Unit 1
Basic Computer Organisation and Design

By- Yadvir Kaur

Contents

Instruction codes

In this chapter we will study about the basic computer and how its operation can be specified with **register transfer statements**(used to describe data flow at the register-transfer level of an architecture).

The internal organization of a digital system is defined by the sequence of microoperations it performs on data stored in its registers.

- The **general purpose digital computer** is capable of executing various microoperations and, in addition., can be instructed as to what specific sequence of operations it must perform. **The user of a computer can control the process by means of a program**.
- 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 microoperations** for the computer.

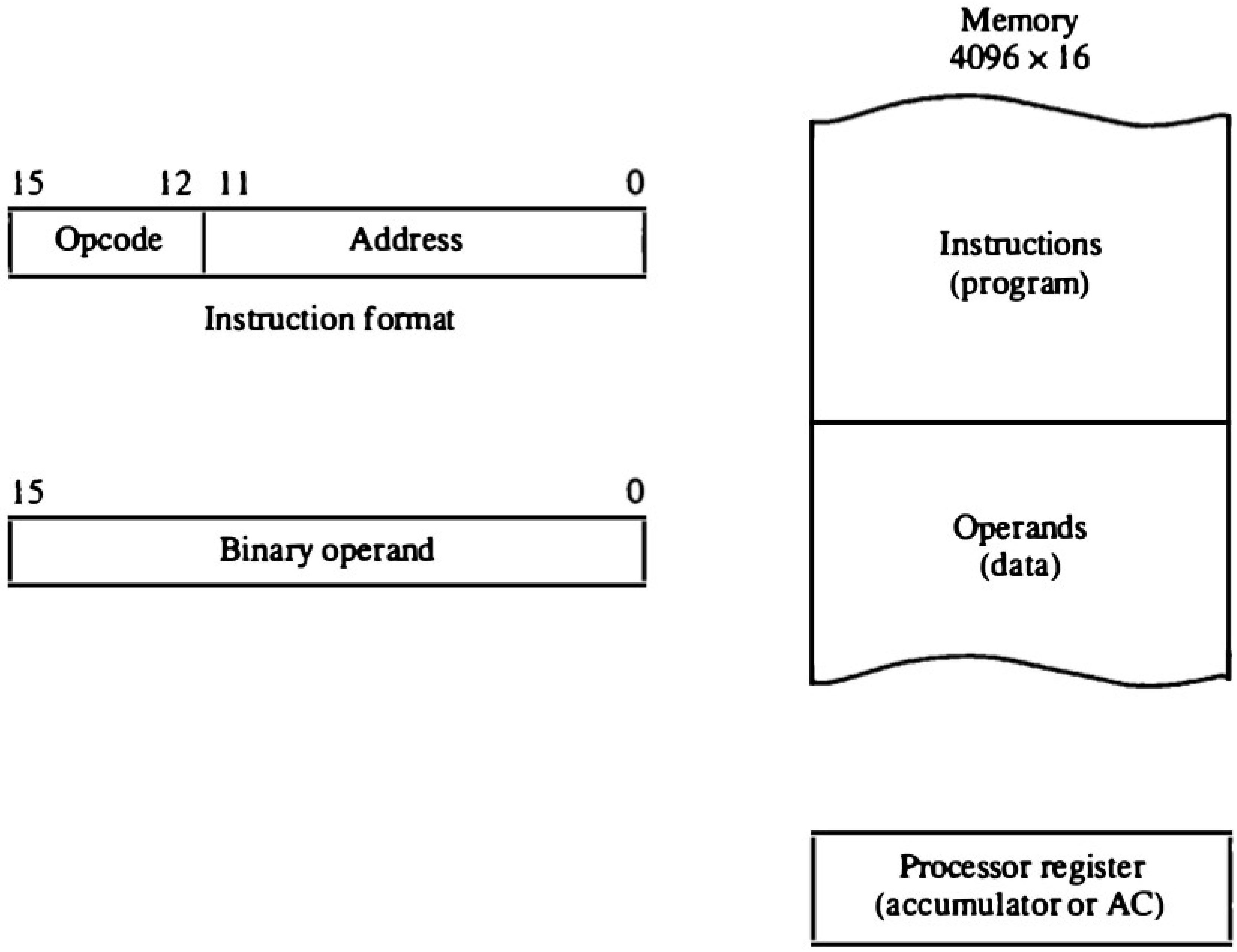
Instruction codes

- Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.
- Instruction code is divided into two parts namely **opcode (operation code)** and **operand bits (address of data)**. Operation code consisting group of bits to define an operation such as add, subtract, multiply etc.
- The **number of bits allocated for the opcode determined how many different instructions the architecture supports.** For example, a 4-bit opcode encoded as a binary number could represent up to 16 different operations.
- The *control unit* is responsible for decoding the opcode and operand bits in the instruction register, and then generating the control signals necessary to drive all other hardware in the CPU to perform the sequence of microoperations that form the instruction.

Stored Program Organization

- The **simplest way to organize a computer** is to have **one processor register** and **an instruction code** format with two parts. The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.
- **Each instruction code is of 16-bit**, 12 bits to specify the address of an operand, we are left with 4 bits for operation code (opcode) to specify one out of 16 possible operations.

Figure 5-1 Stored program organization.



Direct & Indirect Address

- **Accumulator (AC):** Computers that have a single-processor register usually assign to it the name accumulator and label it AC . The operation is performed with the memory operand and the content of AC .
- **Immediate Instruction:**

  » When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.

  » Fast.

  » Operand is constant.

  » No memory refrence is required to access the operand.
- **Effective address**: Where the actual operand is found.
- **General address:** Where the effective address is found.

Direct & Indirect Address

- **Direct address**:

  » When the s**econd part specifies the address of an operand(effective address)**, the instruction is said to have a direct address.

  » Single memory refrence is required to access the operand.

  » No additional calculations to workout effective address.

- **Indirect address:**

  » It is the one where the bits in the second part of the instruction designate an address of a memory word(general address) in which the address(effective address) of the operand is found.

  » **Operand contains the general address. From general address we get the effective address. From effective address we get the data**.

  » Two memory refrences are required to access the operand.

  » Using Indirect address we implement the pointers.

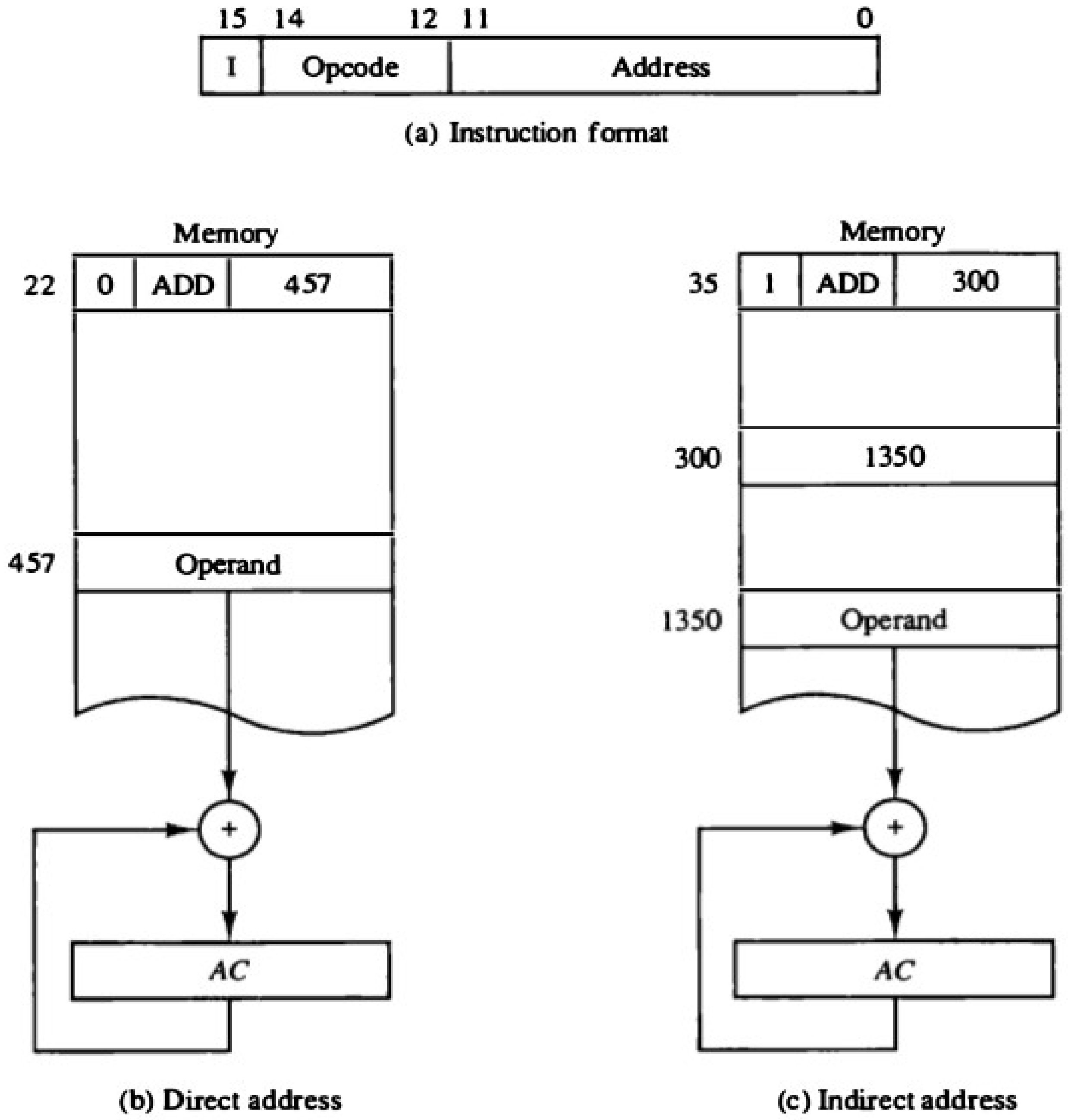


Figure 5-2 Demonstration of direct and indirect address.

Computer Registers

Register: A register is a **very small amount of very fast memory that is built into CPU.**

Registers are normally measured by the no. of bits they can hold, for example, an 8-bit register means it can store 8 bits of data. Registers in a basic computer:

