Chapter 3
Basic Computer Organization and Design

3.1 Introduction: Description of Basic Computer

We introduce here a basic computer whose operation can be specified by the resister transfer statements. Internal organization of the computer is defined by the sequence of microoperations it performs on data stored in its resisters. Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc). Modern processor is a very complex device. It contains:

- Many registers
- Multiple arithmetic units, for both integer and floating point calculations
- The ability to pipeline several consecutive instructions for execution speedup.

However, to understand how processors work, we will start with a simplified processor model. M. Morris Mano introduces a simple processor model; he calls it a "Basic Computer". The Basic Computer has two components, a processor and memory.

- The memory has 4096 words in it
  - $4096 = 2^{12}$, so it takes 12 bits to select an address in memory
- Each word is 16 bits long

Stored Program Organization

The program (instruction) as well as data (operand) is stored in the same memory. If the instruction needs data, the data is found in the same memory and accessed. This feature is called stored program organization.

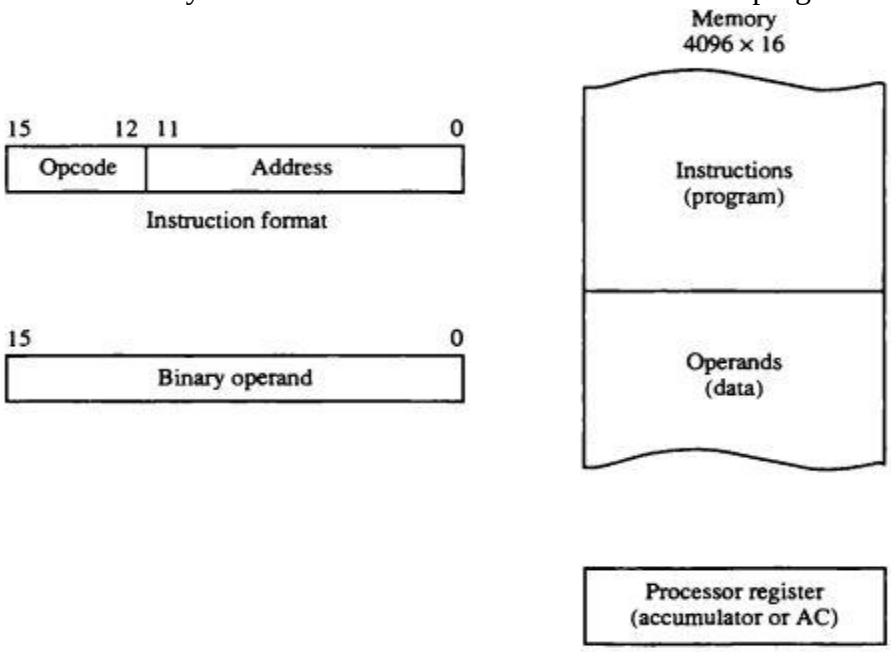


Fig: Stored Program Organization

Instruction Format

A computer instruction is often divided into two parts

- An *op-code* (Operation Code) that specifies the operation for that instruction
- An *address* that specifies the registers and/or locations in memory to use for that operation

In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify the memory address that is used by this instruction. In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing). Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's op-code.

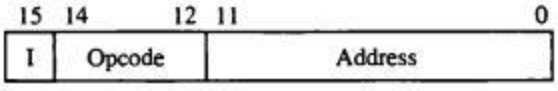


Fig: Instruction Format

Addressing Modes

The address field of an instruction can represent either

- Direct address: the address operand field is effective address (the address of the operand).
- Indirect address: the address operand field contains the memory address of effective address.

Basic Computer Registers

Computer instructions are normally stored in the consecutive memory locations and are executed sequentially one at a time. Thus computer needs processor registers for manipulating data and holding memory address which are shown in the following table:

Symbol	Size	Register Name	Description
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

Since the memory in the Basic Computer only has 4096 ($=2^{12}$) locations, PC and AR only needs 12 bits.
Since the word size of Basic Computer only has 16 bit, the DR, AC, IR and TR needs 16 bits. The Basic Computer uses a very simple model of input/output (I/O) operations.

