UNIT- I Syllabus:

Digital Computers: Introduction, block diagram of digital computer, definition of computer organization and architecture.

Basic Computer Organization and Design: Instruction Codes, Computer Register, Computer Instructions, Instruction Cycle, Memory – Reference Instructions. Input–Output and Interrupt, Complete Computer Description.

Digital Computers

Introduction

***Computer* –** The word "computer "comes from the word "compute "which means to calculate. So a computer is normally considered to be a calculating device that performs arithmetic operations at enormous speed. A computer is an electronic device which is used to perform operation on raw data as per instruction given by user. They are:

1) It accepts data or instructions through input,
2) It stores data,
3) It can process required data by the user,
4) It gives results as production, and
5) It controls all functions inside the computer.

Block diagram of digital computer and the functioning of its blocks

Computer is an electronic device which performs tasks given by user with extremely fast speed and accuracy. Like any other device or machine, a computer system has also a number of parts. A computer system can be blocked into mainly three parts:

1. Input Unit
2. Central Processing Unit
3. Output Unit

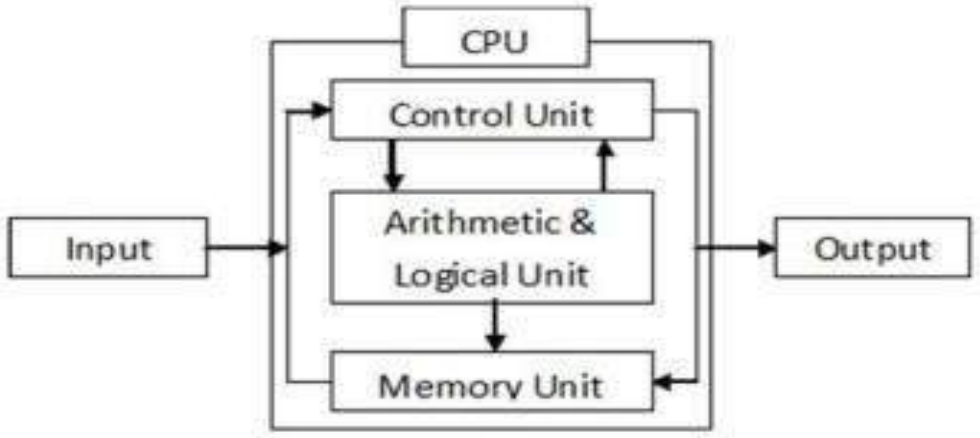


Fig. Block Diagram of Computer

1. Input unit – Input unit is a unit that accepts any input device. The input device is used to input data into the computer system. Function of input unit.

1. It converts inputted data into binary codes.
2. It sends data to main memory of computer .

2. **Central Processing Unit (CPU)** – CPU is called the brain of a computer. An electronic circuitry that carries out the instruction given by a computer program. CPU can be sub classified into three parts.

i. Control unit (CU).
ii. Arithmetic & Logic unit (ALU).
iii. Memory Unit (MU).

i. Control unit (CU)- the control unit manages the various components of the computer. It reads instructions from memory and interpretation and changes in a series of signals to activate other parts of the computer. It controls and co-ordinate is input output memory and all other units.

ii. Arithmetic & Logic unit (ALU) – The arithmetic logic unit (ALU), which performs simple arithmetic operation such as +,-, *, / and logical operation such as >, <, =<, <= etc.

iii. Memory Unit (MU)- Memory is used to store data and instructions before and after processing. Memory is also called Primary memory or internal memory. It is used to store data temporary or permanently.

Function of CPU:

1. It controls all the parts and software and data flow of computer.
2. It performs all operations.
3. It accepts data from input device.
4. It sends information to output device.
5. Executing programs stored in memory.
6. It stores data either temporarily or permanent basis.
7. It performs arithmetical and logical operations.

3. **Output Unit –**Output unit is a unit that constituents a number of output device. An output device is used to show the result of processing.

Function of Output unit:

1. It accepts data or information sends from main memory of computer
2. It converts binary coded information into HLL or inputted languages.

Definition of Computer organization & Architecture

Computer Organization: Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the

organizational structure to verify that the computer parts operate as intended.

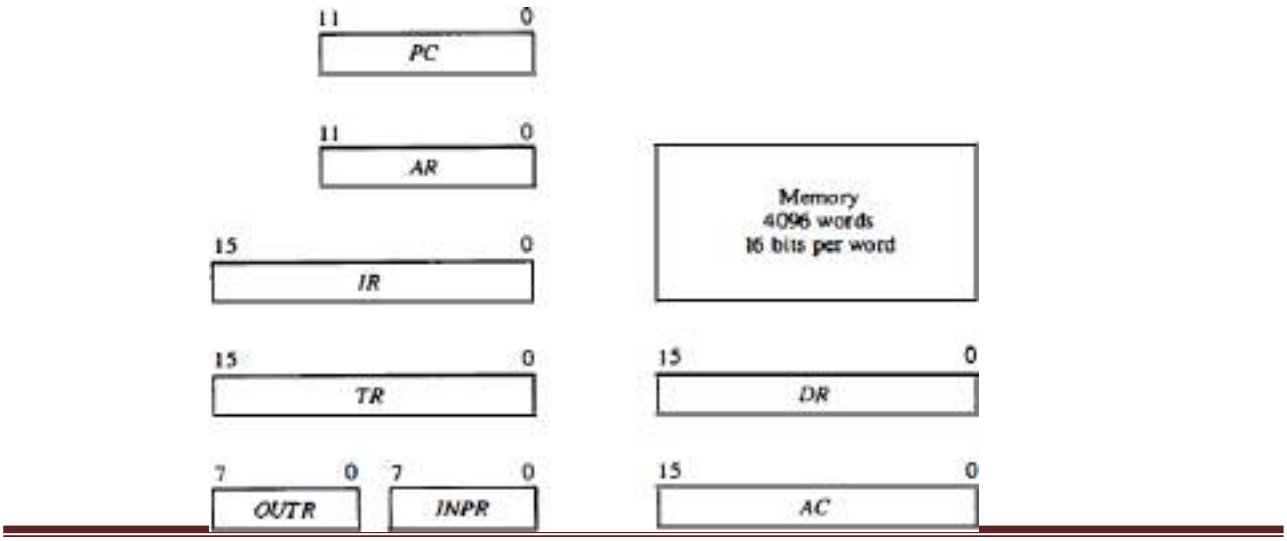
Computer Design: Computer design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system. Computer design is concerned with the determination of what hardware should be used and how the parts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

Computer Architecture: Computer architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

Basic Computer Organization and Design

Computer Registers

The registers available in the computer are.



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

Table (f): List of Registers for the Basic computer.