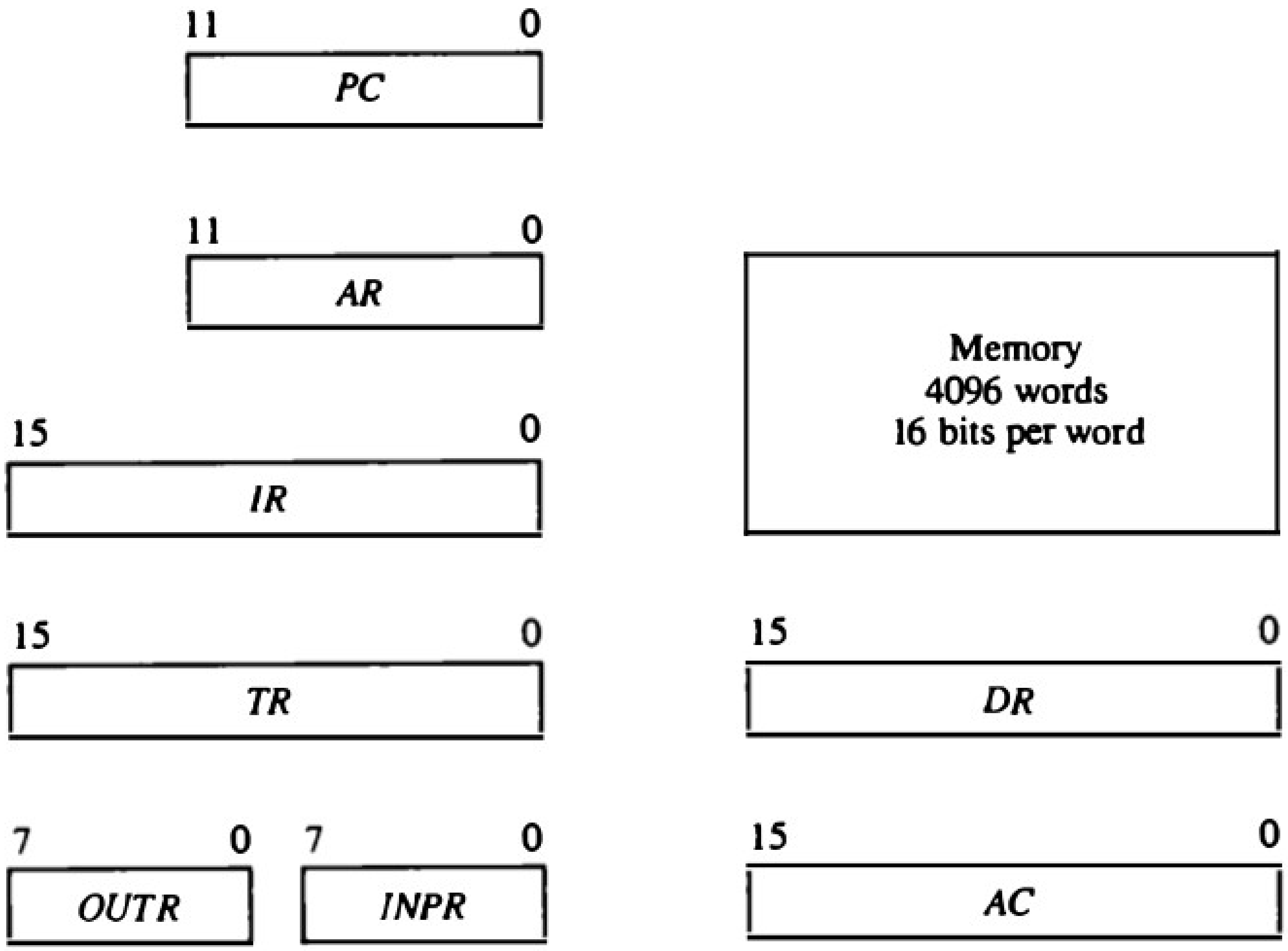


Figure 5-3 Basic computer registers and memory.

Computer Registers

- There are 2 12-bit registers in the commin bus system:

1) **AR (Address register):** Hold a memory address. It points to a memory location where the program being run will fetch some data.
2) **PC (Program Counter)**: It holds the address of the next instruction to be read from memory after the current instruction is executed.

- There are 4 16-bit registers:

1) **IR (Instruction Register): H**old the instruction read from memory.
2) **DR (Data Register)**: Holds the operand(Data) read from memory.
3) **Accumulator (AC)**: The accumulator register is a general purpose processing register. It is used to hold the results/partial results of arithmetic & logic operations.
4) **TR (Temporary Register)**: Used for holding temporary data during the processing.

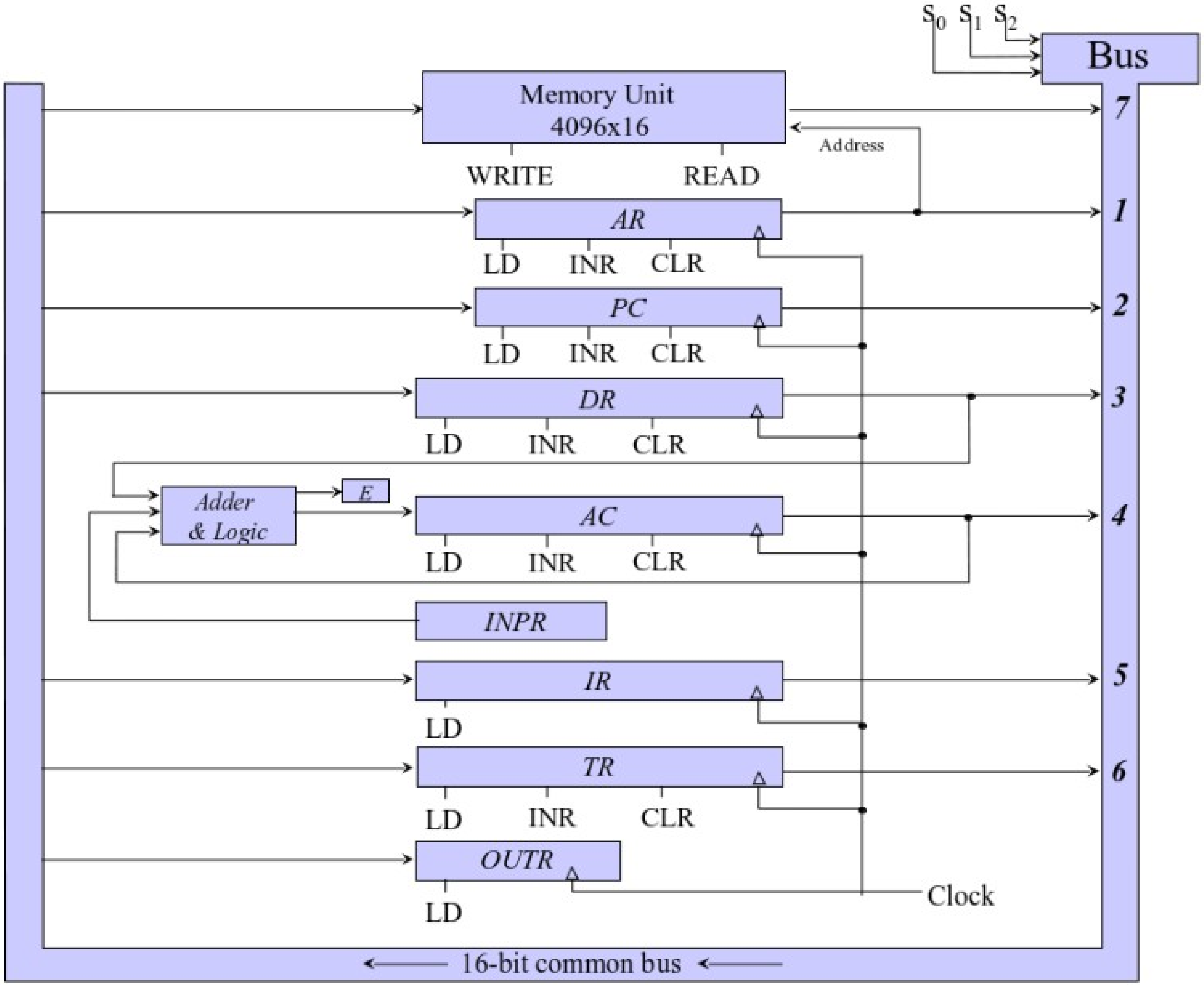
Computer Registers

- There are 2 8-bit registers in the commin bus system:

1) **Input Register(INPR):** Receives an 8-bit character from an input device and delivers it to AC.
2) **Output Register(OUTR):** It receiver an 8-bit character from AC and transfers it to an output device.

Common Bus System

- A wire or a collection of wires that carry some multi-bit information is known as **bus**.
- **Basic computer has eight registers, a memory unit, and a control unit**.
- 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 is to use a common bus.
- Path must be provided to transfer information from one register to another and between memory and registers.



S0 S1 S2
Bus
Memory Unit 4096x16
Address
WRITE
READ
AR
LD INR CLR
PC
LD INR CLR
DR
LD INR CLR
Adder & Logic
E
AC
LD INR CLR
INPR
IR
LD
TR
LD INR CLR
OUTR
LD
Clock
7
1
2
3
4
5
6
16-bit common bus

Common Bus System

- **Selection variables**: Used to specify a register whose output is connected to the common bus at any given time. **To select one register out of 8, we need 3 select variables.**

  For example, if S2S1S0 = 011, the output of DR is directed to the common bus.

- 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 = 1 1 1 .

- **Control inputs: LD, INR, CLR:**

1) **Load input (LD)**: The particular register whose LD input is enabled receives the data from bus.
2) **Increment input (INR)**: Increments the content of a register.
3) **Clear input (CLR)**: Clear the content of a register to zero.

Common Bus System

- **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 3 control inputs: LD(Load), INR(Increment), CLR (clear). Two registers (i.e. IR, OUTR) have only a LD input.**

- **The 16-bit inputs of AC come from an adder and logic circuit. This circuit has three sets of inputs**:

1) **One set of 16-bit inputs come from the outputs of AC**. They are used to implement register microoperations such as complement AC and shift AC.
2) **Another set of 16-bit inputs come from the data register DR**. The inputs from DR and AC are used for arithmetic and logic rnlcrooperations, 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).
3) **A third set of 8-bit inputs come from the input register INPR.**

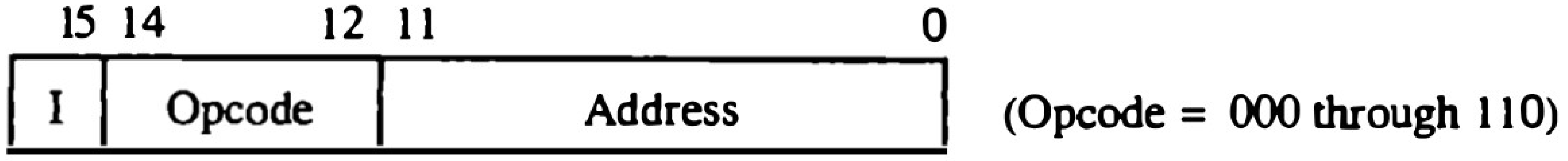
Computer Instructions

- The basic computer has three instruction code formats. 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.

1) Memory-reference instruction:

» Uses 12 bits to specify an address and 3 bits for opcode.

» 1 bit to specify the addressing mode I (I = 0 for direct address, I = 1 for indirect address)



(a) Memory – reference instruction

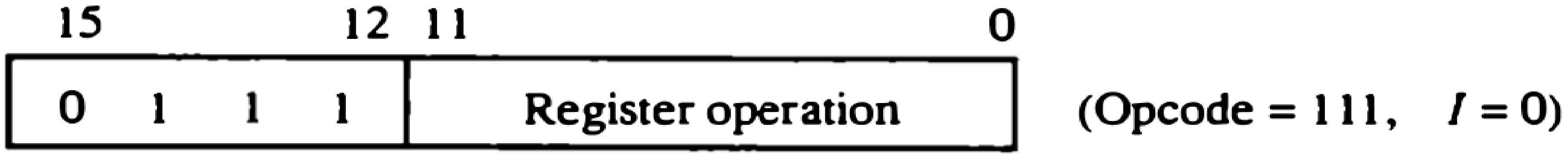
Computer Instructions

2) Register reference instructions:

» 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 the AC register.

» An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.



(b) Register – reference instruction

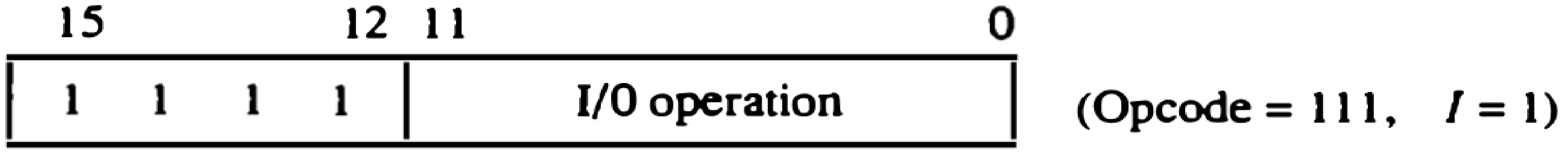
Computer Instructions

3) Input-output instruction:

» Does not need a reference to memory.

» Recognized by the operation code 111 with a 1 in the leftmost bit (bit 15) of the instruction.

