Unit 3
Basic Computer Organization and Design

Introduction

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

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

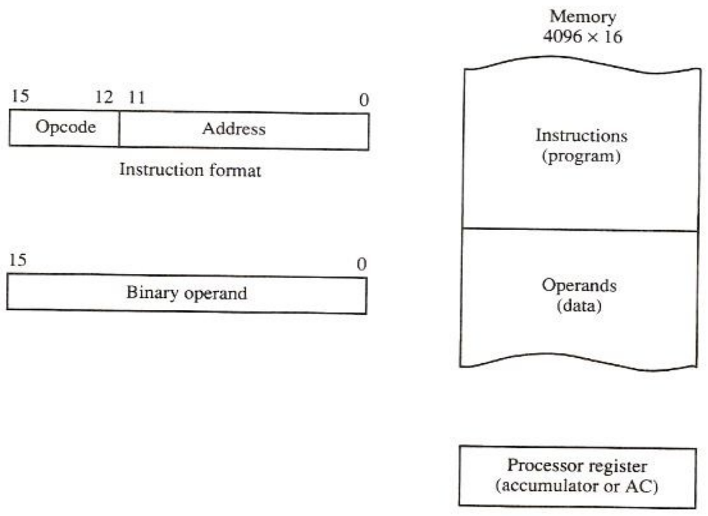
However, to understand how processors work, we will start with a simplified processor model. M. Morris Mano introduces a simple processor model; he calls it a "Basic Computer". The Basic Computer has two components, a processor and memory

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

Instruction code and Stored program organization

Question: *What do you understand by stored program organization?*
Question: *What is instruction and instruction format?*

Instruction code is a group of bits that instructs the computer to perform a specific operation. It is usually divided into parts. Most basic part is operation (**operation code**). Operation code is group of bits that defines operations as add, subtract, multiply, shift, complement etc. The instructions of a program, along with any needed data are stored in memory. The CPU reads the next instruction from memory. It is placed in an *Instruction Register* (IR). Control circuitry in control unit then translates the instruction into the sequence of microoperations necessary to implement it. Stored program concept is the ability to store and execute instructions.

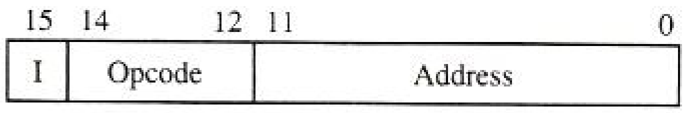


Instruction Format of Basic Computer

A computer instruction is often divided into two parts

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

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

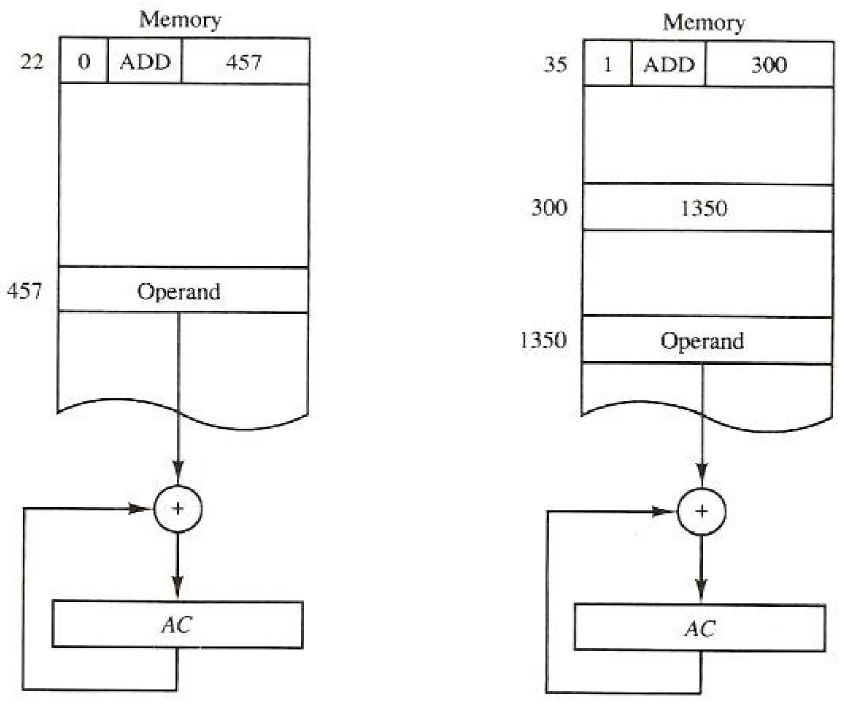


(a) Instruction format

Addressing Modes

The address field of an instruction can represent either

- Direct address: the address operand field is effective address (the address of the operand) or,
- Indirect address: the address in operand field contains the memory address where effective address resides.



(b) Direct address (c) Indirect address

Effective Address (EA): The address, where actual data resides is called effective address.

Basic Computer Registers

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

Symbol	Size	Register Name	Description
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

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

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

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

Common Bus system of Basic computer

The registers in the Basic Computer are connected using a bus. This gives a savings in circuitry over complete connections between registers. Three control lines, S2, S1, and S0 control which register the bus selects as its input.

S_2 S_1 S_0	Register
0 0 0	X (nothing)
0 0 1	AR
0 1 0	PC
0 1 1	DR
1 0 0	AC
1 0 1	IR
1 1 0	TR
1 1 1	Memory

Either one of the registers will have its load signal activated, or the memory will have its read signal activated which will determine where the data from the bus gets loaded. The 12-bit registers, AR and PC, have 0's loaded onto the bus in the high order 4 bit positions. When the 8-bit register OUTR is loaded from the bus, the data comes from the low order 8 bits on the bus.