- Input devices are considered to send 8 bits of character data to the processor
- The processor can send 8 bits of character data to output devices

The Input Register (INPR) holds an 8-bit character gotten from an input device and the Output Register (OUTR) holds an 8-bit character to be sent to an output device.

3.2 Common Bus System

→ The basic computer has eight registers, a memory unit, and a control unit.

→ These registers, memory and control unit are connected using a path (bus) so that information can be transferred to each other.

→ If separate buses are used for connecting each registers, it will cost high.

→ The cost and use of extra buses can be reduced using a special scheme in which many registers use a common bus, called *common bus system*.

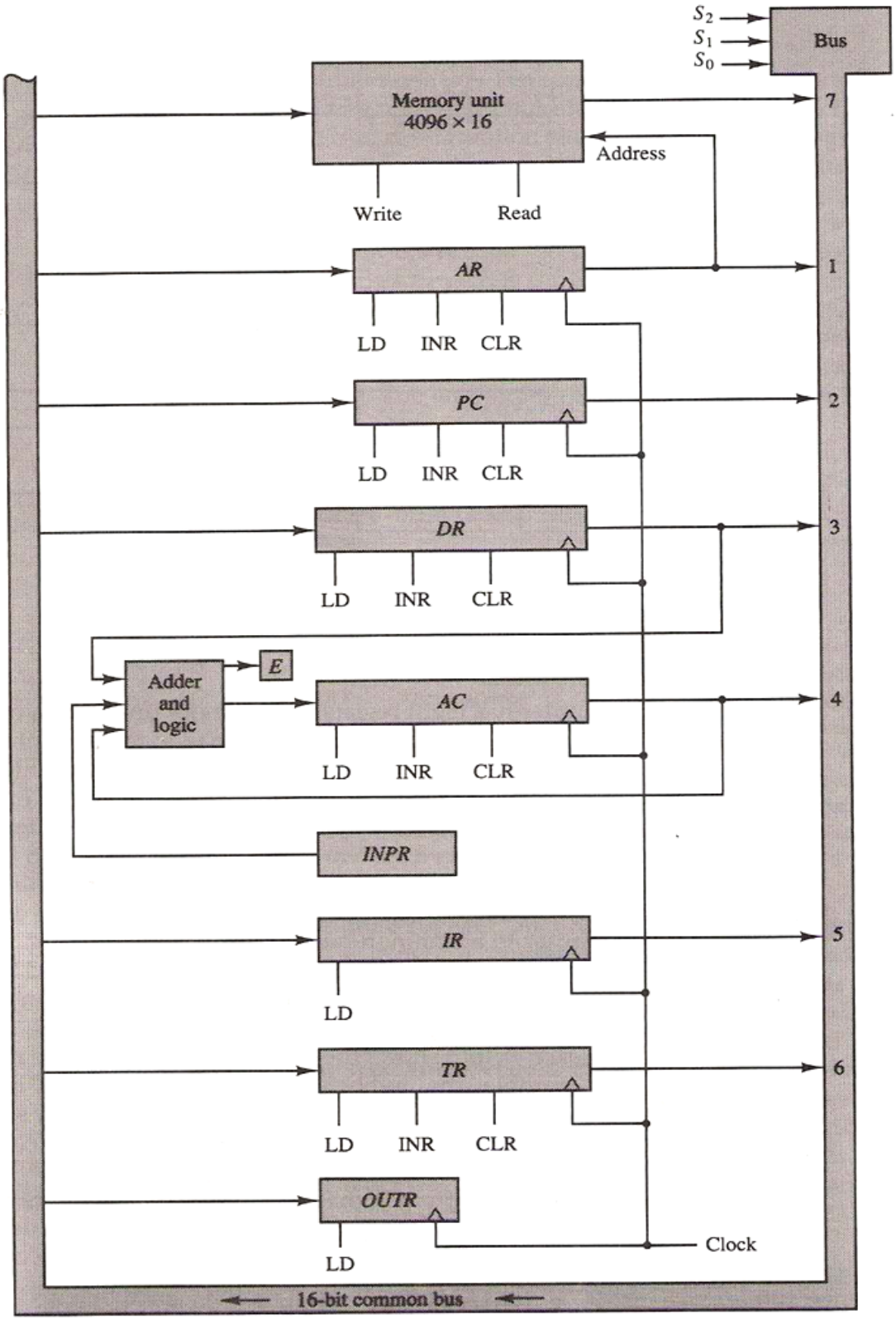


Fig: Common Bus System

→ Three control lines S2, S1 and S0 control the register to be selected as the input by the bus.

S2	S1	S0	Register
0	0	0	X (nothing)
0	0	1	AR

