Basic Computer Organization

In this lecture, we will study

i. Computer Instructions, Instruction Codes, and Instruction Cycles

ii. Timing and Control

iii. Memory-Reference Instructions

iv. Input/Output and Interrupts

v. Complete Computer Description and Design of a Basic Computer

Chapter Exercises

Computer Instructions and Instruction Codes

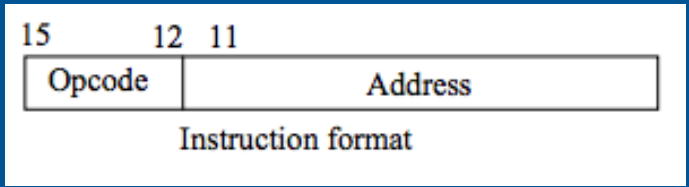
Computer Instructions

A **computer instruction** is a binary code that specifies a *sequence of microoperations* for the computer.

Instruction Codes

An **instruction code** is group of bits that instruct the computer to perform a *specific operation*.

An instruction code is usually divided into different parts, and each part has its own particular interpretation.



Instruction Codes

The most basic part of an instruction code is its OPERATION CODE (or OPCODE); it is a group of bits that define operations such as add, subtract, multiply, shift, and complement.

For instance, the ADD operation can be assigned the opcode 110010. When the computer sees this opcode, it automatically knows to add the operands.

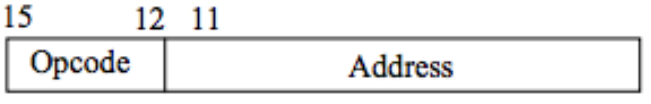
Stored Program Organization

The simplest way to organize a computer is to have
one **processsor register** & an **instruction code format with two parts**.

The first part specifies the operation to be performed
and the second part specifies the address of the required operand.

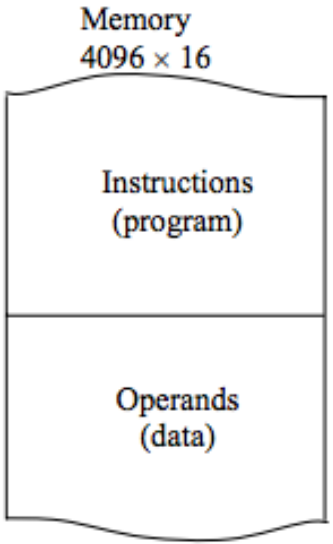
The operand is read from the memory and is then operated on
together with the data stored in the processor register.

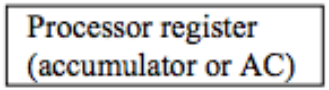
Stored Program Organization



Instruction format

For operations such as 'clear AC' or 'complement AC' (that do not require an operand from the memory, the instruction only has one part.





For a memory unit with 4096 words, we need 12 bits to specify an address (since $2^{12} = 4096$).

If we store each instruction in a 16-bit memory word, then we have 4 bits available for the operation code (opcode).With a 4-bit opcode, we will be able to specify a total of $2^4 = 16$ distinct operations.

Addressing Modes

The first part of an instruction i.e., the opcode, specifies a particular operation to be performed. This operation has to be performed on some operand i.e., data, stored in a computer register or in the main memory.

The way any operand is selected during program execution is dependent on the **addressing mode** being used in the instruction. The three addressing modes we will discuss at this point are:

> Immediate

> Direct

> Indirect

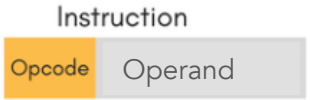
Addressing Modes

Immediate Addressing: When the second part of the instruction code specifies the required operand, the instruction is referred to as an "Immediate Instruction" (because it has an *immediate operand*).

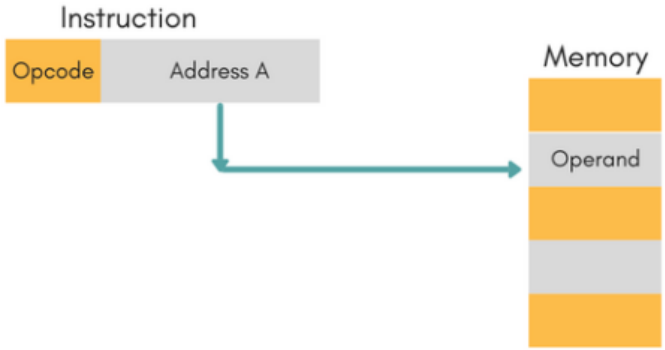
Direct Addresssing: When the second part of the instruction code specifies the address of the required operand, the instruction is said to have a "Direct Address" (because the *address of the operand is directly present in the instruction itself*).

Indirect Addressing: When the second part of the instruction code contains not the address of the operand but rather the address of the memory word in which the address of the operand is present, the instruction is said to have an "Indirect Address".

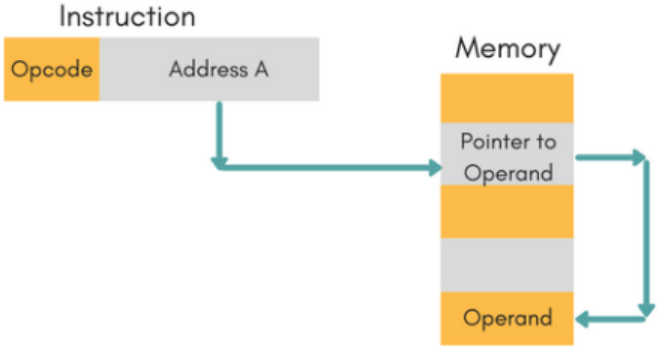
Immediate Addressing



Direct Addressing



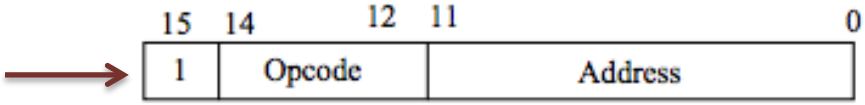
Indirect Addressing



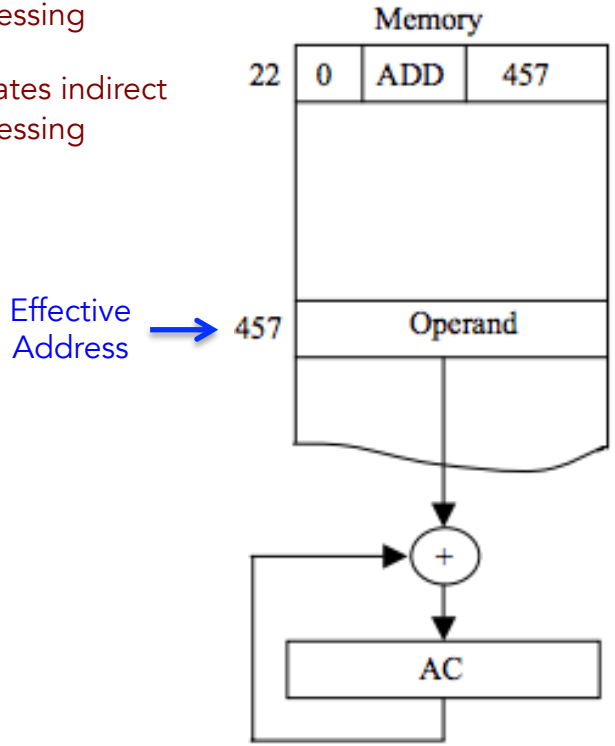
The leftmost bit (I) signifies the addressing mode.

I = 0 indicates direct addressing

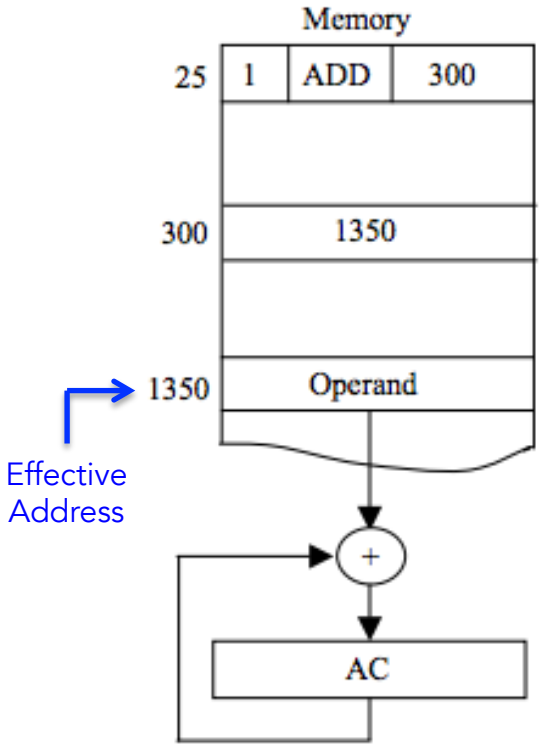
I = 1 indicates indirect addressing



(a) Instruction format



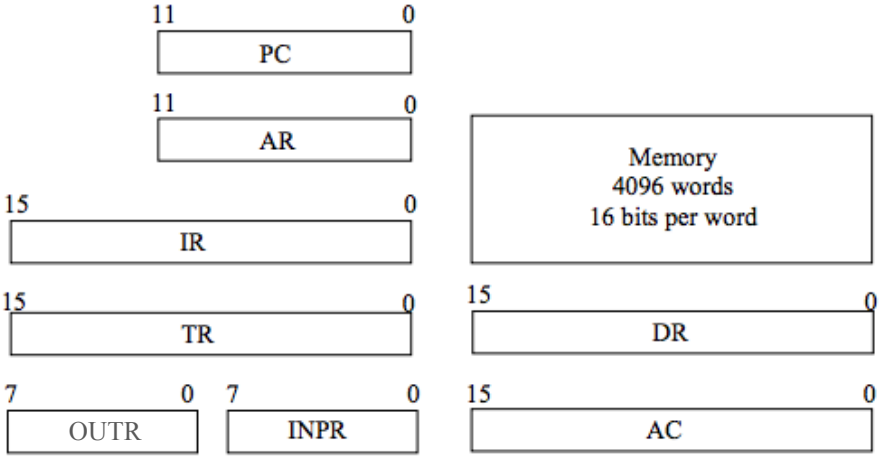
(b) Direct address



(c) Indirect address

Computer Registers

Computer Registers



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

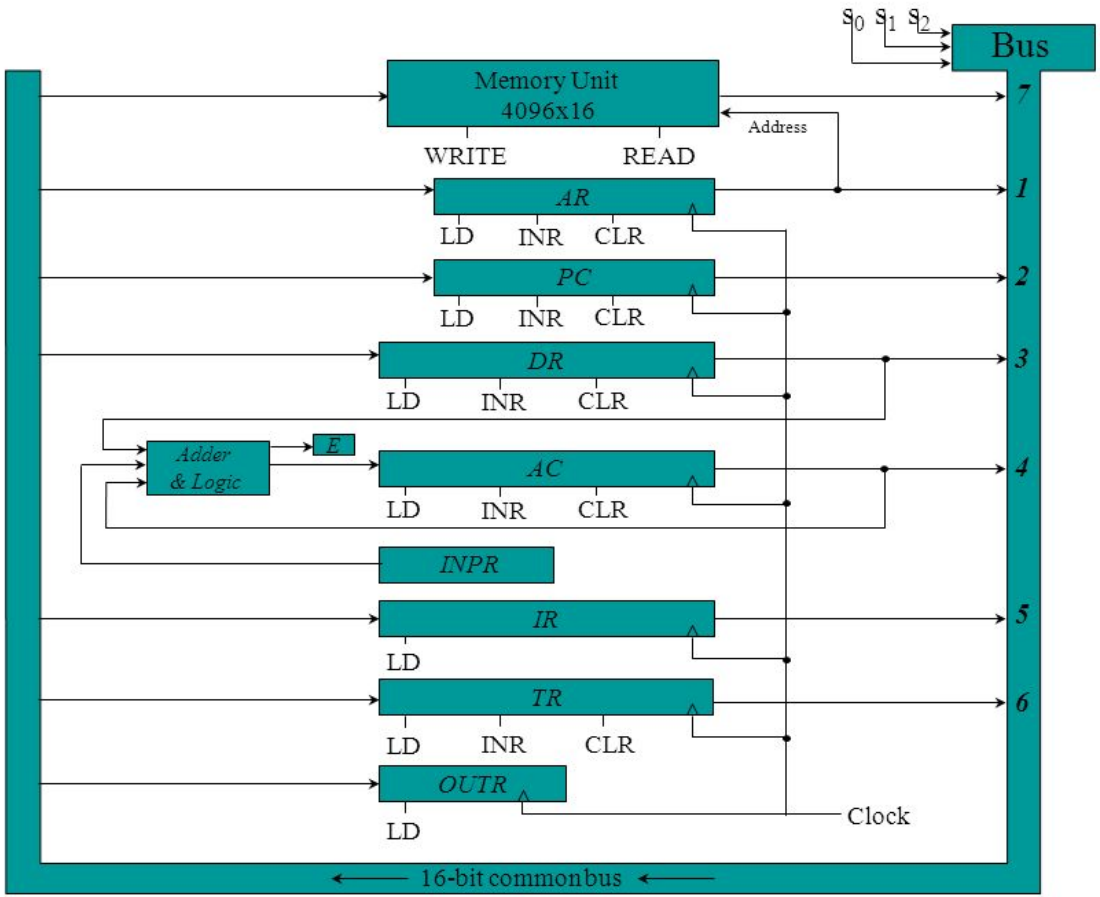
Common Bus System

A basic computer has eight registers, a memory unit, and a control unit; paths must be provided to transfer information from one unit to another.

If separate lines of communication are used between each unit, the number of wires will be excessive.

It is more efficient to use a COMMON BUS SYSTEM.

Common Bus System

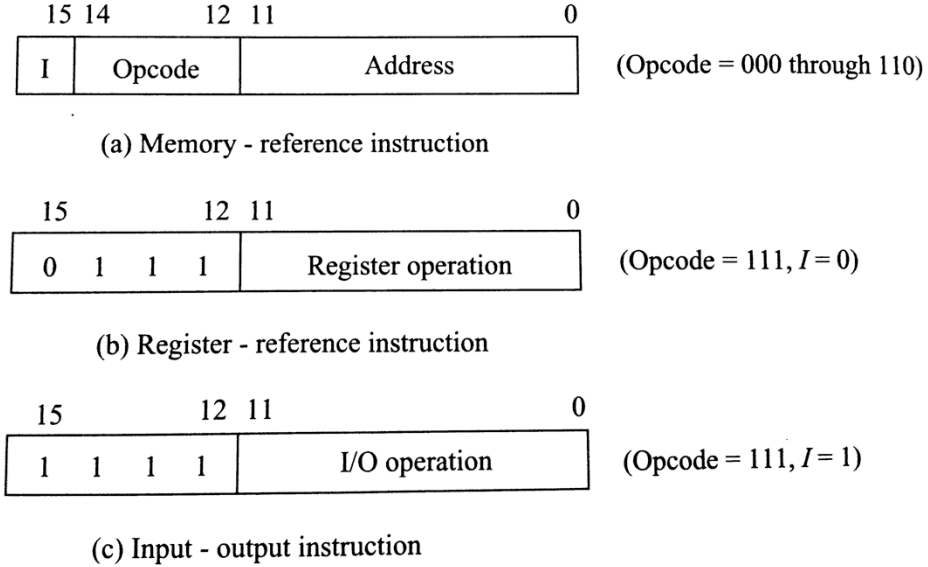


Computer Instructions

Computer Instructions

A basic computer has three instruction code formats:

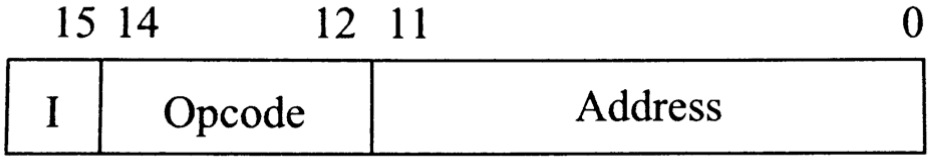
> Memory-Reference Instruction Format
> Register-Reference Instruction Format
> Input-Output Instruction Format



Memory-Reference Instructions

A memory-reference instruction refers to a memory address as operands.

The other operand is always the accumulator.



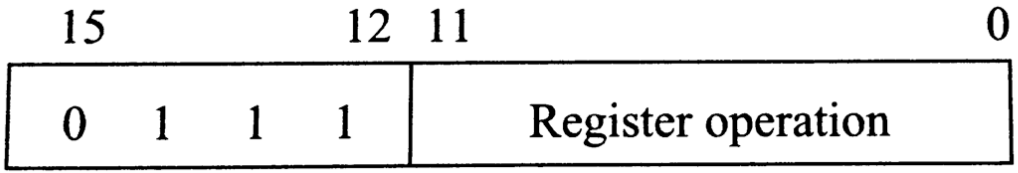
(Opcode = 000 through 110)

A memory-reference instruction uses 12 bits to specify the address of the operand and 1 bit to specify the addressing mode I; I=0 indicates direct addressing mode while I = 1 indicates indirect addressing mode.

The opcode in a memory-reference instruction can take any value from 000 to 110.

Register-Reference Instructions

Register-reference instructions perform operations on registers instead of memory.



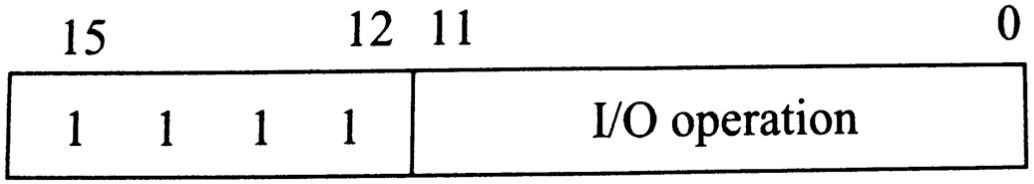
(Opcode = 111, $I = 0$)

A register-reference instruction is identified by the opcode 111, with a 0 in the leftmost bit (i.e., bit 15).

A register-reference instruction specifies an operation on or a test of the accumulator (AC). An operand from memory is not needed, and so the 12 bits (that are generally reserved for the address) are used to specify the register operation.

Input-Output Instruction

These instruction enable communication between the computer and the outside environment.



(Opcode = 111, $I = 1$)

An input-output instruction is identified by the opcode 111, with a 1 in the leftmost bit (i.e., bit 15).

A input-output instruction does not need an operand from the memory, and so the 12 bits (that are generally used to specify the address) are used to specify the input-output operation instead.

Basic Computer Instructions

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

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

Instruction Set Completeness

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

1. *Arithematic, logical, and shift instructions*
2. *Instructions for moving information to/from memory and processor registers*
3. *Program control and status checking instructions*
4. *Input and output instructions*

Timing and Control

Timing and Control

The timing for all registers in a basic computer is controlled by a **Master Clock Generator**.

The clock pulses are applied to all **flip-flops and registers in the system**, as well as, the **flip-flops and registers in the control unit**.

The clock pulses DO NOT change the state of a register **until the register is enabled by a control signal** generated in the control unit.

Two Major Types of Control Organizations

Hardwired Control

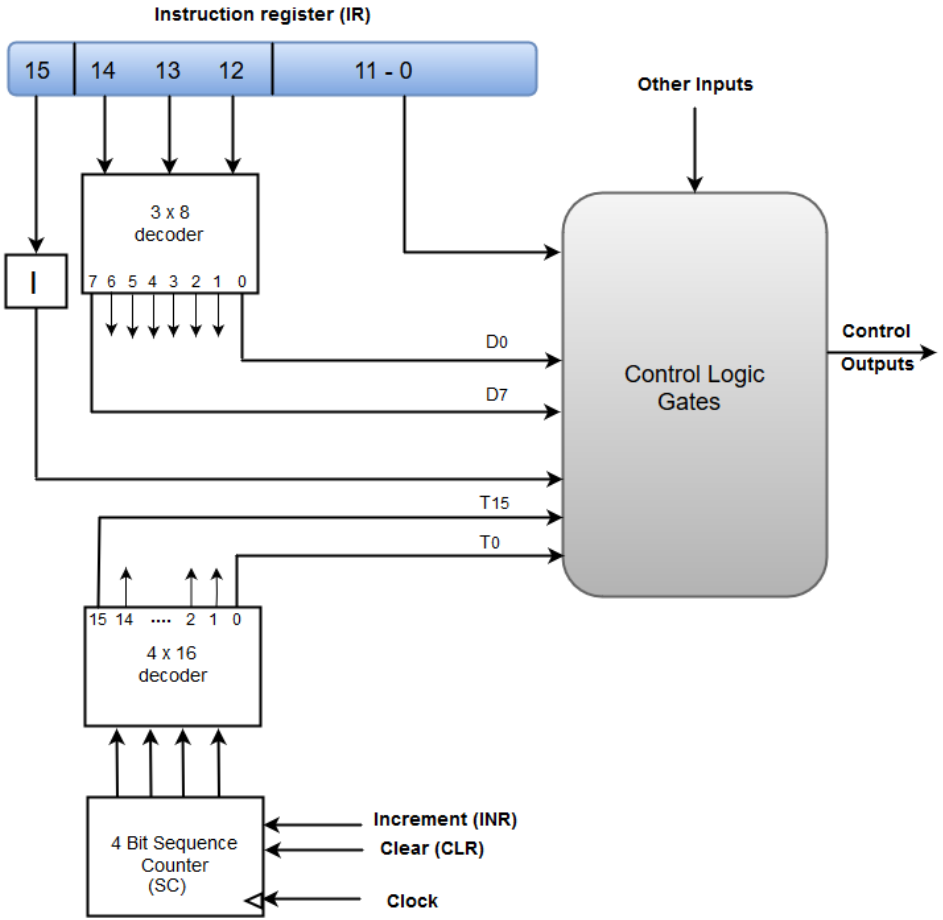
In this type of control organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.