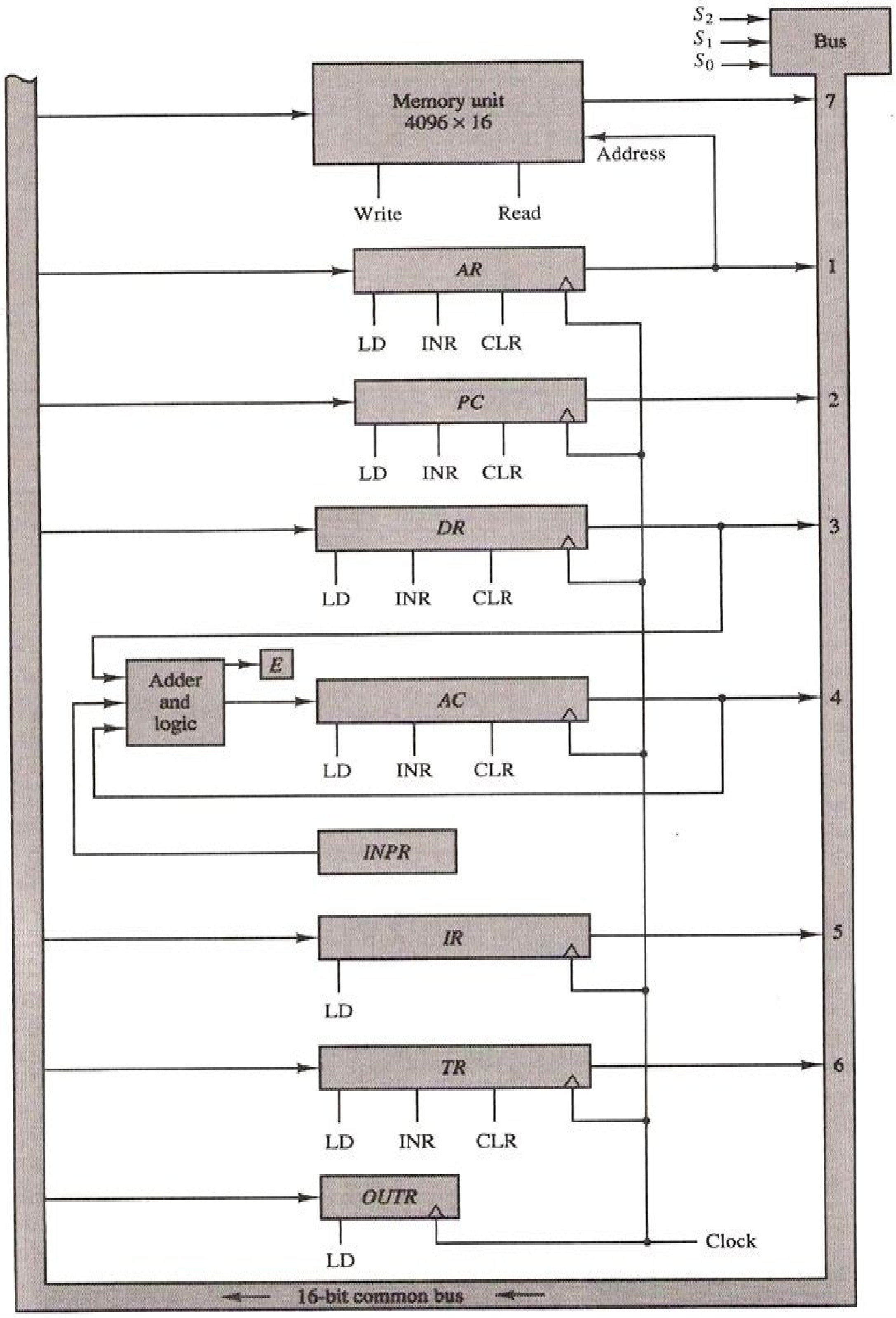


Fig: Basic computer resister connected in a common bus.

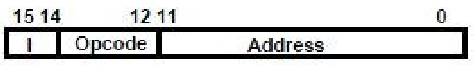
Instruction Formats of Basic Computer

Question: *What are different instruction format used basic computer*?

Question: *What is instruction set completeness? Is instruction set of basic computer complete?*

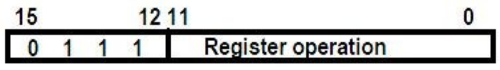
The basic computer has 3 instruction code formats. Type of the instruction is recognized by the computer control from 4-bit positions 12 through 15 of the instruction.

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



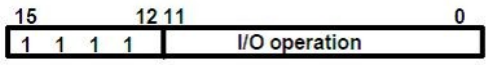
Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment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

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



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 instr. if AC is positive
SNA	7008	Skip next instr. if AC is negative
SZA	7004	Skip next instr. if AC is zero
SZE	7002	Skip next instr. if E is zero
HLT	7001	Halt computer

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



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

Instruction Set Completeness

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

Functional Instructions

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

Transfer Instructions

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

Control Instructions

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

Input/output Instructions

- Input and output
- Examples: INP, OUT

Instruction set of Basic computer is complete because:

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

Instruction Processing & Instruction Cycle (of Basic computer)

Control Unit

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

Hardwired Control

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

Microprogrammed Control

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

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

Question: *How basic computer translates machine instructions to control signals using hardwired control? Explain with block diagram. (OR Discuss hardwired control unit of basic computer?)*

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

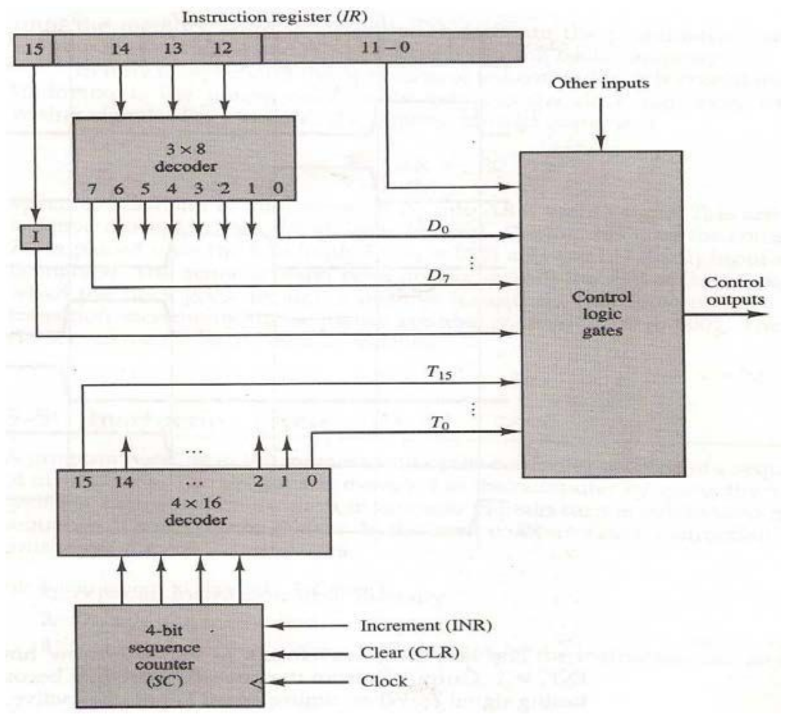


Fig: Control unit of a basic computer

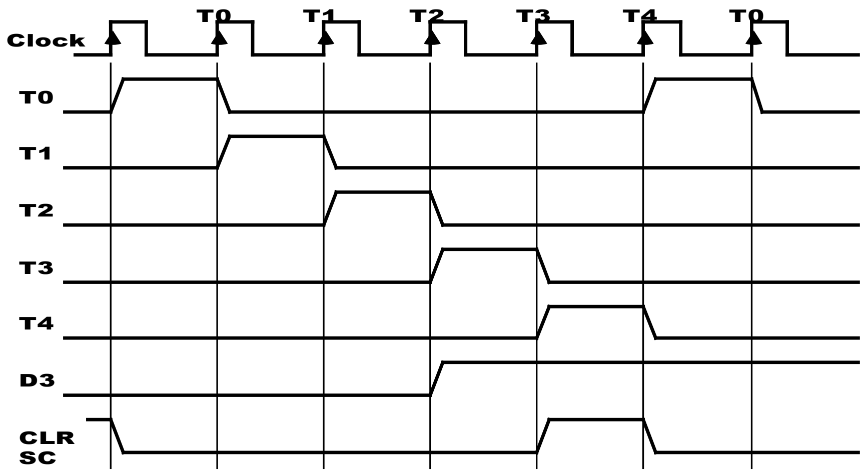
Mechanism:

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

This means instruction cycle of basic computer can not take more than 16 clock cycles.