0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

→ 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 $S_2S_1S_0 = 111$.

3.3 Instruction Formats and their Execution

→ The instruction length of basic computer is 16-bit.
→ 16-bit instruction of basic computer has three fields:

Mode	Op-code	Operand (Address)

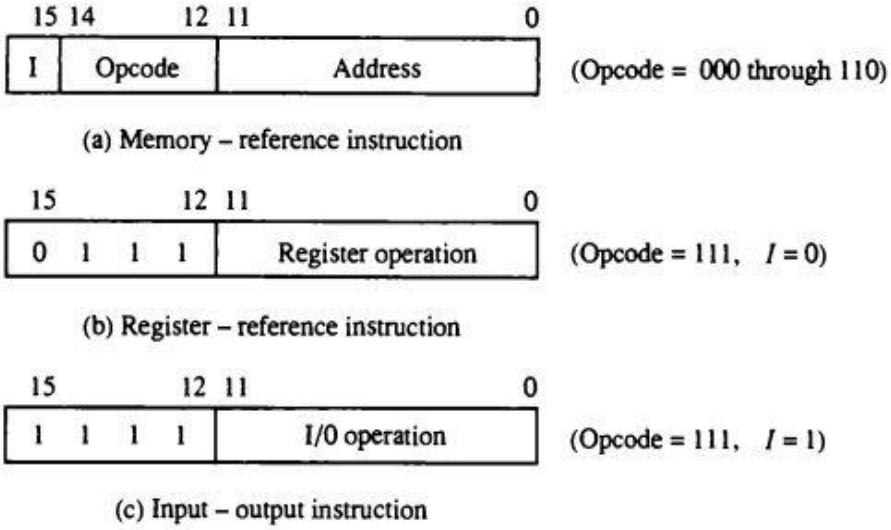
i) Mode field
→ MSB (bit 15) of the 16-bit instruction.
→ Value 0 = direct addressing, value 1 = indirect addressing mode.

ii) Op-code field
→ Defines what operation to be performed.
→ Contains 3 bits (bits 14-12)

iii) Operand (Address) field
→ Contains 12 bits (bits 11-0)

The basic computer has 3 instruction code formats. Type of the instruction is recognized by the computer control from 4-bit positions 12 through 15 of the instruction.

i) Memory-Reference Instructions
ii) Register-Reference Instructions
iii) Input-Output 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

Below is the complete operation that takes place during the instruction:

