UNIT - III

CPU control unit design: Instruction Codes - Computer Registers - Computer Instructions - Timing and Control - Instruction Cycle - Memory-Reference Instructions - Input-Output and Interrupt - Design of Basic Computer - Design of Accumulator Logic.

Microprogrammed Control: Control Memory - Address Sequencing- Microprogram Example - Design of Control Unit.

Pipelining: Basic Concept - Pipeline Organization - Pipelining Issues – Data Dependencies - Memory Delays- Branch Delays - Superscalar Operation - Pipelining in CISC Processors.

Parallel Processors: Hardware Multithreading - Vector (SIMD) Processing - Cache Coherence

CPU control unit design

- There are two major types of control organization: hardwired control and microprogrammed control.
- In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.
- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed.
- In the microprogrammed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
- In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

Instruction Codes

- User of a computer can control the process by means of a program.
- A *program* is a set of instructions that specify the operations, operands, and the processing sequence.
- A *computer instruction* is a binary code that specifies a sequence of micro-operations for the computer.
- Each computer has its unique instruction set. Instruction codes and data are stored in memory.
- Computer reads each instruction from memory and places it in a control register
- Control unit interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations

- Instruction code is a group of bits that instructs the computer to perform a specific operation (sequence of microoperations).

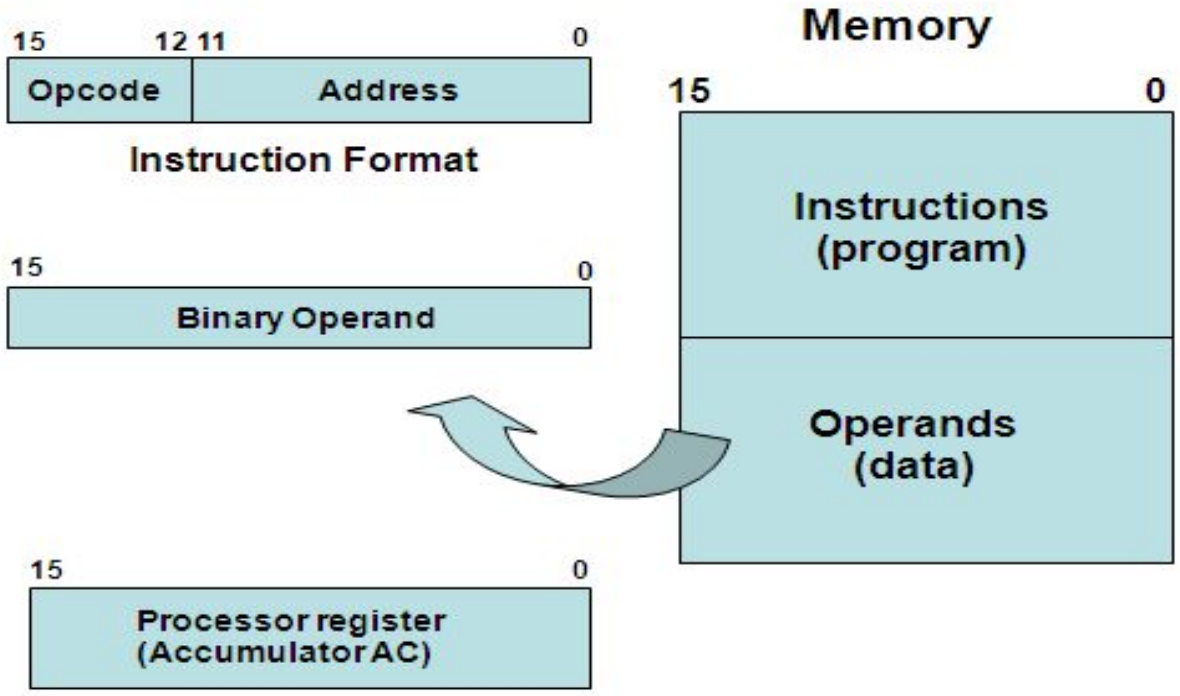
- Operation code of an instruction is a group of bits that defines certain operations such as add, subtract, shift, and complement.