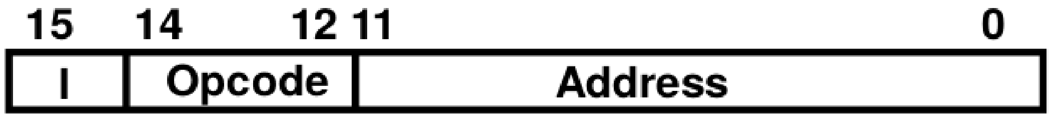
» The remaining 12 bits are used to specify the type of input-output operation or test performed.



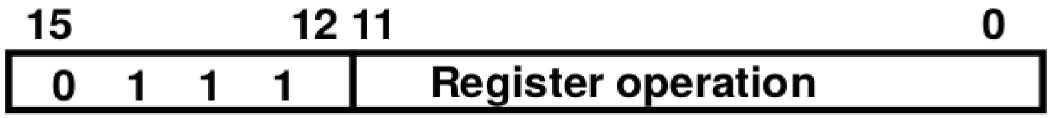
(c) Input – output instruction

Basic Computer Instruction Format

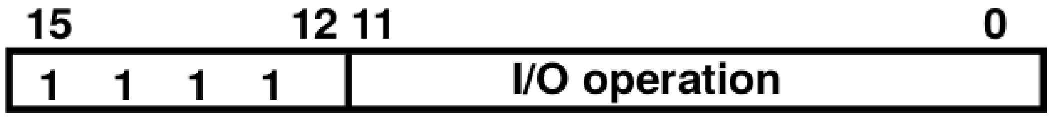
Memory-Reference Instructions **(OP-code = 000 ~ 110)**



Register-Reference Instructions **(OP-code = 111, I = 0)**



Input-Output Instructions **(OP-code =111, I = 1)**



The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction. By using the hexadecimal equivalent we reduced the 16 bits of an instruction code to four digits with each hexadecimal digit being equivalent to four bits.

Memory-reference instruction:

» has an address part of 12 bits which is denoted by three x's and stand for the three hexadecimal digits corresponding to the 12-bit address.

» The last bit of the instruction is designated by the symbol I. When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6. When I = I, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E.

Register reference instructions

» 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7.

» The other three hexadecimal digits give the binary equivalent of the remaining 12 bits.

Input -output instructions

» Use all 16 bits to specify an operation.

» The last four bits are always 1111 , equivalent to hexadecimal F.

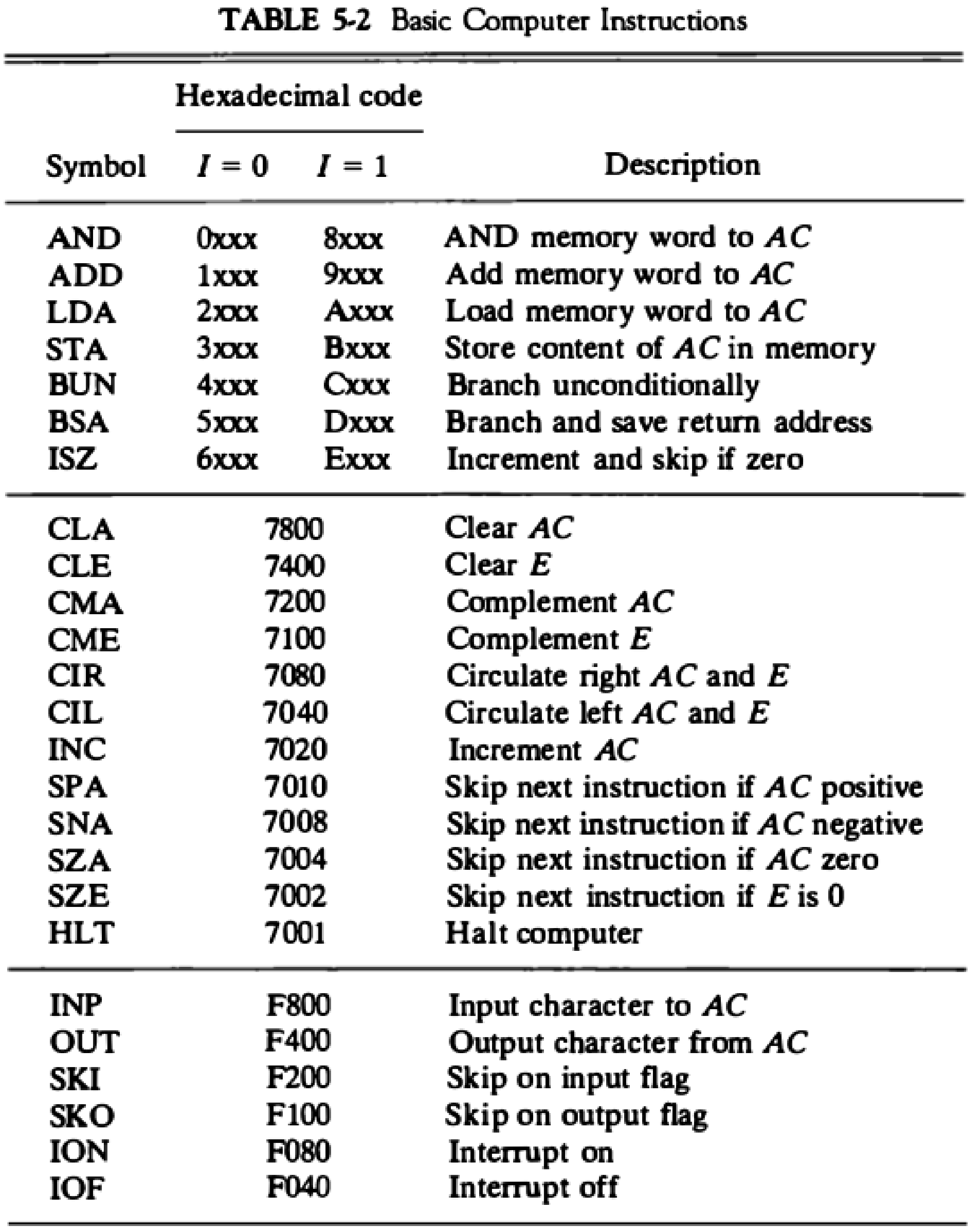


TABLE 5-2 Basic Computer Instructions

Memory-reference instruction:

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

Register-reference instructions:

Symbol	Hexadecimal code	Description
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

Input-output instructions:

Symbol	Hexadecimal code	Description
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

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.
- There are two major types of control organization:
    - **hardwired control and**
    - **microprogrammed control.**

Timing and Control

Hardwired control:

- The **control logic is implemented with gates, flip-flops, decoders, and other digital circuits.**
- A hardwired control, **if the design has to be modified, we need to make the required changes in the wiring among the various components.**

microprogrammed control:

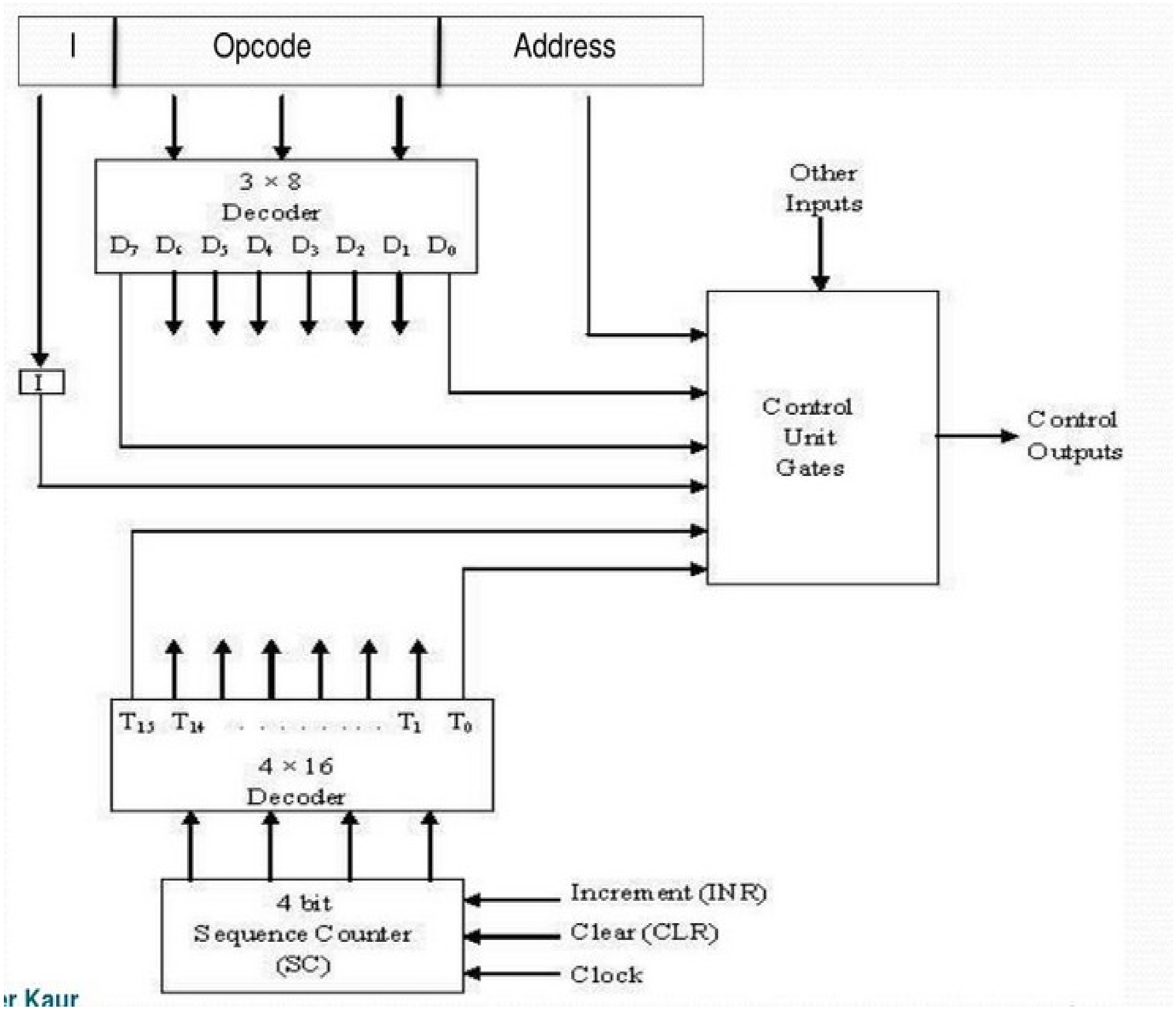
- The control information is stored in a control memory. **The control memory is programmed to initiate the required sequence of microoperations.**
- Any required changes or **modifications can be done by updating the microprogram** in control memory.

Control unit of basic computer

A hardwired control for the basic computer is presented:

- 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). Instruction register 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, which tells about the 8 operations that can be performed on various operands. 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.