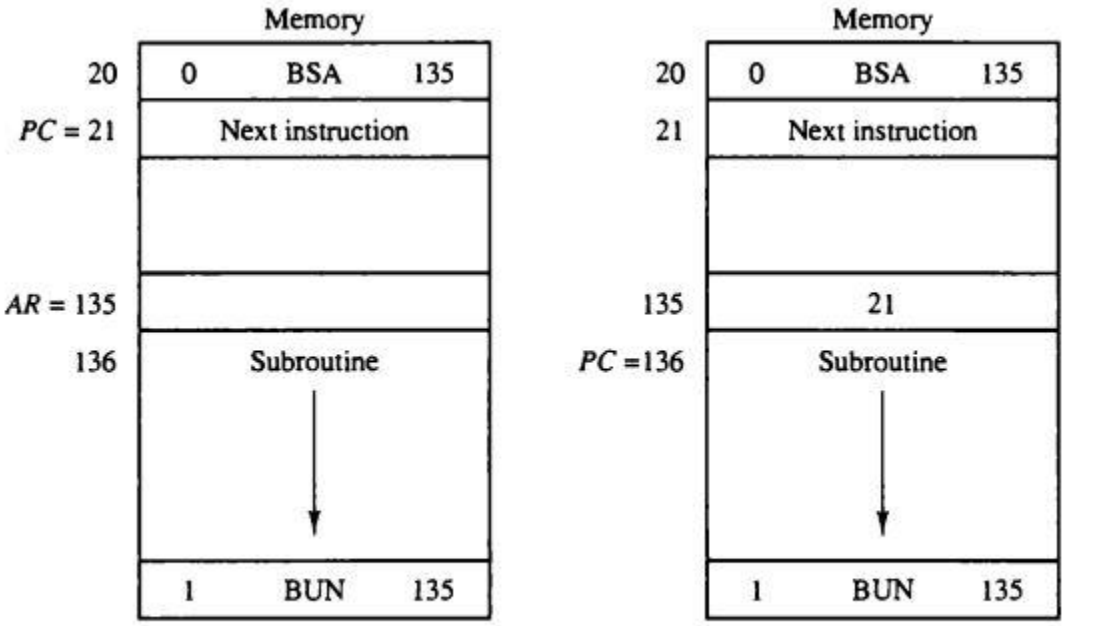
Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6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

Fig: Memory-Reference Instructions

Branch and Save Return Address (BSA)

D5T4: M[AR] <— PC, AR <— AR + 1

D5T5: PC <— AR, SC <— 0



For this example:
M[135] <— 21, AR <— 135 + 1
PC <— 136

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

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow$ 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

Fig: Register-Reference 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

Fig: Input-Output Instructions

Instruction Set Completeness

An instruction set is said to be complete if it contains sufficient instructions to perform operations in following categories:

Functional Instructions

- Arithmetic, logic, and shift instructions
- Examples: ADD, CMA, INC, CIR, CIL, AND, CLA

Transfer Instructions

- Data transfers between the main memory and the processor registers
- Examples: LDA, STA

Control Instructions

- Program sequencing and control
- Examples: BUN, BSA, ISZ

Input/output Instructions

- Input and output
- Examples: INP, OUT

Instruction set of Basic computer is complete because:

- ADD, CMA (complement), INC can be used to perform addition and subtraction and CIR (circular right shift), CIL (circular left shift) instructions can be used to achieve any kind of shift operations. Addition, subtraction and shifting can be used together to achieve multiplication and division. AND, CMA and CLA (clear accumulator) can be used to achieve any logical operations.
- LDA instruction moves data from memory to register and STA instruction moves data from register to memory.
- The branch instructions BUN, BSA and ISZ together with skip instruction provide the mechanism of program control and sequencing.
- INP instruction is used to read data from input device and OUT instruction is used to send data from processor to output device.

3.4 Timing and Control Unit

Control Unit

Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them. There are two types of control organization:

Hardwired Control

- CU is made up of sequential and combinational circuits to generate the control signals.
- If logic is changed, we need to change the whole circuitry.