Timing signals

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



Instruction cycle

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

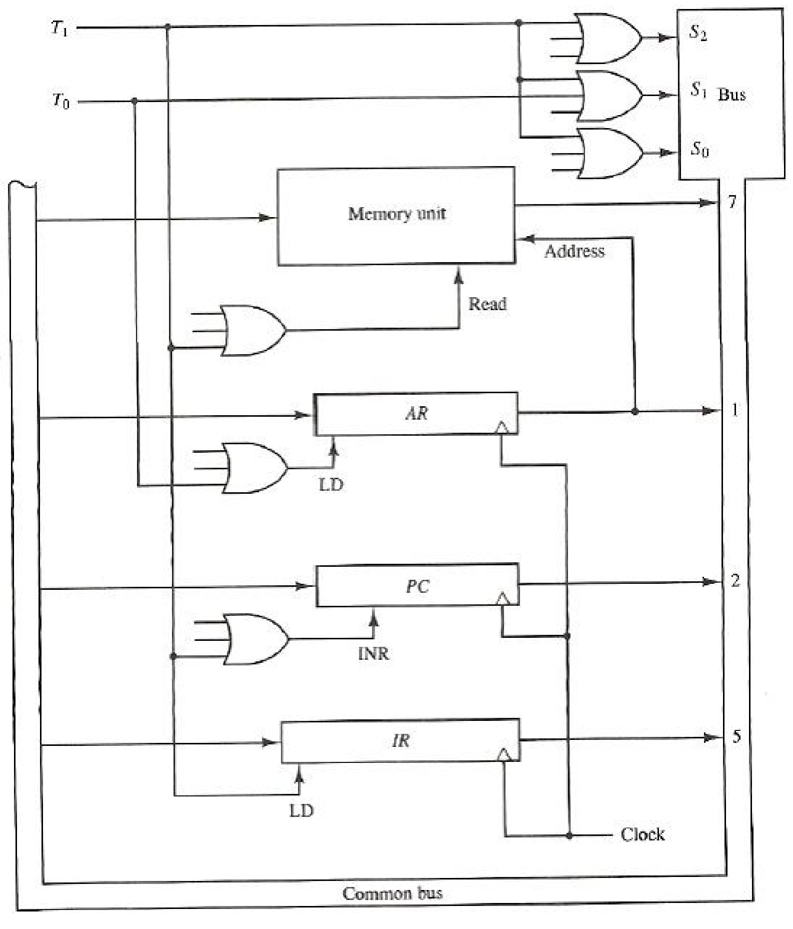
1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address
4. Execute the instruction

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

Fetch and decode

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

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



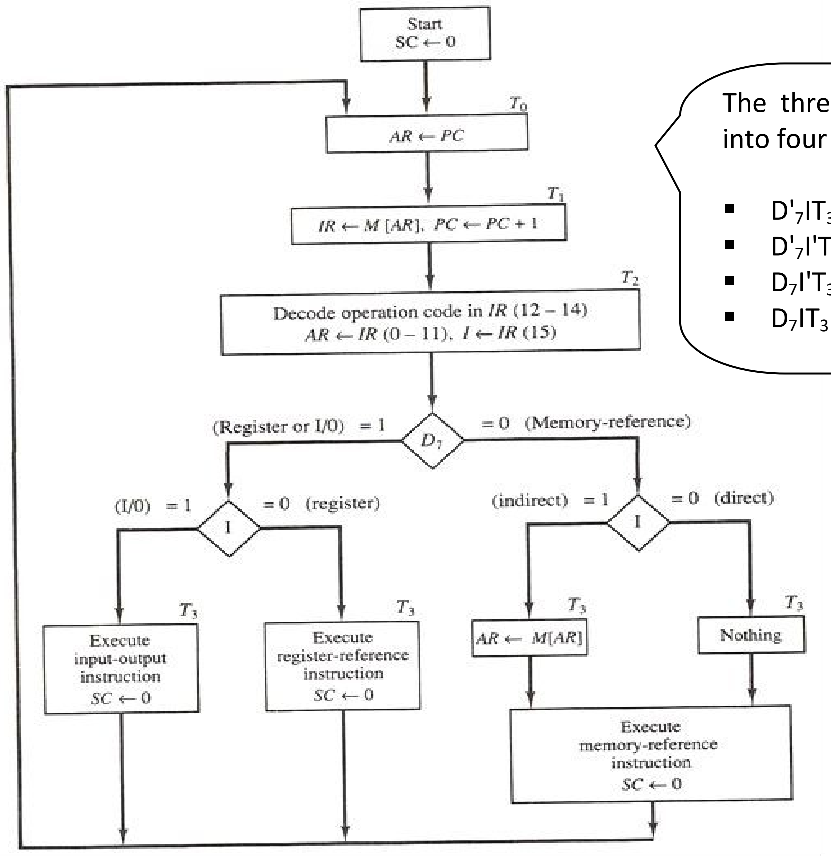
It is necessary to transfer the address from PC to AR during clock transition associated with the timing signal T_0. The instruction read from memory is then placed in IR with clock transition associated with the timing signal T_1. At the same time, PC is incremented by one to prepare for the next instruction in the program. At time T_2, the opcode in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR.

NOTE: SC is incremented after each clock pulse to produce the sequence T_0, T_1 and T_2.

Fig: Resister transfers for the fetch phase

Determine the type of the instruction

The timing signal that is active after decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory. Following flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after decoding.


Start SC ← 0
T0
AR ← PC
T1
IR ← M [AR], PC ← PC + 1
T2
Decode operation code 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
Execute memory-reference instruction SC ← 0

The three instruction types are subdivided into four separate paths:

- D'_7IT_3: AR ←M[AR] (Indirect address)
- $D'_7I'T_3$: Nothing
- $D_7I'T_3$: Execute register-reference instrs.
- D_7IT_3: Execute input-output instructions.

Fig: Flowchart for instruction cycle (Initial configuration)

Resister transfers needed for the execution of resister-reference and memory-reference instructions are explained below: (I/O instructions will be discussed later)

Resister-reference instructions:

Register Reference Instructions are recognized with

- $D_7 = 1$, $I = 0$
- Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR
- Execution starts with timing signal T_3

Let

r = D7 I'T3 => Common to all Register Reference Instruction

B_i = IR (i), i=0, 1, 2... 11. [Bit in IR(0-11) that specifies the operation]

CLA	rB_{11}:	AC ← 0, SC ← 0	Clear AC
CLE	rB_{10}:	E ← 0, SC ← 0	Clear E

CMA	rB_9:	AC ← AC', SC ← 0	Complement AC
CME	rB_8:	E ← E', SC ← 0	Complement E
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ← AC(0), SC ← 0	Circulate right
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15), SC← 0	Circulate Left
INC	rB_5:	AC ← AC + 1, SC ← 0	Increment AC
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1), SC ← 0	Skip if positive
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1), SC ← 0	skip if negative
SZA	rB_2:	if (AC = 0) then (PC ← PC+1), SC ← 0	skip if AC zero
SZE	rB_1:	if (E = 0) then (PC ← PC+1), SC ← 0	skip if E zero
HLT	rB_0:	S ← 0, SC ← 0 (S is a start-stop flip-flop)	Halt computer

Memory-reference instructions

- Once an instruction has been loaded to IR, it may require further access to memory to perform its intended function (direct or indirect).
- The effective address of the instruction is in the AR and was placed their during:
  - Time signal T2 when I = 0 or
  - Time signal T3 when I = 1
- Execution of memory reference instructions starts with the timing signal T4.
- Described symbolically using RTL.