It can be optimized to produce a fast mode of operation.

Microprogrammed Control

In this type of control organization, the control information is stored in a control memory. This memory is programmed to initiate the required sequence of microoperations.

Block Diagram of a Typical Control Unit



Block Diagram of a Typical Control Unit

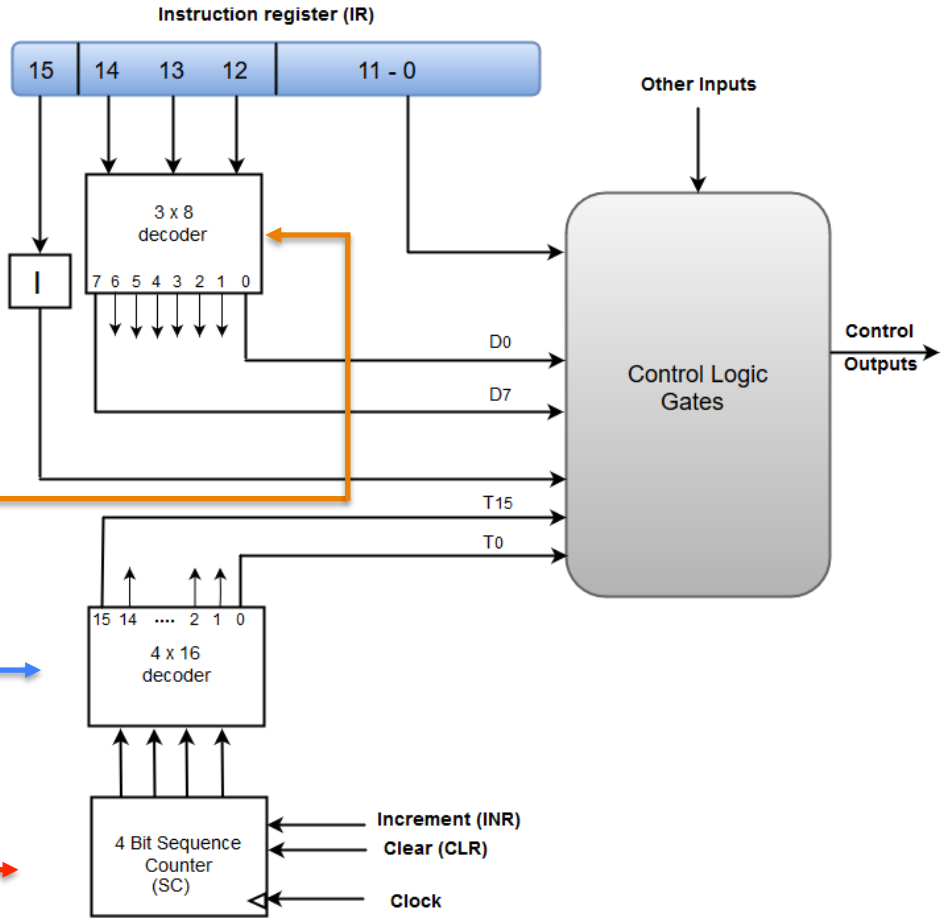
An instruction read from the memory is placed in the IR

I signifies the addressing mode
I = 0 :- direct addressing
I = 1 :- indirect addressing

This decoder decodes the opcode

This decoder decodes outputs of the SC into timing signals

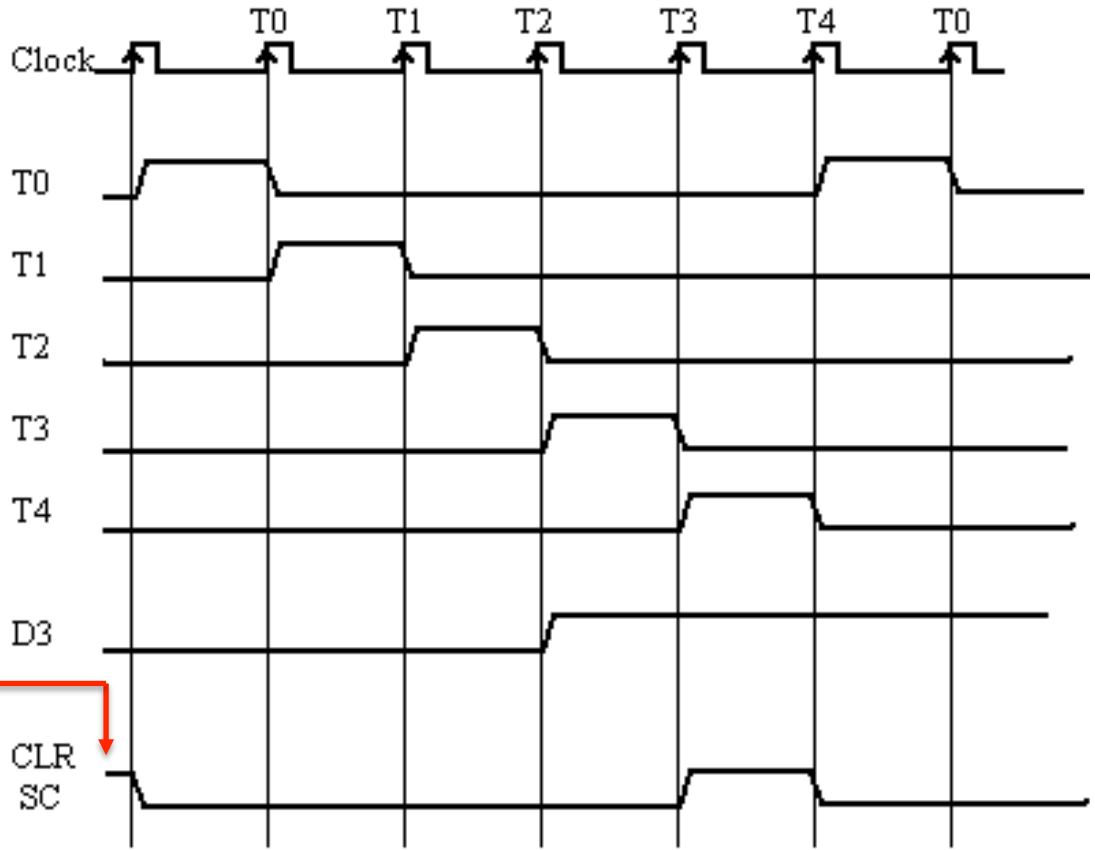
The 4-bit SC can count from 0 to15



Example of Control Timing Signals

The sequence counter SC responds to positive transitions of the clock.

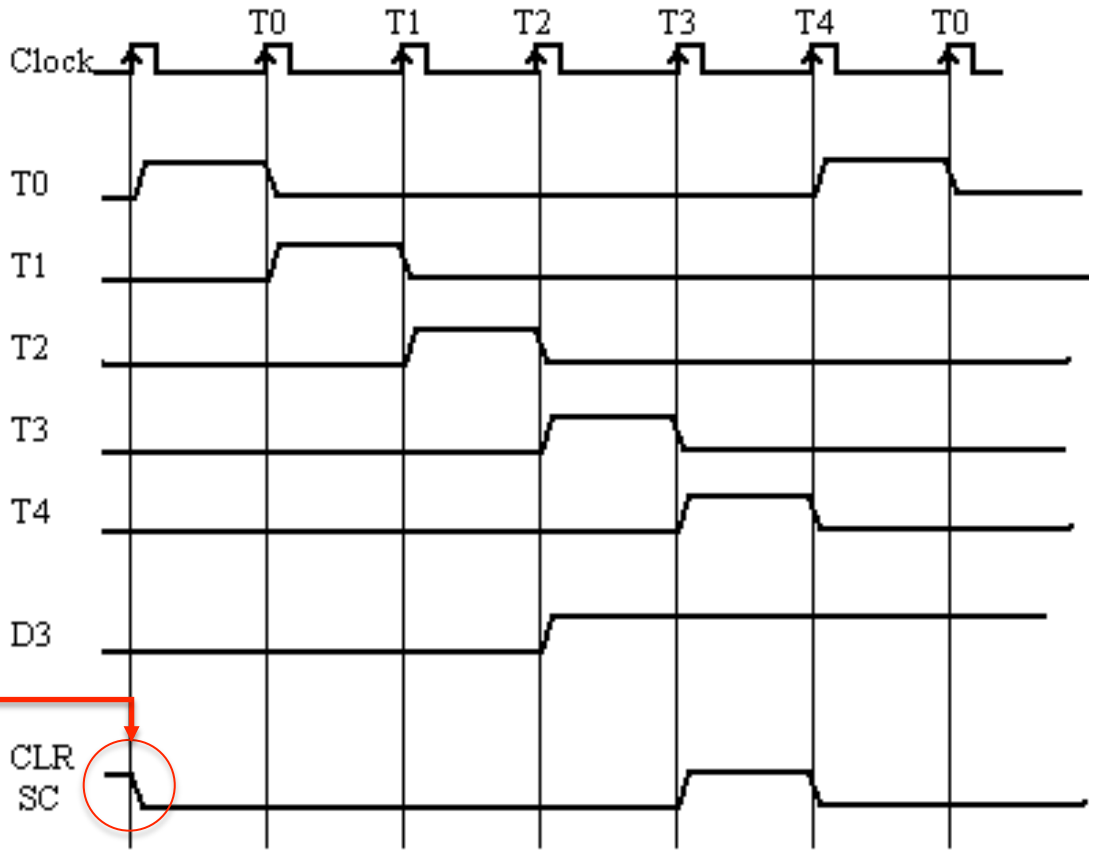
Initially, the CLR input of the SC is active.



Example of Control Timing Signals

Initially, the CLR input of the SC is active.

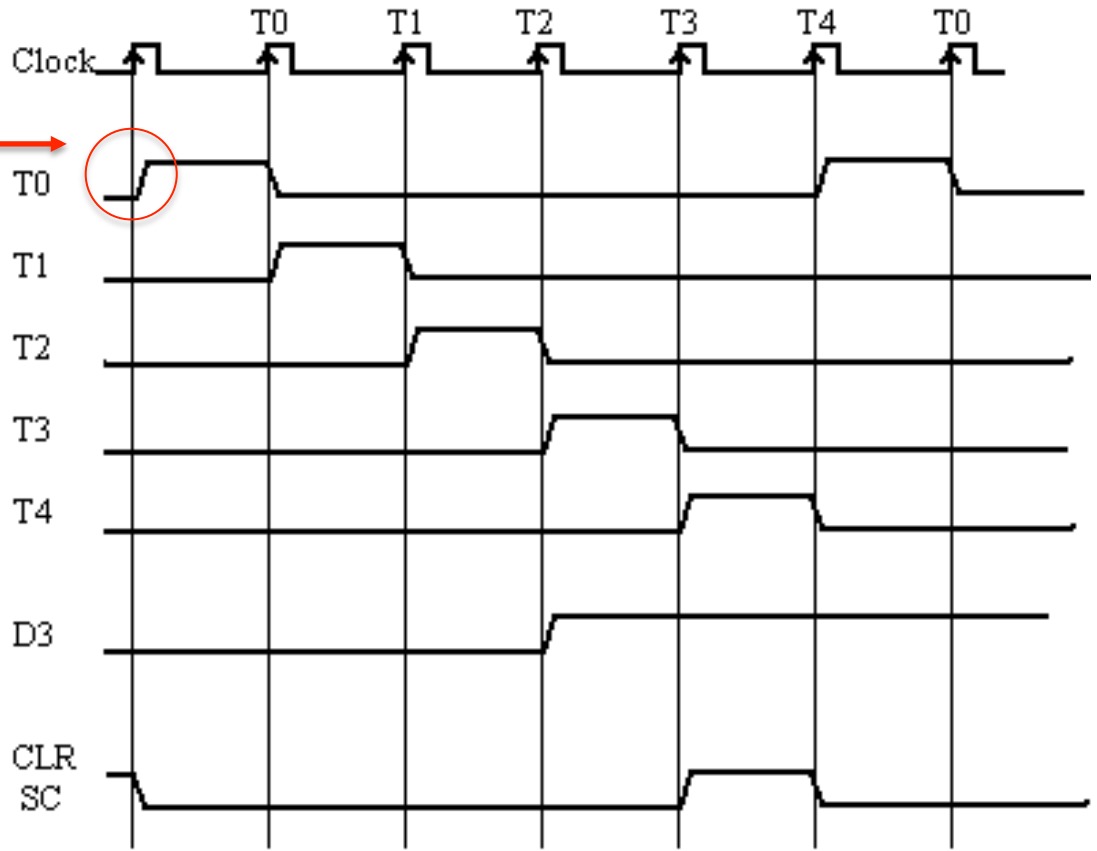
The first +ve transition of the clock clears SC to 0



Example of Control Timing Signals

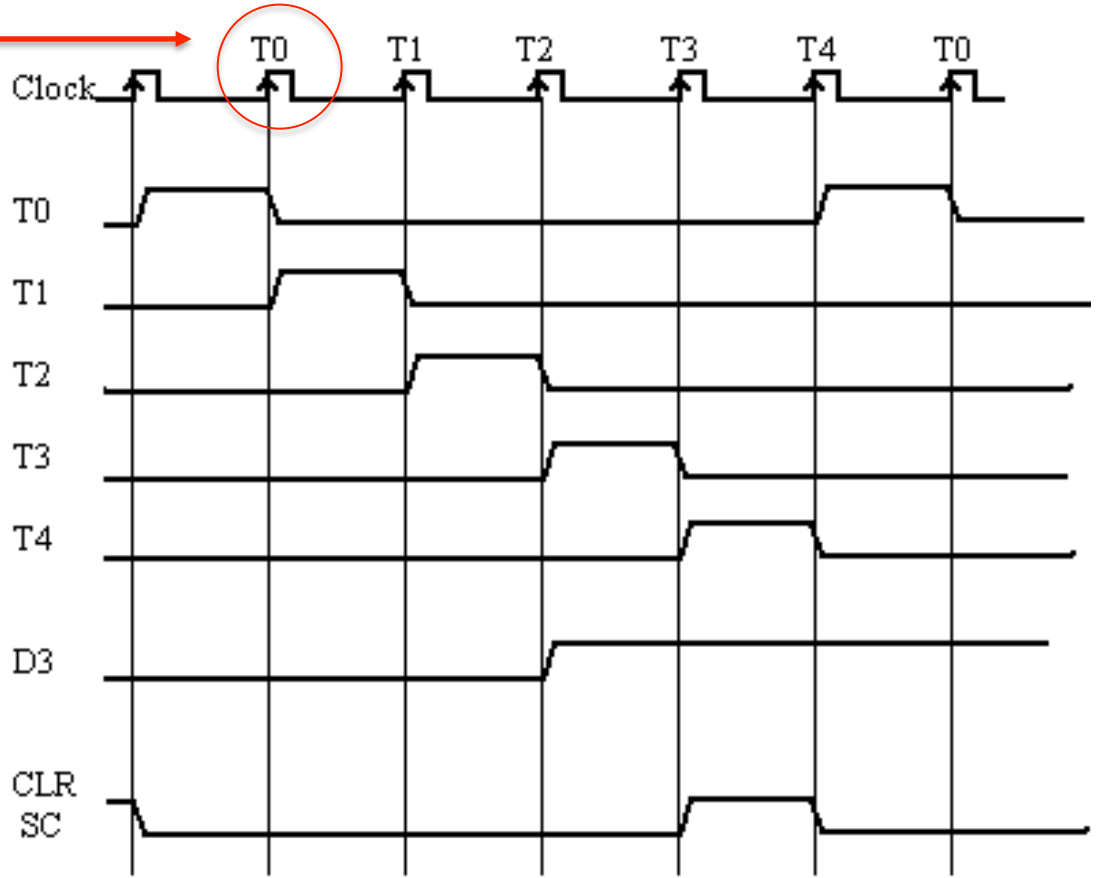
When SC clears to 0, it in turn activates the timing signal T0 out of the decoder.

T0 remains active during one clock cycle.



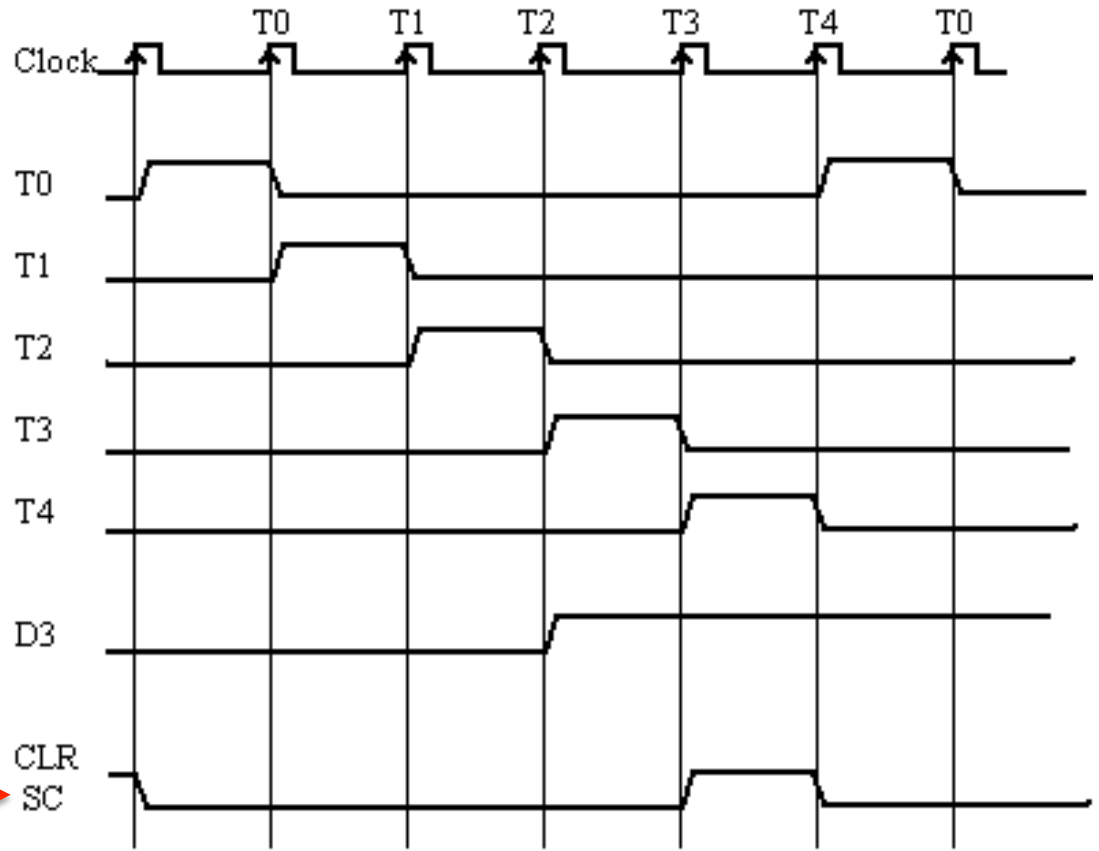
Example of Control Timing Signals

This positive clock transition will activate only those registers whose control inputs are connected to timing signal T0.