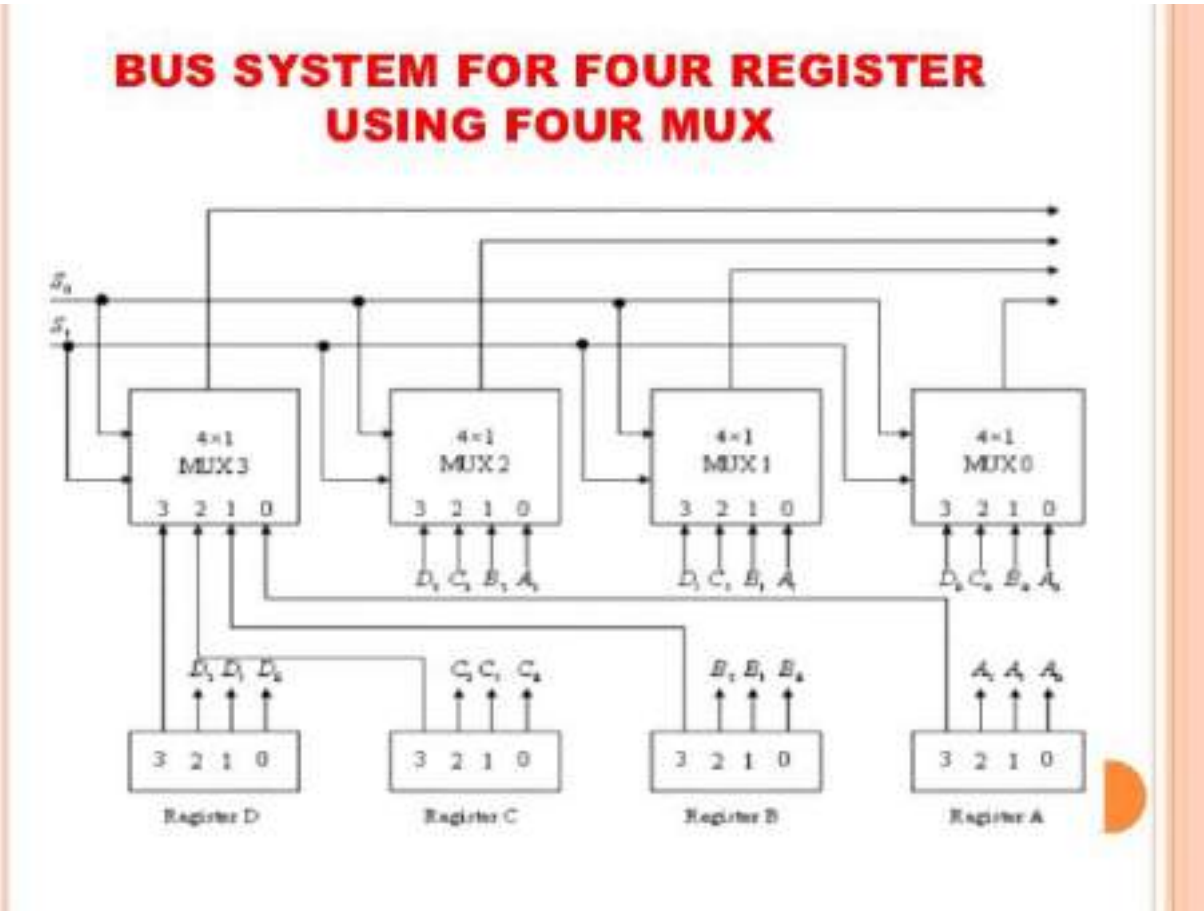
Bus and Memory Transfers

Common bus system for four register

A more efficient scheme for transferring information between registers in a multiple- register configuration is a common bus system.

Control signals determine which register is selected by the bus during each particular register transfer.

One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in figure below. Each register has four bits, numbered 0 through 3.The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S1 and S0.The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.



The two selection lines S1 and S0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four- line common bus.

S1	S2	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Function Table for Bus

When S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.Similarly, register B is selected if S1S0 = 01, and so on.

Table shows the register that is selected by the bus for each of the four possible binary values of the selection lines.

Memory Transfer

Read Operation: The transfer of information from a memory word to the outside environment is called a read operation.

Write Operation: The transfer of new information to be stored into the memory is called a write operation.

A memory word will be symbolized by the letter M.

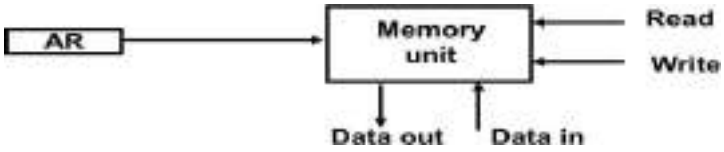
It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the address in square brackets following the letter M. Consider a memory unit that receives the address from a register, called the address register, symbolized by AR. The data are transferred to another register, called the data register, symbolized by DR. The read operation can be stated as follows:

Read: DR ¬M [AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR. The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.Write operation can be stated symbolically as follows:

Write: M[AR] ¬R1

This causes a transfer of information from R1 into memory word M selected by address AR.



Common Bus System

The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers. The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.A more efficient scheme for transferring information in a system with many registers is to use a common bus. The connection of the registers and memory of the basic computer to a common bus system is shown in Fig. 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 S_2, S_1, and S_0.The number along each output shows the decimal equivalent of the required binary selection. For example, the number along the output of DR is 3.The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0 = 011$ 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 $S_2S_1S_0 = 111$.Four registers, DR, AC, IR, and TR, have 16 bits each. 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. INPR is connected to provide information to the bus but OUTR can only receive information from the bus. This is because INPR receives a character from an input device which is then transferred to AC. OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers. The 16 lines of the common bus receive information from six registers and the memory unit. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: LD (load), INR (increment), and CLR (clear).This type of register is equivalent to a binary counter with parallel load and synchronous clear. The increment operation is achieved by enabling the count input of the counter. Two

registers have only a LD input. The input data and output data of the memory are connected to the common bus, but the memory address is connected to AR. Therefore, AR must always be used to specify a memory address.

By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise. 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.Note that the content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle. The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

For example, the two microoperations

DR ← AC and AC ← DR

can be executed at the same time. This can be done by placing the content of AC on the bus (with $S_2S_1S_0 = 100$), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of AC, all during the same clock cycle. **The two transfers** occur upon the arrival of the clock pulse transition at the end of the clock cycle.

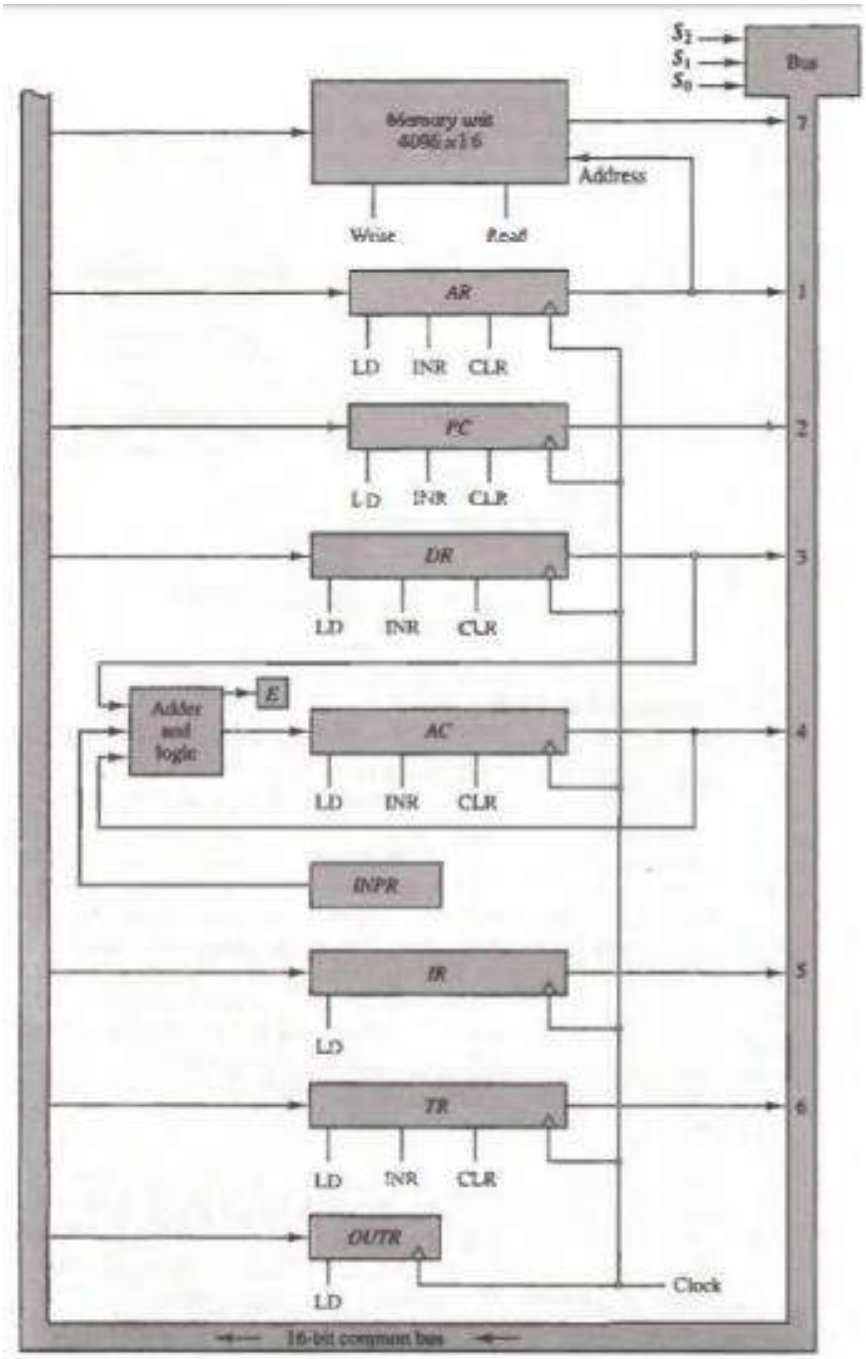


Figure 5-4 Basic computer registers connected to a common bus.