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR]← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR]← PC, PC ← AR + 1
ISZ	D_6	M[AR]← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

AND to AC

This instruction performs the AND logical operation on pairs of bits on AC and the memory word specified by the effective address. The result is transferred to AC. Microoperations that execute these instructions are:

D_0T_4: DR ← M[AR] //Read operand

D_0T_5: AC ←AC ∧ DR, SC ← 0 //AND with AC

ADD to AC

D_1T_4: DR ← M[AR] //Read operand

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0 //Add to AC and stores carry in E

LDA: Load to AC

D_2T_4: DR ← M[AR] //Read operand

D_2T_5: AC ← DR, SC ← 0 //Load AC with DR

STA: Store AC

D_3T_4: M[AR] ← AC, SC ← 0 // store data into memory location

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0 //Branch to specified address

BSA: Branch and Save Return Address

D_5T_4: M[AR] ← PC, AR ← AR + 1 // save return address and increment AR

D_5T_5: PC ← AR, SC ← 0 // load PC with AR

ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR] //Load data into DR

D_6T_5: DR ← DR + 1 // Increment the data

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

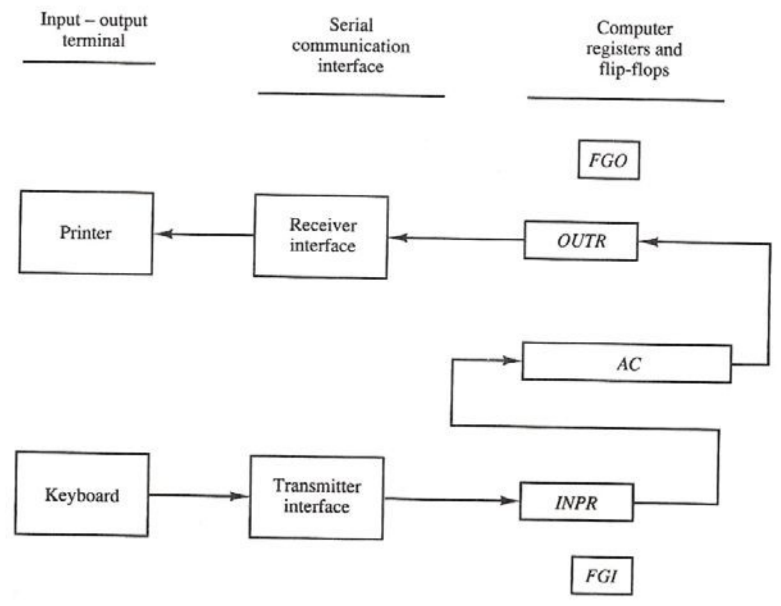
// if DR=0 skip next instruction by incrementing PC

Input-Output and Interrupt

In computer, instructions and data stored in memory come from some input device and Computational results must be transmitted to the user through some output device.

Input-output configuration

The terminal sends and receives serial information. Each quantity of information has 8 bits of an alphanumeric code. Two basic computer resisters INPR and OUTR communicate with a communication interfaces.

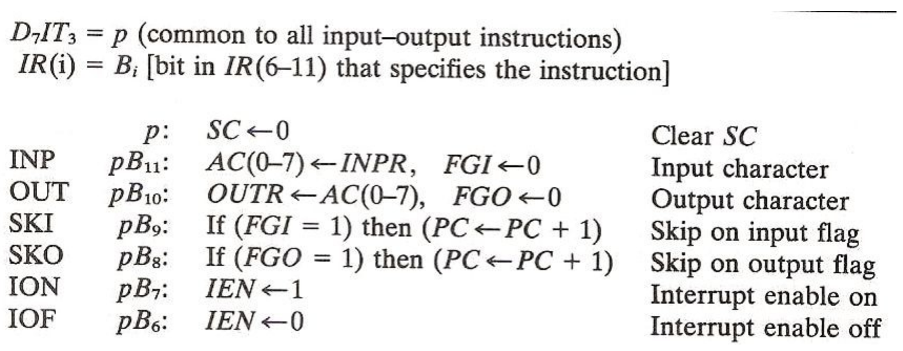


- **INPR:** Input register - 8 bits
- **OUTR:** Output register- 8 bits
- **FGI:** Input flag - 1 bit (Is a control flip-flop, set to 1 when new information is available)
- **FGO:** Output flag - 1 bit
- **IEN**: Interrupt enable - 1 bit

Fig: Input-output configuration

Scenario1: when a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1. As long as the flag is set, the information in INPR can not be changed by striking another key. The control checks the flag bit, if 1, contents of INPR is transferred in parallel to AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.
Scenario2: OUTR works similarly but the direction of information flow is reversed. Initially FGO is set to 1. The computer checks the flag bit; if it is 1, the information is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character and when operation is completed, it sets FGO to 1.

Input-output Instructions
I/O instructions are needed to transferring information to and form AC register, for checking the flag bits and for controlling the interrupt facility.



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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR, \quad FGI \leftarrow 0$	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

- Input and Output interactions with electromechanical peripheral devices require huge processing times compared with CPU processing times
  - I/O (milliseconds) versus CPU (nano/micro-seconds)
- Interrupts permit other CPU instructions to execute while waiting for I/O to complete
- 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.

Scenario3: consider a computer which completes instruction cycle in 1µs. Assume I/O device that can transfer information at the maximum rate of 10 characters/sec. Equivalently, one character every 100000µs. Two instructions are executed when computer checks the flag bit and decides not to transfer information. Which means computer will check the flag 50000 times between each transfer. Computer is wasting time while checking the flag instead of doing some useful processing task.

- IEN (Interrupt-enable flip-flop)
  - can be set and cleared by instructions
  - When cleared, the computer cannot be interrupted

Interrupt cycle

This is a hardware implementation of a branch and save return address operation.