Control unit of basic computer



Control unit of basic computer

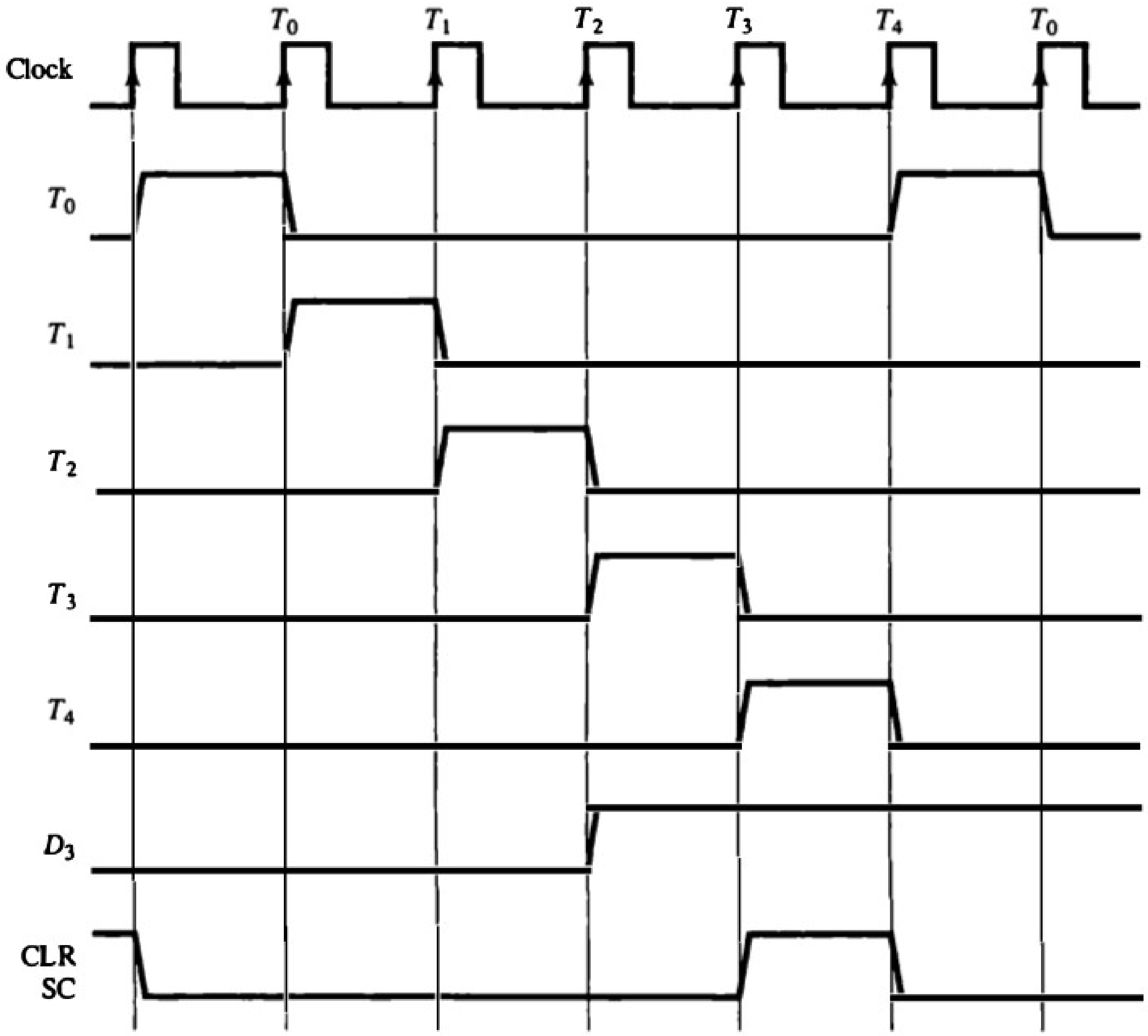
- The **Sequence counter** is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.
- **Once in awhile, the counter is cleared to 0, causing the next active timing signal to be To.**
- As an example, consider the case where SC is incremented to provide timing T1, T2, T3, and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement:

  D_3T_4:   SC <-- 0

Example of control timing signals

- The timing diagram of statement: D_3T_4: SC <-- 0
- SC is incremented with every positive clock transition, unless its CLR input is active. This produces the sequence of timing signals T_0. T_1, T_2, T_3, T_4, and so on.
- Initially, the CLR input of SC is active. The first positive transition of the clock clears SC to 0, which in tum activates the timing signal T_0 out of the decoder.
- Output D3 from the operation decoder becomes active at the end of timing signal T_2.
- When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active. This signal is applied to the CLR input of SC.
- On the next positive clock transition the counter is cleared to 0. This causes the timing signal To to become active instead of T5 that would have been active if SC were incremented instead of cleared.

Example of control timing signals



INSTRUCTION CYCLE

- A program consists of a sequence of instructions. When the program is to be executed, every instruction present in the program has to be executed.
- **Every instruction goes through a cycle, which in turn is subdivided into a 4 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

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 To.**
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements:

1) T_0: AR <-- PC

2) T_1: IR <-- M [AR], PC <- PC + 1

3) T_2: D_0, . . ., D_7 <-- Decode IR(12-14), AR <--- IR(0-11), I <--- IR(15)

1) **T0: AR <-- PC** (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 To) Timing signal T0 to achieve the following connection:

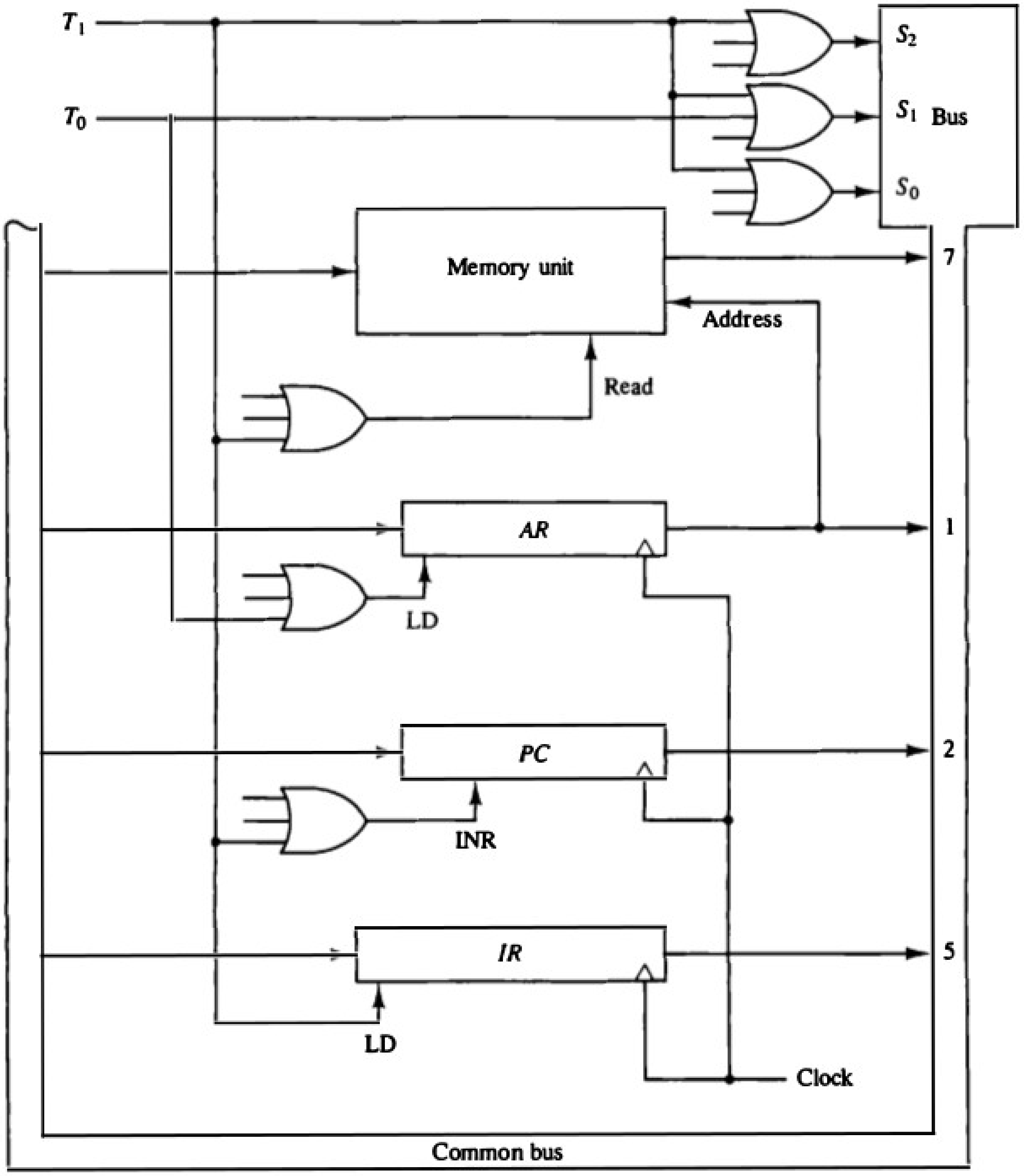
- **Place the content of PC onto the bus by making the bus selection inputs S2S1S0 = 010.**
- **Transfer the content of the bus to AR by enabling the LD input of AR .**

2) **T1: IR <-- M [AR], PC <-- PC + 1** (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.) Timing signal T1 to provide the following connections:

- **Enable the read input of memory.**
- **Place the content of memory onto the bus by making S2S1S0 = 111.**
- **Transfer the content of the bus to IR by enabling the LD input of IR .**
- **Increment PC by enabling the INR input of PC .**

3) **T2: D0, . . ., D7 <-- Decode IR(12-14), AR <--- IR(0-11), I <--- IR(15)** (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**.)

Register transfers for the fetch phase



Determine the Type of Instruction

- After decoding, T_3 is active. **During time T_3, the control unit determines the type of instruction that was just read from memory.**
- **If the operation code = 111, i.e. decoder output D7 = 1, the instruction must be 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.**
- 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:

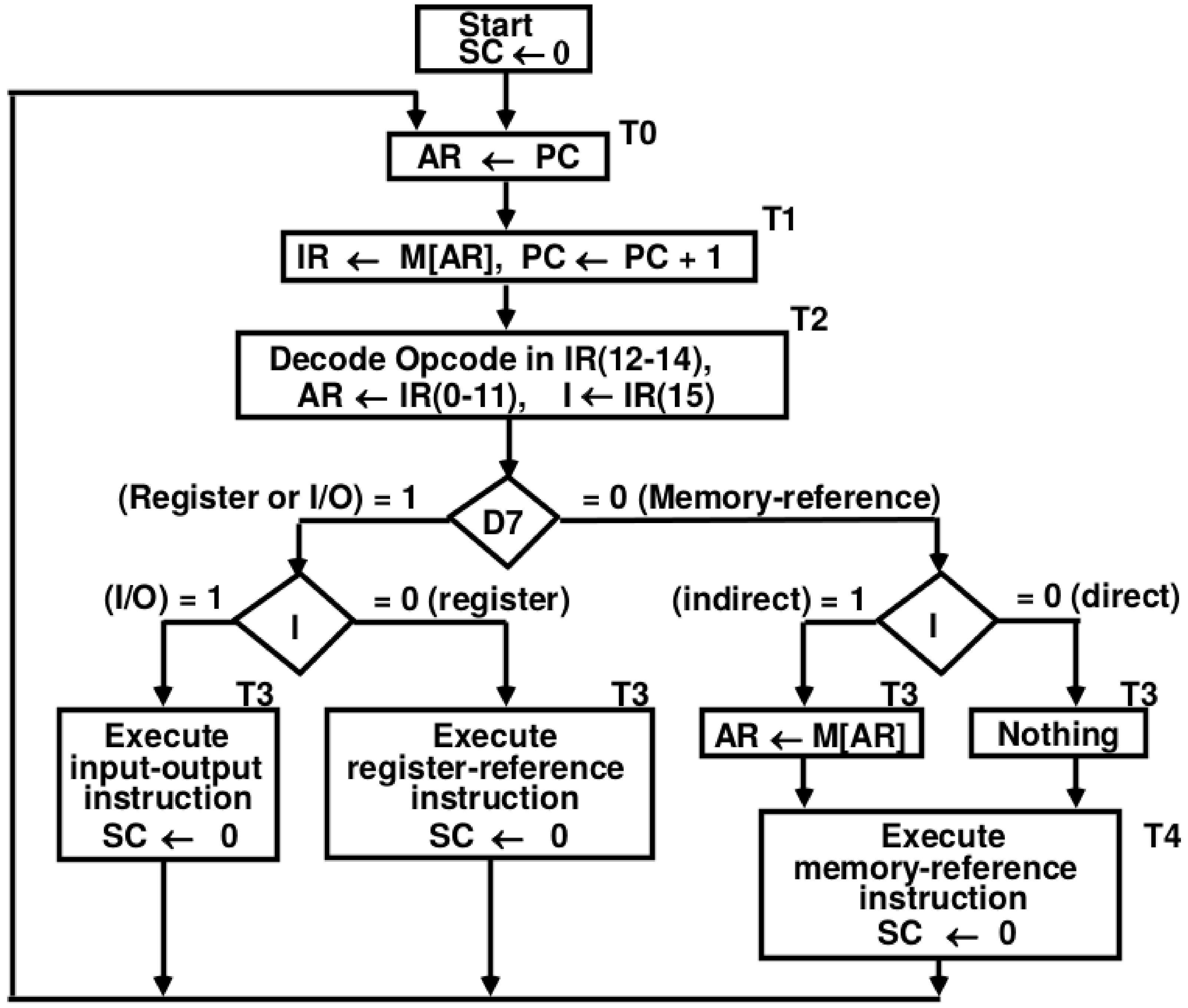
1) **$D'_7 I T_3$: AR <-- M [AR]**

2) **$D'_7 I' T_3$: Nothing**

3) **$D_7 I' T_3$: Execute a register-reference instruction**

4) **$D_7 I T_3$: Execute an input -output instruction**

DETERMINE THE TYPE OF INSTRUCTION


Start
SC ← 0
T0
AR ← PC
T1
IR ← M[AR], PC ← PC + 1
T2
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M[AR]
T3
Nothing
T4
Execute memory-reference instruction SC ← 0

Memory-Reference Instructions

- **The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1**

- **AND to AC**

  This instruction performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.