- Number of bits required for the operation code depends on the total number of operations available in the computer

- 2^n (or little less) distinct operations n bit operation code

Stored Program Organization

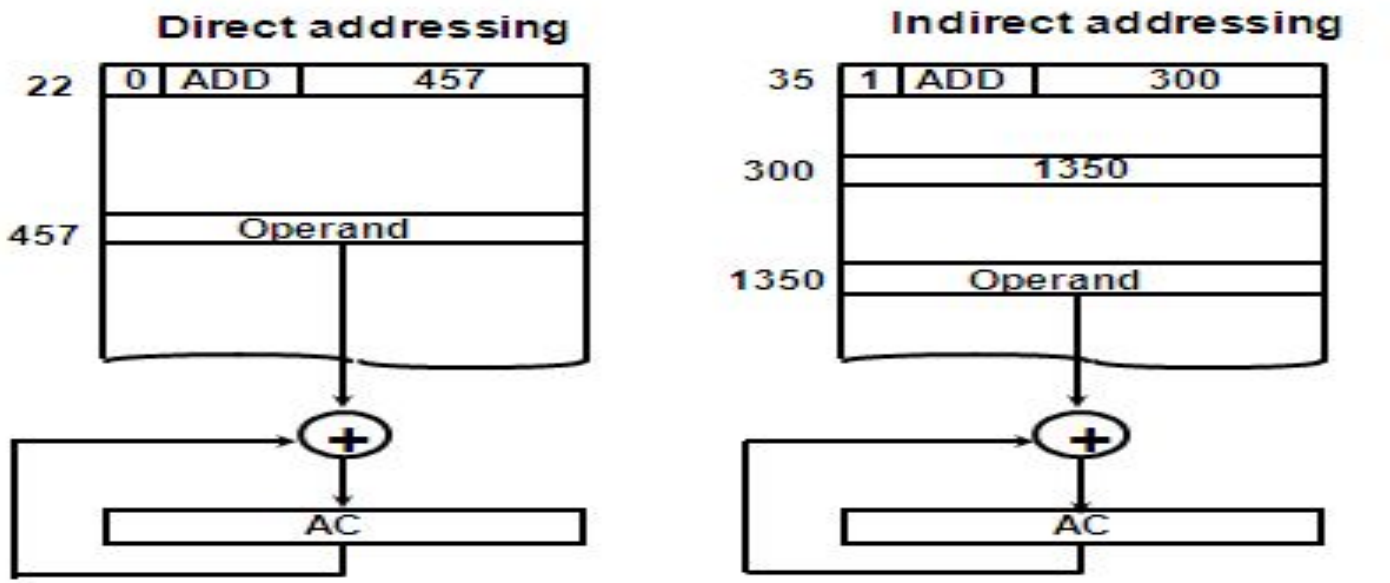
- Instruction code is usually divided into operation code, operand address, addressing mode, etc.
- Simplest way to organize a computer is to have one processor register (accumulator AC) and an instruction code format with two parts (op code, address)



ADDRESSING MODES

Direct address: the address in memory of the data to use (the address of the operand)

Indirect address: the address in memory of the address in memory of the data to use