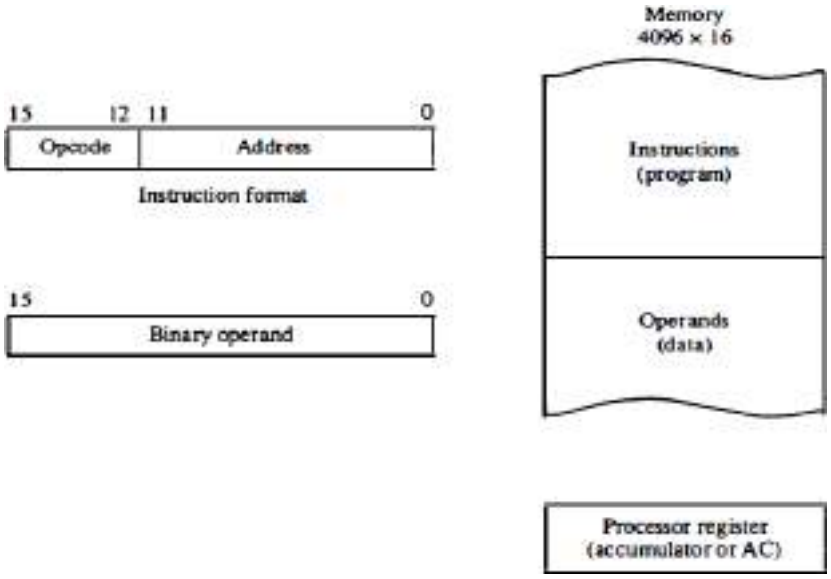
Instruction Codes

A **program** is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur. 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. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro operations.

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 **operation part** of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory.

.

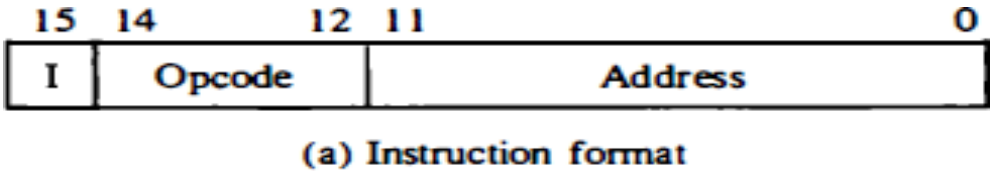


Stored program organization

Indirect Address

When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand. When the second part specifies the address of an operand, the instruction is said to have a direct address. When 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, the instruction is said to an indirect address. One bit of the instruction code can be used to distinguish between a direct and an indirect address.

An **effective address** is the address of the operand.



(a) Instruction format

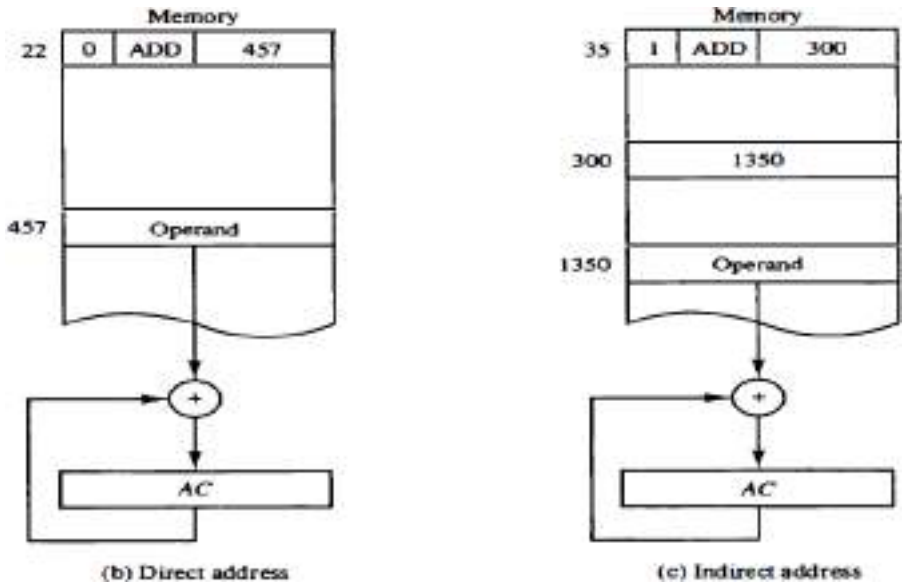


Figure (l): Demonstration of direct and indirect address.

Computer Instructions

The basic computer has **three types of instruction code formats**,

1. Memory-reference instruction.
2. Register-reference instruction.
3. An input-output instruction.

Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

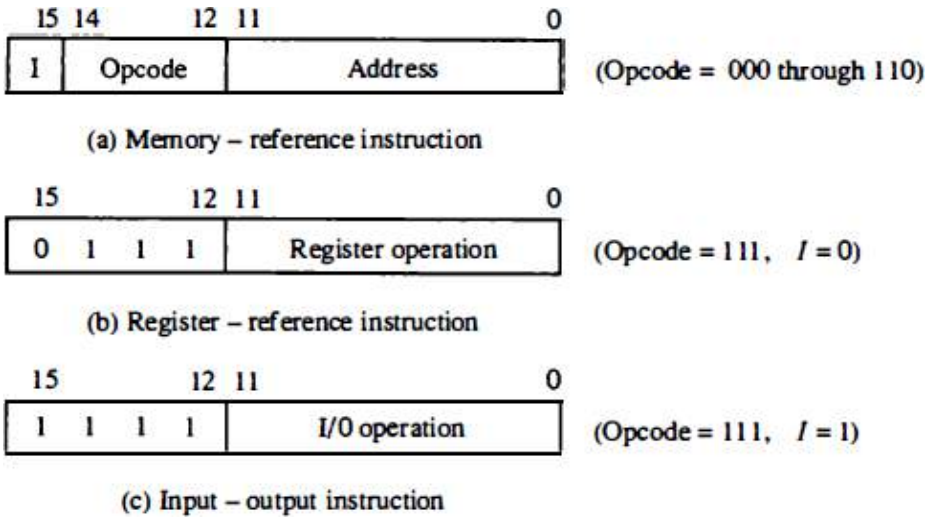


Figure (n): Basic computer instruction formats

If the three opcode bits in positions 12 to 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 I. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I = 0 for direct address and I = 1 for indirect address.

If the 3-bit opcode = 111, control then inspects the bit in position 15. If this bit = 0, the instruction is a **register-reference type**. These instructions use 16 bits to specify an operation.

If the bit I = 1, the instruction is **an input-output type**. These instructions also use all 16 bits to specify an operation.

The instructions for the computer are listed in Table (g, h, i).

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

Table (g): Memory-reference instructions

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

Table (h): Register-reference instructions

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

Table (i): Input-output instructions

Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:

1. **Fetch** an instruction from memory.
2. **Decode** the instruction.
3. **Read** the effective address from memory if the instruction has an indirect address.
4. **Execute** the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode:

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

T0:AR←PC
T1: IR←M[AR], PC ← PC + 1

Determine the Type of Instruction

The timing signal that is active after the decoding is T_3. During time T_3 the control unit determines the type of instruction that was just read from memory.

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111.

- If $D_7 = 1$, the instruction must be a register-reference or input-0utput type.
- If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.