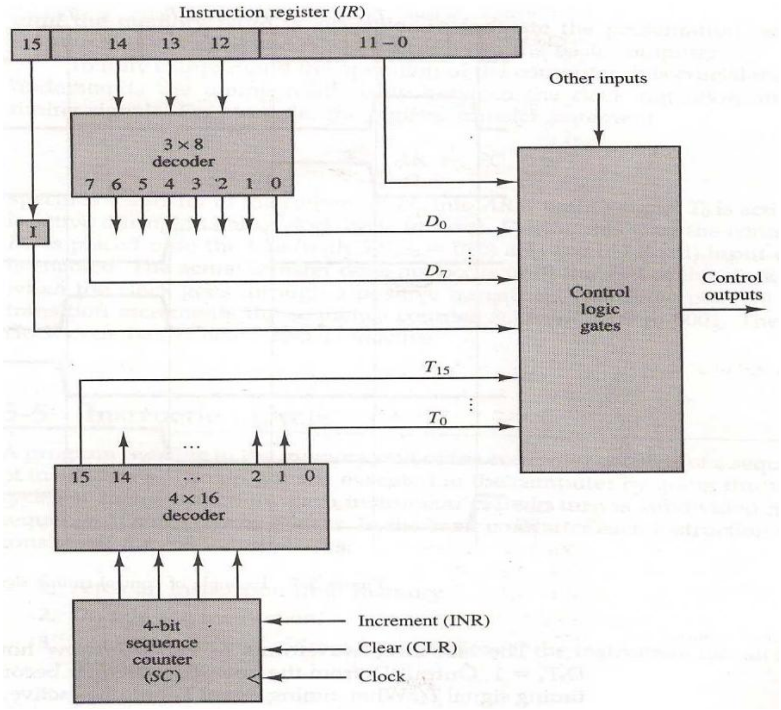
- Expensive
- Fast

Microprogrammed Control

- A control memory on the processor contains microprograms that activate the necessary control signals.
- If logic is changed, we only need to change the microprogram.
- Cheap
- Slow

NOTE: Microprogrammed control unit will be discussed in next chapter.

The block diagram of a hardwired control unit is shown below. It consists of two decoders, a sequence counter, and a number of control logic gates.



Mechanism:

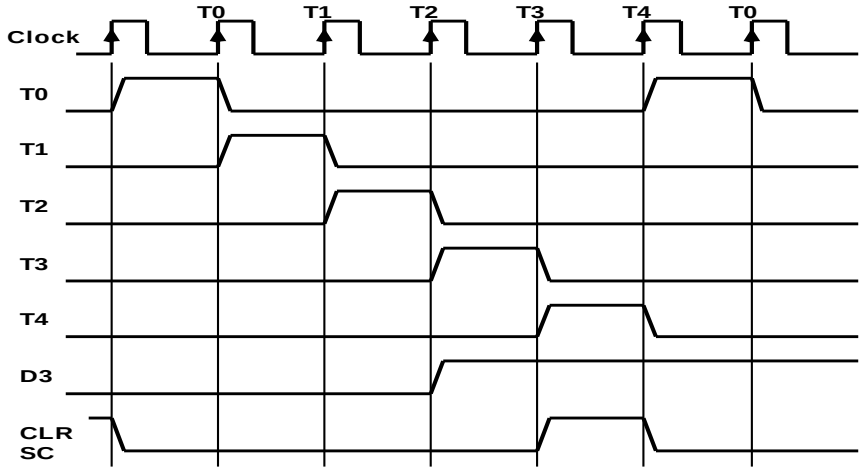
- An instruction read from memory is placed in the instruction resister (IR) where it is decoded into three parts: **I** bit, **operation code** and bits **0 through 11**.
- The operation code bit is decoded with 3 x 8 decoder producing 8 outputs D_0 through D_7.
- Bit 15 of the instruction is transferred to a flip-flop I.
- And operand bits are applied to control logic gates.
- The 16 outputs of 4-bit sequence counter (SC) are decoded into 16 timing signals T_0 through T_{15}.

This means instruction cycle of basic computer cannot take more than 16.

Fig: Control unit of a basic computer

Timing signals

- Generated by 4-bit sequence counter and 4x16 decoder.
- The SC can be incremented or cleared.
- Example: T_0, T_1, T_2, T_3, T_4, T_0, T_1 . . .
  Assume: At time T4, SC is cleared to 0 if decoder output D3 is active: D_3T_4: SC ← 0



3.5 Instruction Cycle

→ Processing required for complete execution of an instruction is called instruction cycle.