  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, transfering control to timing signal T0 to start a new instruction cycle.

D_0T_4:	$DR \leftarrow M[AR]$	Read operand
D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$	AND with AC

- **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 Cout is transferred to the E (extended accumulator) flip-flop.

  D_1T_4: $DR \leftarrow M[AR]$ Read operand

  D_1T_5: $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$ Add to AC and store carry in E

- **LDA: Load to AC**

  This instruction **transfers the memory word specified by the effective address to AC.**

  As **there is no direct path from the bus into AC . The adder and logic circuit receive information from DR which can be transferred into AC** .

  D_2T_4: $DR \leftarrow M[AR]$

  D_2T_5: $AC \leftarrow DR$, $SC \leftarrow 0$

- **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: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

- **BUN: Branch Unconditionally**

  Remember that PC holds the address of the instruction to be read from memory in the next instruction cycle.

  The **BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally.**

  D_4T_4: PC $\leftarrow$ AR, SC $\leftarrow$ 0

- **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.

  Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.

  D_6T_4: DR $\leftarrow$ M[AR]

  D_6T_5: DR $\leftarrow$ DR + 1

  D_6T_4: M[AR] $\leftarrow$ DR, if (DR = 0) then (PC $\leftarrow$ PC + 1), SC $\leftarrow$ 0

- **BSA: Branch and Save Return Address**

  This instruction is **useful for branching to a portion of the program called a subroutine or procedure.**

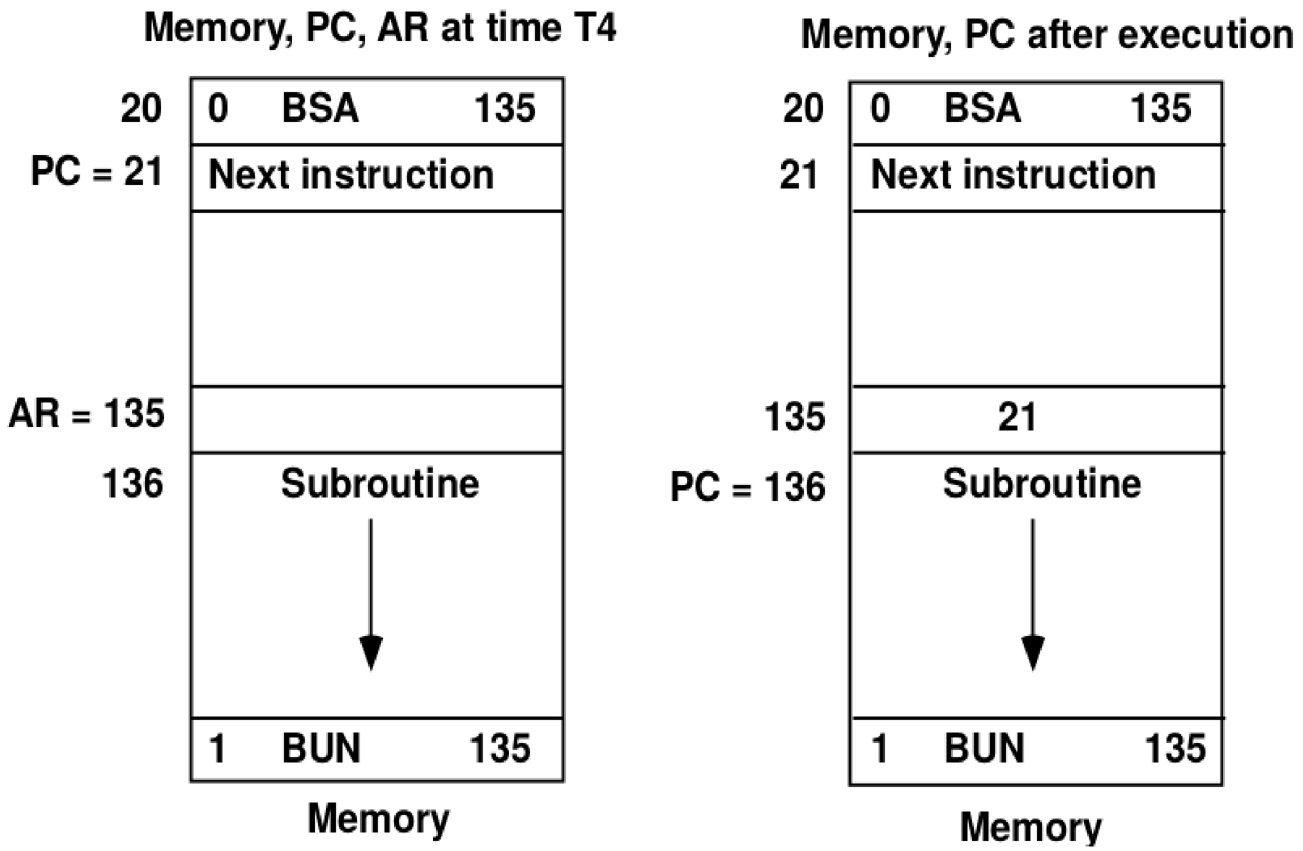
  BSA instruction **stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the EA.**

  The **EA +1 is then transferred to PC to serve as the address of the first instruction in the subroutine.**

  The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return.

D_5T_4: $M[AR] \leftarrow PC,$
$AR \leftarrow AR + 1$

D_5T_5: $PC \leftarrow AR,$
$SC \leftarrow 0$



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 instructions are executed with the clock transition associated with timing variable T3.
- 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 Bi to bit i of IR, all control functions can be simply denoted by rBi.

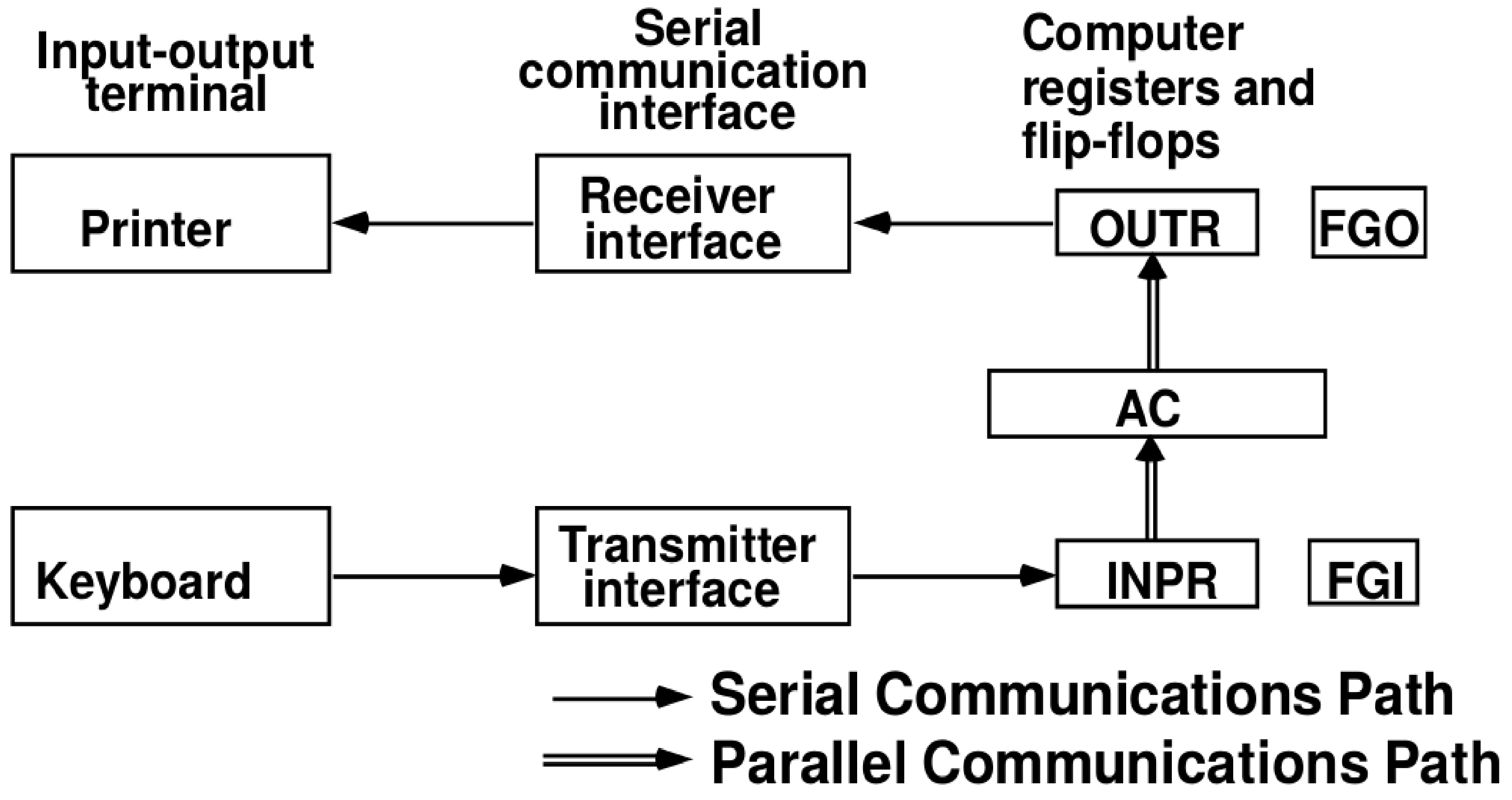
TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (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:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	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

INPUT-OUTPUT AND INTERRUPT

- **Input-Output Configuration**
- INPR- Input register - 8 bits, OUTRO- output register - 8 bits
- FGI- Input flag - 1 bit, FGO- Output flag - 1 bit, IEN- Interrupt enable - 1 bit



Input-output terminal
Serial communication interface
Computer registers and flip-flops
Printer
Receiver interface
OUTR
FGO
AC
Keyboard
Transmitter interface
INPR
FGI
Serial Communications Path
Parallel Communications Path

INPUT-OUTPUT AND INTERRUPT

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into input register INPR
- The serial info. for the printer is stored in the output register OUTR
- **INPR and OUTR communicate with the terminal serially and with the AC in parallel.**
- **The flags are needed to *synchronize* the timing difference between I/O device and the computer**
- **FGI : *set* when INPR is ready, *clear* when INPR is empty** : Initially, the input flag FGI is cleared to 0. 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.
- **FGO : *set* when operation is completed, *clear* when output device is in the process of printing**: 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 .