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

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when $D_7'T_3 = 1$, so that the execution of the memory-reference instruction can be continued with timing variable T4. A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.

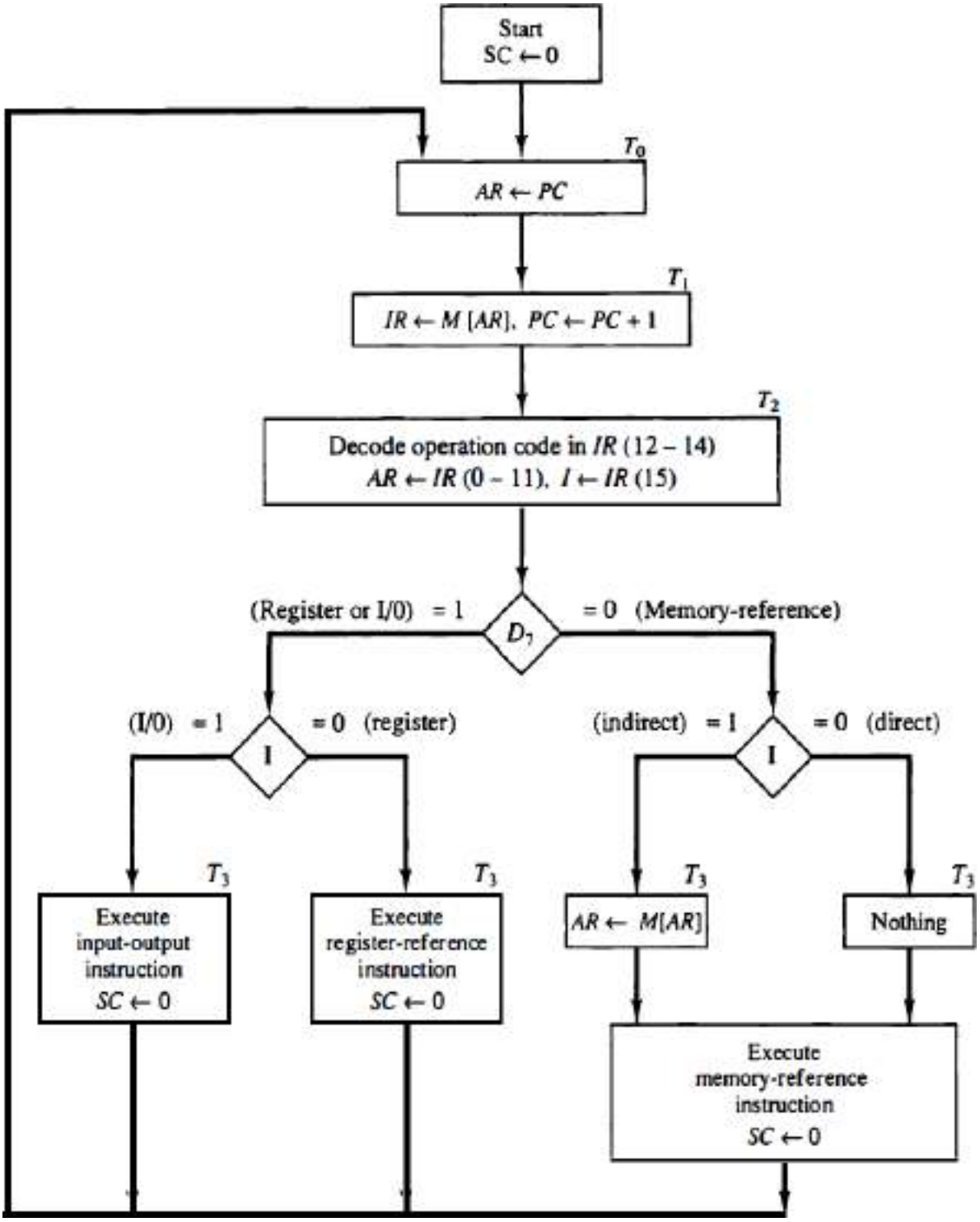


Figure : Flowchart for instruction cycle (initial configuration).

Register-Reference Instructions

Register-reference instructions are recognized by the control when $D_7 = 1$ 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 T_2.

- Each control function needs the Boolean relation D_7 I' T_3, which we designate for convenience by the symbol r . The control function is distinguished by one of the bits in IR(0-11). By assigning the symbol B_i to bit i of IR, all control functions can be simply denoted by rB_i.
- **For example**, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000.
  i. The first bit is a zero and is equivalent to I'.
  ii. The next three bits constitute the operation code and are recognized from decoder output D7.
  iii. Bit 11 in IR is 1 and is recognized from B_{11}.

The control function that initiates the rnicrooperation for this instruction is D_7 I' T_3 B_{11} = r B_{11}

$D_7IT_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

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

Table : Execution of Register-Reference Instructions

Memory-Reference Instructions

- ✓ The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.
- ✓ The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1. The execution of the memory-reference instructions starts with timing signal T4.

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 $PC \leftarrow PC + 1$

Memory-Reference Instructions

AND : 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 rnicrooperations that execute this instruction are:

D_0T_4: DR ← M[AR]
D_0T_5: AC ← AC Λ DR, SC ← 0

ADD : ADD to AC

This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C,., is transferred to the E (extended accumulator) flip-flop. The rnicrooperations needed to execute this instruction are:

D_1T_4 : DR ← M[AR]
D_1T_5: AC ← AC + DR, E ← Cout, SC ← 0

LDA: Load to AC

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

D_2T_4: DR ← M[AR]
D_2T_5: AC ← DR, SC ← 0

STA: Store AC

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

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

BUN: Branch Unconditionally

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

BSA: Branch and Save Return Address

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

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

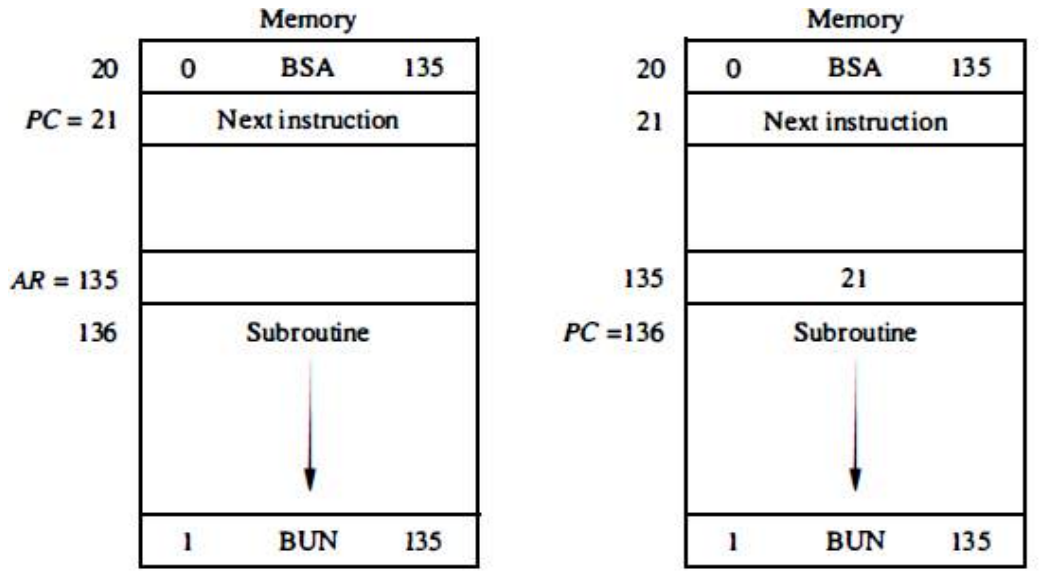
BSA: Branch and Save Return Address

EX:

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

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

The result of this operation is shown in part (b) of the figure. The return 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.



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

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

$$D_5T_4: \quad M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$$
$$D_5T_5: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

Timing signal T4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR . The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T5 to transfer the content of AR to PC.

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.