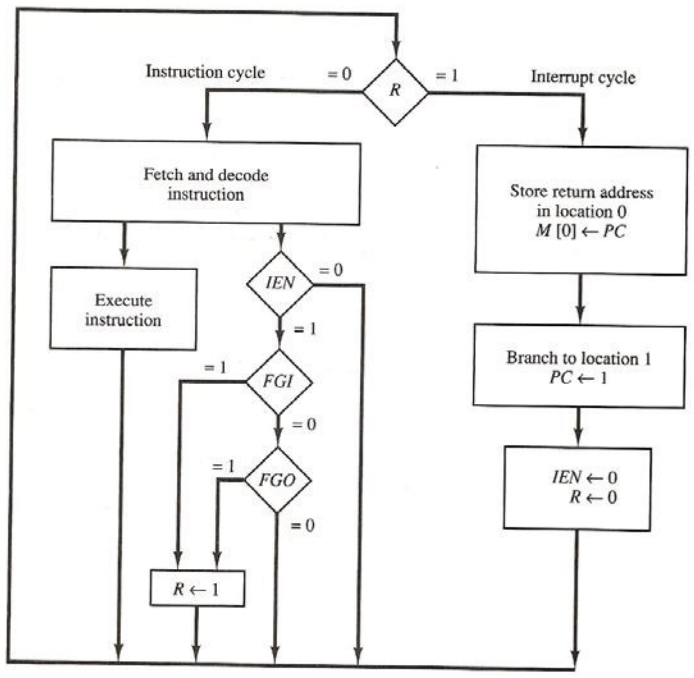
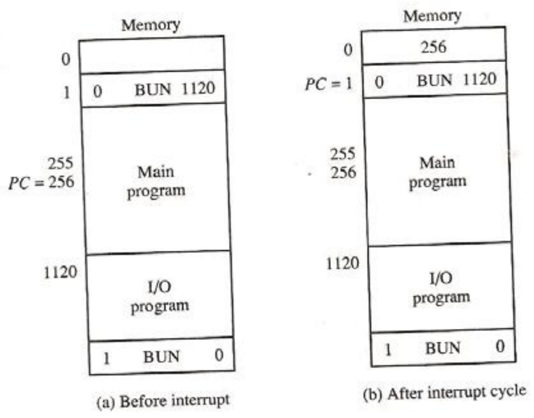


Fig: flowchart of interrupt cycle



- At the beginning of the instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

Fig: Demonstration of interrupt cycle

Resister transfer operations in interrupt cycle

Register Transfer Statements for Interrupt Cycle

- R F/F ← 1 if IEN (FGI + FGO) T0'T1'T2' ↔ $T_0'T_1'T_2'$ (IEN) (FGI + FGO): R ← 1

- The fetch and decode phases of the instruction cycle must be modified: Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle : RT_0: *AR ← 0, TR ← PC*