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.

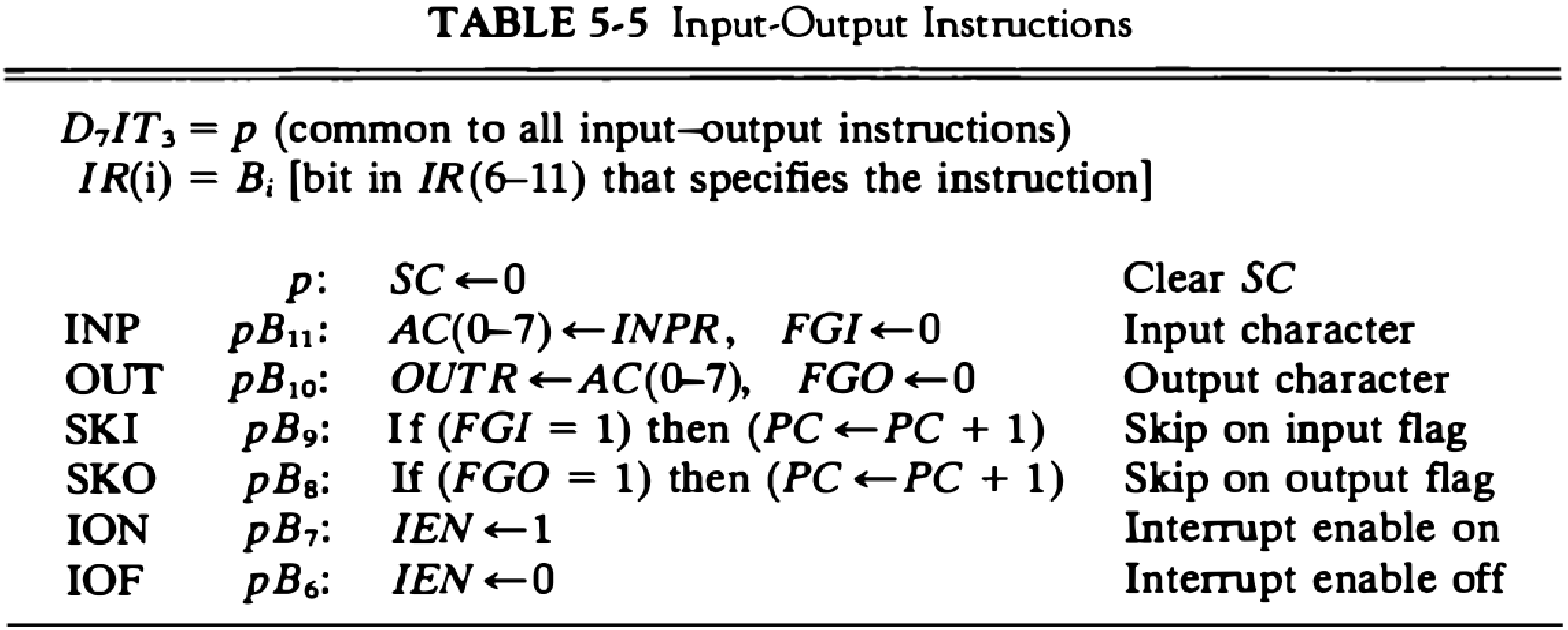


TABLE 5-5 Input-Output Instructions

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

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

Program Interrupt

- **Program-controlled I/O**: In this the **computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.** The computer is wasting time while checking the flag instead of doing some other useful processing task.
- **Interrupt initiated I/O:** An alternative to the programmed controlled procedure is to let the **external device inform the computer when it is ready for the transfer.**
- The ***I/O interface***, instead of the CPU, monitors the I/O device.
- When the interface founds that the I/O device is ready for data transfer, it generates an ***interrupt request*** to the CPU
- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.
- IEN (Interrupt-enable flip-flop): can be set and cleared by instructions. When cleared, the computer cannot be interrupted.

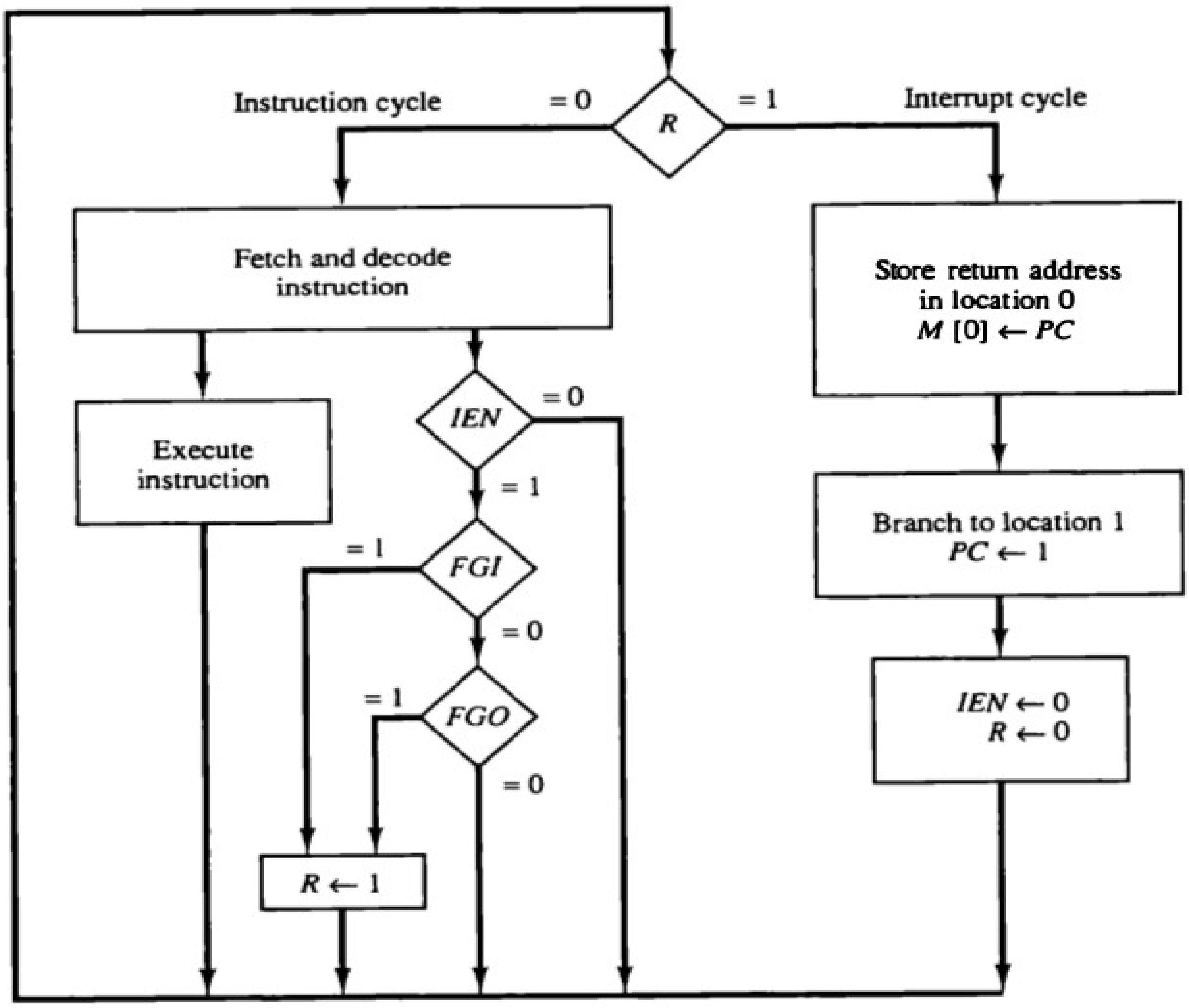


Figure 5-13 Flowchart for interrupt cycle.

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 IEN =0: it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.

  - If IEN=1: control checks the flag bits. If either flag is set to 1, flip-flop R is set to 1.
- **At the end of the execute phase, control checks the value of R**

  - **If R= to 1, it goes to an interrupt cycle instead of an instruction cycle.**

  - If R= to 0: Instruction cycle.
- **Demonstration of the interrupt cycle**

  - The interrupt cycle 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.

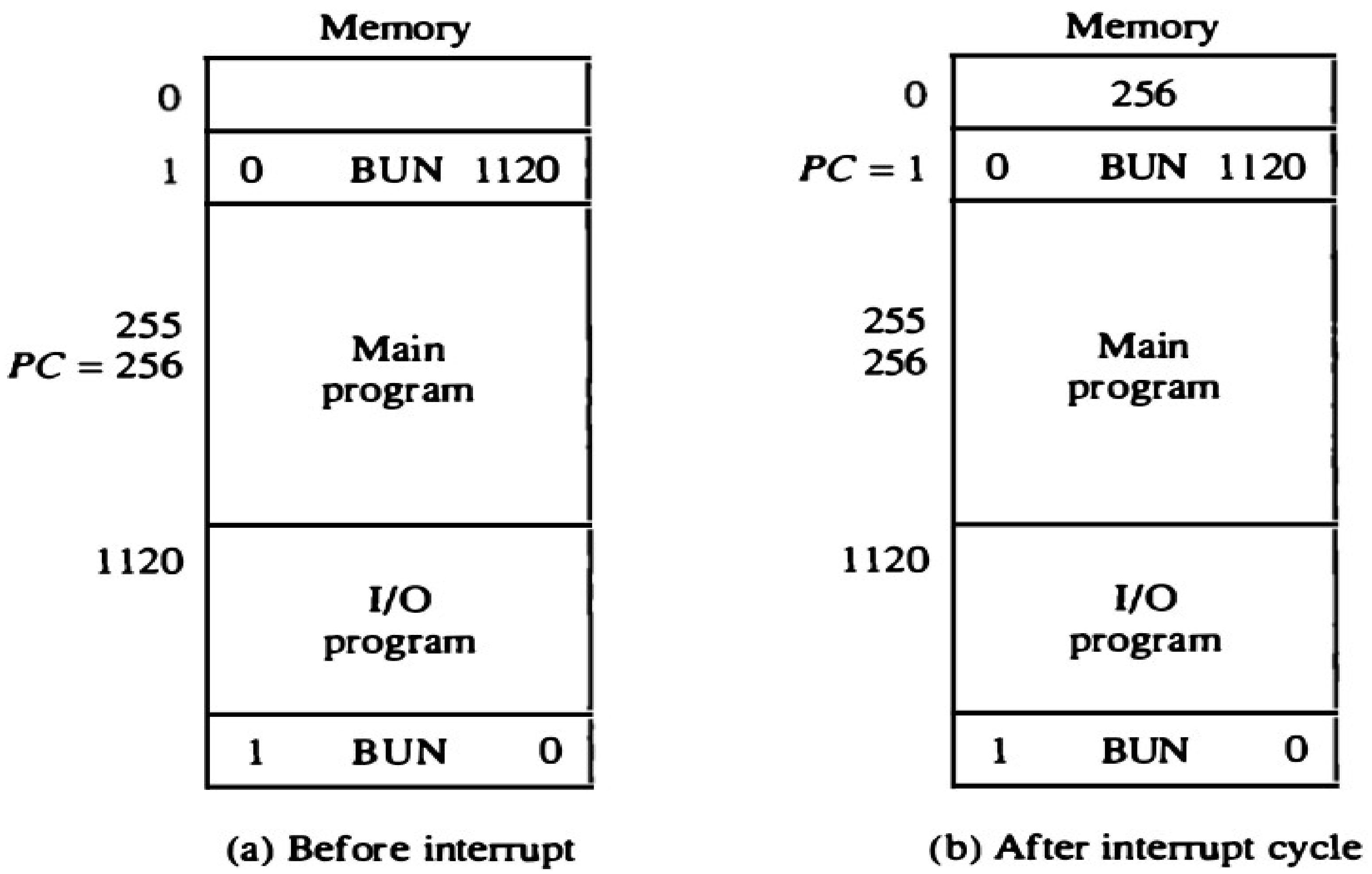
  - The memory location at address 0 is used as the place for storing the return address.