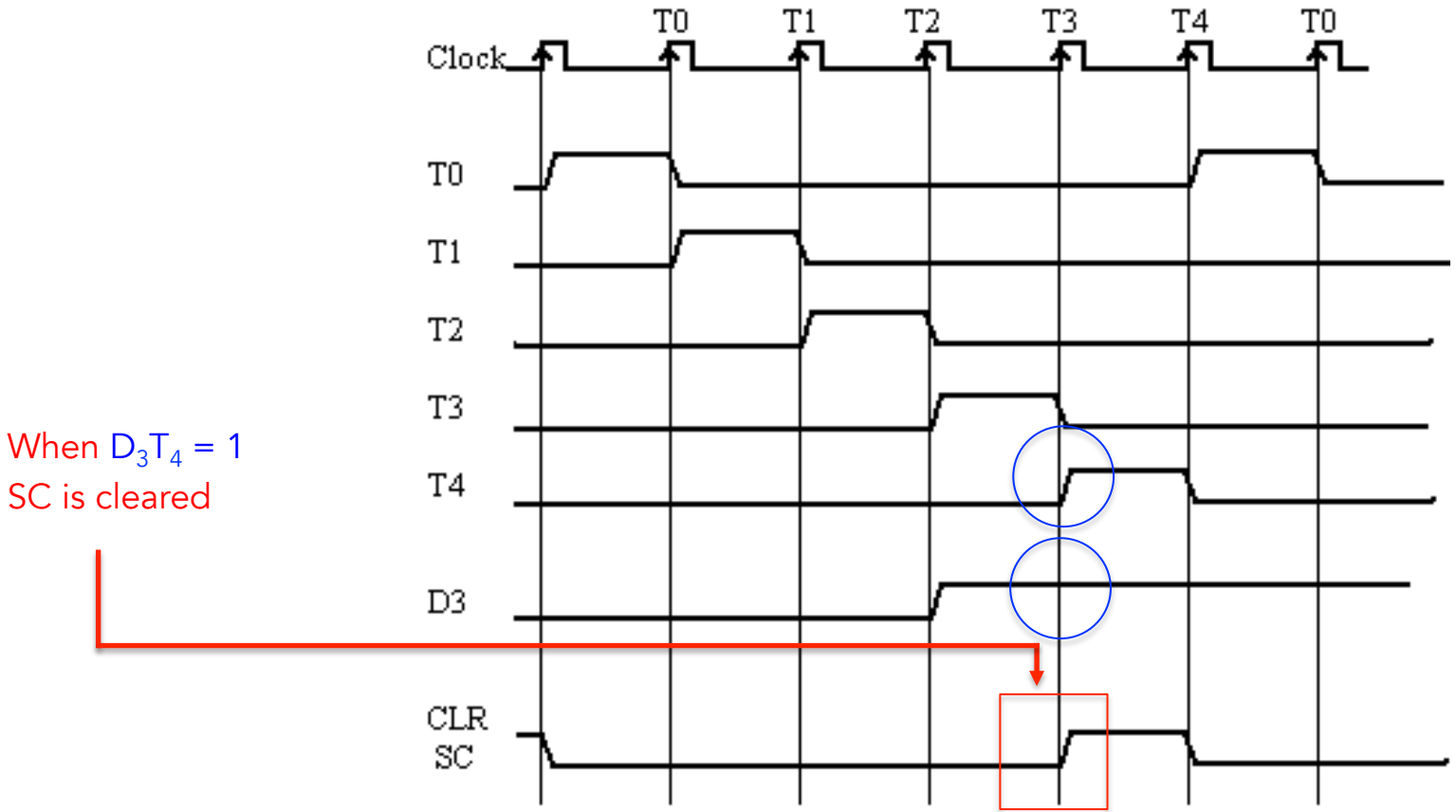


Example of Control Timing Signals

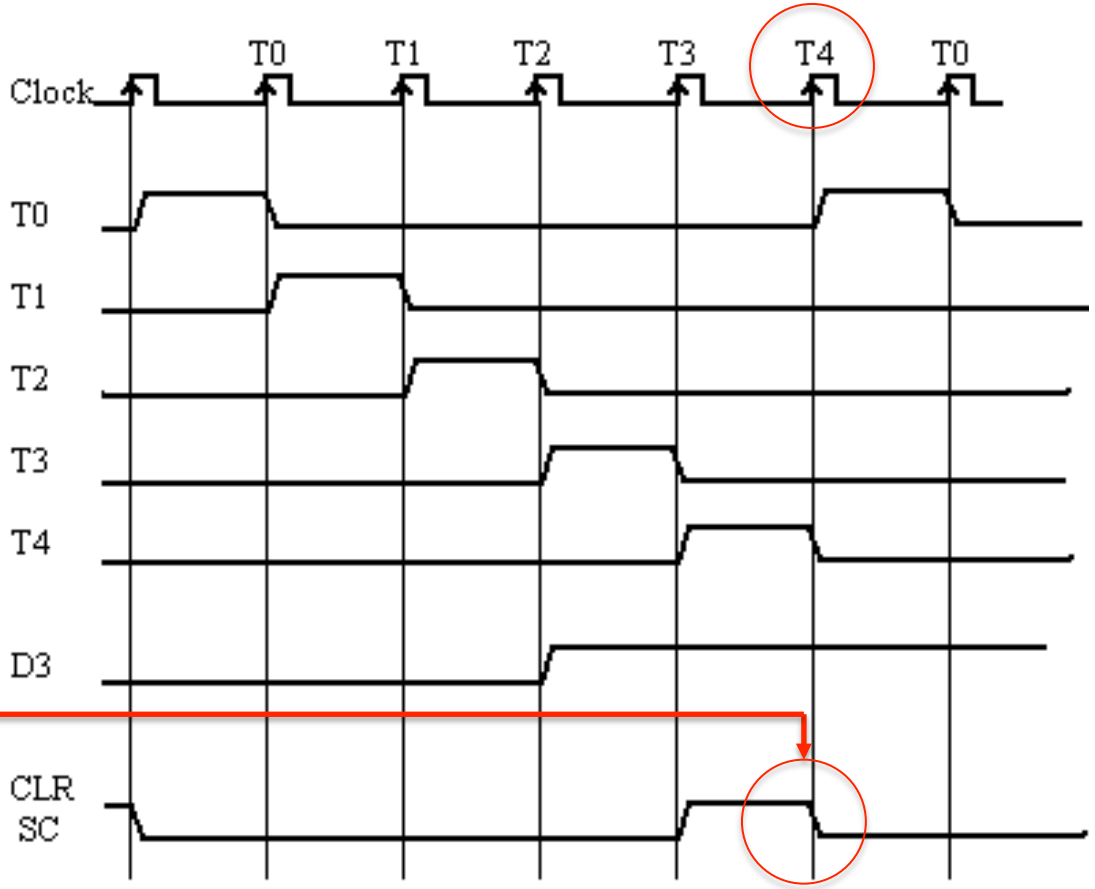
SC is incremented with every positive clock transition, unless its CLR input is active.



Example of Control Timing Signals



Example of Control Timing Signals



And so, at the next positive clock transition (T4), the counter is cleared.

Relationship Between *Data Transfer*, *Clock Transitions*, and *Timing Signals*

In order to fully comprehend the operation of a computer, it is crucial to understand the timing relationship between clock transitions and timing signals.

For instance, the register transfer statement

$$\mathbf{T_0 : AR \leftarrow PC}$$

specifies a transfer of contents of PC into AR **if the timing signal T_0 is active.**

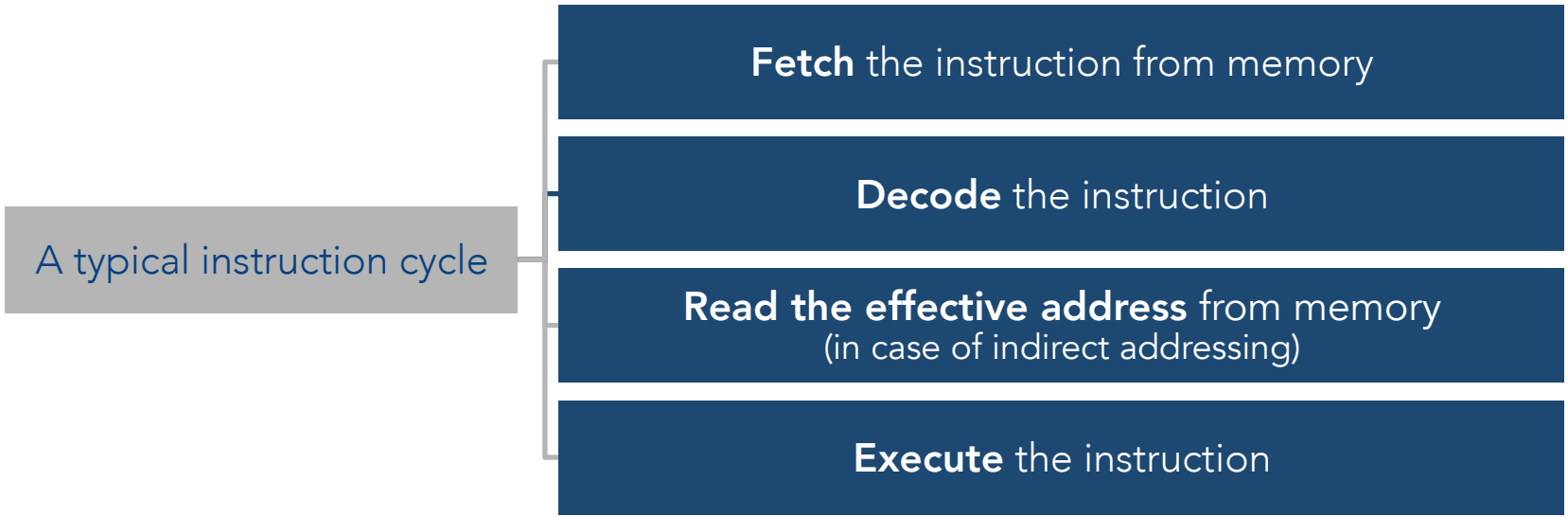
T_0 is active during an entire clock cycle interval; during this time, the contents of PC are placed on the bus and the 'load' input of the AR is enable.

But the actual transfer does not occur until the end of the clock cycle when the clock goes through the next positive transition.

Instruction Cycle

Instruction Cycle

A program residing in the memory is basically a sequence of instructions, and a specific cycle has to be followed for the execution of each instruction, as illustrated below:



Instruction Cycle: Fetch and Decode

Initially, the PC is loaded with the address of the first instruction in the program. The sequence counter (SC) is cleared to 0, providing a decoded timing signal T_0.
After each clock pulse, the SC is incremented by 1, so that the next timing signals T_1 , T_2, T_3 and so on can be generated.

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

T_0 : AR ← PC

T_1 : IR ← M[AR], PC ← PC+1

T_2 : D_0, D_1,....., D_7 ← Decode IR (12-14), AR ← IR (0-11), I ← IR(15)

Fetch and Decode: Microoperation 1

$$T_0 : AR \leftarrow PC$$

The address of the instruction is transferred from PC to AR.

Fetch and Decode: Microoperation 2

T_1 : IR ← M[AR], PC ← PC+1

The contents of the memory word M, whose address is present in AR, is moved to IR, and the PC is incremented by one (so that it now points to the next instruction to be fetched). This happens with the clock transition associated with the next timing signal, i.e., T_1.

Fetch and Decode: Microoperation 3

T_2 : D_0, D_1,....., D_7 ← Decode IR (12-14), AR ← IR (0-11), I ← IR(15)

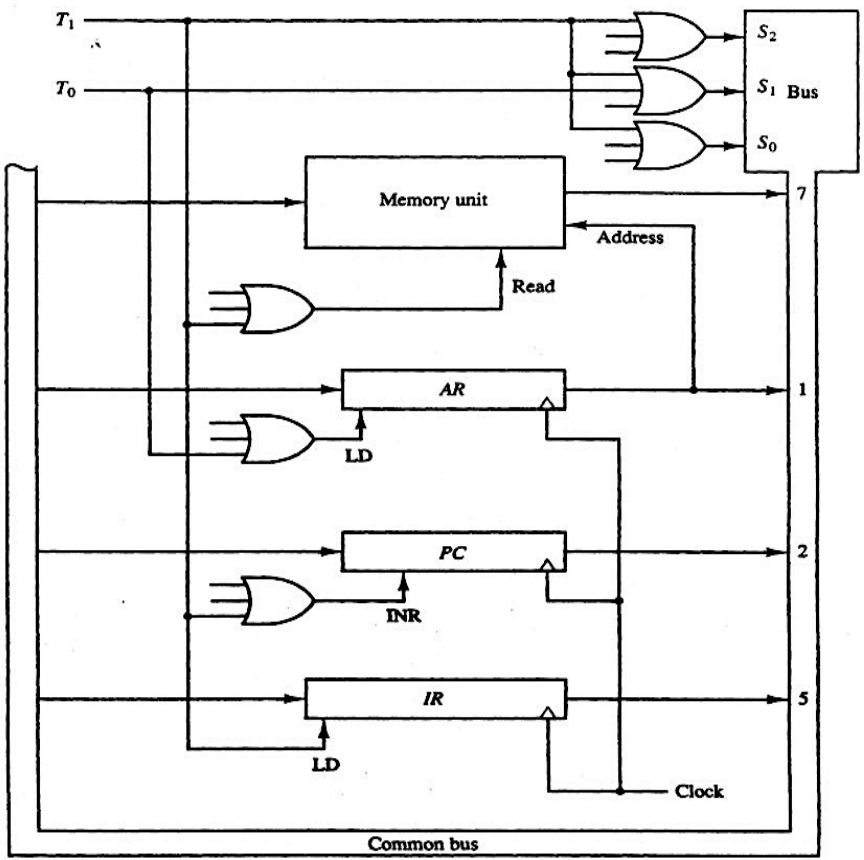
At time T_2 :

i) the opcode (which is present in bits 12, 13, and 14 of the IR) is decoded.

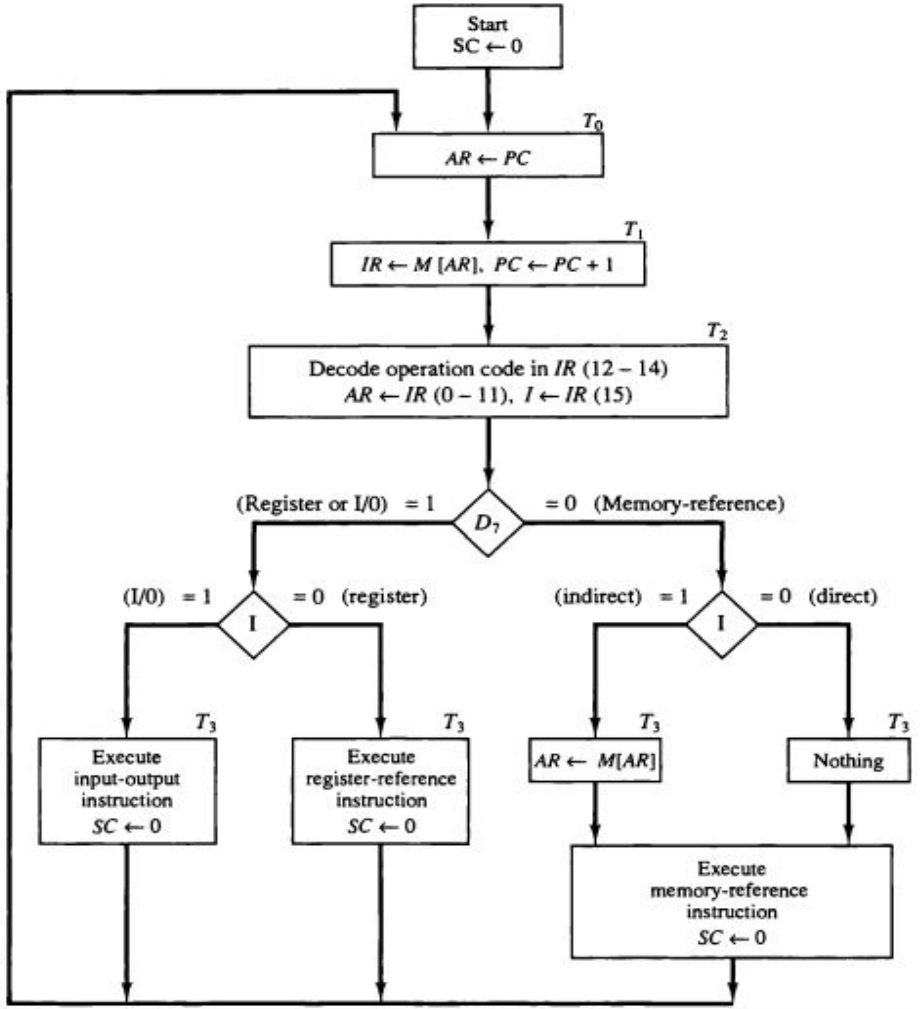
ii) the address of the operand (which is present in bits 0 – 11 of the IR) is transferred to AR.

iii) the single bit (bit 15 of the IR) which indicates the indirect addressing mode is transferred to flip-flop I.

Diagramatic Representation of The Fetch Phase



Flowchart for Instruction Cycle: Initial Configuration



Memory-Reference Instructions

Memory-Reference Instructions

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

Execution of a memory-reference instruction starts with the timing signal T_4.

Memory-Reference Instructions

1. AND to AC

$$AC \leftarrow AC \wedge M[AR]$$

This instruction performs the AND logic operation on pairs of bits in the AC (accumulator) and the memory word specified by the effective address present in AR. The result is then transferred to AC.

The microoperations that execute this instruction are:

$$D_0T_4: DR \leftarrow M[AR]$$

$$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$$

Memory-Reference Instructions

2. ADD to AC

$$AC \leftarrow AC + M[AR],\ E \leftarrow C_{out}$$

This instruction adds the contents of the memory word specified by the effective address to the value of AC. The sum is transferred to AC and the output carry is transferred to E (extended accumulator) flip-flop.

The microoperations needed to execute this instruction are:

$$D_1T_4: DR \leftarrow M[AR]$$
$$D_1T_5: AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$$

Memory-Reference Instructions

3. LDA: Load to AC

$$AC \leftarrow M[AR]$$

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

The microoperations needed to execute this instruction are:

$$D_2T_4: DR \leftarrow M[AR]$$
$$D_2T_5: AC \leftarrow DR, SC \leftarrow 0$$

Memory-Reference Instructions

4. STA: Store AC

$$M[AR] \leftarrow AC$$

This instruction stores the contents of AC into the memory word specified by the effective address.

The microoperation needed to execute this instruction is:

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

Memory-Reference Instructions

5. BUN: Branch Unconditionally

$$PC \leftarrow AR$$

This instruction transfers the control of the program to the instruction specified the effective address.

The microoperation needed to execute this instruction is:

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

Memory-Reference Instructions

6. BSA: Branch and Save Return Address

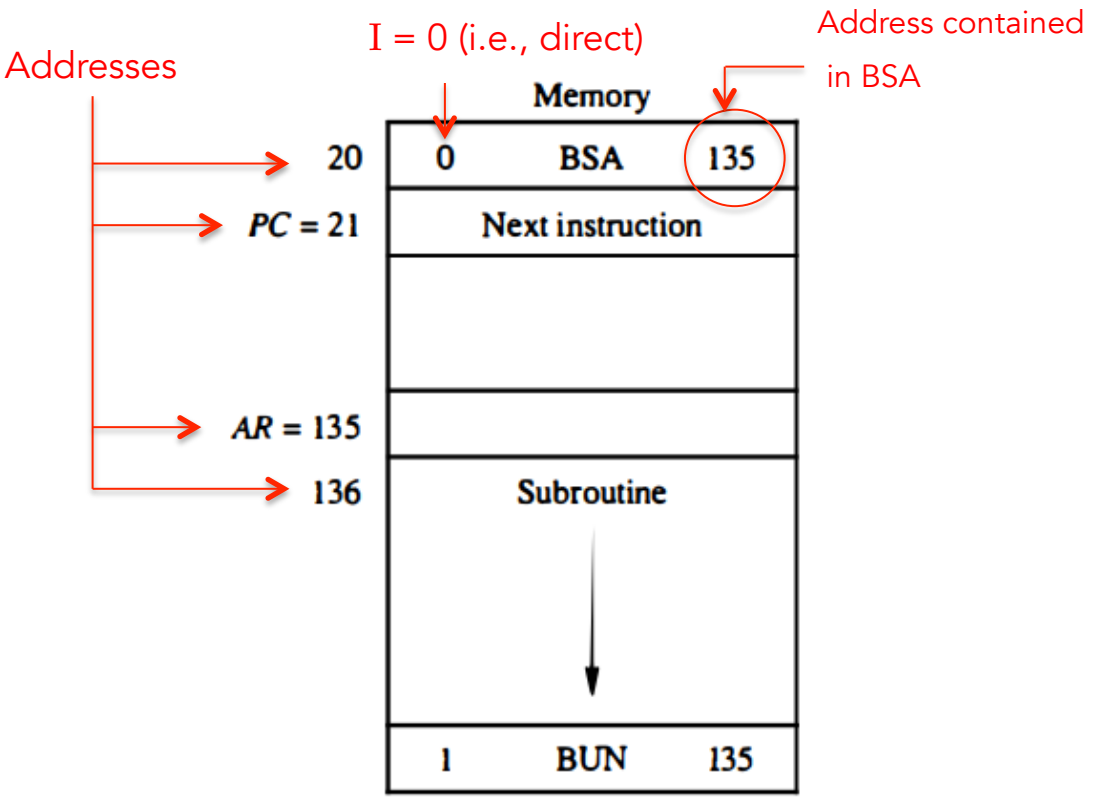
M[AR] ← PC, PC ← AR+1

This instruction is useful for branching to a subroutine or procedure.

When executed, this intruction stores the address of the next instruction into a memory location specified by the effective address. The effective address plus one is then added to PC (which will serve as the first address in the subroutine).