→ In Basic Computer, a machine instruction is executed in the following cycle:

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, control goes back to step 1 to fetch, decode and execute the next instruction. This process is continued indefinitely until HALT instruction is encountered.

Fetch and Decode

→ Sequence of steps required for fetching instruction from memory to CPU internal register is known as fetch cycle.

```
T0: AR ← PC  (S0S1S2=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)
```

→ For fetching and decoding, the steps are:

i) Initially, PC holds the address of next instruction to fetch. With timing signal T_0, address pointed by PC is transferred to the AR.

ii) The processor fetches instruction to IR from memory location referenced by AR and increment PC for next instruction. This happens with timing signal T_1.

iii) Processor interprets instruction and performs required action i.e. decoding during time period T_2.

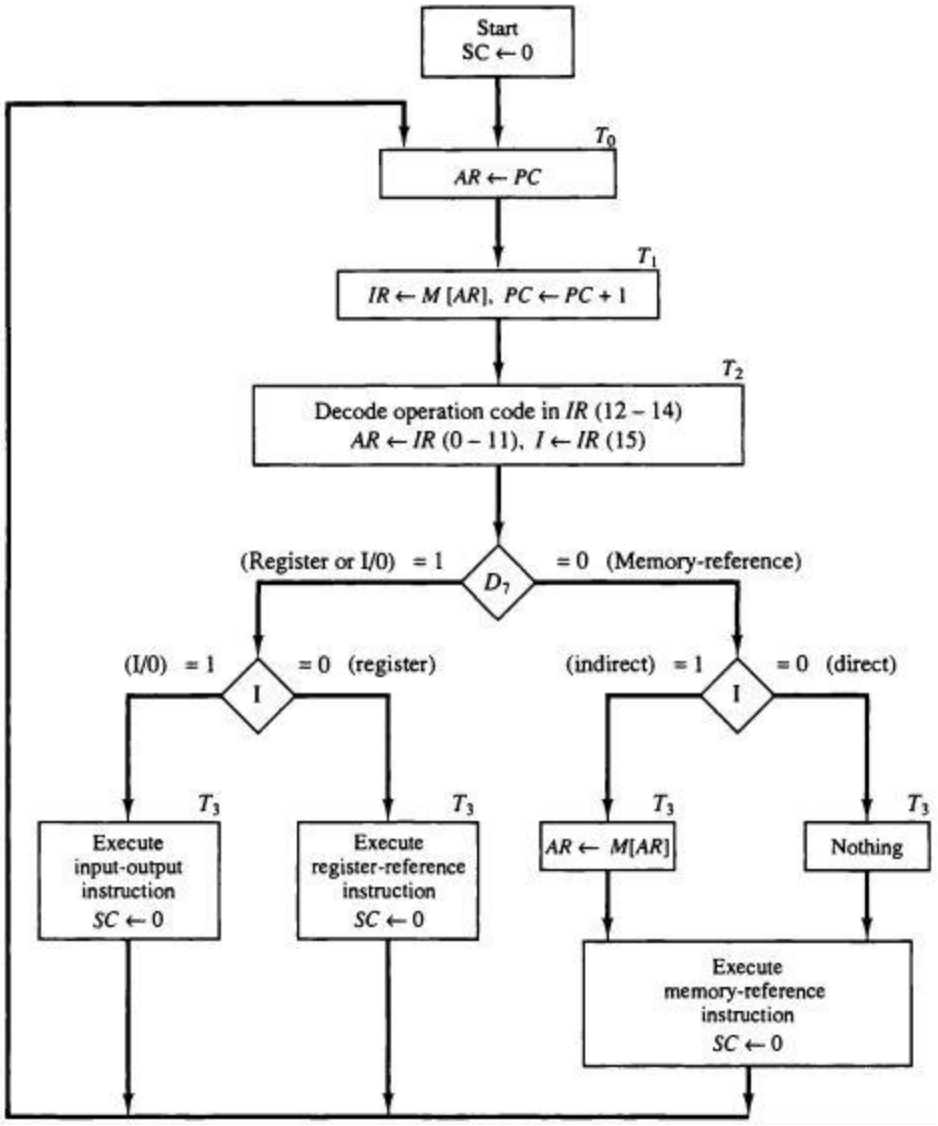


Fig: Flowchart of Instruction Cycle

→ Then, among decoded, D7 determines which type of instruction.