  - 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.

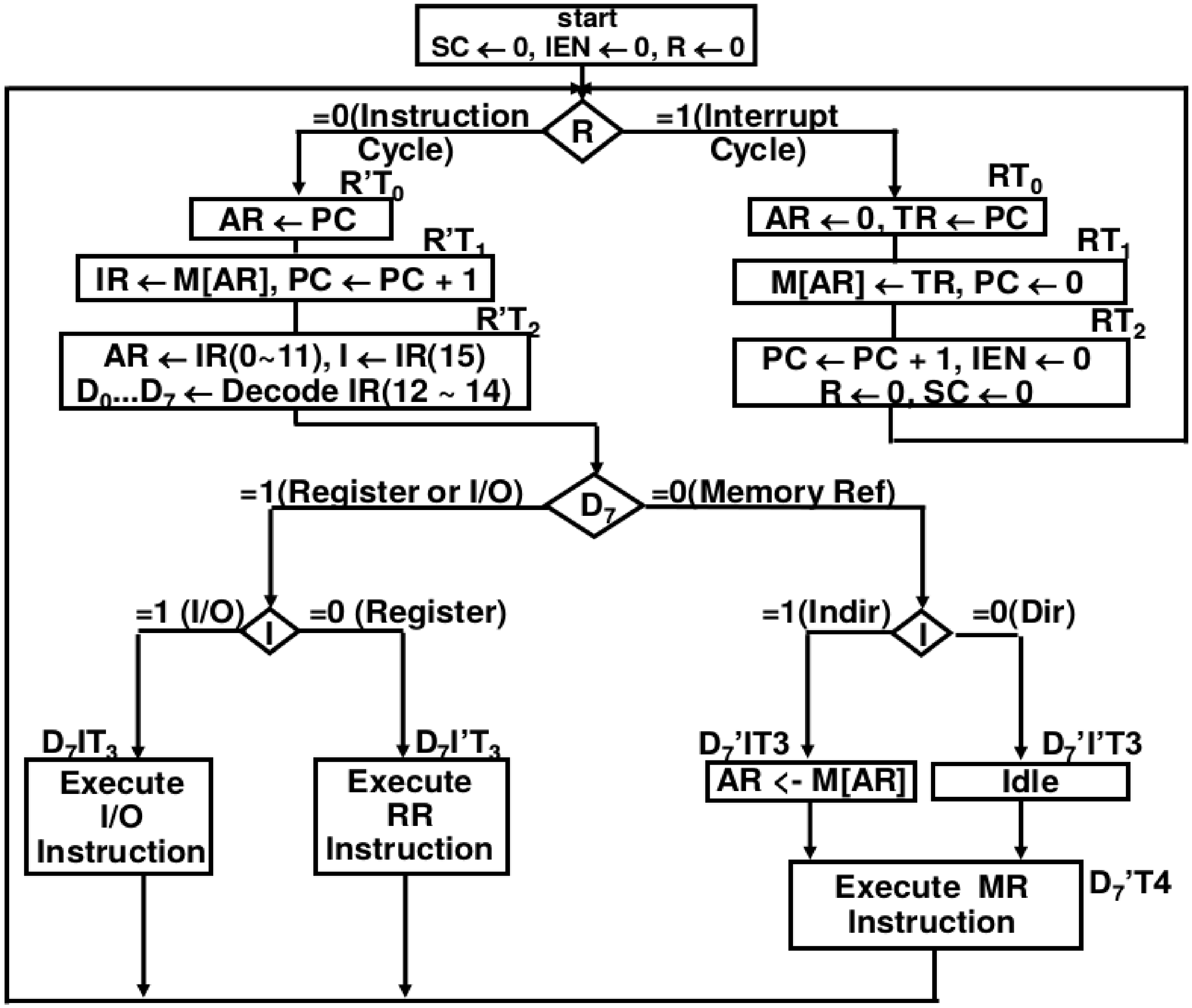
- The condition for R = 1: $T_0'T_1'T_2'(IEN)(FGI + FGO)\text{:} \quad R \leftarrow 1$
- Modified Fetch and Decode Phase:

RT_0: $AR \leftarrow 0, \quad TR \leftarrow PC$
RT_1: $M[AR] \leftarrow TR, \quad PC \leftarrow 0$
RT_2: $PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0$

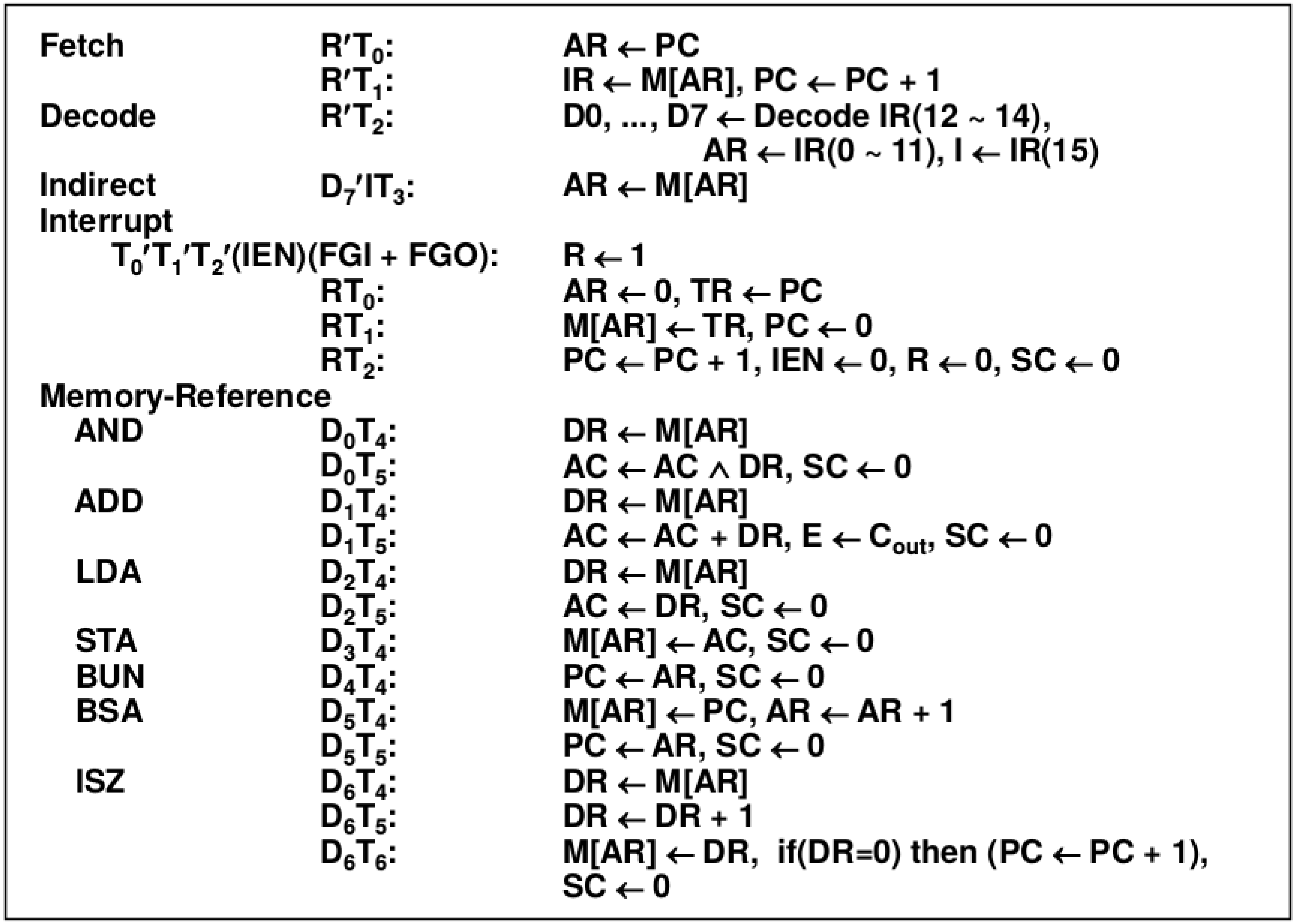
Figure 5-14 Demonstration of the interrupt cycle.