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

A flowchart showing all micro operations for the execution of the seven memory- reference instructions is shown in Figure (q). The control functions are indicated on top of each box. The microoperations that are performed during time T4, T5, or T6, depend on the operation code value. The sequence counter SC is cleared to 0 with the last timing signal in each case. This causes a transfer of control to timing signal T0 to start the next instruction cycle.

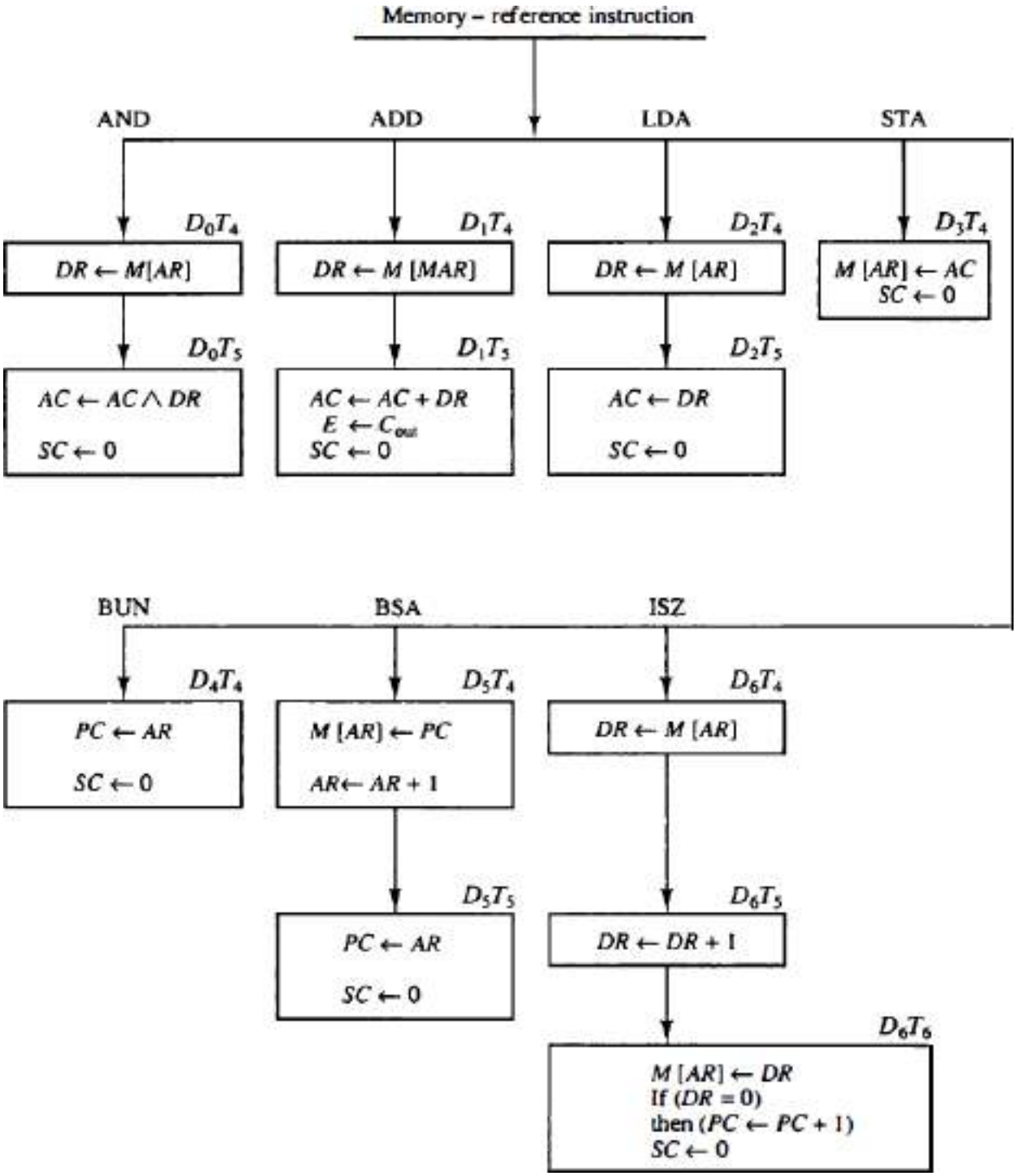


Figure : Flowchart for Memory-reference instructions

Input-Output and Interrupt

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. Commercial computers include many types of input and output devices.

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.

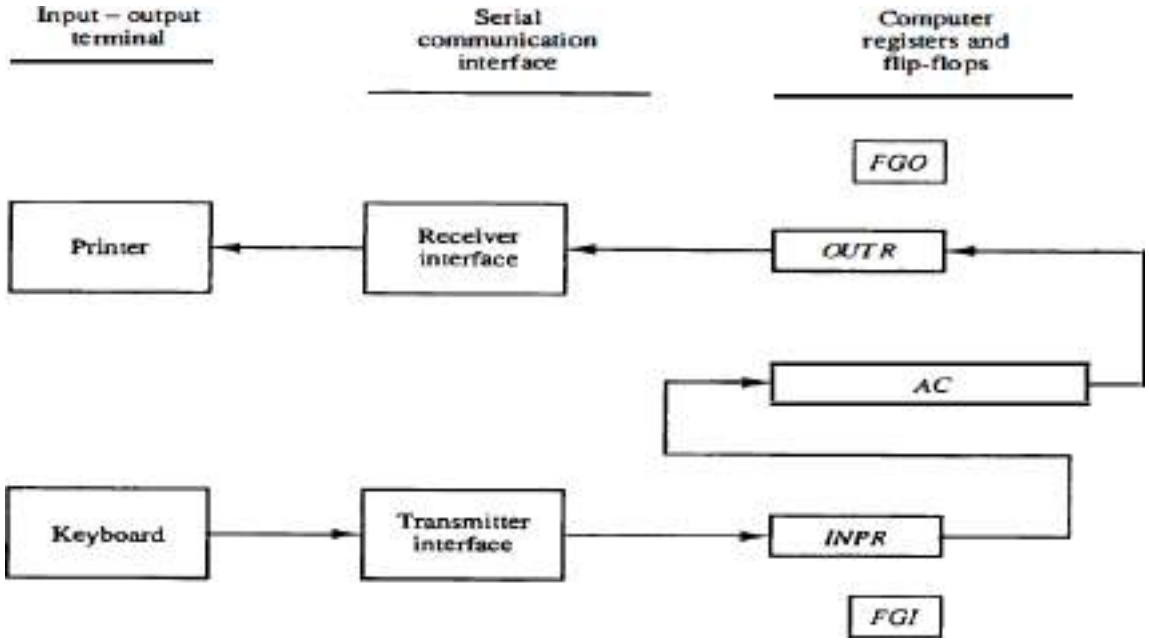


Figure : Input-Output configuration

The process of information transfer is as follows: 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. 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. Once the flag is cleared, new information can be shifted into INPR by striking another key.

Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1. 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.

$D_7IT_3 = p$ (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

Table: Input-Output instructions

Program Interrupt

The process of communication discussed so far 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 the input-output device makes this type of transfer inefficient.

To see why this is inefficient, 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 100,000 μs. 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.

An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.

While the computer is running a program, it does not check the flags. However, when a flag is set , the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that a flag has been set. 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 lEN can be set and cleared with two instructions (IOF and ION instructions).