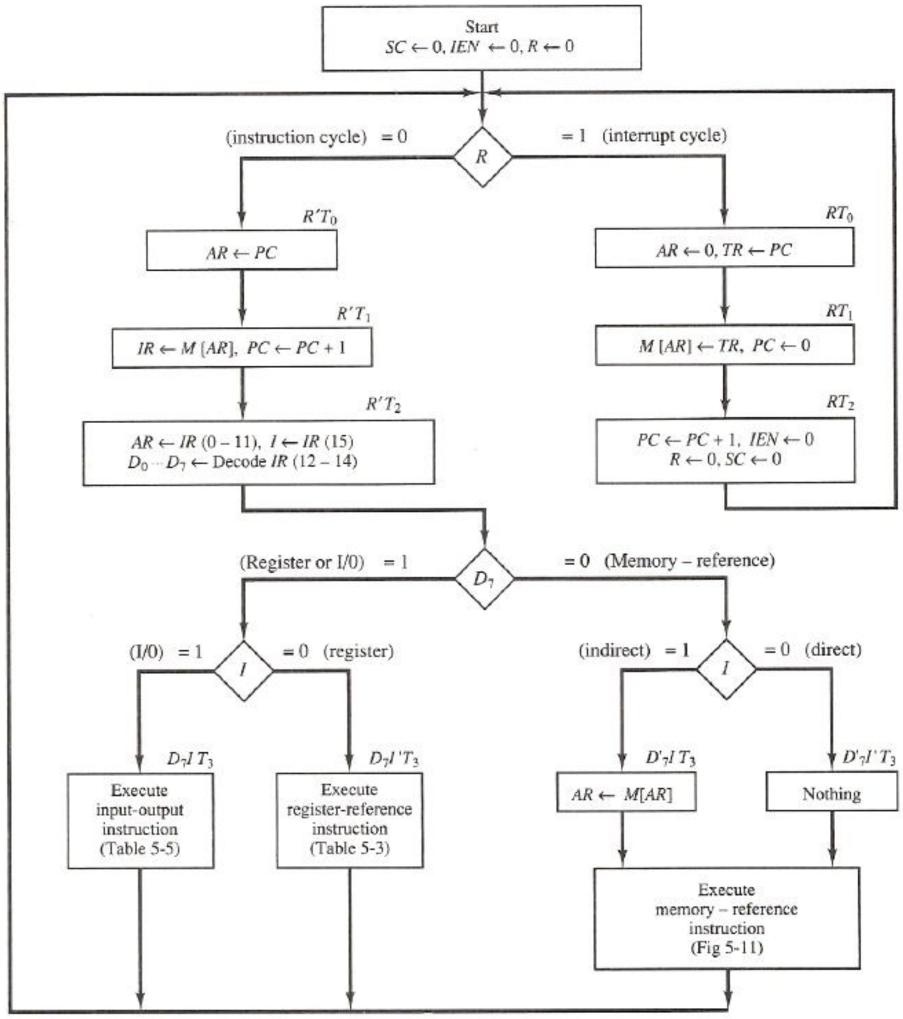
  RT_1: *M[AR] ← TR, PC ← 0*

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

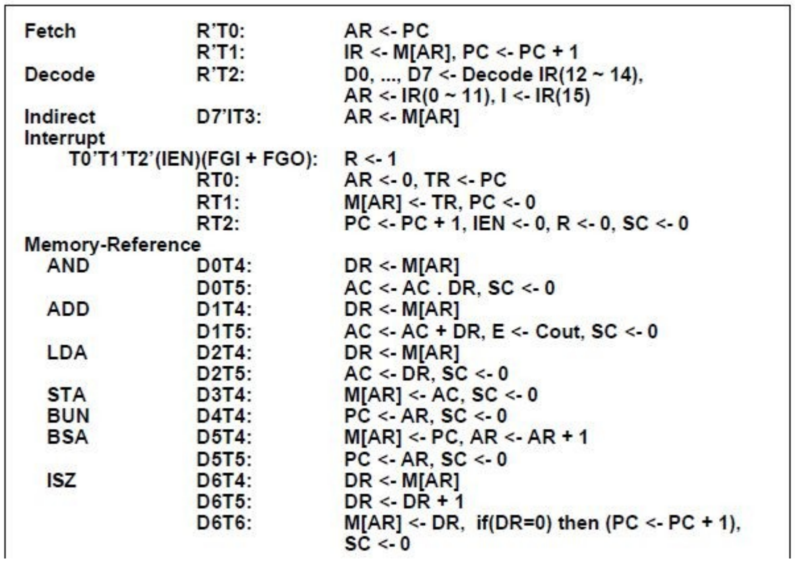
Complete computer description

Flowchart

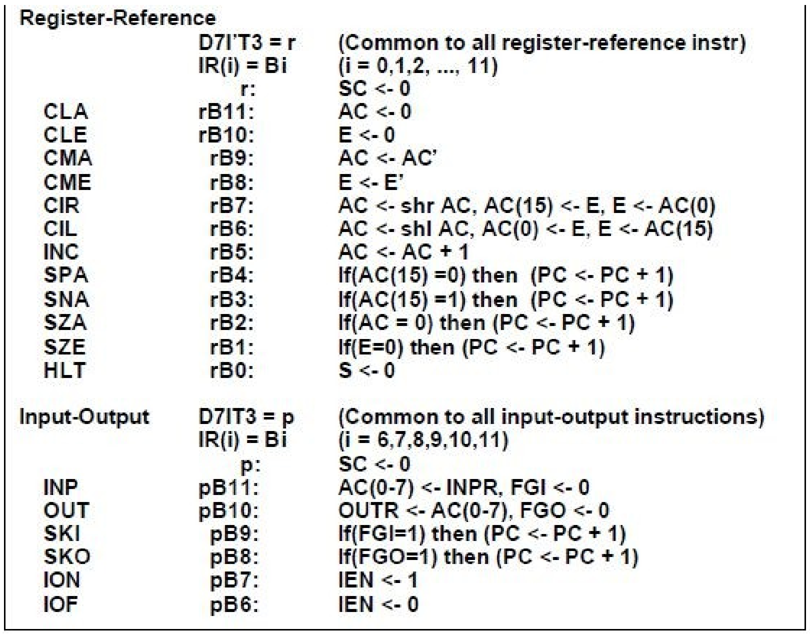
This is the final flowchart of the instruction cycle including interrupt cycle for the basic computer.



Microoperations



Fetch	R'T0:	AR <- PC
	R'T1:	IR <- M[AR], PC <- PC + 1
Decode	R'T2:	D0, ..., D7 <- Decode IR(12 ~ 14), AR <- IR(0 ~ 11), I <- IR(15)
Indirect	D7'IT3:	AR <- M[AR]
Interrupt		
T0'T1'T2'(IEN)(FGI + FGO):		R <- 1
	RT0:	AR <- 0, TR <- PC
	RT1:	M[AR] <- TR, PC <- 0
	RT2:	PC <- PC + 1, IEN <- 0, R <- 0, SC <- 0
Memory-Reference		
AND	D0T4:	DR <- M[AR]
	D0T5:	AC <- AC . DR, SC <- 0
ADD	D1T4:	DR <- M[AR]
	D1T5:	AC <- AC + DR, E <- Cout, SC <- 0
LDA	D2T4:	DR <- M[AR]
	D2T5:	AC <- DR, SC <- 0
STA	D3T4:	M[AR] <- AC, SC <- 0
BUN	D4T4:	PC <- AR, SC <- 0
BSA	D5T4:	M[AR] <- PC, AR <- AR + 1
	D5T5:	PC <- AR, SC <- 0
ISZ	D6T4:	DR <- M[AR]
	D6T5:	DR <- DR + 1
	D6T6:	M[AR] <- DR, if(DR=0) then (PC <- PC + 1), SC <- 0



Register-Reference		
	D7I'T3 = r	(Common to all register-reference instr)
	IR(i) = Bi	(i = 0,1,2, ..., 11)
	r:	SC <- 0
CLA	rB11:	AC <- 0
CLE	rB10:	E <- 0
CMA	rB9:	AC <- AC'
CME	rB8:	E <- E'
CIR	rB7:	AC <- shr AC, AC(15) <- E, E <- AC(0)
CIL	rB6:	AC <- shl AC, AC(0) <- E, E <- AC(15)
INC	rB5:	AC <- AC + 1
SPA	rB4:	If(AC(15) =0) then (PC <- PC + 1)
SNA	rB3:	If(AC(15) =1) then (PC <- PC + 1)
SZA	rB2:	If(AC = 0) then (PC <- PC + 1)
SZE	rB1:	If(E=0) then (PC <- PC + 1)
HLT	rB0:	S <- 0
Input-Output	D7IT3 = p	(Common to all input-output instructions)
	IR(i) = Bi	(i = 6,7,8,9,10,11)
	p:	SC <- 0
INP	pB11:	AC(0-7) <- INPR, FGI <- 0
OUT	pB10:	OUTR <- AC(0-7), FGO <- 0
SKI	pB9:	If(FGI=1) then (PC <- PC + 1)
SKO	pB8:	If(FGO=1) then (PC <- PC + 1)
ION	pB7:	IEN <- 1
IOF	pB6:	IEN <- 0

Design of Basic Computer (BC)

Hardware Components of BC

1. A memory unit: 4096 x 16.
2. Registers:
   AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
3. Flip-Flops(Status):
   I, S, E, R, IEN, FGI, and FGO
4. Decoders: A 3x8 Opcode decoder
   A 4x16 timing decoder
5. Common bus: 16 bits
6. Control logic gates
7. Adder and Logic circuit: Connected to AC

Control Logic Gates

Inputs:

1. Two decoder outputs
2. I flip-flop
3. IR(0-11)
4. AC(0-15)
   - To check if AC = 0
   - To detect sign bit AC(15)
5. DR(0-15)
   - To check if DR = 0
6. Values of seven flip-flops

Outputs:

1. Input Controls of the nine registers
2. Read and Write Controls of memory
3. Set, Clear, or Complement Controls of the flip-flops
4. S_2, S_1, S_0 Controls to select a register for the bus
5. AC, and Adder and Logic circuit

Control of resisters and memory
The control inputs of the resisters are LD (load), INR (increment) and CLR (clear).

- Address Resister (AR)
  To derive the gate structure associated with the control inputs of AR: we find all the statements that change the contents of AR.

$R'T_0$:	$AR \leftarrow PC$	LD(AR)
$R'T_2$:	$AR \leftarrow IR(0\text{-}11)$	LD(AR)
D'_7IT_3:	$AR \leftarrow M[AR]$	LD(AR)
RT_0:	$AR \leftarrow 0$	CLR(AR)
D_5T_4:	$AR \leftarrow AR + 1$	INR(AR)

⇩

$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$
$CLR(AR) = RT_0$
$INR(AR) = D_5T_4$

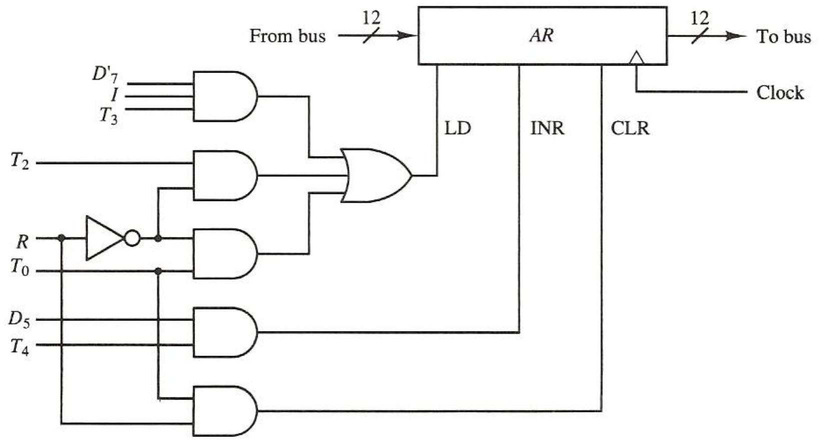


Fig: Control gates associated with AR

Similarly, control gates for the other resisters as well as the read and write inputs of memory can be derived. Viz. the logic gates associated with the read inputs of memory is derived by scanning all statements that contain a read operation. (Read operation is recognized by the symbol 🡐M[AR]).

$$\text{Read} = R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$$

The output of the logic gates that implement the Boolean expression above must be connected to the read input of memory.