PROCESSOR REGISTERS

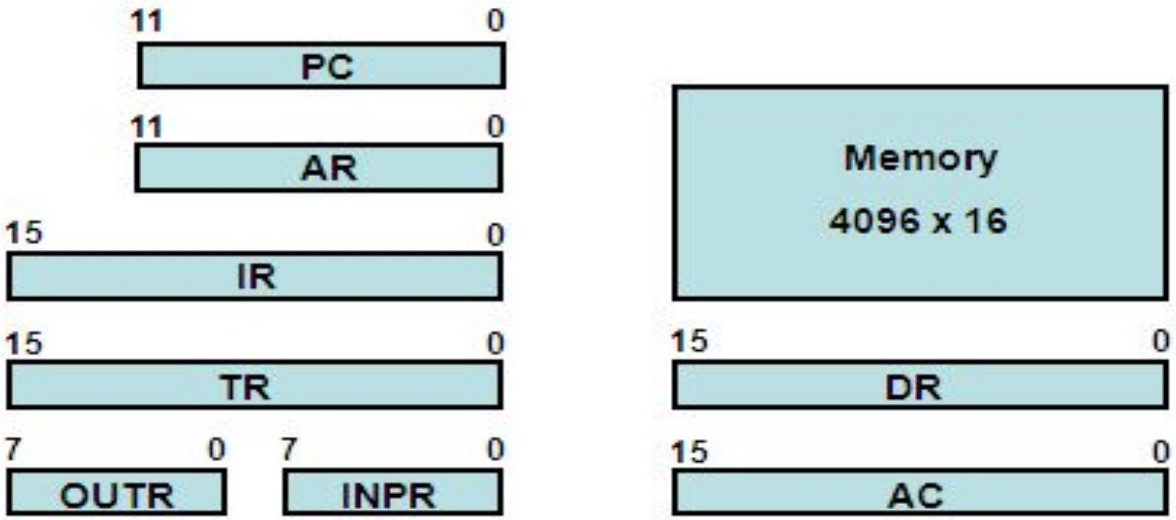
- Processor has many registers to hold instructions, addresses, data, etc
- Processor has a register, the Program Counter (PC) that holds the memory address of the next instruction
- Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits
- In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The Address Register (AR) is used for this and the AR is a 12 bit register in the Basic Computer
- When an operand is found, using either direct or indirect addressing, it is placed in the Data Register (DR). The processor then uses this value as data for its operation
- Basic Computer has a single general purpose register – the Accumulator (AC)

- The significance of a general purpose register is that it can be used for loading operands and storing results

  e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location

- Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR)
- 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
- *Input Register* (INPR) holds an 8 bit character gotten from an input device
- *Output Register* (OUTR) holds an 8 bit character to be send to an output device