i) If D7 = 1, it will be either register-reference or input-output instruction.
   a) If I = 1, input-output instruction is executed during T3.
   b) If I = 0, register-reference instruction is executed during T3.

ii) If D7 = 0, it will be memory-reference instruction.
   a) If I = 1, indirect addressing mode instruction during T3.
   b) If I = 0, direct addressing mode instruction during T3.

→ The SC is reset after executing each instruction.

3.6 Input-Output Configuration

The terminal sends and receives serial information. Each quantity of information has 8 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 figure. 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.

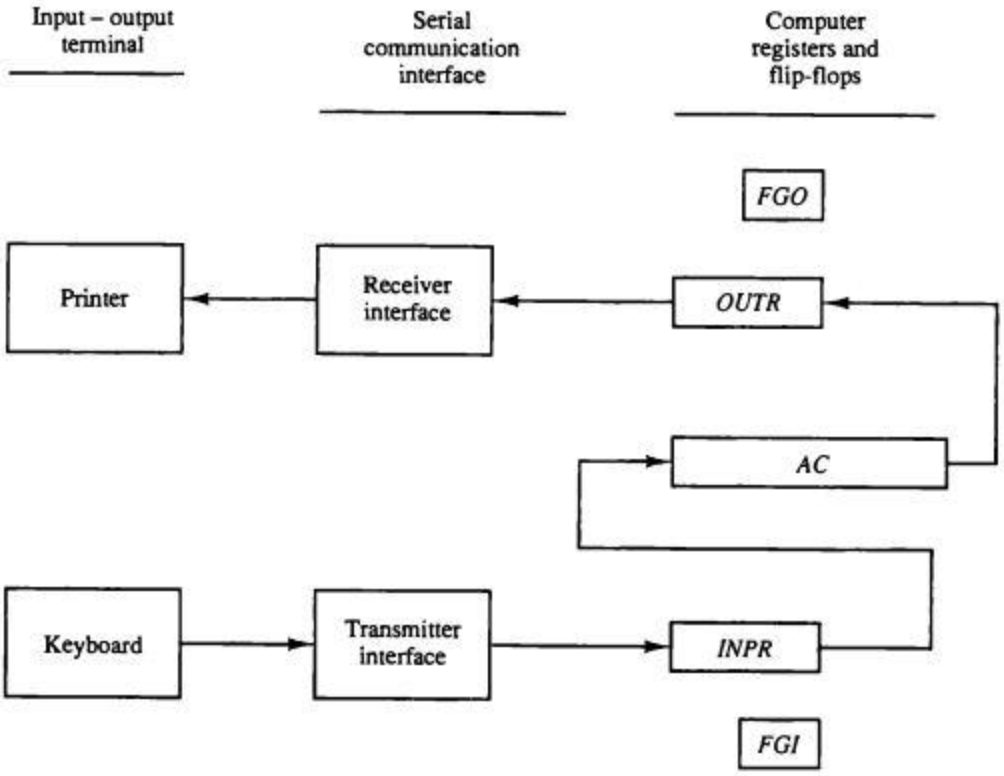


Fig: Input-Output Configuration

Scenario1: 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 control checks the flag bit, if 1, contents of INPR is transferred in parallel to AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

Scenario2: OUTR works similarly but the direction of information flow is reversed. Initially FGO is set to 1. The computer checks the flag bit; if it is 1, the information 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 operation is completed, it sets FGO to 1.