Example of BSA Instruction Execution

Memory, PC, and AR
Before Execution of BSA



Example of BSA Instruction Execution

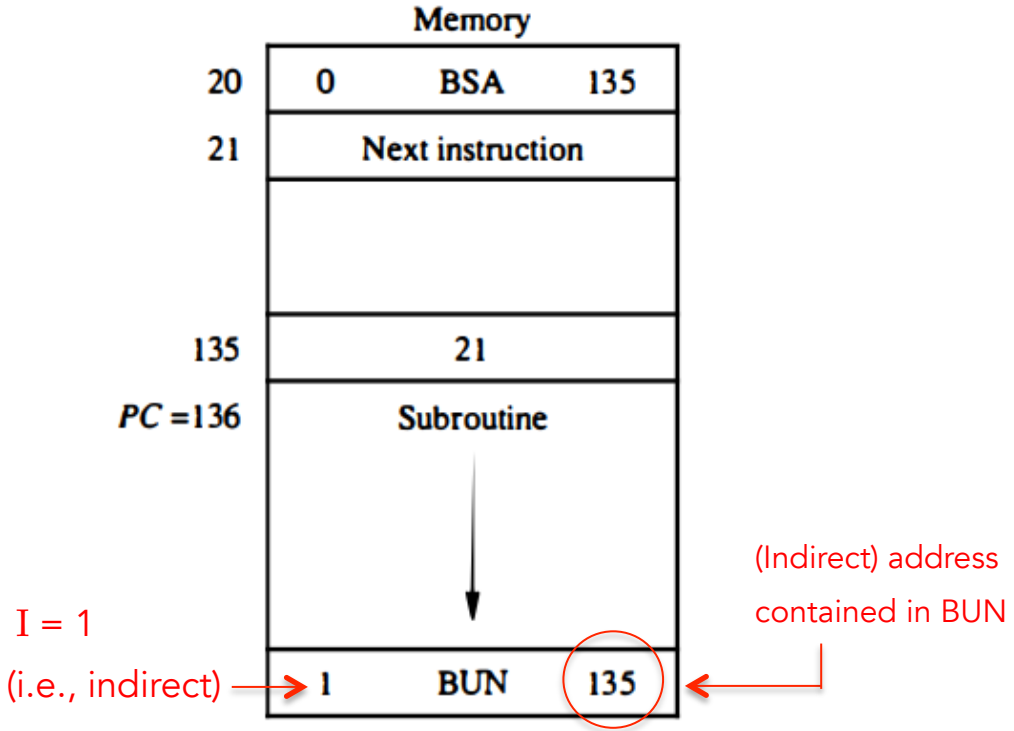
During Execution:

M[AR] ← PC, PC ← AR+1

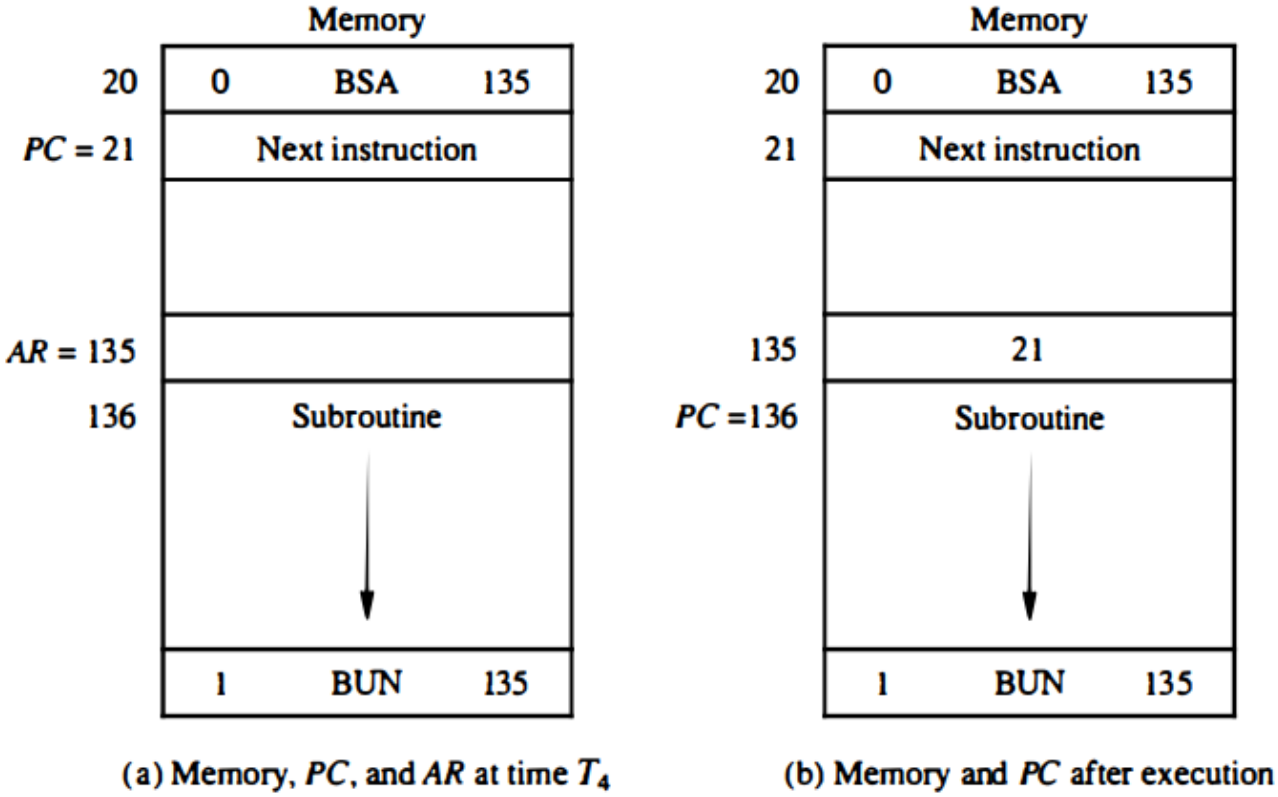
i.e.,

M[135] ← 21, PC ←135+1 = 136

Memory and PC After Execution of BSA



Example of BSA Instruction Execution



Memory-Reference Instructions

6. BSA: Branch and Save Return Address

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

The microoperations needed to execute the BSA instruction are:

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

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

Memory-Reference Instructions

7. ISZ: Increment and Skip if Zero (this is the longest instruction)

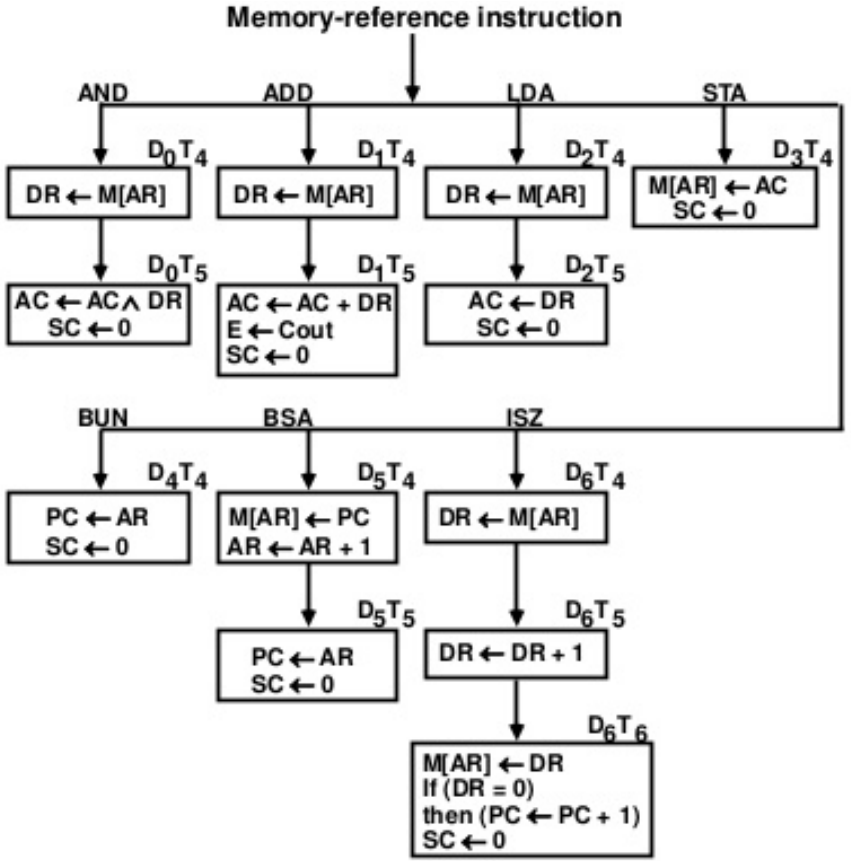
$$M[AR] \leftarrow M[AR] + 1$$
$$\text{If } M[AR] + 1 = 0 \text{ then } PC \leftarrow PC + 1$$

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 microoperations needed to execute this instruction are:

$$D_6T_4: DR \leftarrow M[AR]$$
$$D_6T_5: DR \leftarrow DR + 1$$
$$D_5T_6: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC+1), SC \leftarrow 0$$

Control Flowchart of Memory-Reference Instructions



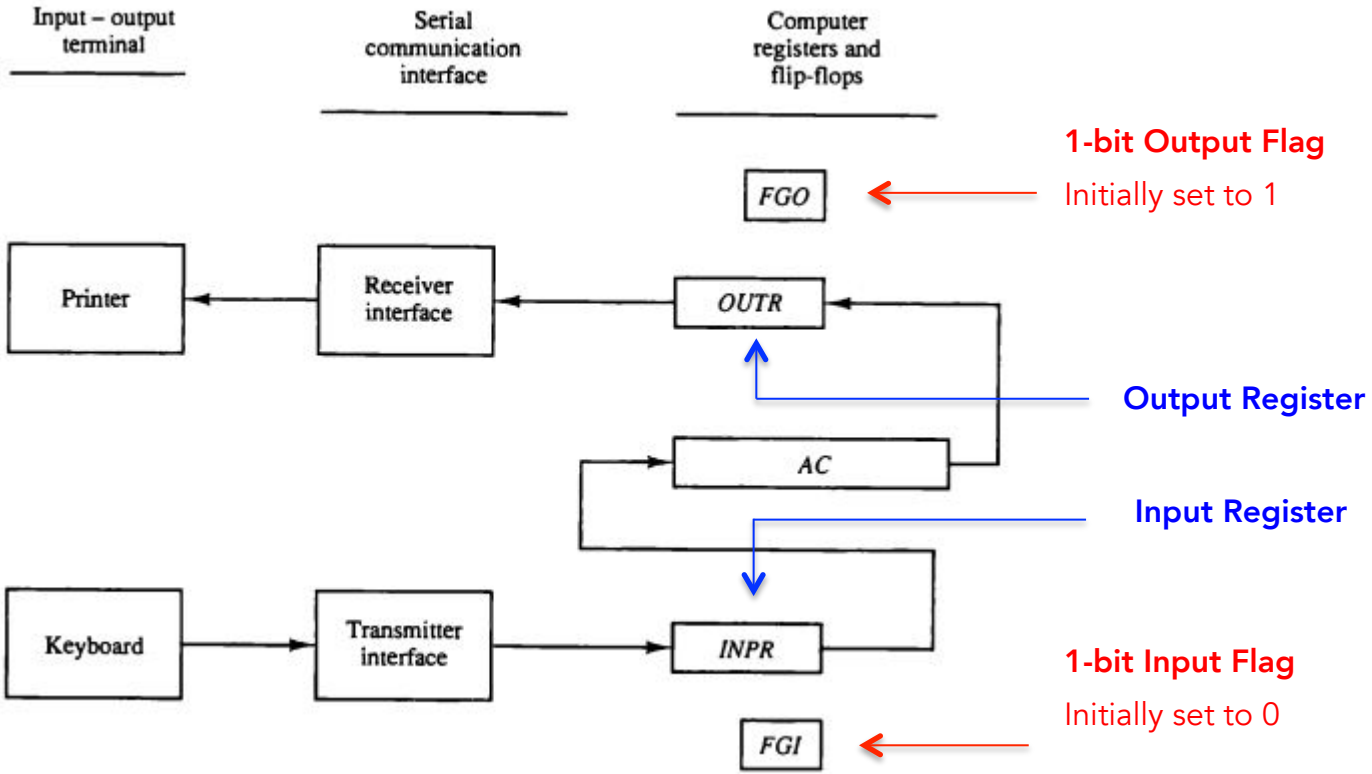
Input and Output

Input and Output

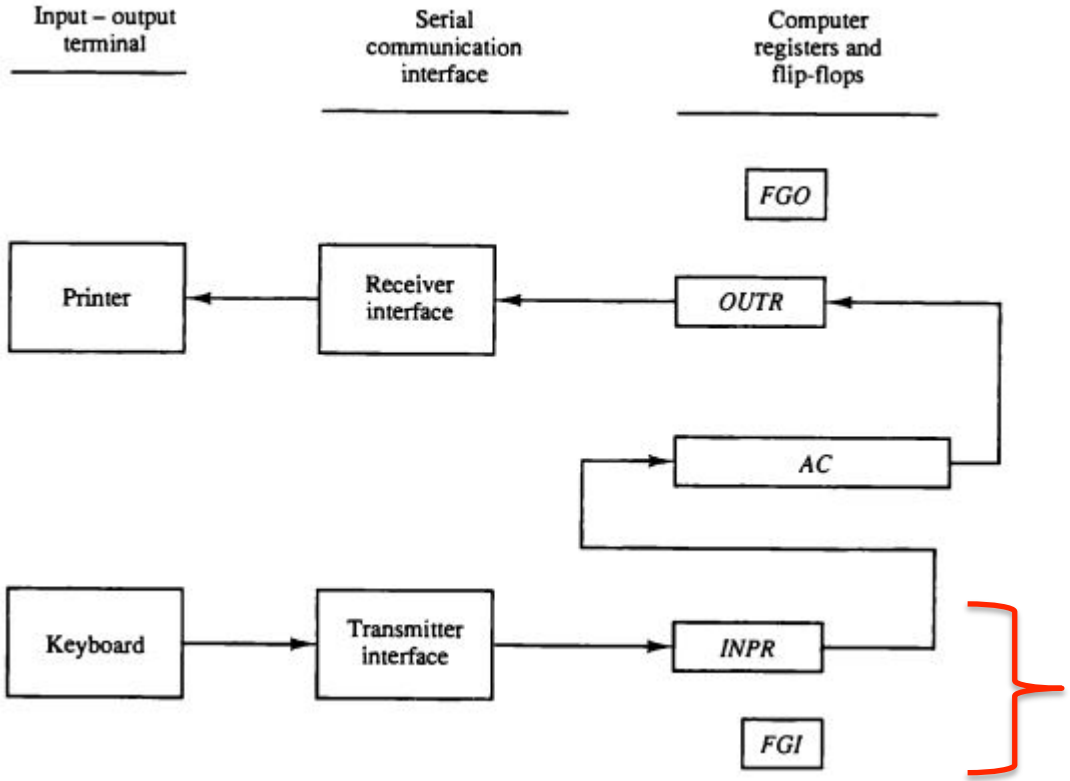
A computer can serve no useful purpose
unless it communicates with the external environment.

Instructions and data stored in the memory
must come from some input device,
and the results of the computations
must be transmitted to the user through some output device.

Input-Output Configuration

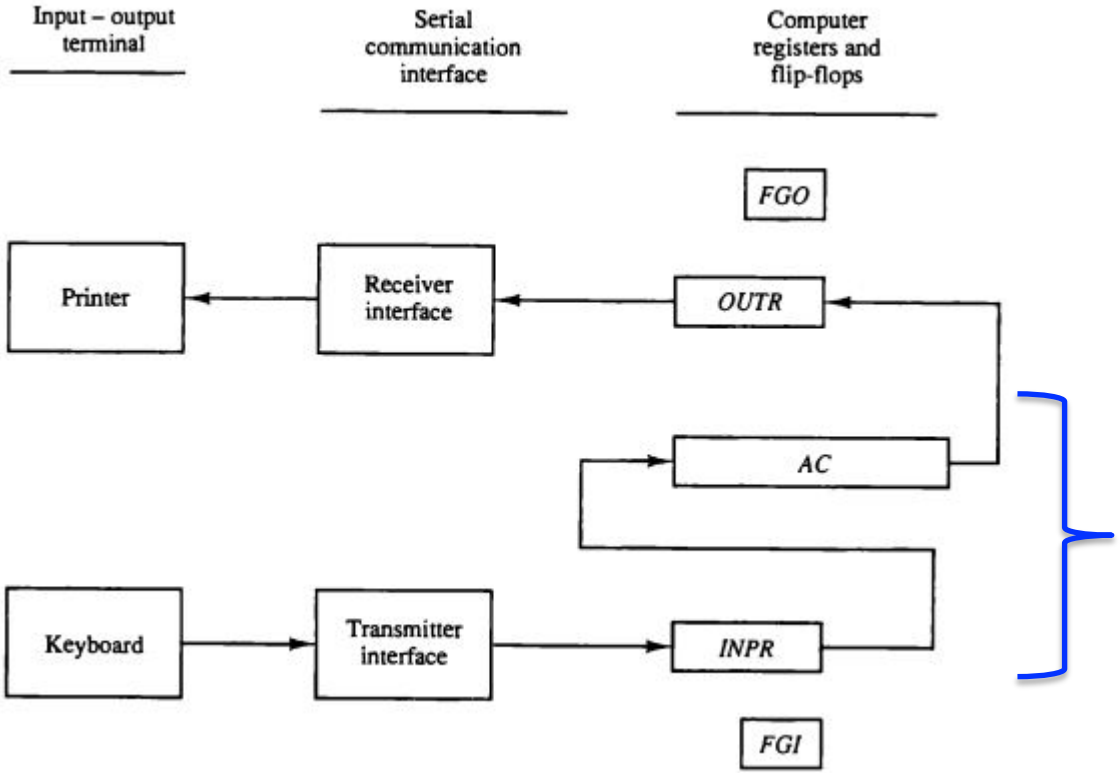


Input-Output Configuration: Flow of Information Transfer



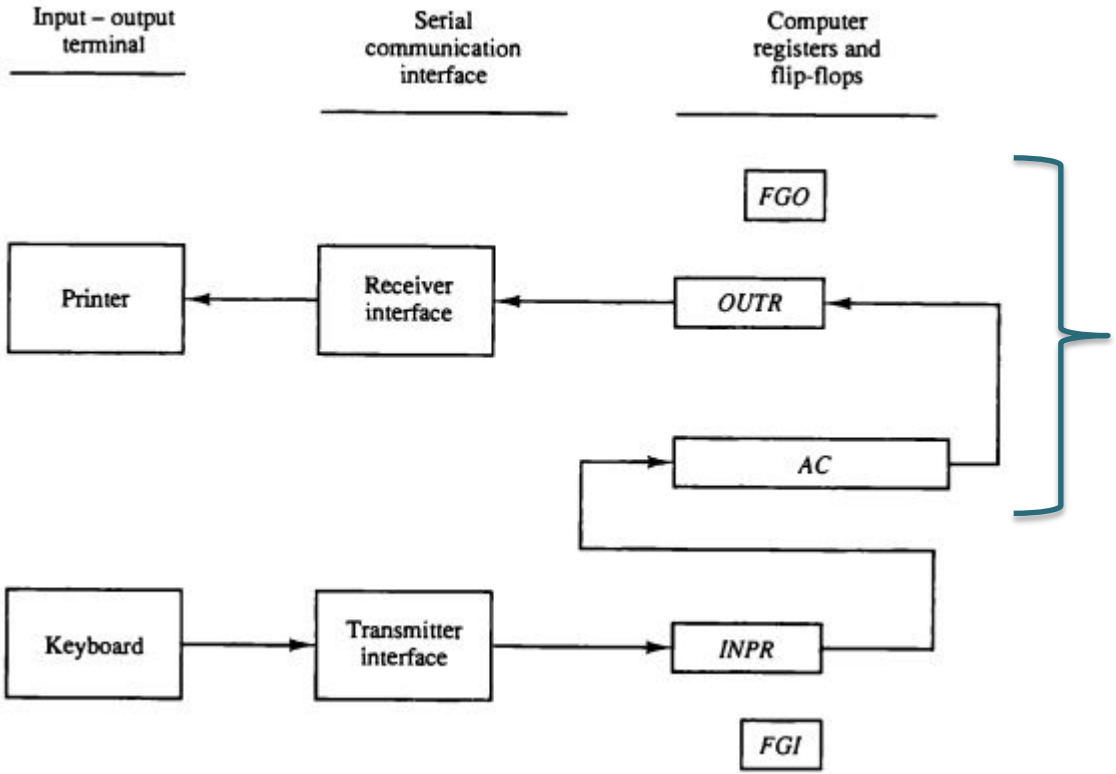
Initially, FGI is set to 0. When you press a key on the keyboard, an 8-bit alphanumeric code is shifted into INPR and FGI is set to 1. (As long as the flag is set, pressing another key does not change the information in INPR).

Input-Output Configuration: Flow of Information Transfer



The computer checks the flag; if FGI = 1, information from INPR is transferred in parallel to AC and FGI is cleared to 0.

Input-Output Configuration: Flow of Information Transfer



Initially, FGO is set to 1. The computer checks the flag; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The printer accepts the coded info, prints the corresponding character, and when this is done, FGO is set to 1.

(As long as FGO = 0, no new info goes into OUTR)

Input/Output Instructions

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

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

Programmed Control Transfer

The process of communication just described is referred to as PROGRAMMED CONTROL TRANSFER.

In this case, the computer keeps checking the flag bit, and when it finds that the bit is set, it initiates an information transfer.

Programmed Control Transfer

Programmed control transfer is **INEFFICIENT**,
because of the difference in the information flow rates
between the processor and the input/output device.

Program Interrupt: An Alternative to Programmed Control Transfer

An alternative to programmed control transfer is to

let the input/output device inform the processor

when it is ready for data transfer,

and in the meantime, the processor is free

to perform other useful tasks.

THIS IS KNOWN AS “PROGRAM INTERRUPT”.

Program Interrupt: How It Works

Step 1

When the computer is running a program, it does not check the flags.

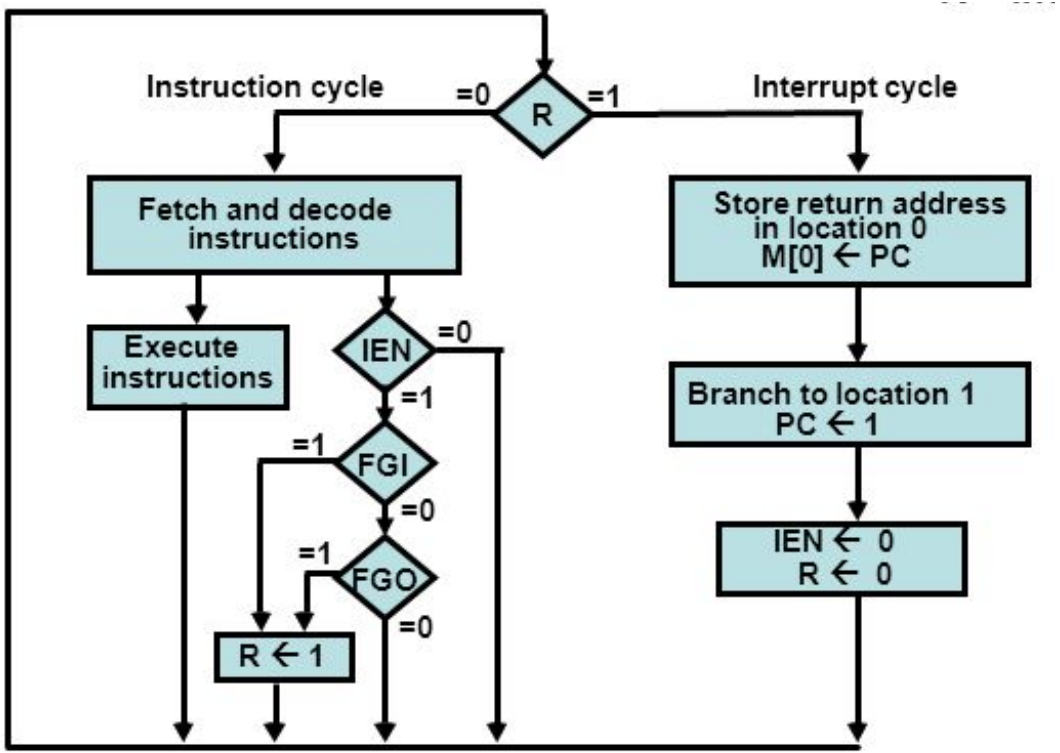
Step 2

When a flag is set, the computer is interrupted from its current program, and is informed that a flag has been set .