Registers in the Basic Computer



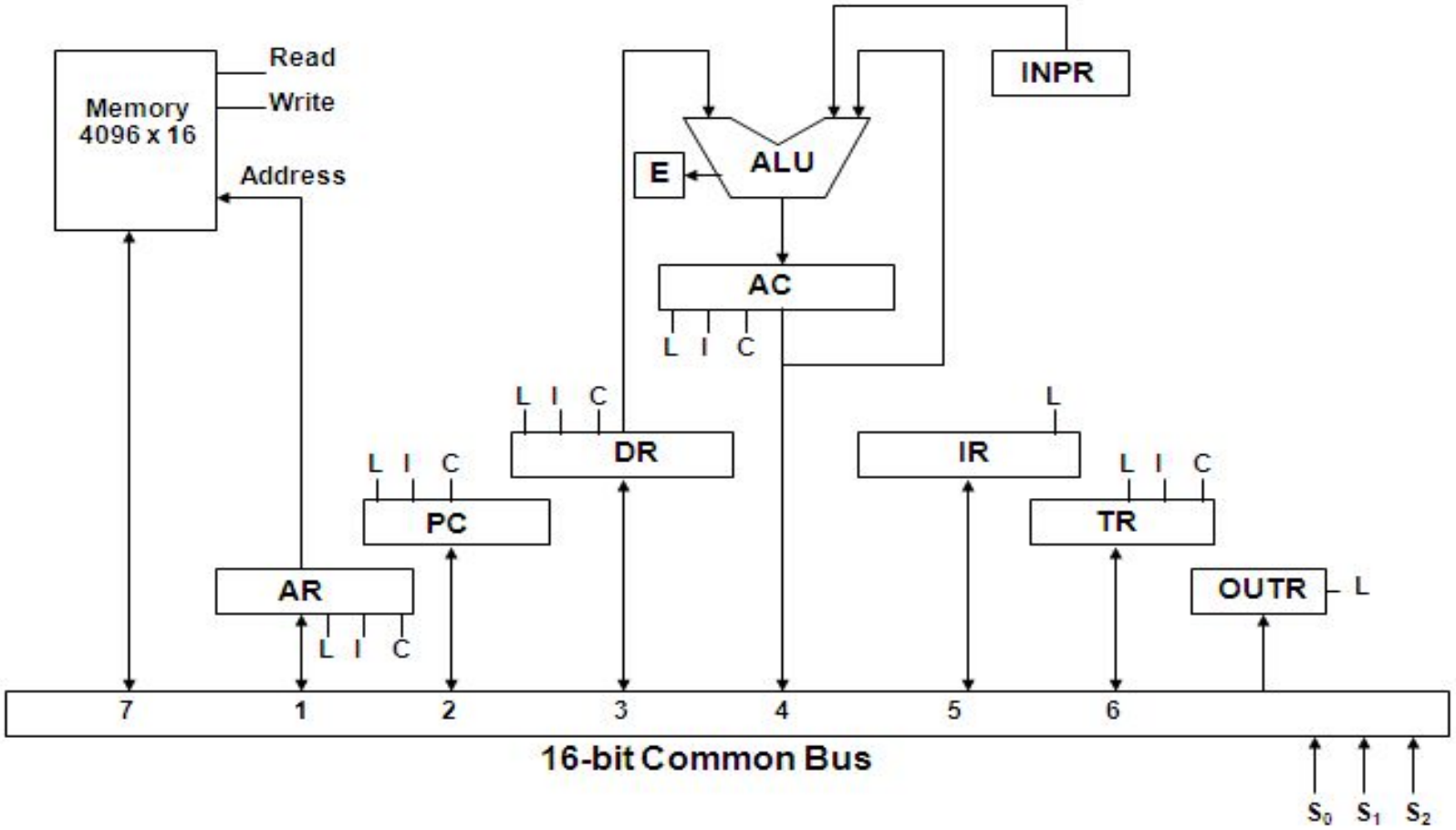
List of BC Registers

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

The registers in the Basic Computer are connected using a bus

This gives a savings in circuitry over complete connections between registers



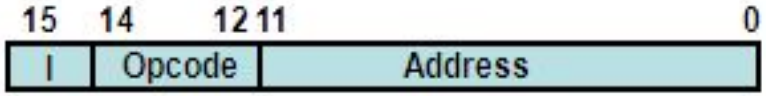
S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

- Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input
- Either one of the registers will have its load signal activated, or the memory will have its read signal activate will determine where the data from the bus gets loaded
- 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

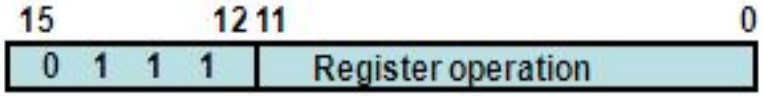
Computer Instructions

Basic Computer Instruction code format

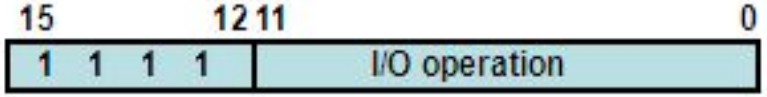
Memory-Reference Instructions (OP-code = 000 ~ 110)



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



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



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
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
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

Set of instructions using which user can construct machine language programs to evaluate any computable function.

Instruction Types

Functional Instructions

- Arithmetic, logic, and shift instructions
- ADD, CMA, INC, CIR, CIL, AND, CLA (other than ADD/AND?)

Transfer Instructions

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

Control Instructions

- Program sequencing and control
- BUN, BSA, ISZ

Input/Output Instructions

- Input and output
- INP, OUT

CONTROL UNIT

Control unit (CU) of a processor translates from machine instructions to the control signals (for the microoperations) that implement them.