Complete Computer Description Flowchart of Operations

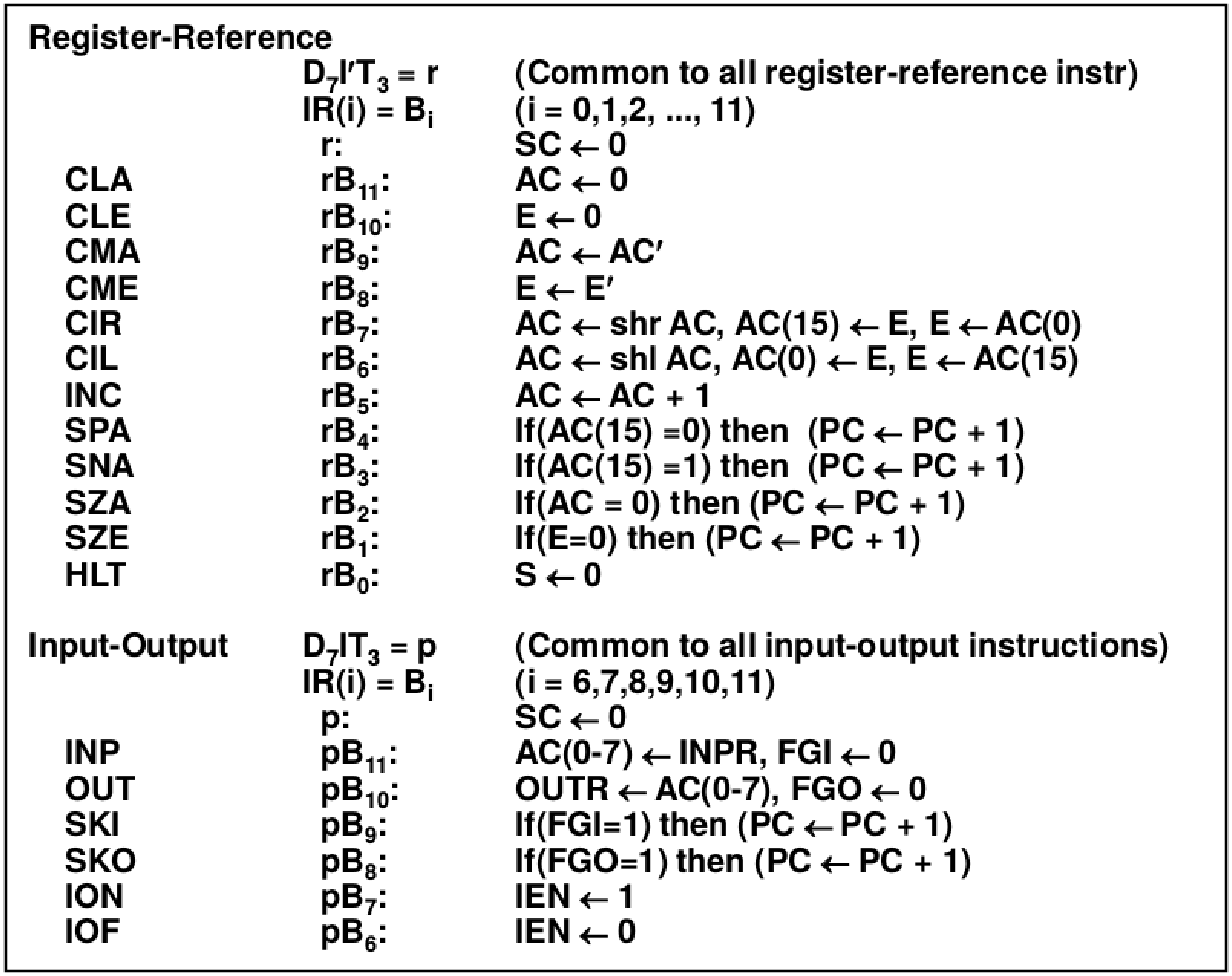


Control Functions and Microoperations for the Basic Computer



Fetch	$R'T_0$:	AR ← PC
	$R'T_1$:	IR ← M[AR], PC ← PC + 1
Decode	$R'T_2$:	D0, ..., D7 ← Decode IR(12 ~ 14), AR ← IR(0 ~ 11), I ← IR(15)
Indirect	$D_7'IT_3$:	AR ← M[AR]
Interrupt		
	$T_0'T_1'T_2'$(IEN)(FGI + FGO):	R ← 1
	RT_0:	AR ← 0, TR ← PC
	RT_1:	M[AR] ← TR, PC ← 0
	RT_2:	PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0
Memory-Reference		
AND	D_0T_4:	DR ← M[AR]
	D_0T_5:	AC ← AC $\wedge$ DR, SC ← 0
ADD	D_1T_4:	DR ← M[AR]
	D_1T_5:	AC ← AC + DR, E ← C_{out}, SC ← 0
LDA	D_2T_4:	DR ← M[AR]
	D_2T_5:	AC ← DR, SC ← 0
STA	D_3T_4:	M[AR] ← AC, SC ← 0
BUN	D_4T_4:	PC ← AR, SC ← 0
BSA	D_5T_4:	M[AR] ← PC, AR ← AR + 1
	D_5T_5:	PC ← AR, SC ← 0
ISZ	D_6T_4:	DR ← M[AR]
	D_6T_5:	DR ← DR + 1
	D_6T_6:	M[AR] ← DR, if(DR=0) then (PC ← PC + 1), SC ← 0

Control Functions and Microoperations for the Basic Computer



Register-Reference		
	$D_7I'T_3 = r$	(Common to all register-reference instr)
	$IR(i) = B_i$	(i = 0,1,2, ..., 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If($AC(15)$ =0) then ($PC \leftarrow PC + 1$)
SNA	rB_3:	If($AC(15)$ =1) then ($PC \leftarrow PC + 1$)
SZA	rB_2:	If($AC = 0$) then ($PC \leftarrow PC + 1$)
SZE	rB_1:	If($E=0$) then ($PC \leftarrow PC + 1$)
HLT	rB_0:	$S \leftarrow 0$
Input-Output	$D_7IT_3 = p$	(Common to all input-output instructions)
	$IR(i) = B_i$	(i = 6,7,8,9,10,11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If($FGI=1$) then ($PC \leftarrow PC + 1$)
SKO	pB_8:	If($FGO=1$) then ($PC \leftarrow PC + 1$)
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

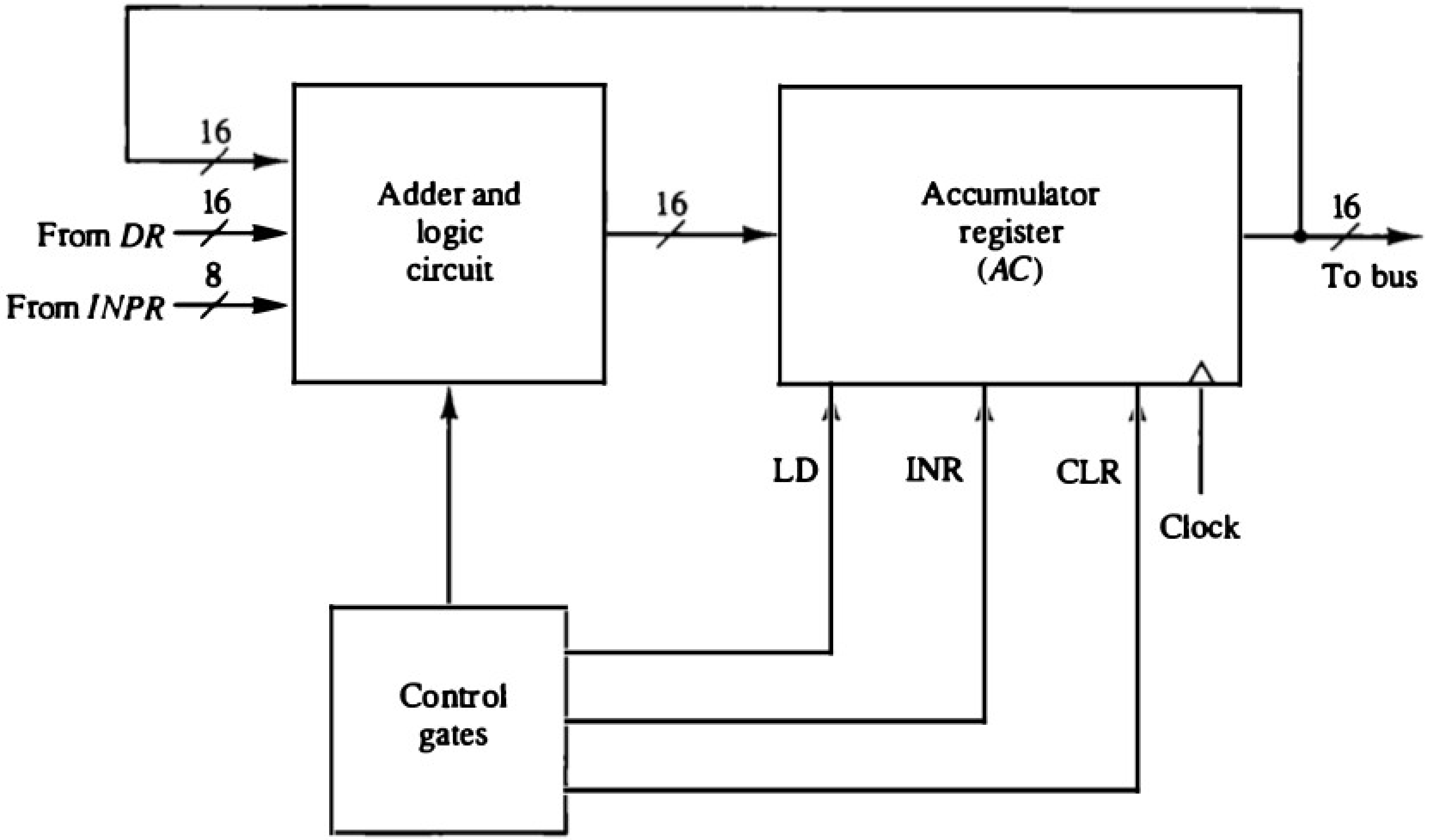
Design of Accumulator Logic

- The circuits associated with the AC register are shown in Fig.
- The adder and logic circuit has three sets of inputs.
    - One set of 16 inputs comes from the outputs of AC .
    - Another set of 16 inputs comes from the data register DR .
    - A third set of eight inputs comes from the input register INPR .

- The outputs of the adder and logic circuit provide the data inputs for the AC register.
- In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the AC register and for controlling the operation of the adder and logic circuit.

Circuit of Accumulator Logic

Figure 5-19 Circuits associated with AC.

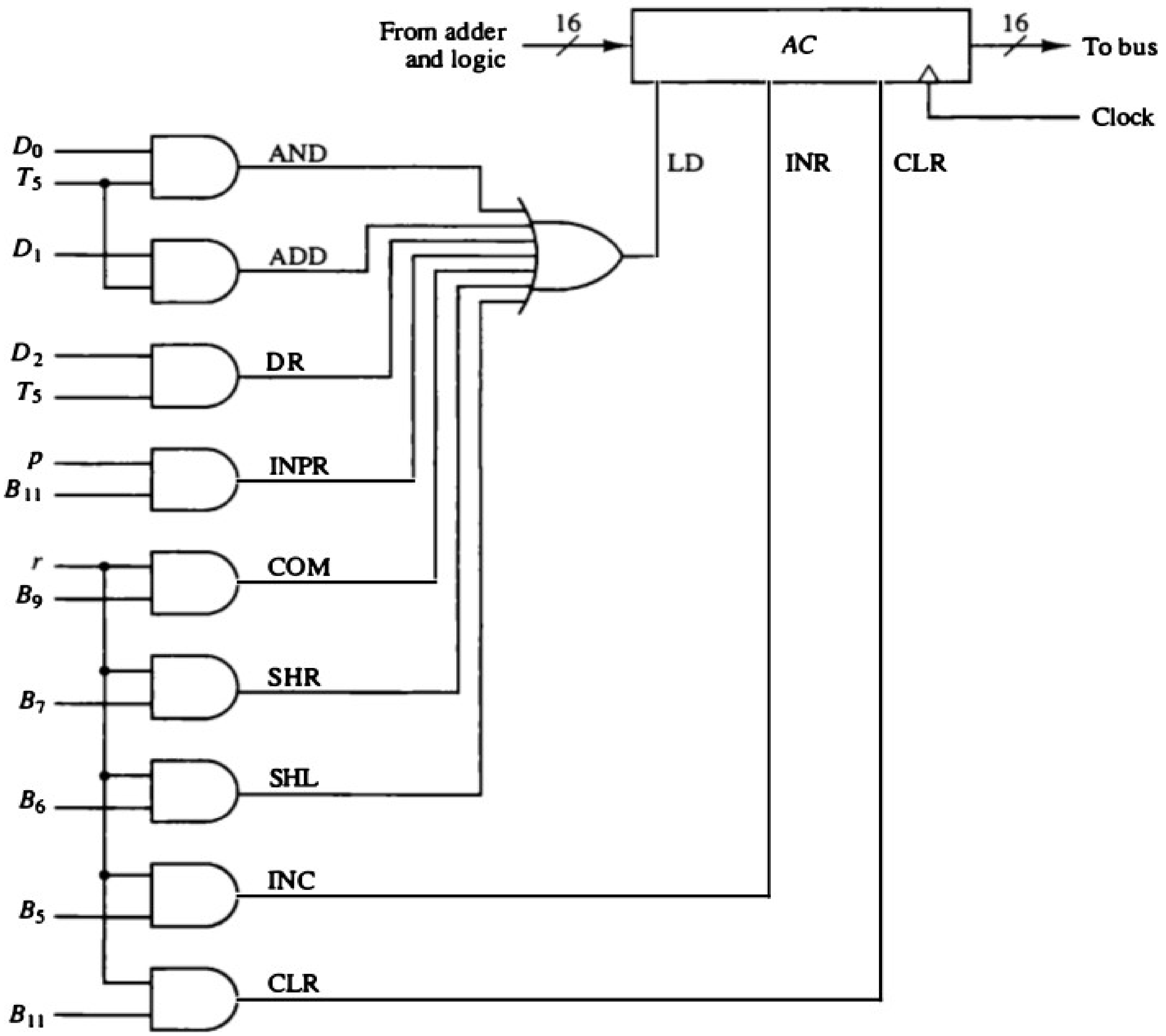


Control of AC Register

- The gate structure that controls the LD, INR, and CLR inputs of AC is shown in Fig.
- In order to design the logic associated with AC, it is necessary to go over the register transfer statements that change the content of AC. The gate configuration is derived from the control functions in the following list:

D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$	Add with *DR*
D_2T_5:	$AC \leftarrow DR$	Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

Figure 5-20 Gate structure for controlling the LD, INR, and CLR of AC.


From adder and logic
16
AC
16
To bus
Clock
LD
INR
CLR
D_0
T_5
AND
D_1
ADD
D_2
T_5
DR
p
B_{11}
INPR
r
B_9
COM
B_7
SHR
B_6
SHL
B_5
INC
B_{11}
CLR