Step 3

The computer momentarily deviates from what it is doing to take care of the input/output operation, before returning to its original task.

Flowchart of Interrupt Cycle



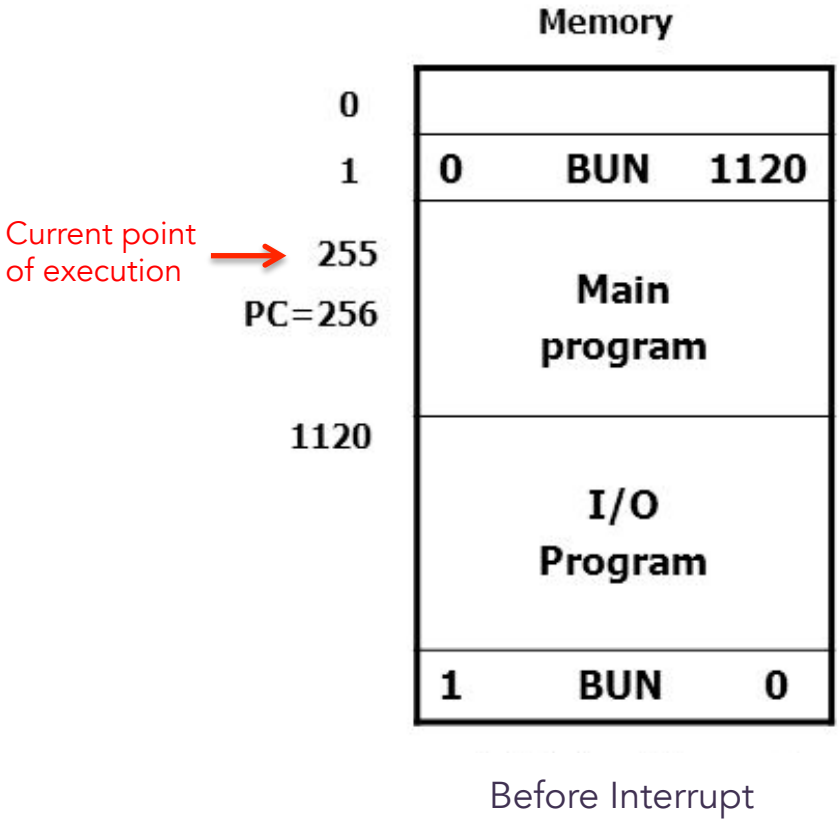
R= Interrupt Flip-Flop

IEN = Interrupt Enable Flip-Flop

When IEN = 0, the flags cannot interrupt the computer.

When IEN = 1, the computer can be interrupted.

Demonstration of Interrupt Cycle

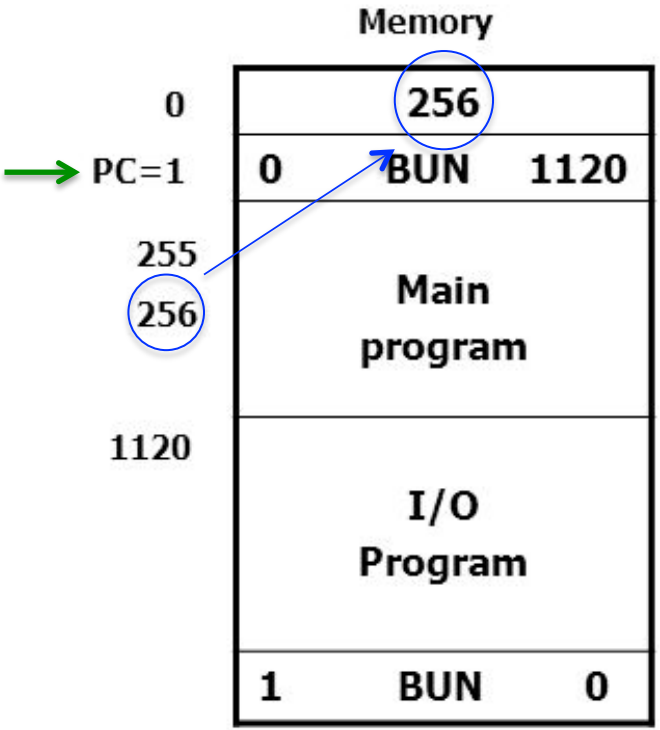


Before Interrupt

Before the interrupt, the main program is being executed (specifically, the instruction at location 255 is being executed).

When R = 1, the interrupt cycle begins…

Demonstration of Interrupt Cycle



After Interrupt

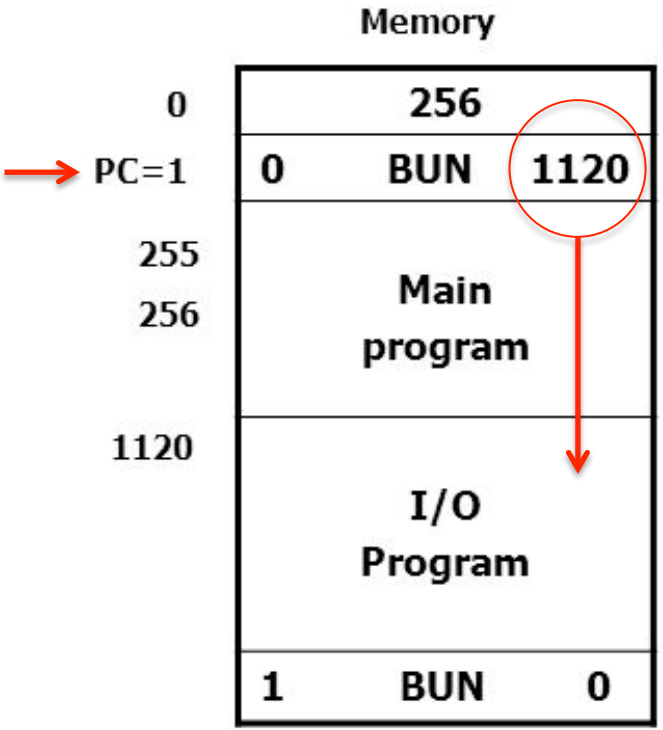
When the interrupt cycle begins…

the content of PC (i.e., 256) is placed in memory location 0

PC is set to 1

and R is cleared to 0

Demonstration of Interrupt Cycle

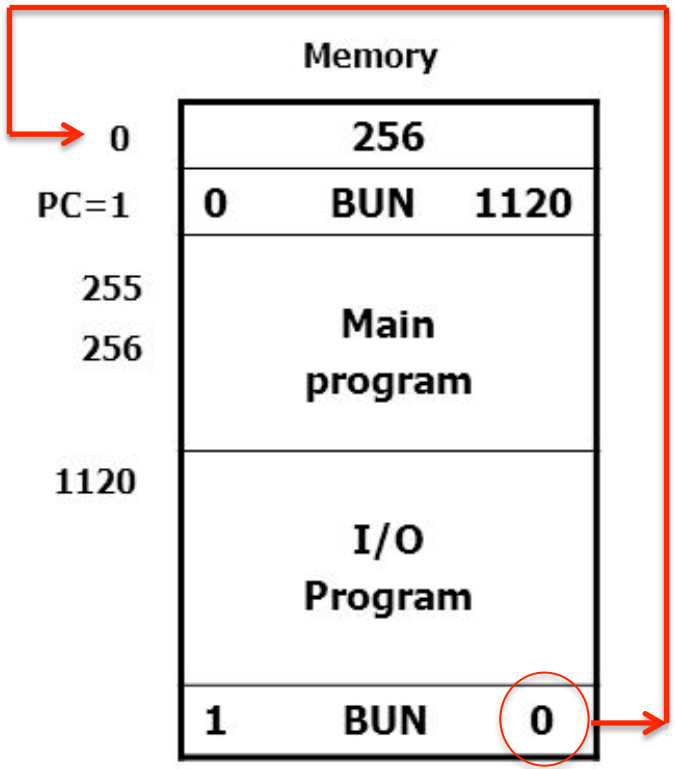


After Interrupt

At the beginning of the next intruction cycle, the instruction whose address is stored in PC is read.

This instruction takes the control to the I/O Program.

Demonstration of Interrupt Cycle



After Interrupt

When the I/O program is finished, the control goes back to memory location 0, which further takes the control to location 256.

Microoperations for Interrupt Cycle

1. **RT_0: AR ← 0, TR ← PC**

(During the first timing signal, AR is cleared to 0, and the content of PC is transferred to a temporary register TR.)

2. **RT_1: M[AR] ← TR, PC ← 0**

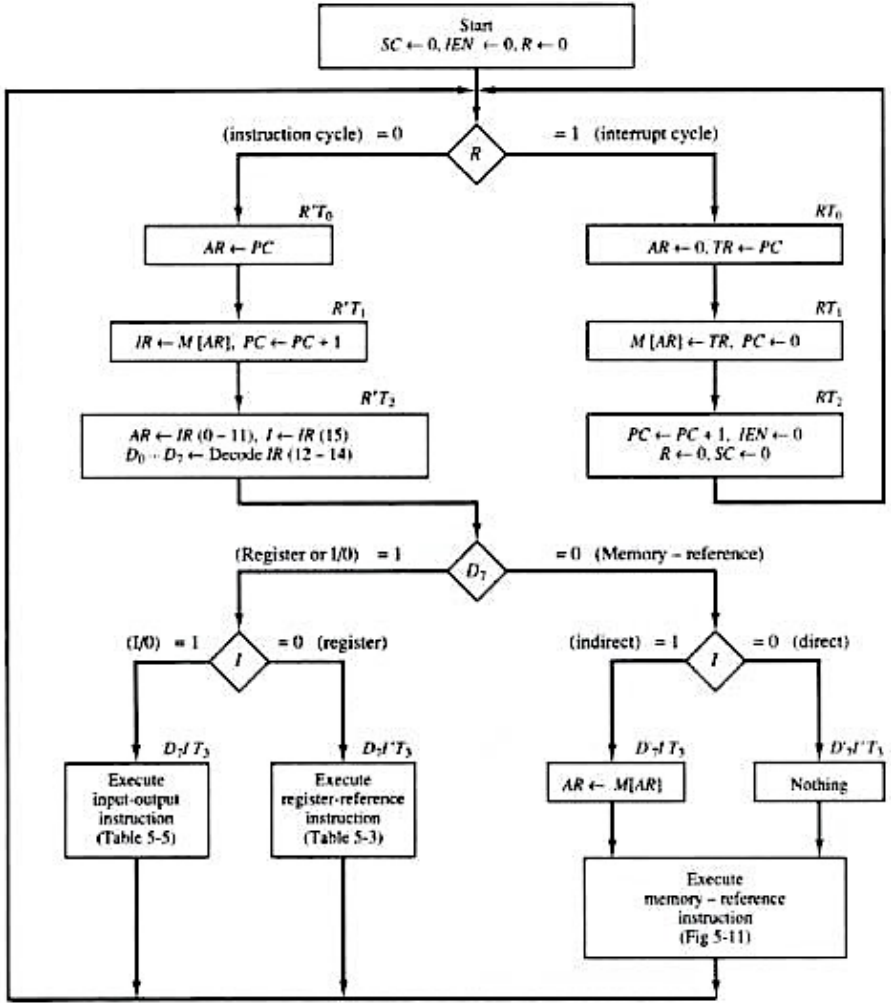
(During the second timing signal, the return address is stored in the memory location 0, and PC is cleared to 0.)

3. **RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0**

(During the third timing signal, PC is incremented, IEN, R, and SC are cleared to 0.)

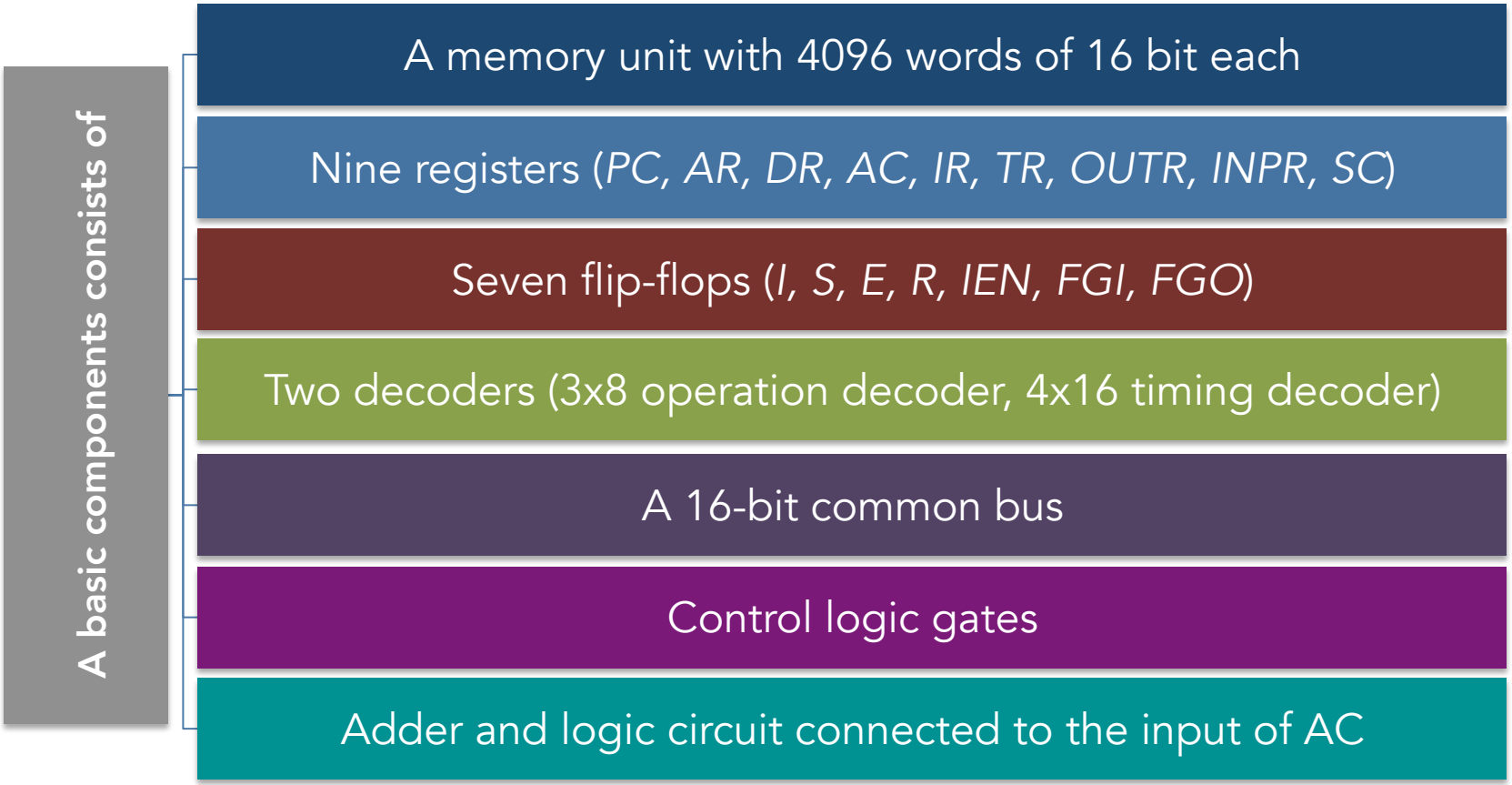
Complete Computer Description

Complete Flowchart for Computer Operation



Design of a Basic Computer

Components of a Basic Computer



Control Functions and Microoperations for Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$,
		$AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	$D_7'IT_3$:	$AR \leftarrow M[AR]$
Interrupt:		
$T_0'T_1'T_2'(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

Control Functions and Microoperations for Basic Computer

Register-reference:

$D_7I'T_3 = r$ (common to all register-reference instructions)

$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$

	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
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:	$IEN \leftarrow 0$

Exercises

1.

A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.

a. How many bits are there in the operation code, the register code part, and the address part?
b. Draw the instruction word format and indicate the number of bits in each part.
c. How many bits are there in the data and address inputs of the memory?

Solution:

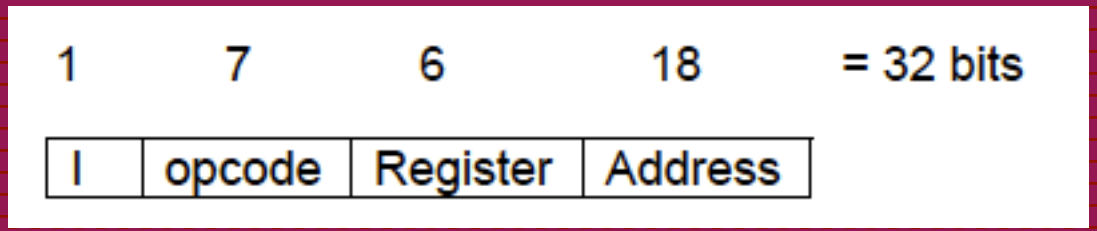
$256\ K = 2^8 \times 2^{10} = 2^{18}$

$64 = 2^6$

a. Address part: 18 bits, Register code part: 6 bits, Indirect bit: 1 bit

32 - (18+6+1) = 32 - 25 = 7 bits for opcode.

b.



c. No. of bits in data input: 32 bits, No. of bits in address input: 18 bits

2.

What is the difference between a direct and an indirect address instruction? How many references to memory are needed for each type of instruction to bring an operand into a processor register?

Solution:

A direct address instruction needs TWO references to memory: **one to read the instruction and one to read the operand.**

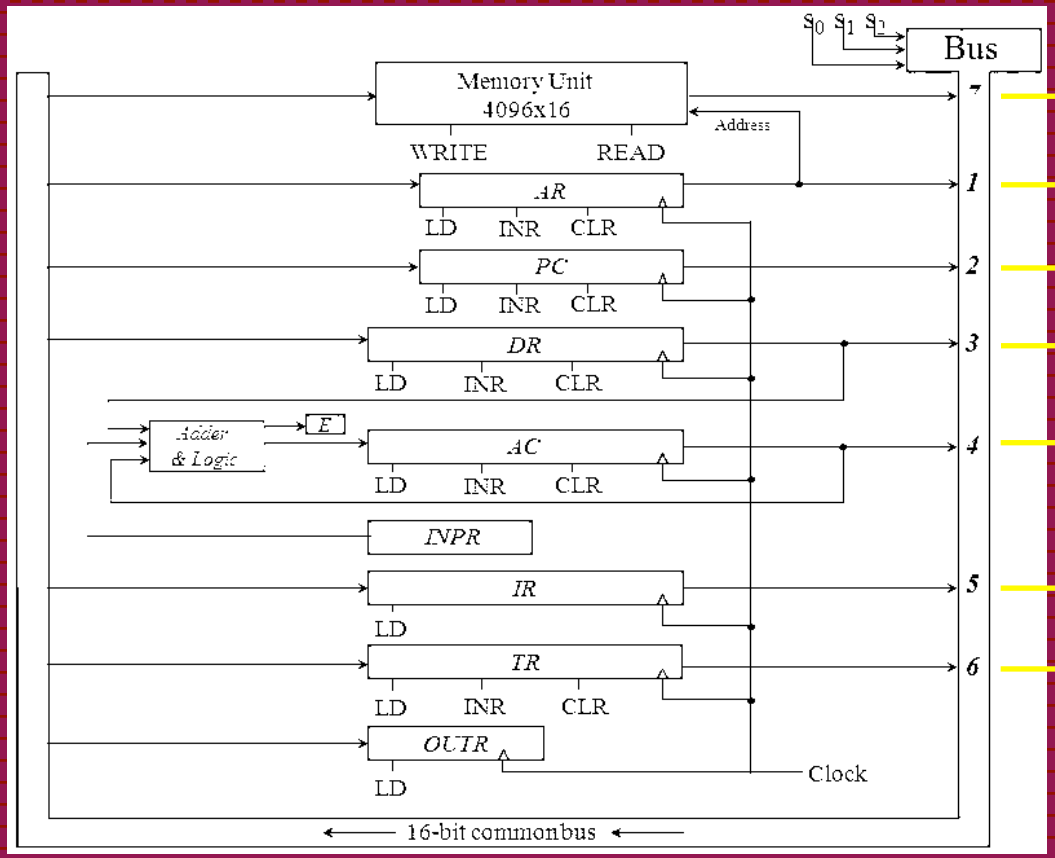
An indirect address instruction needs THREE references to memory: **one to read the instruction, one to read the effective address, and one to read the operand.**

3.

The following control inputs are active in the bus system shown in Fig. 5-4. For each case, specify the register transfer that will be executed during the next clock transition.

	S_2	S_1	S_0	LD of register	Memory	Adder
a.	1	1	1	*IR*	Read	—
b.	1	1	0	*PC*	—	—
c.	1	0	0	*DR*	Write	—
d.	0	0	0	*AC*	—	Add

Fig. 5-4.