Control units are implemented in one of two ways

***Hardwired* Control**

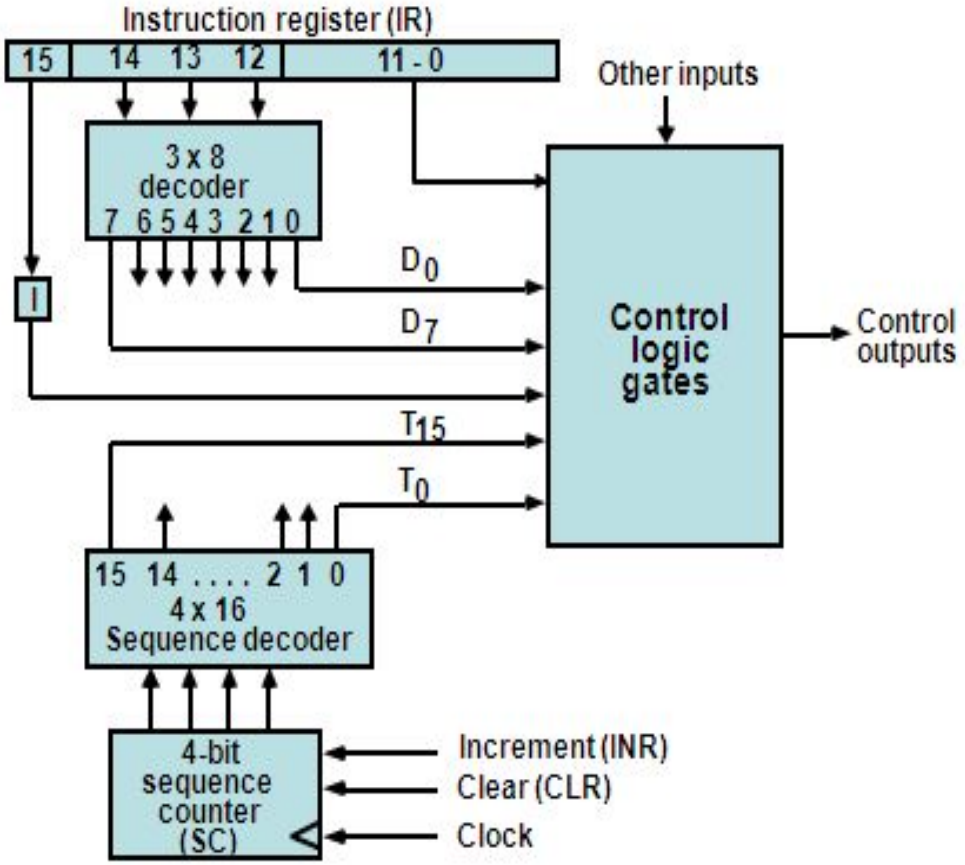
CU is made up of sequential and combinational circuits to generate the control signals

***Microprogrammed* Control**

A control memory on the processor contains microprograms that activate the necessary control signals

We will consider a hardwired implementation of the control unit for the Basic Computer