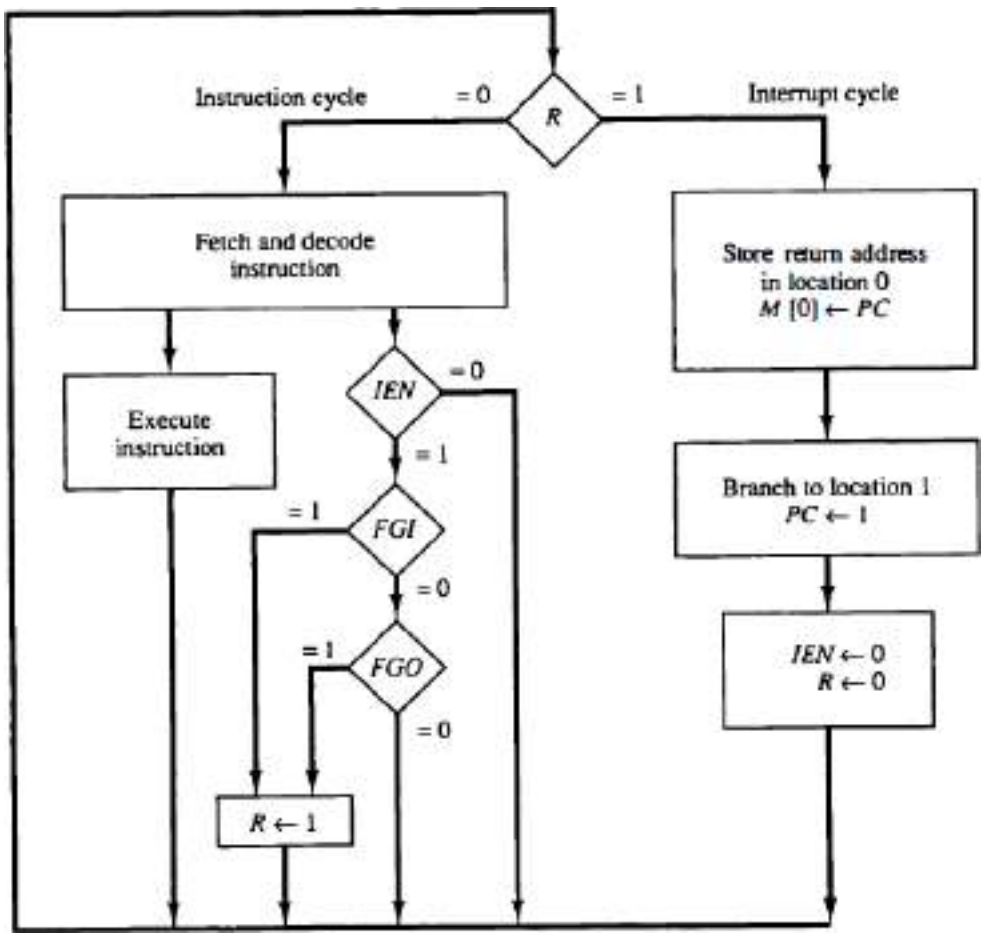


Figure: Flowchart for interrupt cycle

The way that the interrupt is handled by the computer can be explained by means of the flowchart of Figure.

- 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 lEN 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 lEN 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 lEN = 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(BSA)** operation.

EX:

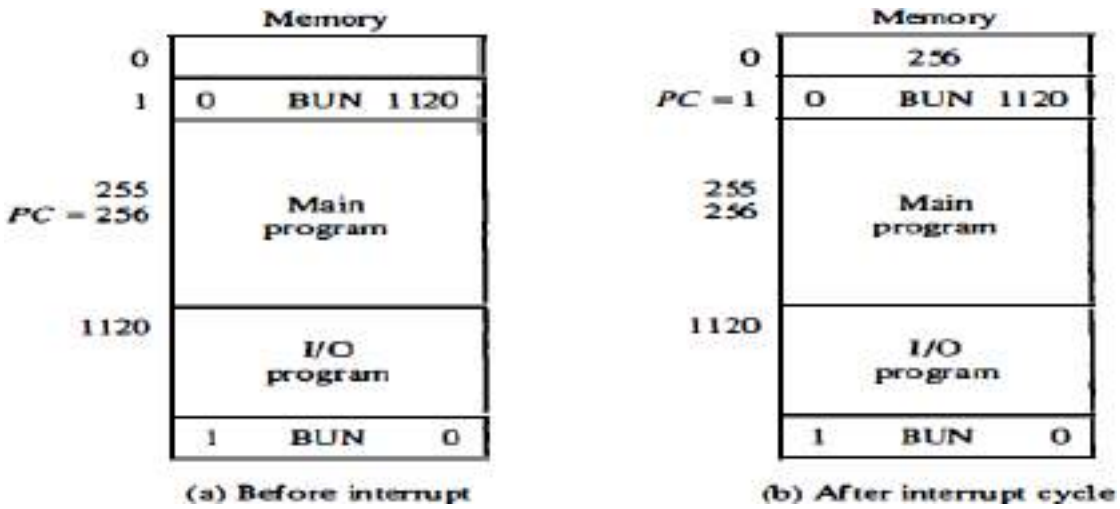


Figure : Demonstration of Interrupt Cycle

An example that shows what happens during the interrupt cycle is shown in Figure (t). Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Figure (a).

When control reaches timing signal T0 and finds that R = 1, it proceeds with the interrupt cycle. The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120. This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the program returns to the location where it was interrupted. This is shown in Figure (b).

Interrupt Cycle

The interrupt cycle is initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if lEN = 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 condition for setting flip-flop R to 1 can be expressed with the following register transfer statement:

$$T_0'T_1'T_2'(IEN)(FGI + FGO): \quad R \leftarrow 1$$

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 lEN and R, and control goes back to T0 by clearing SC to 0. The beginning of the next instruction cycle has the condition R' T0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

$$\begin{aligned} RT_0&: \quad AR \leftarrow 0, \quad TR \leftarrow PC \\ RT_1&: \quad M[AR] \leftarrow TR, \quad PC \leftarrow 0 \\ RT_2&: \quad PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0 \end{aligned}$$

Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Fetch and Decode

1. **Initially, the program counter PC** is loaded with the address of the first instruction in the program.
2. **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.
3. **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)

4. **Since only AR** is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.

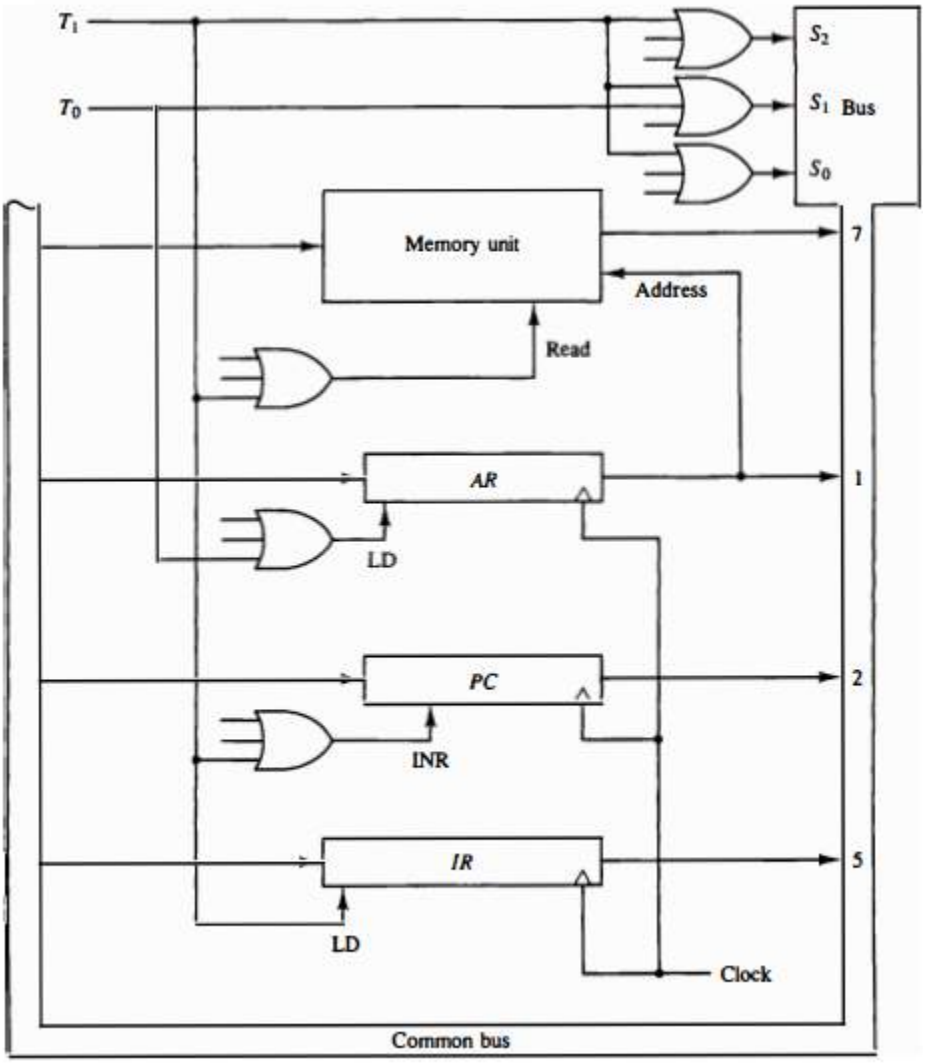


Figure Register transfers for the fetch phase.