S_0 S_1 S_2
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
16-bit commonbus
7
1
2
3
4
5
6
111
001
010
011
100
101
110

Solution:

	S_2	S_1	S_0	LD of register	Memory	Adder
a.	1	1	1	*IR*	Read	—
b.	1	1	0	*PC*	—	—
c.	1	0	0	*DR*	Write	—
d.	0	0	0	*AC*	—	Add

a. Memory read to bus and load to IR: IR ← M[AR]

b. TR to bus and load to PC: PC ← TR

c. AC to bus, write to memory, and load to DR:

DR ← AC, M[AR]← AC

d. Add DR (or INPR) to AC: AC ← AC + DR

4.

The following register transfers are to be executed in the system of Fig. 5-4. For each transfer, specify: (1) the binary value that must be applied to bus select inputs S_2, S_1, and S_0; (2) the register whose LD control input must be active (if any); (3) a memory read or write operation (if needed); and (4) the operation in the adder and logic circuit (if any).

a. $AR \leftarrow PC$
b. $IR \leftarrow M[AR]$
c. $M[AR] \leftarrow TR$
d. $AC \leftarrow DR$, $DR \leftarrow AC$ (done simultaneously)

Solution:

		(1) $S_2S_1S_0$	(2) Load(LD)	(3) Memory	(4) Adder
(a)	AR ← PC	010 (PC)	AR	—	—
(b)	IR ← M[AR]	111 (M)	IR	Read	—
(c)	M[AR] ← TR	110 (TR)	—	Write	—
(d)	DR ← AC AC ← DR	100 (AC)	DR and AC	—	Transfer DR to AC

Solution:

		(1) $S_2S_1S_0$	(2) Load(LD)	(3) Memory	(4) Adder
(a)	AR ← PC	010 (PC)	AR	—	—
(b)	IR ← M[AR]	111 (M)	IR	Read	—
(c)	M[AR] ← TR	110 (TR)	—	Write	—
(d)	DR ← AC AC ← DR	100 (AC)	DR and AC	—	Transfer DR to AC

The contents of AC are placed on the bus (with $S_2S_1S_0 = 100$), LD of DR is enabled, contents of DR are moved to AC through the adder and logic circuit, and LD of AC is enable, ***all during the same clock cycle***.

5.

Explain why each of the following microoperations cannot be executed during a single clock pulse in the system shown in Fig. 5-4. Specify a sequence of microoperations that will perform the operation.

a. $IR \leftarrow M[PC]$

b. $AC \leftarrow AC + TR$

c. $DR \leftarrow DR + AC$ (AC does not change)

Solution

a. IR ← M[PC]

PC cannot provide address to memory; address must be transferred to AR first.

AR ← PC

IR ← M[AR]

b. AC ← AC+TR

Add operation must be performed with DR; contents of TR must first be moved to DR.

DR ← TR

AC ← AC+DR

Solution (continued)

c. DR ← DR+AC *(AC does not change)*

Result of addition are transferred to AC (not DR). So to make sure that the contents of AC do not change, its contents must first be transferred to DR or TR.

AC ← DR, DR ← AC	(switch the contents of AC and DR)
AC ← AC+DR	(perform the addition)
AC ← DR, DR ← AC	(switch back)

Solution (continued)

c. Suppose AC = 10 and DR = 5.

AC ← DR, DR ← AC ➡ AC = 5, DR = 10

AC ← AC+DR ➡ AC = 10 + 5 = 15

AC ← DR, DR ← AC ➡ AC = 10, DR = 15

6.

Consider the instruction formats of the basic computer shown in Fig. 5-5 and the list of instructions given in Table 5-2. For each of the following 16-bit instructions, give the equivalent four-digit hexadecimal code and explain in your own words what it is that the instruction is going to perform.

a. 0001 0000 0010 0100
b. 1011 0001 0010 0100
c. 0111 0000 0010 0000

Fig. 5-5

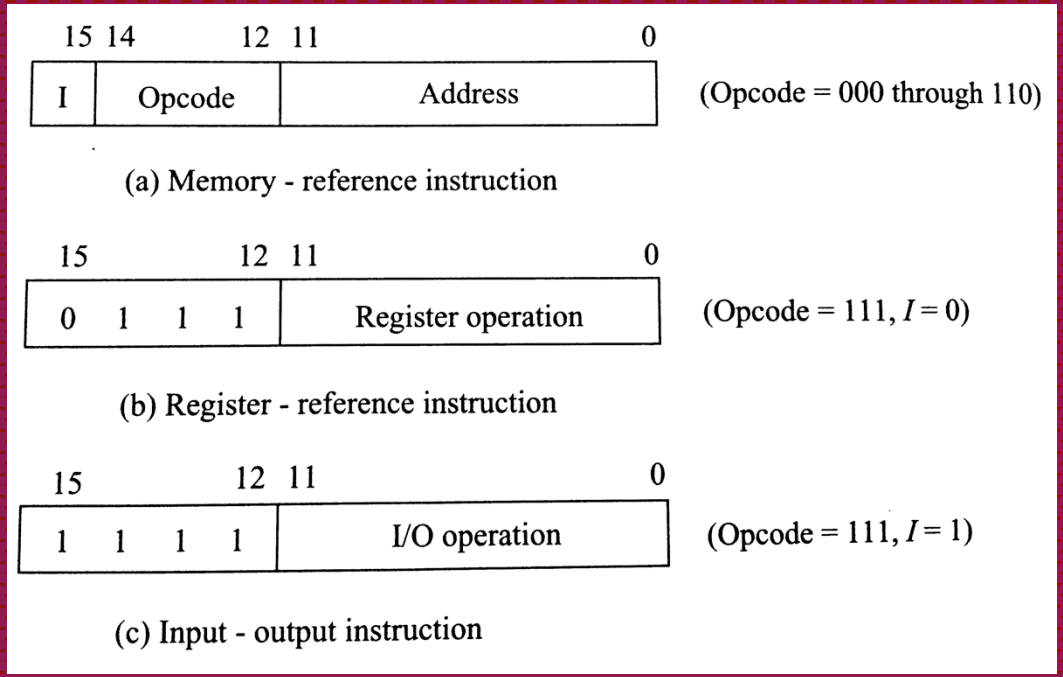

15 14 12 11 0
I
Opcode
Address
(Opcode = 000 through 110)
(a) Memory - reference instruction
15 12 11 0
0 1 1 1
Register operation
(Opcode = 111, I = 0)
(b) Register - reference instruction
15 12 11 0
1 1 1 1
I/O operation
(Opcode = 111, I = 1)
(c) Input - output instruction

Table 5-2

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
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

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

Solution:

a.	0001	0000	0010	0100
b.	1011	0001	0010	0100
c.	0111	0000	0010	0000

(a) 0001 0000 0010 0100 = $(1024)_{16}$
ADD $(024)_{16}$
ADD content of M[024] to AC ADD 024

(b) 1 011 0001 0010 0100 = $(B124)_{16}$
I STA $(124)_{16}$
Store AC in M[M[124]] STA I 124

(c) 0111 0000 0010 0000 = $(7020)_{16}$
Register Increment AC INC

Table 5-2

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
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

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

Solution:

a.	0001	0000	0010	0100
b.	1011	0001	0010	0100
c.	0111	0000	0010	0000

(a) 0001 0000 0010 0100 = $(1024)_{16}$
ADD $(024)_{16}$
ADD content of M[024] to AC ADD 024

(b) 1 011 0001 0010 0100 = $(B124)_{16}$
I STA $(124)_{16}$
Store AC in M[M[124]] STA I 124

(c) 0111 0000 0010 0000 = $(7020)_{16}$
Register Increment AC INC

Table 5-2

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
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

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

Solution:

a.	0001	0000	0010	0100
b.	1011	0001	0010	0100
c.	0111	0000	0010	0000

(a) 0001 0000 0010 0100 = $(1024)_{16}$
ADD $(024)_{16}$
ADD content of M[024] to AC ADD 024

(b) 1 011 0001 0010 0100 = $(B124)_{16}$
I STA $(124)_{16}$
Store AC in M[M[124]] STA I 124

(c) 0111 0000 0010 0000 = $(7020)_{16}$
Register Increment AC INC

Table 5-2

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
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

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

7.

What are the two instructions needed in the basic computer in order to set the *E* flip-flop to 1?

Solution:

CLE: Clear E

CME: Complement E

Table 5-2

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
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

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

8.

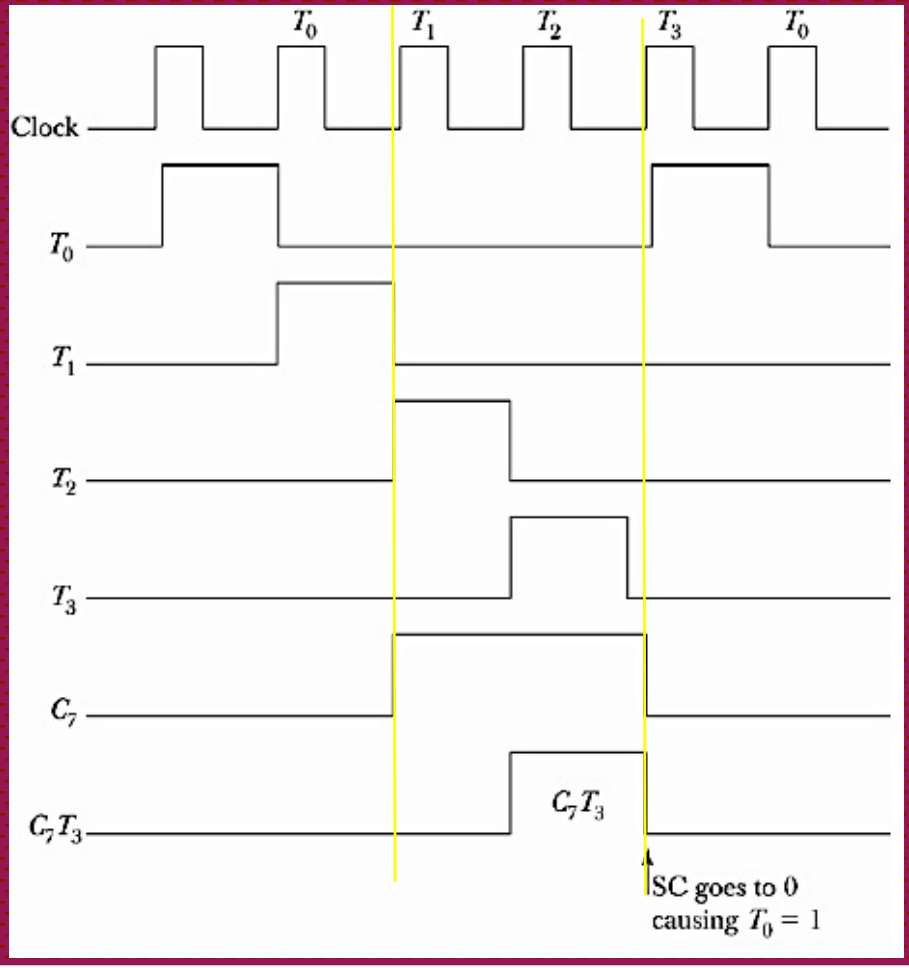
Draw a timing diagram similar to Fig. 5-7 assuming that SC is cleared to 0 at time T_3 if control signal C_7 is active.

$$C_7T_3: \quad SC \leftarrow 0$$

C_7 is activated with the positive clock transition associated with T_1.

Solution:

C_7T_3: $SC \leftarrow 0$



9.

The content of *AC* in the basic computer is hexadecimal A937 and the initial value of *E* is 1. Determine the contents of *AC*, *E*, *PC*, *AR*, and *IR* in hexadecimal after the execution of the CLA instruction. Repeat 11 more times, starting from each one of the register-reference instructions. The initial value of *PC* is hexadecimal 021.

Solution

1. CLA: Clear AC

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CLA	1	0000	022	800	7800

Hexadecimal code for CLA is 7800, where '7' is the opcode and '800' is the address.

2. CLE: Clear E

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CLE	0	A937	022	400	7400

Hexadecimal code for CLE is 7400, where '7' is the opcode and '400' is the address.

Solution (continued)

3. CMA: Complement AC

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CMA	1	56C8	022	200	7200

Hexadecimal code for CMA is 7200, where '7' is the opcode and '200' is the address.

	A	9	3	7
	1 0 1 0	1 0 0 1	0 0 1 1	0 1 1 1
Complement:	0 1 0 1	0 1 1 0	1 1 0 0	1 0 0 0
	5	**6**	**C**	**8**

Solution (continued)

4. CME: Complement E

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CME	0	A937	022	100	7100

Hexadecimal code for CME is 7100, where '7' is the opcode and '100' is the address.

Solution (continued)

5. CIR: Circulate Right AC and E

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CIR	1	D49B	022	080	7080

Hexadecimal code for CIR is 7080, where '7' is the opcode and '080' is the address.

CIR : AC ← shr AC, AC(15) ← E, E ← AC(0)

	A	9	3	7	
AC :	1 0 1 0	1 0 0 1	0 0 1 1	0 1 1 1	E = 1
shr, AC, AC(15) ← E, E ← AC(0) :	1 1 0 1	0 1 0 0	1 0 0 1	1 0 1 1	E = 1
	D	**4**	**9**	**B**	

Solution (continued)

6. CIL: Circulate Left AC and E

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CIL	1	526F	022	040	7040

Hexadecimal code for CIL is 7040, where '7' is the opcode and '050' is the address.

CIL : AC ← shl AC, AC(0) ← E, E ← AC(15)

	A	9	3	7	
AC :	1 0 1 0	1 0 0 1	0 0 1 1	0 1 1 1	E = 1
shl AC, AC(0) ← E, E ← AC(15) :	0 1 0 1	0 0 1 0	0 1 1 0	1 1 1 1	E = 1
	5	**2**	**6**	**F**	

Solution (continued)

7. INC: Increment AC

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
INC	1	A938	022	020	7020

Hexadecimal code for INC is 7020, where '7' is the opcode and '020' is the address.

8. SPA: Skip next instruction if AC is positive

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
SPA	1	A937	022	010	7010

Hexadecimal code for SPA is 7010, where '7' is the opcode and '010' is the address.

Solution (continued)

9. SNA: Skip next instruction if AC is negative

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
SNA	1	A937	023	008	7008

Hexadecimal code for SNA is 7008, where '7' is the opcode and '008' is the address.

10. SZA: SZA: Skip next instruction if AC = 0

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
SZA	1	A937	022	004	7004

Hexadecimal code for SZA is 7004, where '7' is the opcode and '004' is the address.

Solution (continued)

11. SZE: Skip next instruction if E = 0

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
SZE	1	A937	022	002	7002

Hexadecimal code for SZE is 7002, where '7' is the opcode and '002' is the address.

12. HTL: Halt computer

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
HLT	1	A937	022	001	7001

Hexadecimal code for HLT is 7001, where '7' is the opcode and '001' is the address.

Solution

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CLA	1	0000	022	800	7800
CLE	0	A937	022	400	7400
CMA	1	56C8	022	200	7200
CME	0	A937	022	100	7100
CIR	1	D49B	022	080	7080
CIL	1	526F	022	040	7040
INC	1	A938	022	020	7020
SPA	1	A937	022	010	7010
SNA	1	A937	023	008	7008
SZA	1	A937	022	004	7004
SZE	1	A937	022	002	7002
HLT	1	A937	022	001	7001

10.

An instruction at address 021 in the basic computer has $I = 0$, an operation code of the AND instruction, and an address part equal to 083 (all numbers are in hexadecimal). The memory word at address 083 contains the operand B8F2 and the content of *AC* is A937. Go over the instruction cycle and determine the contents of the following registers at the end of the execute phase: *PC*, *AR*, *DR*, *AC*, and *IR*. Repeat the problem six more times starting with an operation code of another memory-reference instruction.

Solution

Instruction at address 021 has *I* = 0 and opcode of the AND instruction. The address part is 083.

Also, M[083] = B8F and AC = A937.

1. AND: AND memory word to AC

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
AND	022	083	B8F2	A832	0083

AND: DR ← M[AR]

AC ← AC ∧ DR

When I = 0, AND instruction has the hexadecimal code 0xxx

AC:	A	9	3	7	
	1 0 1 0	1 0 0 1	0 0 1 1	0 1 1 1	
DR:	B	8	F	2	
	1 0 1 1	1 0 0 0	1 1 1 1	0 0 1 0	
AC ∧ DR:	**1 0 1 0**	**1 0 0 0**	**0 0 1 1**	**0 0 1 0**	**(= A832)**

Solution (continued)

2. ADD: ADD memory word to AC

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
ADD	022	083	B8F2	6229	1083

ADD: DR ← M[AR]

AC ← AC + DR, E ← C_{out}

When I = 0, ADD instruction has the hexadecimal code 1xxx

AC:	A	9	3	7	
	1 0 1 0	1 0 0 1	0 0 1 1	0 1 1 1	
DR:	B	8	F	2	
	1 0 1 1	1 0 0 0	1 1 1 1	0 0 1 0	
AC + DR:	**0 1 1 0**	**0 0 1 0**	**0 0 1 0**	**1 0 0 1**	**(= 6229) , E = 1**

Solution (continued)

3. LDA: Load memory word to AC

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
LDA	022	083	B8F2	B8F2	2083

LDA: AC ← M[AR]

When I = 0, LDA has the hexadecimal code 2xxx

4. STA: Store content of AC in memory

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
STA	022	083	—	A937	3083

STA: M[AR] ← AC

When I = 0, STA has the hexadecimal code 3xxx

Solution (continued)

3. LDA: Load memory word to AC

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
LDA	022	083	B8F2	B8F2	2083

LDA: DR ← M[AR]

AC ← DR

When I = 0, LDA has the hexadecimal code 2xxx

4. STA: Store content of AC in memory

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
STA	022	083	—	A937	3083

STA: M[AR] ← AC

When I = 0, STA has the hexadecimal code 3xxx

Solution (continued)

5. BUN: Branch Unconditionally

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
BUN	083	083	—	A937	4083

BUN: PC ← AR

When I = 0, BUN has the hexadecimal code 4xxx

5. BSA: Branch and save return address

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
BSA	084	084	—	A937	5083

BSA: M[AR] ← PC, PC ← AR+1

AR ← PC

When I = 0, BSA has the hexadecimal code 5xxx

Solution (continued)

7. ISZ: Increment and skip if zero

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
ISZ	022	083	B8F3	A937	6083

When I = 0, ISZ has the hexadecimal code 6xxx

ISZ: M[AR] ← M[AR] + 1 (M[AR] = DR)

If M[AR] + 1 = 0 then PC ← PC + 1

Solution

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
AND	022	083	B8F2	A832	0083
ADD	022	083	B8F2	6229	1083
LDA	022	083	B8F2	B8F2	2083
STA	022	083	—	A937	3083
BUN	083	083	—	A937	4083
BSA	084	084	—	A937	5083
ISZ	022	083	B8F3	A937	6083

11.

Show the contents in hexadecimal of registers *PC*, *AR*, *DR*, *IR*, and *SC* of the basic computer when an ISZ indirect instruction is fetched from memory and executed. The initial content of *PC* is 7FF. The content of memory at address 7FF is EA9F. The content of memory at address A9F is 0C35. The content of memory at address C35 is FFFF. Give the answer in a table with five columns, one for each register and a row for each timing signal. Show the contents of the registers after the positive transition of each clock pulse.

Solution

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Solution

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1

Solution (continued)

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2

Solution (continued)

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3

Solution (continued)

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4

Solution (continued)

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4
T_4	800	C35	FFFF	EA9F	5

Solution (continued)

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4
T_4	800	C35	FFFF	EA9F	5
T_5	800	C35	0000	EA9F	6

Address	Memory
PC = 7FF	EA9F
A9F	0C35
C35	FFFF

Solution (continued)

Indirect ISZ: DR ← M[AR]

DR ← DR + 1

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

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4
T_4	800	C35	FFFF	EA9F	5
T_5	800	C35	0000	EA9F	6
T_6	801	C35	0000	EA9F	0

12.

The content of *PC* in the basic computer is 3AF (all numbers are in hexadecimal). The content of *AC* is 7EC3. The content of memory at address 3AF is 932E. The content of memory at address 32E is 09AC. The content of memory at address 9AC is 8B9F.

a. What is the instruction that will be fetched and executed next?
b. Show the binary operation that will be performed in the *AC* when the instruction is executed.
c. Give the contents of registers *PC*, *AR*, *DR*, *AC*, and *IR* in hexadecimal and the values of *E*, *I*, and the sequence counter *SC* in binary at the end of the instruction cycle.

Solution

a. PC contains 3AF, and at memory location 3AF,

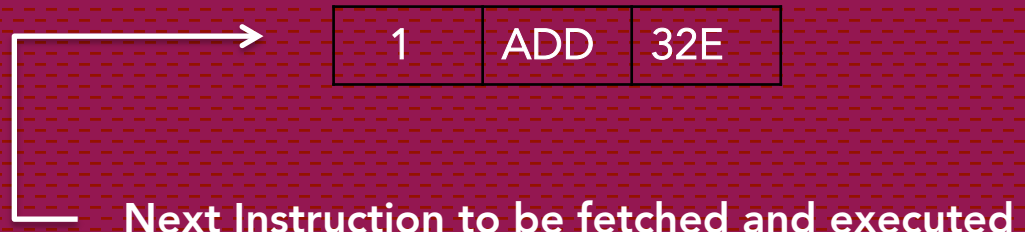
we have the value 932E

9 = 1001

So, 932E becomes 1001 32E = 1 001 32E = 1 ADD 32E

Memory

3AF	932E
32E	09AC
9AC	8B9F



Solution (continued)

b. AC = 7EC3

Next instruction = 

09AC

DR ⟶ 8B9F

Memory

3AF	932E
32E	09AC
9AC	8B9F

We will ADD 7EC3 and 8B9F => 7EC3 + 8B9F = 0A62

So, AC = 0A62 and Carry = 1 (carry is moved to E, So E = 1)

Solution (continued)

c.