TIMING AND CONTROL



Hardwired Control Organization

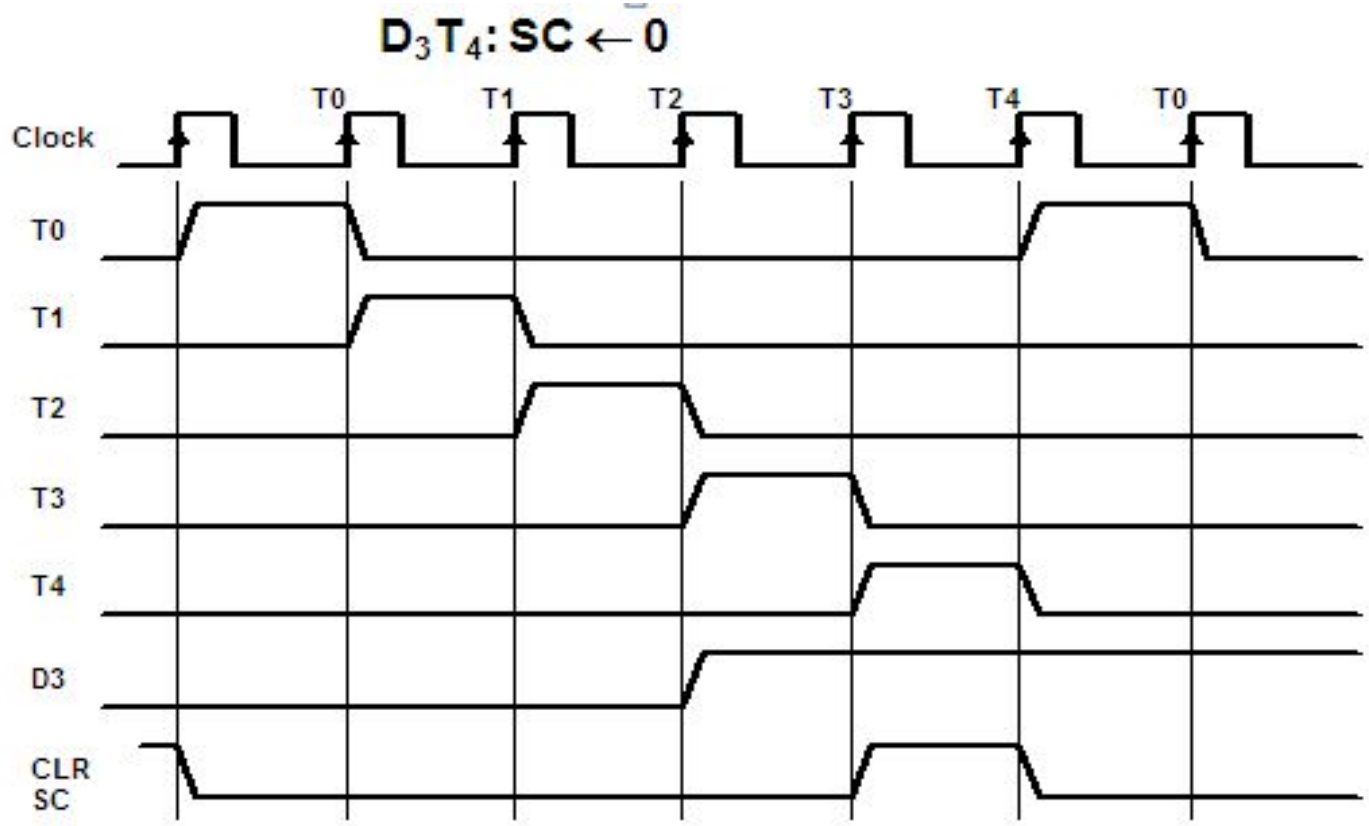
TIMING SIGNALS

Generated by 4-bit sequence counter and 4×16 decoder

- The SC can be incremented or cleared.

- Example: $T_0, T_1, T_2, T_3, T_4, T_0, T_1, \ldots$

  Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.



INSTRUCTION CYCLE

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

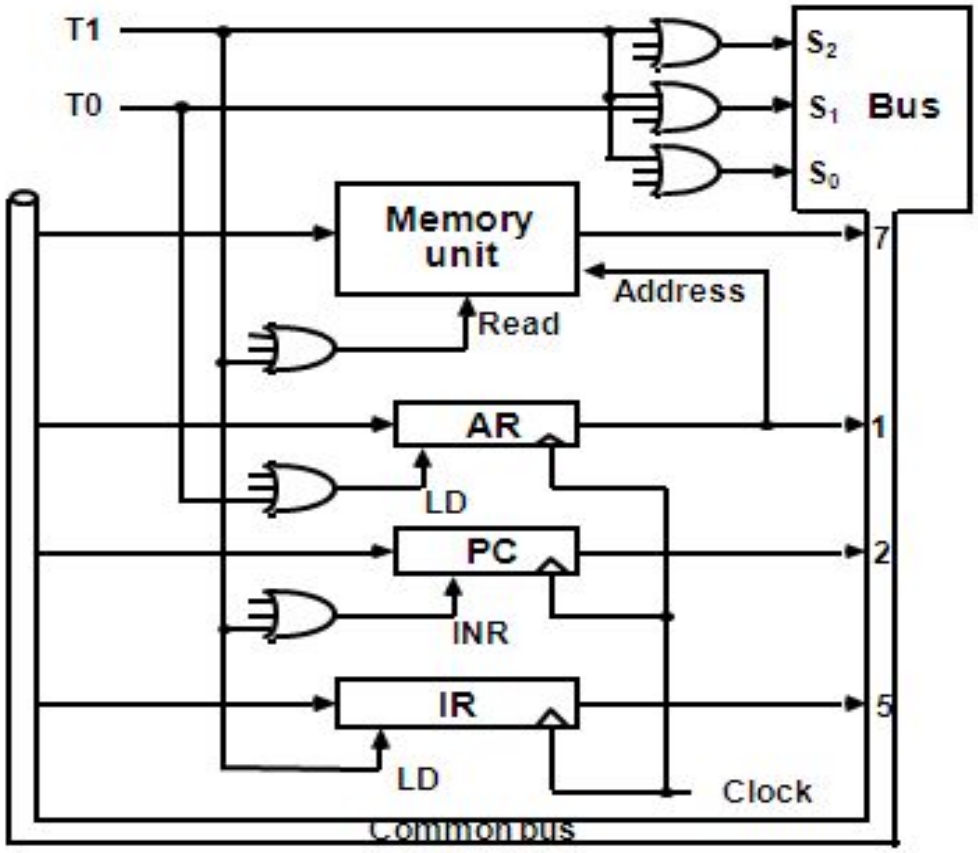
1. Fetch an instruction from memory
2. Decode the instruction and calculate effective address (EA)
3. Read the EA from memory if the instruction has an indirect address (Fetch operand)
4. Execute the instruction