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.**Figure above** shows how the first two register transfer statements are implemented in the bus system. **To provide** the data path for the transfer of PC to AR we must apply timing signal T0 to achieve the following connection:

Place the content of PC onto the bus by making the bus selection inputs

$S_2S_1S_0$ equal to 010.

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 use timing signal T1 to provide the following connections in the bus system.

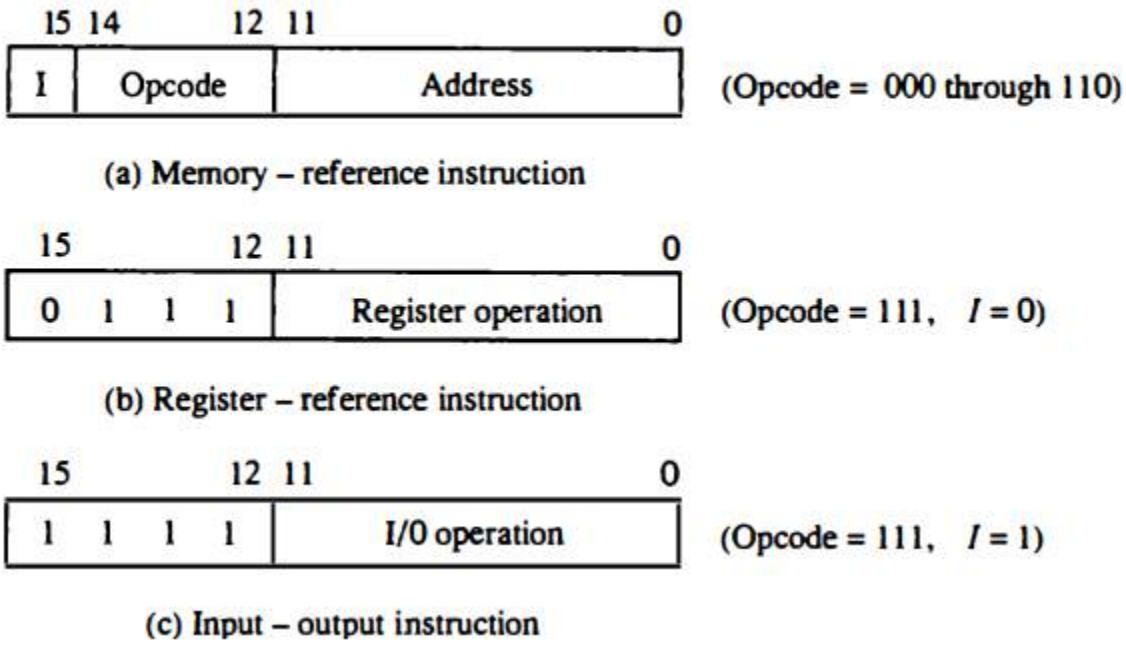
1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. 4. Increment PC by enabling the INR input of PC

The next clock transition initiates the read and increment operations since T1 = 1.Figure above duplicates a portion of the bus system and shows how T0 and T1 are connected to the control inputs of the registers, the memory, and the bus selection inputs. Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the Type of Instruction

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory.The flowchart of Fig. below presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding. The three possible instruction types available in the basic computer are specified in Fig. on basic computer formats.

Figure Basic computer instruction formats.



Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. From Fig. on basic computer formats we determine that if $D_7 = 1$, the instruction must be a register-reference or input-output type. If $D_7 = 0$, 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. It is then necessary to read the effective address from memory. The micro operation for the indirect address condition can be symbolized by the register transfer statement: AR ← M[AR].Initially, AR holds the address part of the instruction. This address is used during the memory read operation. The word at the address given by AR is read from memory and placed on the common bus. The LD input of AR is then enabled to receive the indirect address that resided in the 12 least significant bits of the memory word.

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

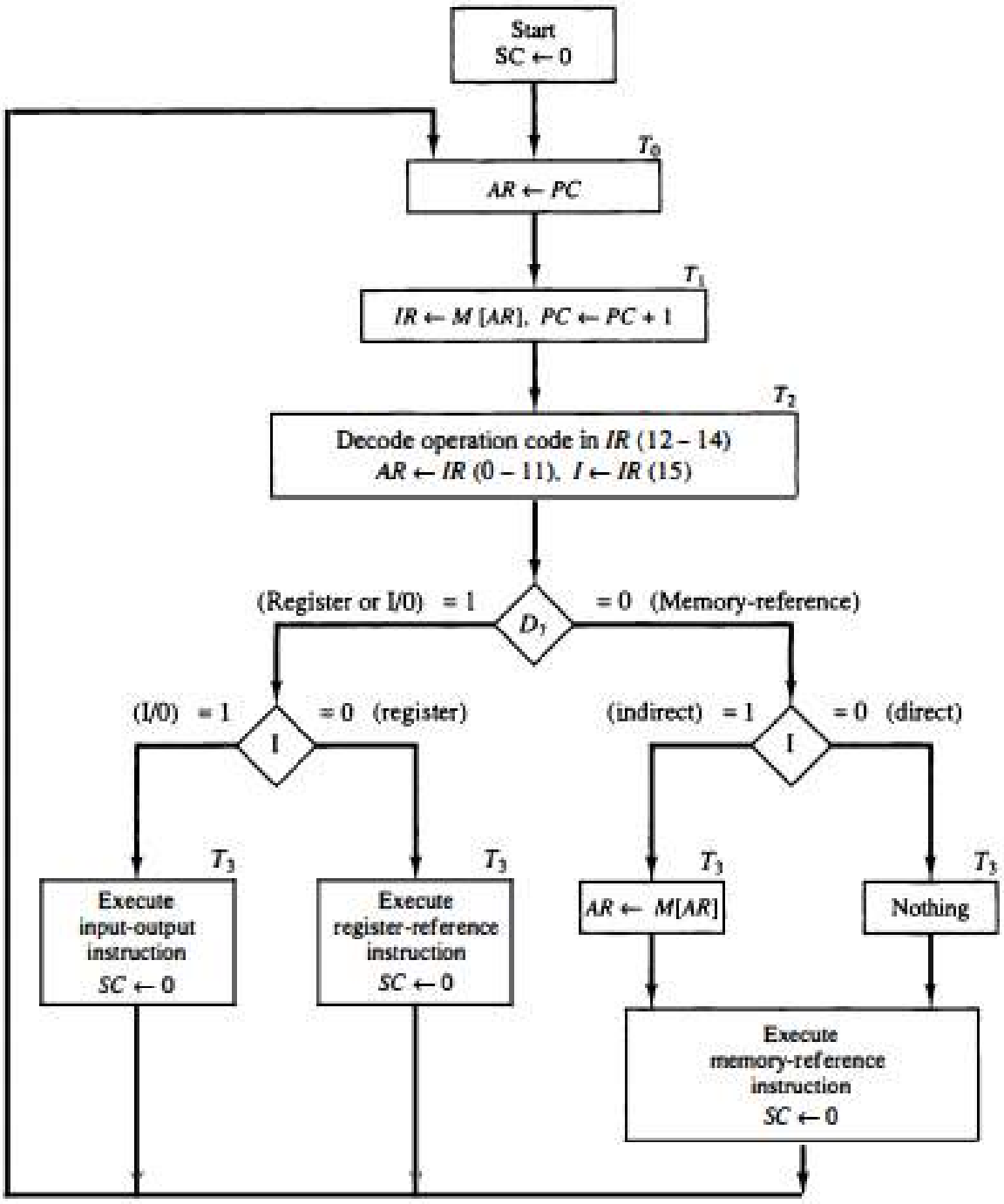


Figure Flowchart for instruction cycle (initial configuration).