PC = 3AF + 1 = 3B0

IR = 932E

AR = 9AC

DR = 8B9F

AC = 0A62

E = 1

I = 1

SC = 0000

Memory

3AF	932E
32E	09AC
9AC	8B9F

13.

Assume that the first six memory-reference instructions in the basic computer listed in Table 5-4 are to be changed to the instructions specified in the following table. *EA* is the effective address that resides in *AR* during time T_4. Assume that the adder and logic circuit in Fig. 5-4 can perform the exclusive-OR operation $AC \leftarrow AC \oplus DR$. Assume further that the adder and logic circuit cannot perform subtraction directly. The subtraction must be done using the 2's complement of the subtrahend by complementing and incrementing *AC*. Give the sequence of register transfer statements needed to execute each of the listed instructions starting from timing T_4. Note that the value in *AC* should not change unless the instruction specifies a change in its content. You can use *TR* to store the content of *AC* temporary or you can exchange *DR* and *AC*.

Table 5-4

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 $PC \leftarrow PC + 1$

13.

Symbol	Opcode	Symbolic designation	Description in words
XOR	000	$AC \leftarrow AC \oplus M[EA]$	Exclusive-OR to AC
ADM	001	$M[EA] \leftarrow M[EA] + AC$	Add AC to memory
SUB	010	$AC \leftarrow AC - M[EA]$	Subtract memory from AC
XCH	011	$AC \leftarrow M[EA], \quad M[EA] \leftarrow AC$	Exchange AC and memory
SEQ	100	If $(M[EA] = AC)$ then $(PC \leftarrow PC + 1)$	Skip on equal
BPA	101	If $(AC > 0)$ then $(PC \leftarrow EA)$	Branch if AC positive and non-zero

Solution

Original Instruction and Microoperations	New Instruction and Microoperations
$AC \leftarrow AC \wedge M[AR]$	$AC \leftarrow AC \oplus M[EA]$
D_0T_4: $DR \leftarrow M[AR]$	D_0T_4: $DR \leftarrow M[AR]$
D_0T_5: $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$	D_0T_5: $AC \leftarrow AC \oplus DR$, $SC \leftarrow 0$

Solution (continued)

Original Instruction and Microoperations	New Instruction and Microoperations
	$M[EA] \leftarrow M[EA] + AC$
$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$	
D_1T_4: $DR \leftarrow M[AR]$	D_1T_4: $DR \leftarrow M[AR]$
D_1T_5: $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$	D_1T_5: $DR \leftarrow AC, AC \leftarrow AC + DR$
	D_1T_6: $M[AR] \leftarrow AC, AC \leftarrow DR, SC \leftarrow 0$

Solution (continued)

Original Instruction and Microoperations	New Instruction and Microoperations
$AC \leftarrow M[AR]$	$AC \leftarrow AC - M[EA]$
D_2T_4: $DR \leftarrow M[AR]$	D_2T_4: $DR \leftarrow M[AR]$
D_2T_5: $AC \leftarrow DR$, $SC \leftarrow 0$	D_2T_5: $DR \leftarrow AC$, $AC \leftarrow DR$
	D_2T_6: $AC \leftarrow \overline{AC}$
	D_2T_7: $AC \leftarrow AC + 1$
	D_2T_8: $AC \leftarrow DR + AC$, $SC \leftarrow 0$

Solution (continued)

Original Instruction and Microoperations	New Instruction and Microoperations
$M[AR] \leftarrow AC$	$AC \leftarrow M[EA], M[EA] \leftarrow AC$
D_3T_4: $M[AR] \leftarrow AC, SC \leftarrow 0$	D_3T_4: $DR \leftarrow M[AR]$
	D_3T_5 : $M[AR] \leftarrow AC, AC \leftarrow DR, SC \leftarrow 0$

Solution (continued)

Original Instruction and Microoperations	New Instruction and Microoperations
PC ← AC	If (M[EA] = AC) then (PC ← PC + 1)
D_4T_4: PC ← AR, SC ← 0	D_4T_4: DR ← M[AR]
	D_4T_5: TR ← AC, AC ← AC ⊕ DR
	D_4T_6: If (AC = 0) then PC ← PC + 1, AC ← TR, SC ← 0

Solution (continued)

Original Instruction and Microoperations	New Instruction and Microoperations
M[AR] ← PC, PC ← AR+1	If (AC > 0) then PC ← EA
D_5T_4: M[AR] ← PC, AR ← AR + 1	If （AC (15) = 0)
D_5T_5: PC ← AR, SC ← 0	then (PC ←AR), SC ← 0

14.

Make the following changes to the basic computer.

1. Add a register to the bus system CTR (count register) to be selected with $S_2S_1S_0 = 000$.
2. Replace the ISZ instruction with an instruction that loads a number into CTR.

LDC Address $CTR \leftarrow M[Address]$

3. Add a register reference instruction ICSZ: Increment *CTR* and skip next instruction if zero. Discuss the advantage of this change.

Solution:

This change converts the ISZ instruction from a memory-reference instruction to a register-reference instruction.

The new instruction ICSZ can be executed at time T_3 instead of time T_6, which helps save 3 clock cycles.

15.

The memory unit of the basic computer shown in Fig. 5-3 is to be changed to a 65,536 × 16 memory, requiring an address of 16 bits. The instruction format of a memory-reference instruction shown in Fig. 5-5(a) remains the same for $I = 1$ (indirect address) with the address part of the instruction residing in positions 0 through 11. But when $I = 0$ (direct address), the address of the instruction is given by the 16 bits in the next word following the instruction. Modify the microoperations during time T_2, T_3, (and T_4 if necessary) to conform with this configuration.

Fig. 5.3

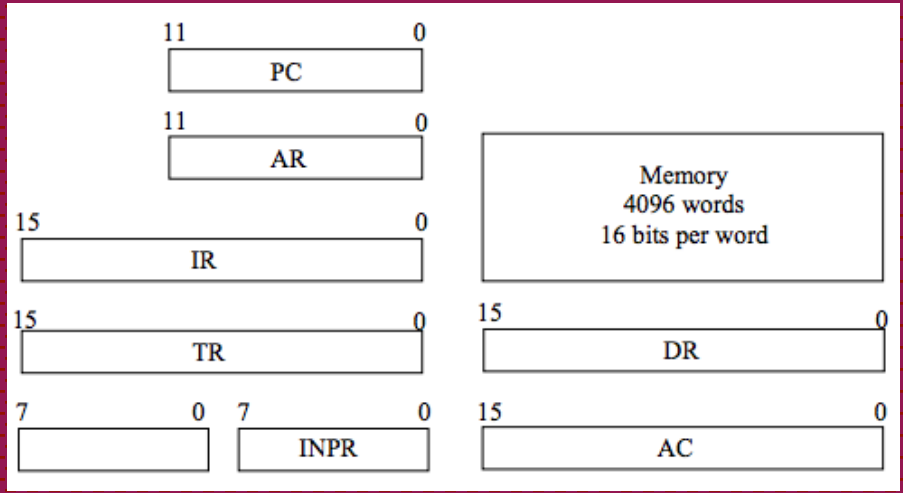
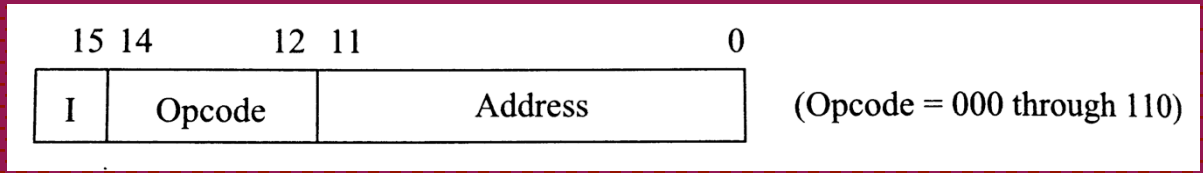
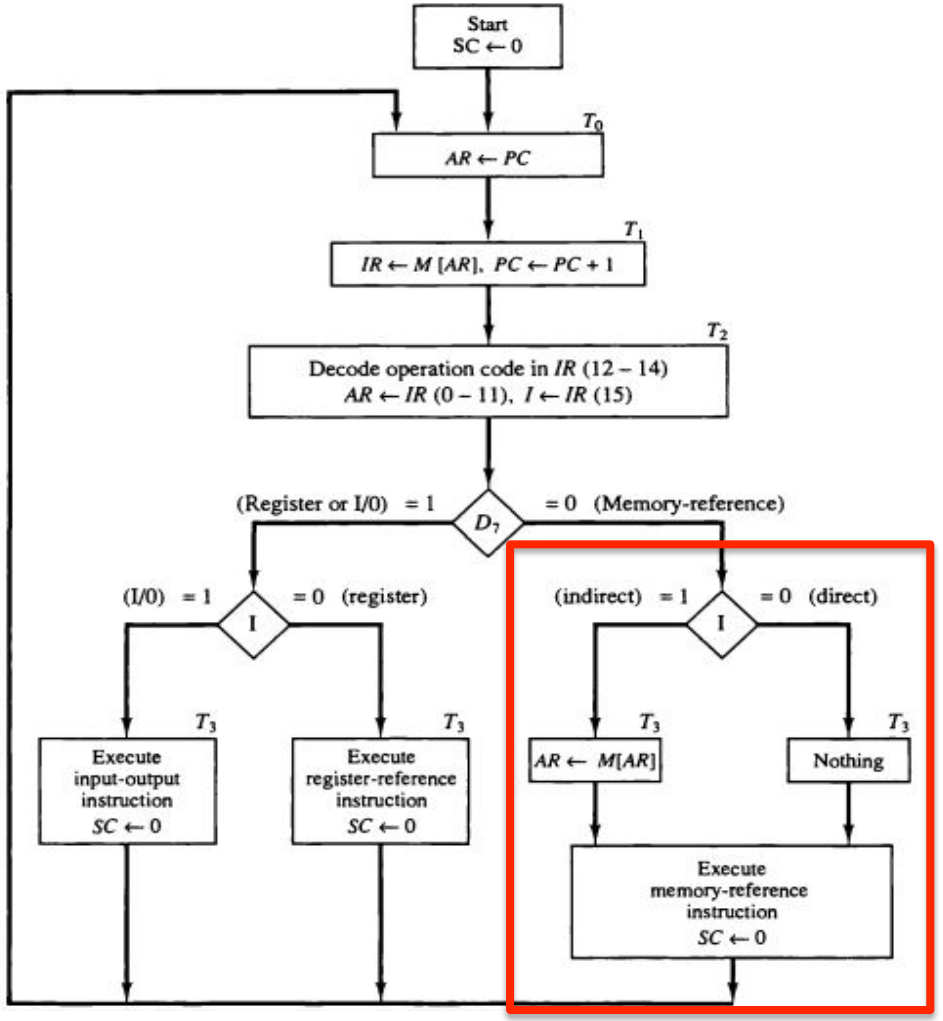


Fig. 5.5(a)

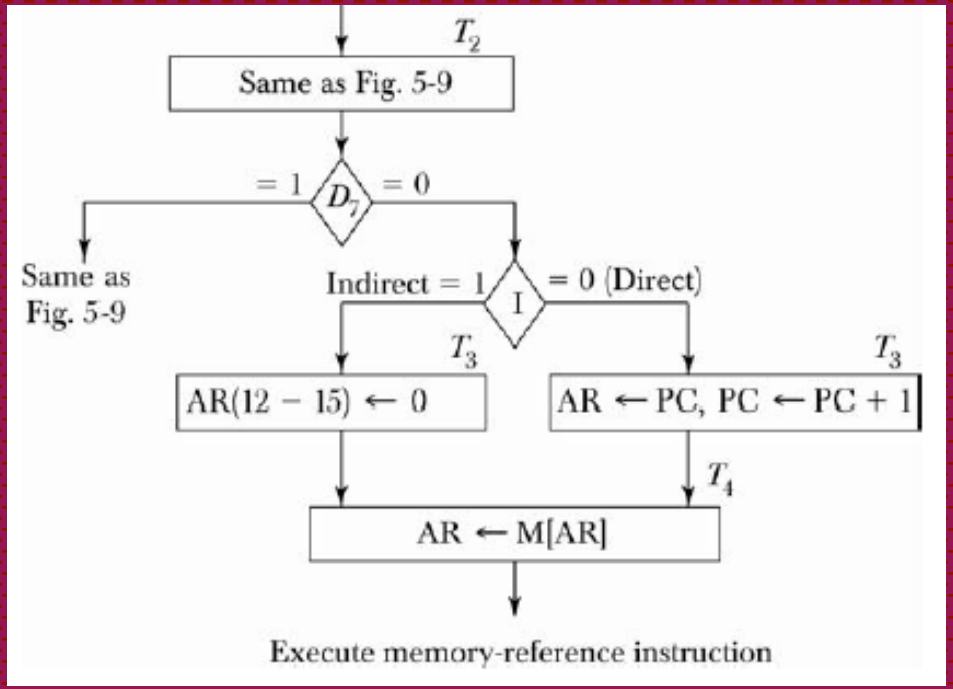


Flowchart for Instruction Cycle: Initial Configuration

(Fig. 5-9)



Solution:


T_2
Same as Fig. 5-9
= 1
D_7
= 0
Same as Fig. 5-9
Indirect = 1
I
= 0 (Direct)
T_3
T_3
AR(12 – 15) ← 0
AR ← PC, PC ← PC + 1
T_4
AR ← M[AR]
Execute memory-reference instruction

16.

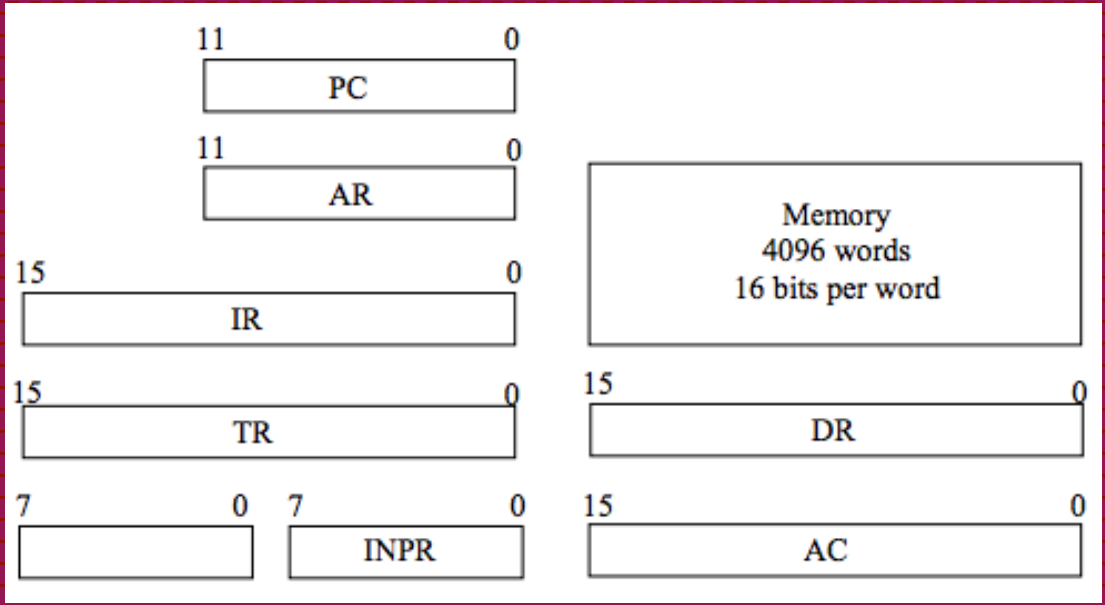
A computer uses a memory of 65,536 words with eight bits in each word. It has the following registers: *PC*, *AR*, *TR* (16 bits each), and *AC*, *DR*, *IR* (eight bits each). A memory-reference instruction consists of three words: an 8-bit operation-code (one word) and a 16-bit address (in the next two words). All operands are eight bits. There is no indirect bit.

a. Draw a block diagram of the computer showing the memory and registers as in Fig. 5-3. (Do not use a common bus).

b. Draw a diagram showing the placement in memory of a typical three-word instruction and the corresponding 8-bit operand.

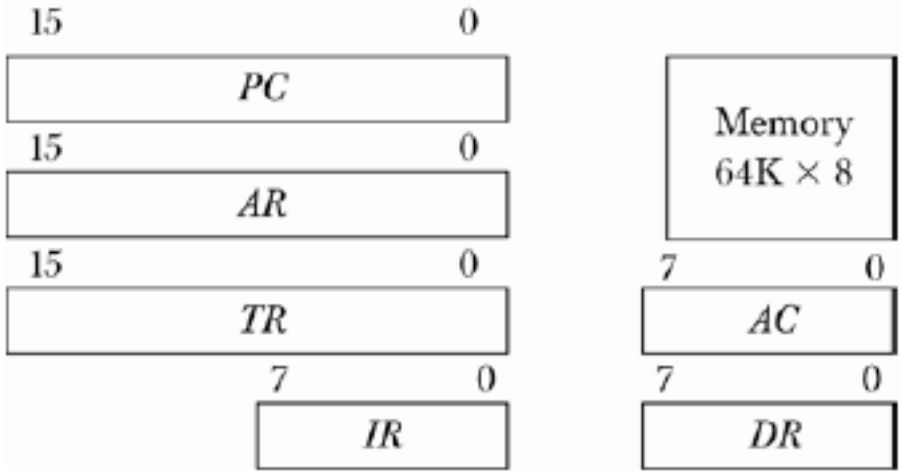
c. List the sequence of microoperations for fetching a memory reference instruction and then placing the operand in *DR*. Start from timing signal T_0.

Fig. 5.3


11
0
PC
11
0
AR
15
0
IR
15
0
TR
7
0
7
0
INPR
Memory
4096 words
16 bits per word
15
0
DR
15
0
AC

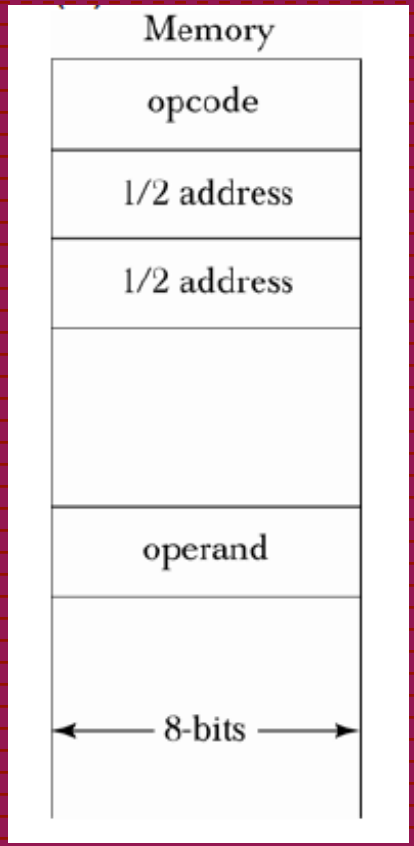
Solution

a. Diagram of the computer showing memory and registers.



Solution (continued)

b. Placement of a typical three-word instruction and the corresponding operand in memory.



Solution (continued)

c. Microoperations needed to fetch a memory-reference instruction and place the operand in DR.

T_0: IR ← M(PC), PC ← PC + 1

T_1: AR(0–7) ← M[PC], PC ← PC + 1

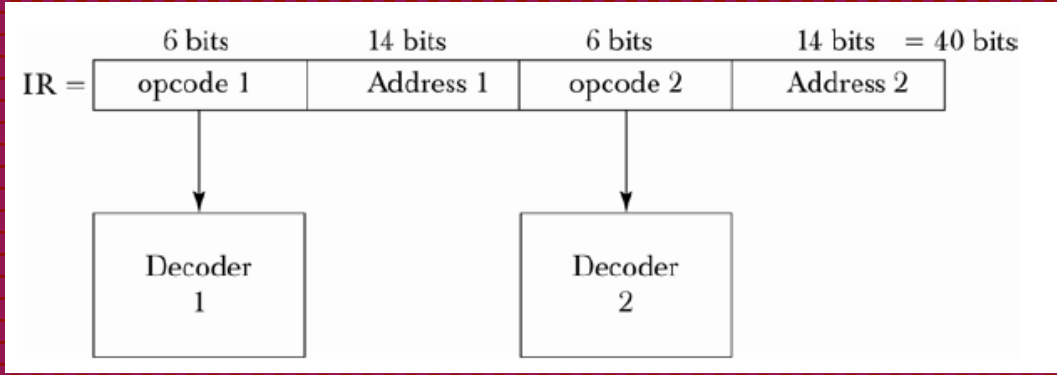
T_2: AR(8–15) ← M[PC], PC ← PC +1

T_3: DR ← M [AR]

17.

A digital computer has a memory unit with a capacity of 16,384 words, 40 bits per word. The instruction code format consists of six bits for the operation part and 14 bits for the address part (no indirect mode bit). Two instructions are packed in one memory word, and a 40-bit instruction register *IR* is available in the control unit. Formulate a procedure for fetching and executing instructions for this computer.

Solution



1. Read 40-bit double instruction from memory to IR, increment PC
2. Decode opcode 1
3. Execute instruction 1 using address 1
4. Decode opcode 2
5. Execute instruction 2 using address 2
6. Go to step 1

18.

An output program resides in memory starting from address 2300. It is executed after the computer recognizes an interrupt when *FGO* becomes a 1 (while *IEN* = 1).

a. What instruction must be placed at address 1?

b. What must be the last two instructions of the output program?