Control of flip-flops
The control gates for the seven flip-flops can be determined in a similar manner.
Example:

- IEN(Interrupt Enable Flag)

$pB7$: $IEN \leftarrow 1$ (I/O Instruction)
$pB6$: $IEN \leftarrow 0$ (I/O Instruction)
RT_2: $IEN \leftarrow 0$ (Interrupt)

These three instructions can cause IEN flag to change its value.

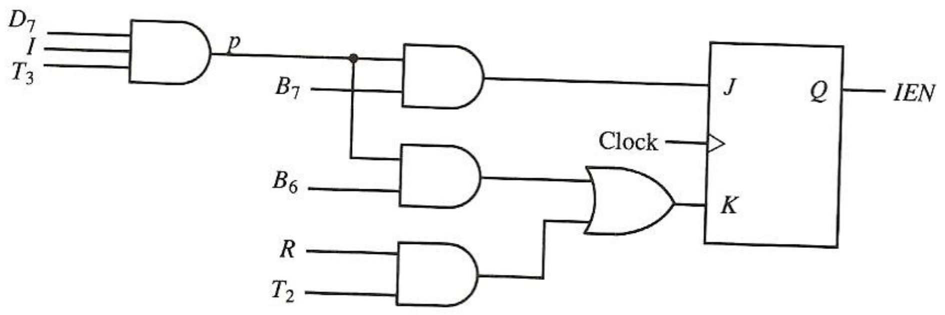


Fig: control inputs for IEN

Control of Common Bus

The 16-bit common bus is controlled by the selection inputs S_2, S_1 and S_0. Binary numbers for $S_2S_1S_0$ is associated with a Boolean variable x_1 through x_7, which must be active in order to select the resister or memory for the bus.

Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	Memory

Fig: Encoder for Bus Selection Circuit

Example: when x1 = 1, $S_2S_1S_0$ must be 001 and thus output of AR will be selected for the bus.

To determine the logic for each encoder input, it is necessary to find the control functions that place the corresponding resister onto the bus.

Example: to find the logic that makes x1 = 1, we scan all resister transfer statements that have AR as a source.

$$D_4T_4: \quad PC \leftarrow AR$$
$$D_5T_5: \quad PC \leftarrow AR$$

Therefore the Boolean function for x1 is,

$$x_1 = D_4T_4 + D_5T_5$$

Similarly, for memory read operation,

$$x_7 = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$$

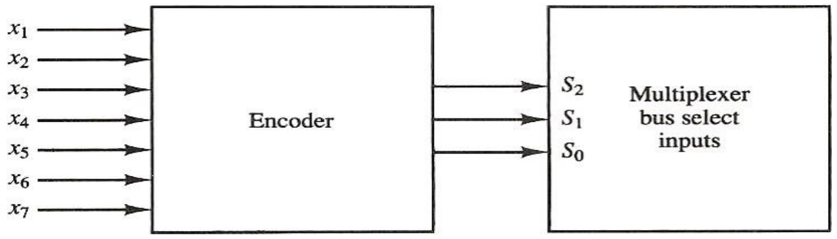
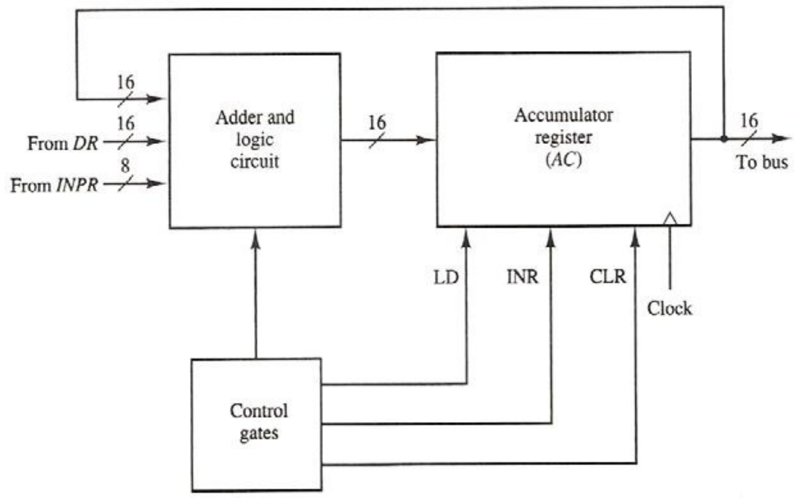
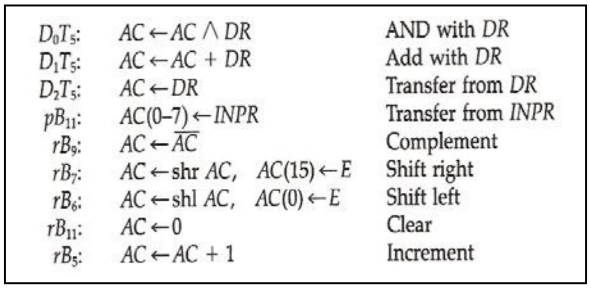


Fig: Encoder for bus selection inputs

Design of Accumulator Logic

To design the logic associated with AC, we extract all resister transfer statements that change the contents of AC. The circuit associated with the AC resister is shown below:





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

Fig: circuits associated with AC

Control of AC Resister

The gate structure that controls the LD, INR and CLR inputs of AC is shown below:

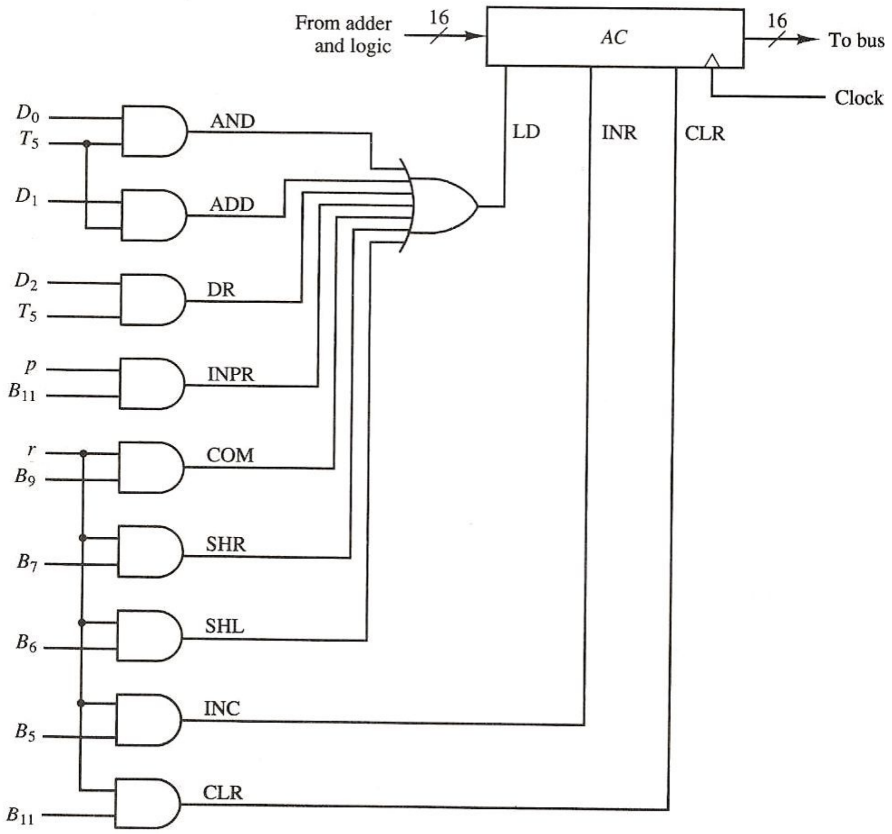


Fig: Gate structure for controlling LD, INR and CLR of AC

Adder and Logic Circuit

The adder and logic circuit can be subdivided into 16 stages, with each bit corresponding to one bit of AC.