$D'_7 IT_3$: AR ← M[AR]
$D'_7 I'T_3$: Nothing
$D_7 I'T_3$: Execute a register-reference instruction
D_7IT_3: Execute an input-output instruction

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when D'_7T_3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T_4 .A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T_0 = 1.Note that the sequence counter SC is either incremented or cleared to 0 with every positive clock

transition. We will adopt the convention that if SC is incremented, we will not write the statement SC ← SC + 1, but it will be implied that the control goes to the next timing signal in sequence. When SC is to be cleared, we will include the statement SC ← 0. The register transfers needed for the execution of the register-reference instructions are presented in this section.

Timing and Control

The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro operations for the accumulator.

There are two major types of control organization: hardwired control and micro programmed control.

In the hardwired organization, 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.

In the micro programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro operations.

A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the micro programmed control, any required changes or modifications can be done by updating the micro program in control memory.

The block diagram of the control unit is shown in Fig. below. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR).The position of this register in the common bus system is indicated in Fig. previously seen. The instruction register is shown again in Fig. below, where it is divided into three parts: the I bit, the operation code, and bits 0 through 11.The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15.The sequence counter SC can be incremented or cleared synchronously Most of the time; the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T0.As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active.

This is expressed symbolically by the statement D3T4: SC ← 0. The timing diagram of Fig. below shows the time relationship of the control signals.

The sequence counter SC responds to the positive transition of the clock. Initially, the CLR input of SC is active. The first positive transition of the clock clears SC to 0, which in turn activates the timing signal T0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labeled T0 in the

diagram will trigger only those registers whose control inputs are connected to timing signal T0.SC is incremented with every positive clock unless its CLR input is active. This produces the sequence of timing signals T0,T1,T2,T3,T4 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 T15 and back to T_0.

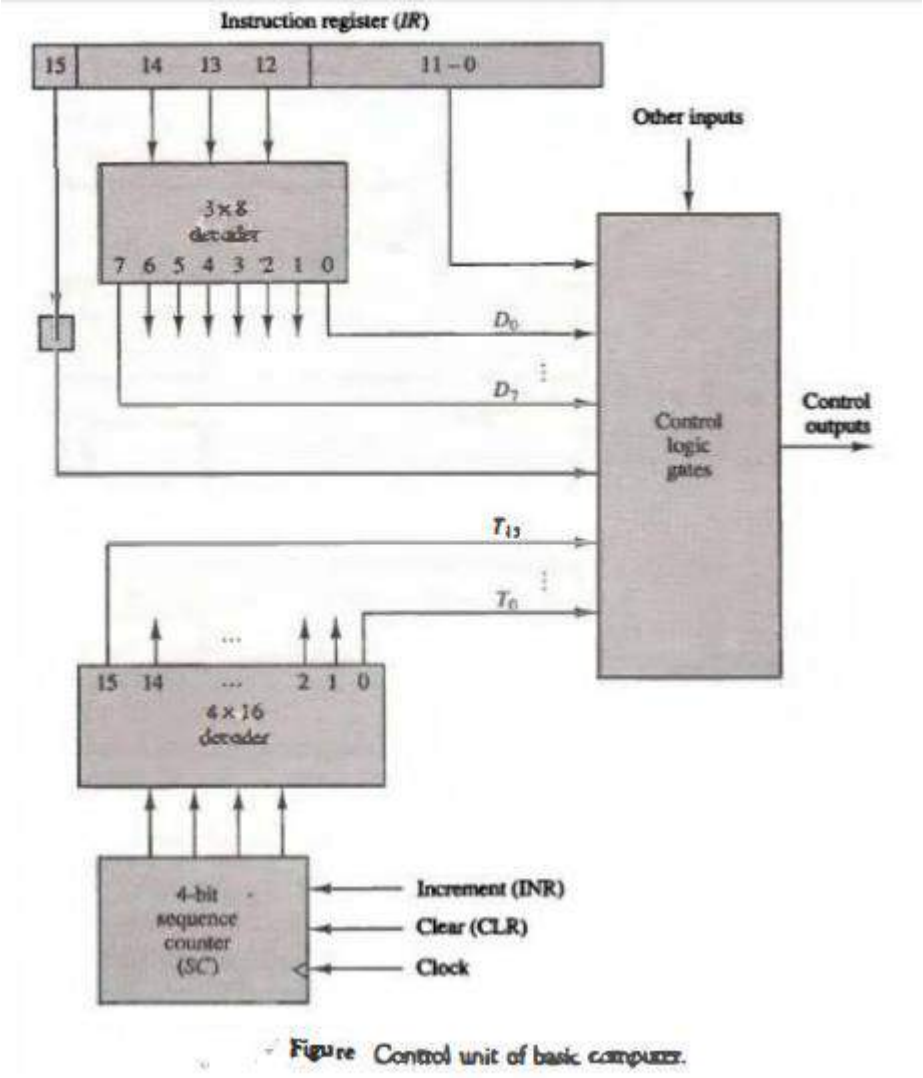


Figure Control unit of basic computer.

Clock clears SC to 0, which in turn activates the timing signal T0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labeled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.SC is incremented with every positive clock unless its CLR input is active. This produces the sequence of timing signals T0, T1, T2, T3, and T4 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 T15 and back to T_0. The last three waveforms in Fig. above show how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active.

This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared. A memory read or writes cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or writes cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle.

In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer. 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 transfers statement T0: AR ← PC specifies a transfer of the content of PC into AR if timing signal T0 is active. T0 is active during an entire clock cycle interval. During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) 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.

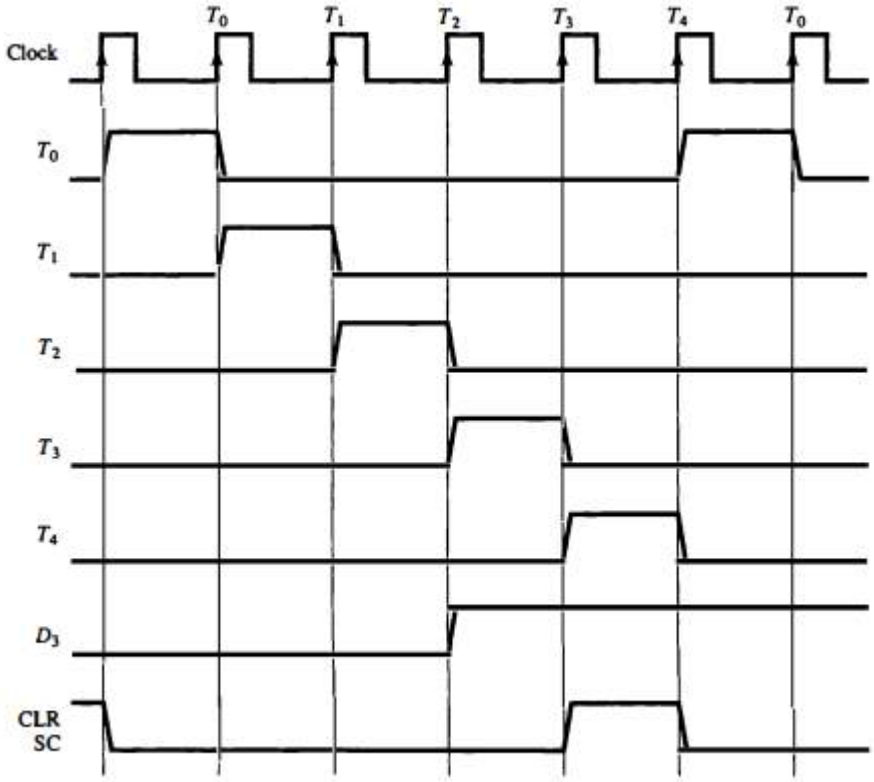


Figure Example of control timing signals.