After an instruction is executed, the cycle starts again at step 1, for the next instruction

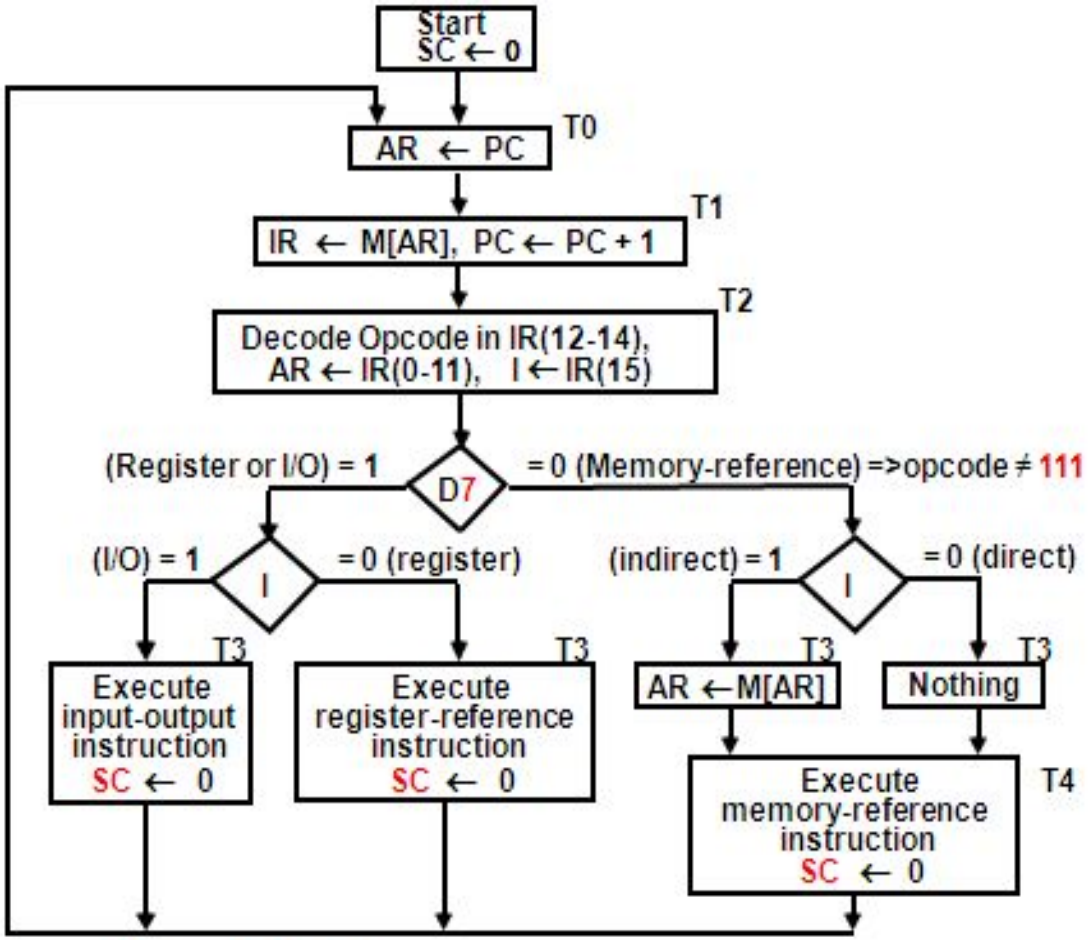
Note: Every different processor has its own (different) instruction cycle