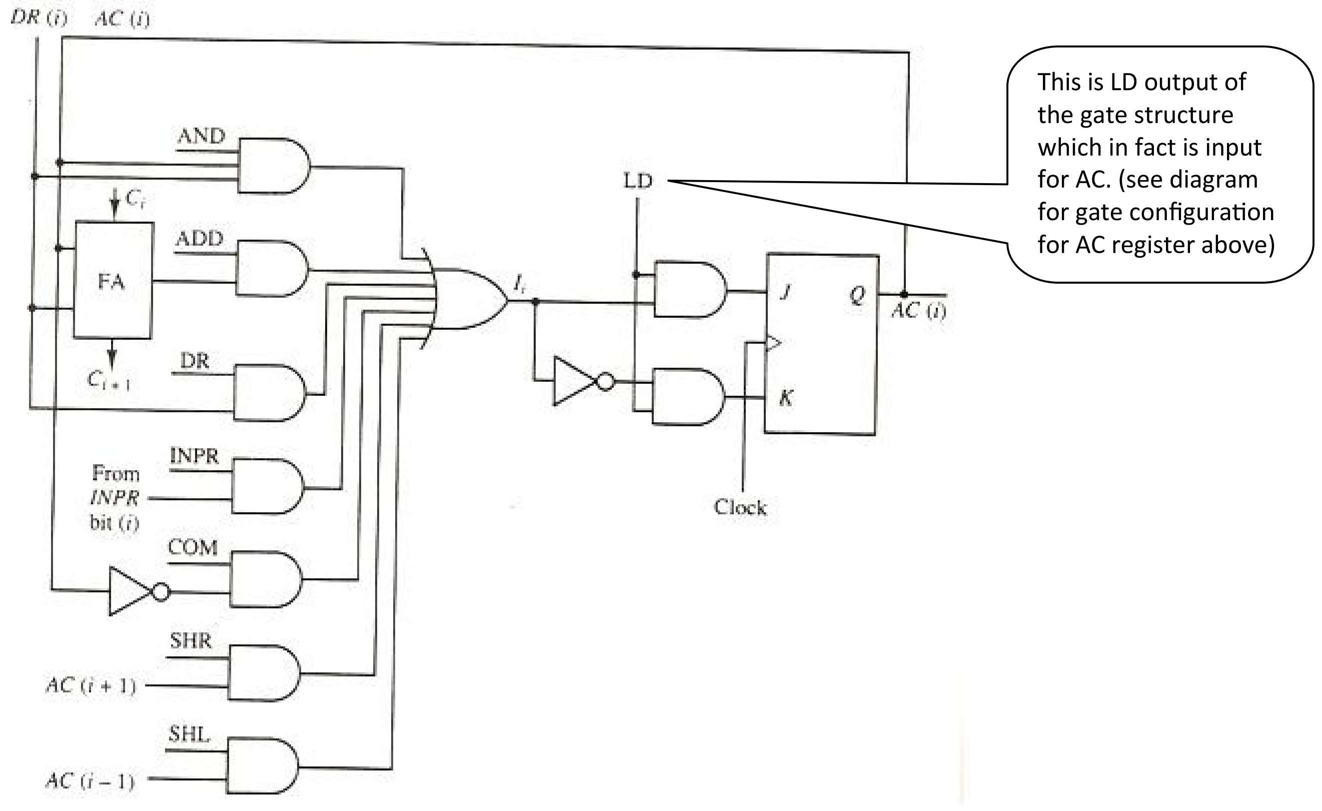

DR (i)
AC (i)
AND
C_i
FA
ADD
C_{i+1}
DR
INPR
From INPR bit (i)
COM
SHR
AC (i + 1)
SHL
AC (i − 1)
I_i
LD
J
Q
K
AC (i)
Clock
This is LD output of the gate structure which in fact is input for AC. (see diagram for gate configuration for AC register above)

Fig: One stage of adder and logic circuit

- One stage of the adder and logic circuit consists of seven AND gates, one OR gate and a full adder (FA) as shown above.
- The input is labeled I_i output AC(i).
- When LD input is enabled, the 16 inputs I_i for i = 0, 1, 2… 15 are transferred to AC(i).
- The AND operation is achieved by ANDing AC(i) with the corresponding bit in DR(i).
- The transfer from INPR to AC is only for bits 0 through 7.
- The complement microoperation is obtained by inverting the bit value in AC.
- Shift-right operation transfers bit from AC(i+1) and shift-left operation transfers the bit from AC(i-1).

HEY! : The complete adder and logic circuit consists of 16 stages connected together.

EXERCISES: Textbook chapter 5→ 5.1, 5.2, 5.10, 5.23

5.1(solution)

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

$64 = 2^6$

(a) Address : 18 bits
Register code : 6 bits
Indirect bit : 1 bit

$\overline{}$ 25 $\qquad 32 - 25 = 7$ bits for opcode.

(b)

1	7	6	18
I	opcode	Register	Address

= 32 bits

(c) Data ; 32 bits ; address : 18 bits.

5-2

A direct address instruction needs two references to memory : (1) Read instruction ; (2) Read operand.
An indirect address instruction needs three references to memory : (1) Read instruction ; (2) Read effective address ; (3) Read operand.

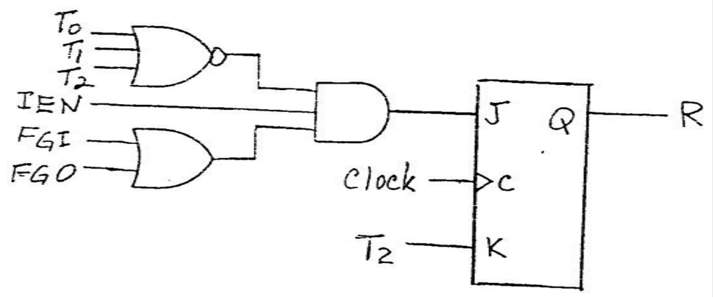
5.10 (Solution)

	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

5.23 (Solution)

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

$RT_2 : \quad R \leftarrow 0$



T_0
T_1
T_2
IEN
FGI
FGO
Clock
T_2
J
Q
C
K
R