Solution

a. BUN 2300 0 (Branch direct with address 2300)

b. ION (Interrupt Enable On)

BUN 0 1 (Branch indirect with address 0)

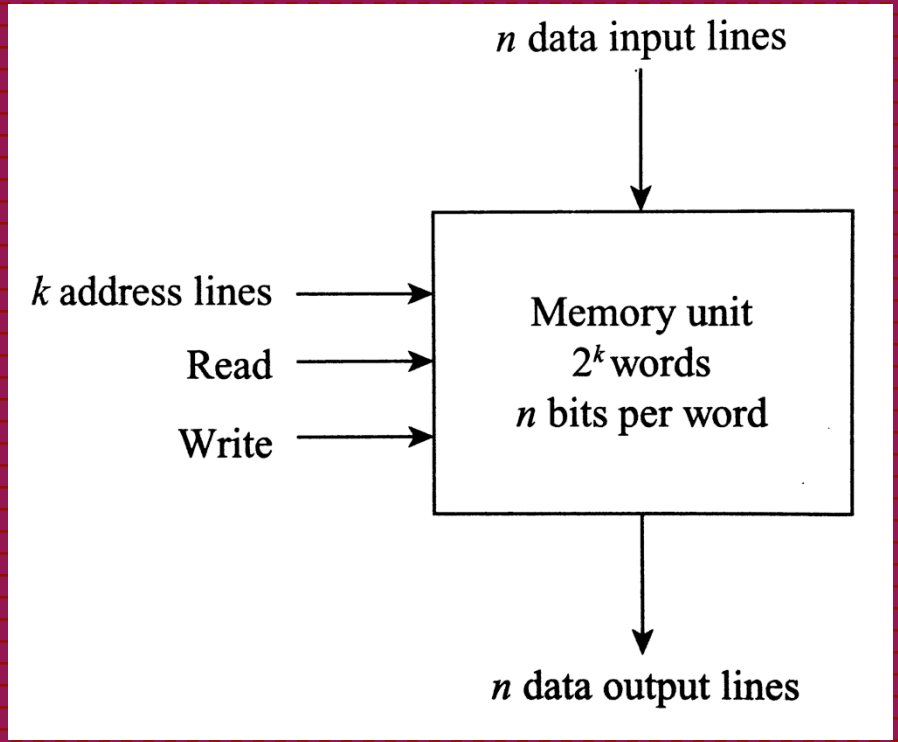
19.

The register transfer statements for a register R and the memory in a computer are as follows (the X's are control functions that occur at random):

$X_3' X_1$:	$R \leftarrow M[AR]$	Read memory word into R
$X_1' X_2$:	$R \leftarrow AC$	Transfer AC to R
$X_1' X_3$:	$M[AR] \leftarrow R$	Write R to memory

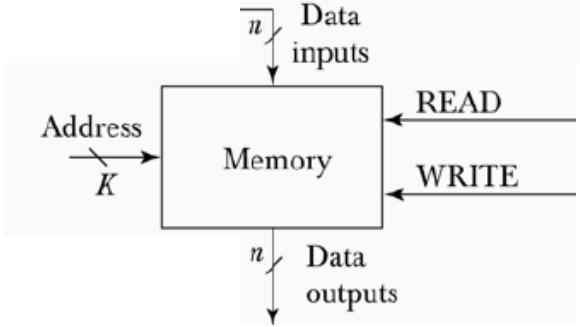
The memory has data inputs, data outputs, address inputs, and control inputs to read and write as in Fig. 2-12. Draw the hardware implementation of R and the memory in block diagram form. Show how the control functions X_1 through X_3 select the load control input of R, the select inputs of multiplexers that you include in the diagram, and the read and write inputs of the memory.

Fig. 2-12


n data input lines
k address lines
Read
Write
Memory unit
2^k words
n bits per word
n data output lines

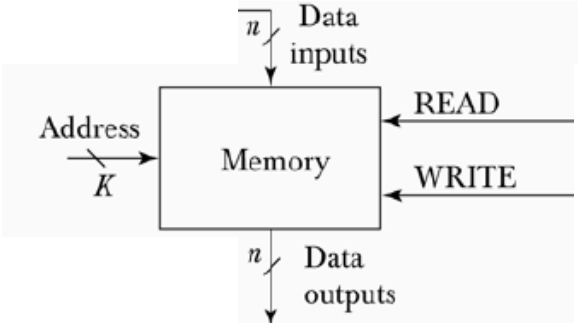
Solution (Step 1)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



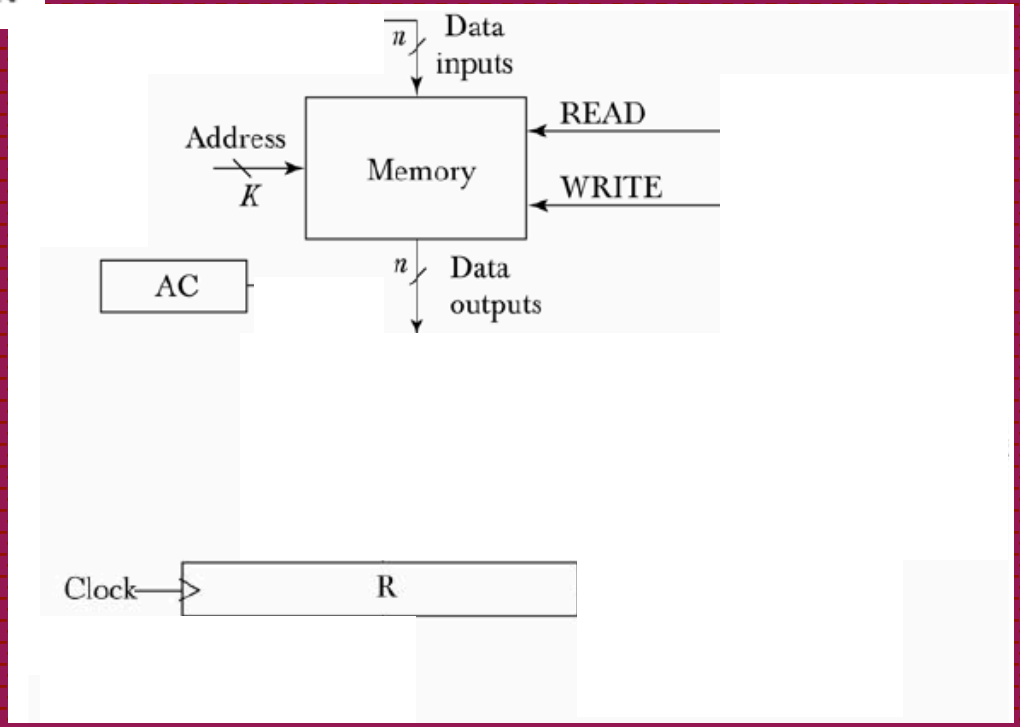
Solution (Step 2)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



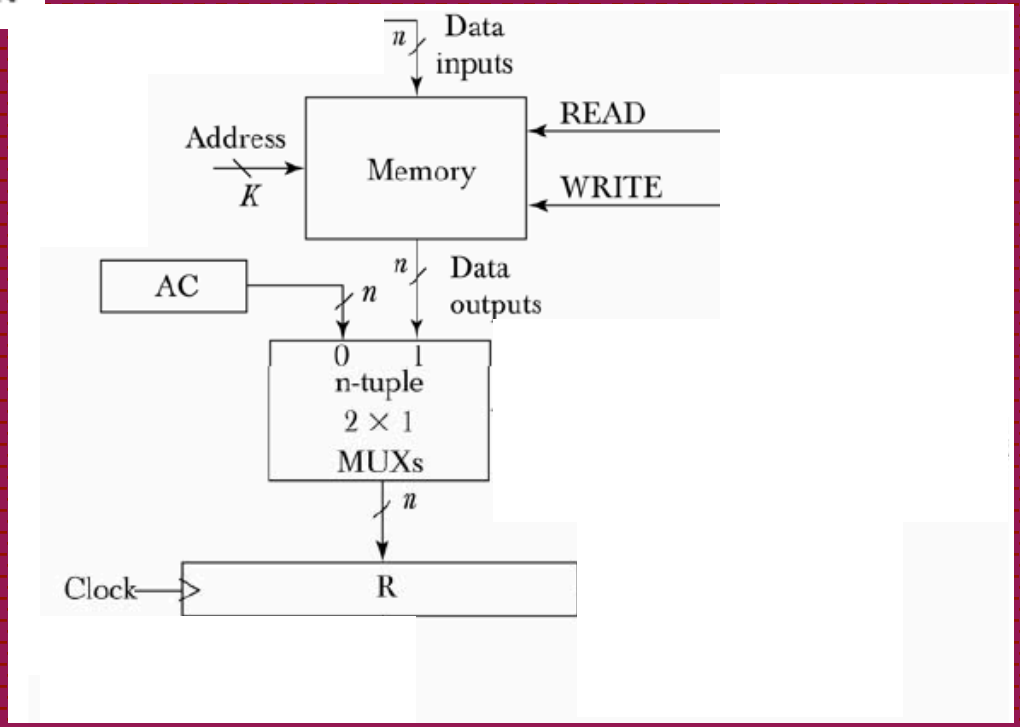
Solution (Step 3)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



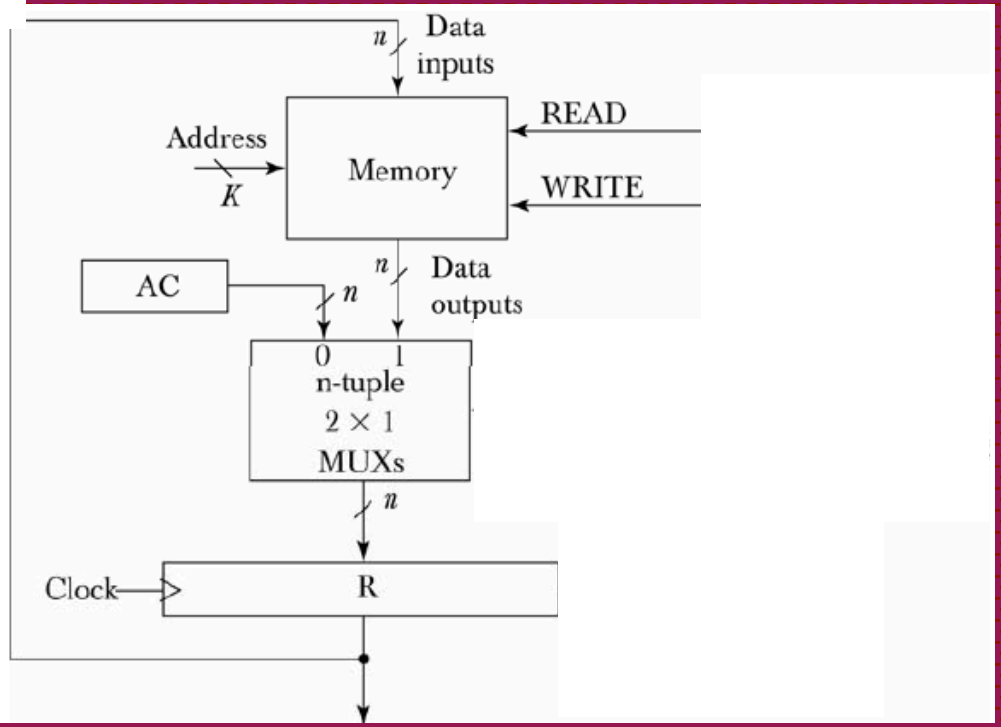
Solution (Step 4)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



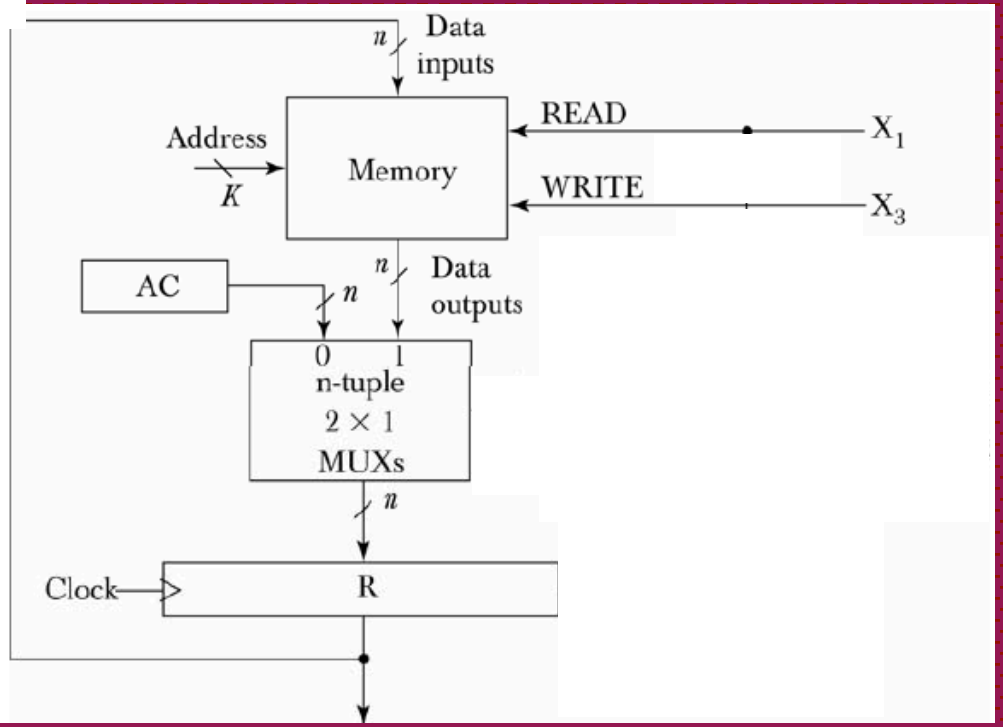
Solution (Step 5)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



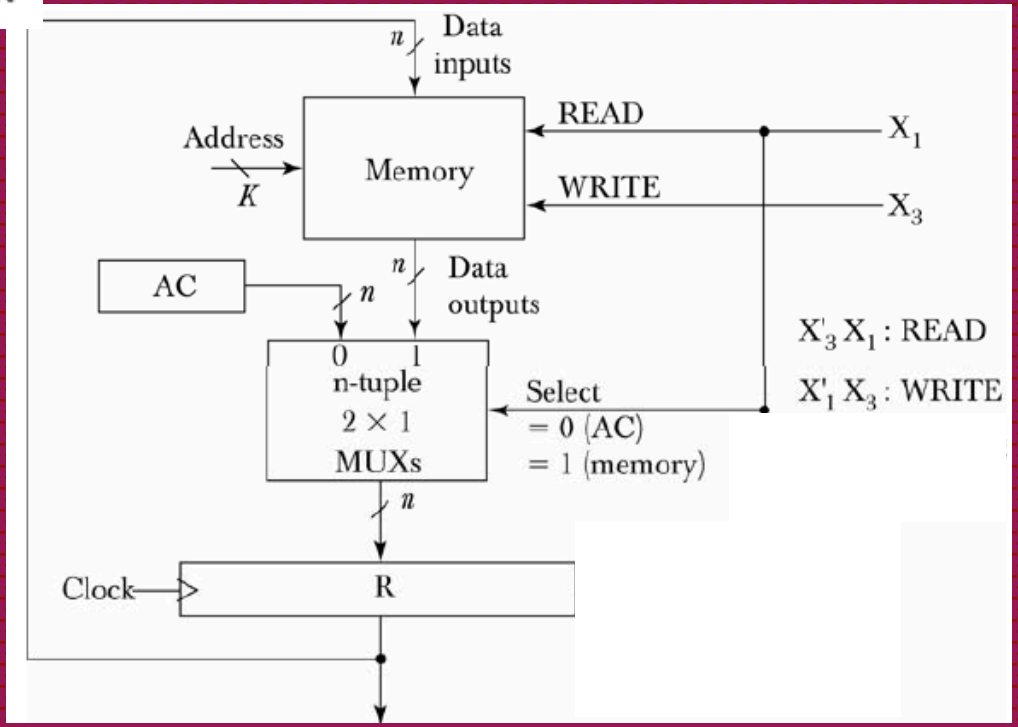
Solution (Step 6)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



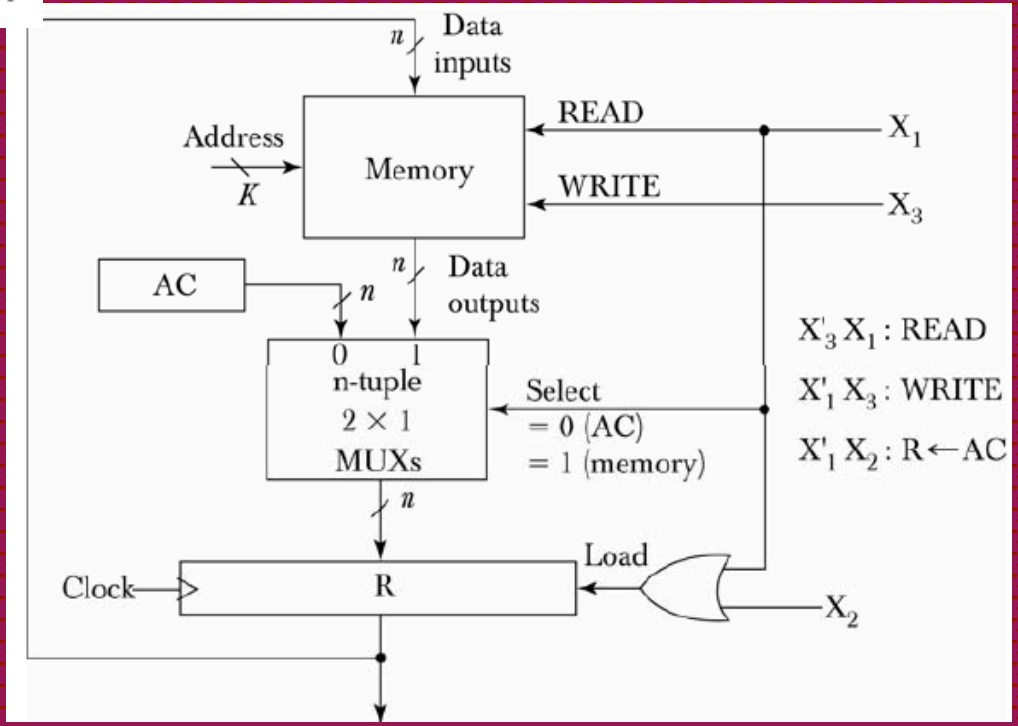
Solution (Step 7)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



Solution (Step 8)

$X_3' X_1$: $R \leftarrow M[AR]$
$X_1' X_2$: $R \leftarrow AC$
$X_1' X_3$: $M[AR] \leftarrow R$



22.

Derive the control gates for the write input of the memory in the basic computer.

Solution

Memory's "write" operation pertains to all those operations that involve a transfer like this: M[AR] ← xx

$$\text{Write} = D_3T_4 + D_5T_4 + D_6T_6 + RT_1$$ (from Table 5.6)

↓ STA ↓ BSA ↓ ISZ ↓ one of the microoperations of the interrupt operation

23.

Show the complete logic of the interrupt flip-flops *R* in the basic computer. Use a *JK* flip-flop and minimize the number of gates.

Solution

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

$RT_2 : R \leftarrow 0$

or

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

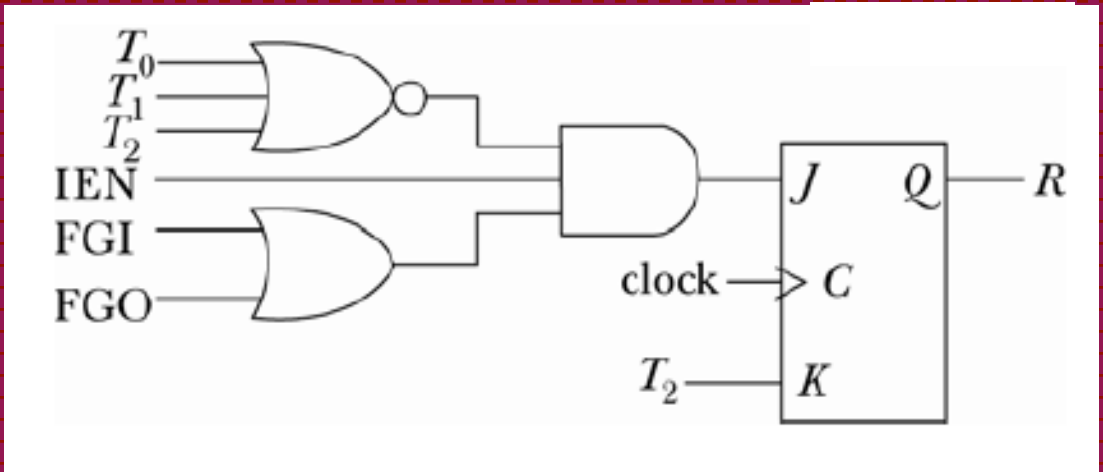
$RT_2 : R \leftarrow 0$

} from Table 5.6

Solution (continued)

$(T_0 + T_1 + T_2)'$ (IEN) (FGI + FGO) : R ← 1

RT_2 : R ← 0



25.

Derive the Boolean expression for the gate structure that clears the sequence counter SC to 0. Draw the logic diagram of the gates and show how the output is connected to the INR and CLR inputs of SC (see Fig. 5-6). Minimize the number of gates.

Solution

Clearing the sequence counter means CLR (SC), i.e., any microoperation that involves the part "SC ← 0".

$$CLR(SC) = RT_2 + D_7T_3 (I'+I) + (D_0 + D_1 + D_2 + D_5) T_5 + (D_3 + D_4) T_4 + D_6T_6$$

(from Table 5.6)

Solution (continued)

$$CLR(SC) = RT_2 + D_7T_3(I'+I) + (D_0 + D_1 + D_2 + D_5)T_5 + (D_3 + D_4)T_4 + D_6T_6$$