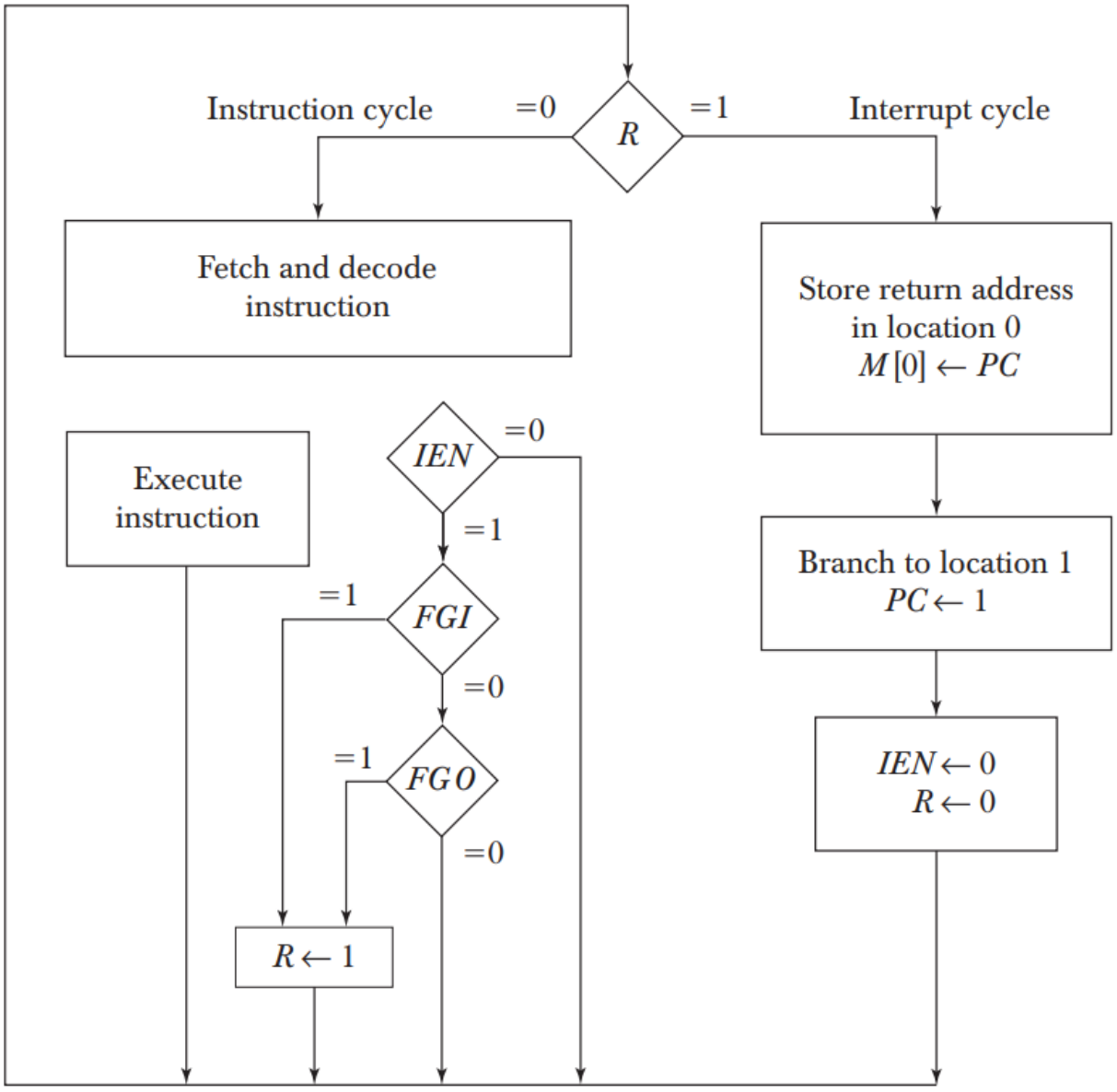


Figure 5-13 Flowchart for interrupt cycle.

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 IEN = 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. 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 where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.

Interrupt Cycle

The interrupt cycle is initiated after the last execute phase if the interrupt flipflop R is equal to 1. This flip-flop is set to 1 if IEN 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T0, T1, or T2 are active.

The interrupt cycle stores the return address (available in PC) into memory location 0, branches to memory location 1, and clears IEN, R, and SC to 0. This can be done with the following sequence of micro-operations:

RT0: AR ← 0, TR ← PC

RT1: M [AR] ← TR, PC ← 0

RT2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR. With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0. The third timing signal increments PC to 1, clears IEN and R, and control goes back to T0 by clearing SC to 0.