FETCH and DECODE

T0: AR ← PC ($S_2S_1S_0$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 ($S_2S_1S_0$=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



DETERMINE THE TYPE OF INSTRUCTION



D'_7IT_3: $AR \leftarrow M[AR]$

$D'_7I'T_3$: Nothing

$D_7I'T_3$: Execute a register-reference instr.

D_7IT_3: Execute an input-output instr.

REGISTER REFERENCE INSTRUCTIONS

Register Reference Instructions are identified when

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

$r = D_7 I' T_3$ => Register Reference Instruction

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

	r:	SC ← 0
CLA	rB_{11}:	AC ← 0
CLE	rB_{10}:	E ← 0
CMA	rB_9:	AC ← AC'
CME	rB_8:	E ← E'
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15)
INC	rB_5:	AC ← AC + 1
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1)
SZA	rB_2:	if (AC = 0) then (PC ← PC+1)
SZE	rB_1:	if (E = 0) then (PC ← PC+1)
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)

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 $PC \leftarrow PC+1$

Effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1.

- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

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 store carry in E

MEMORY REFERENCE INSTRUCTIONS

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

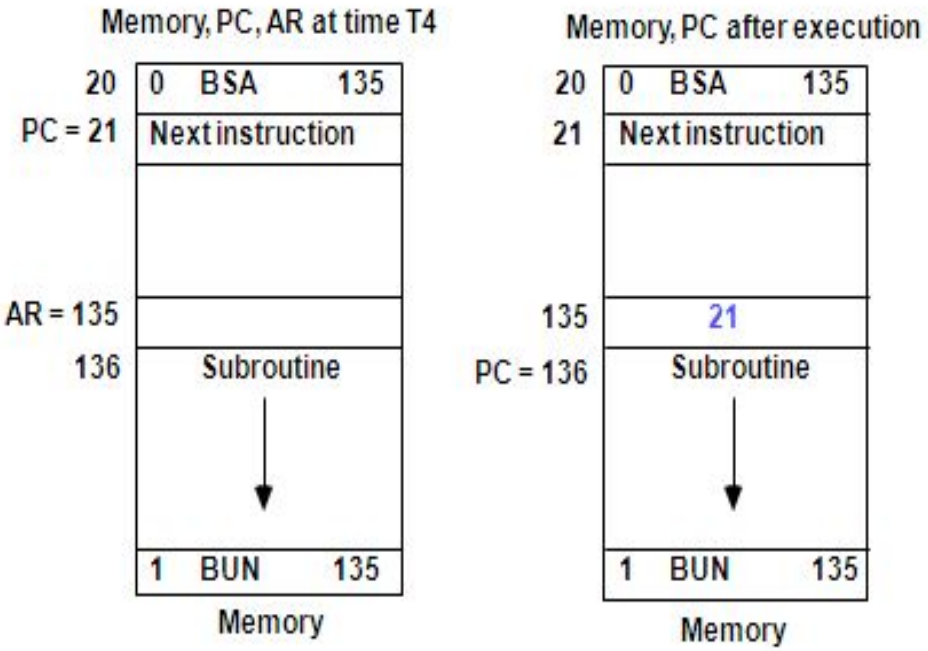
STA: Store AC

D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address



MEMORY REFERENCE INSTRUCTIONS

BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1

D_5T_5: PC ← AR, SC ← 0

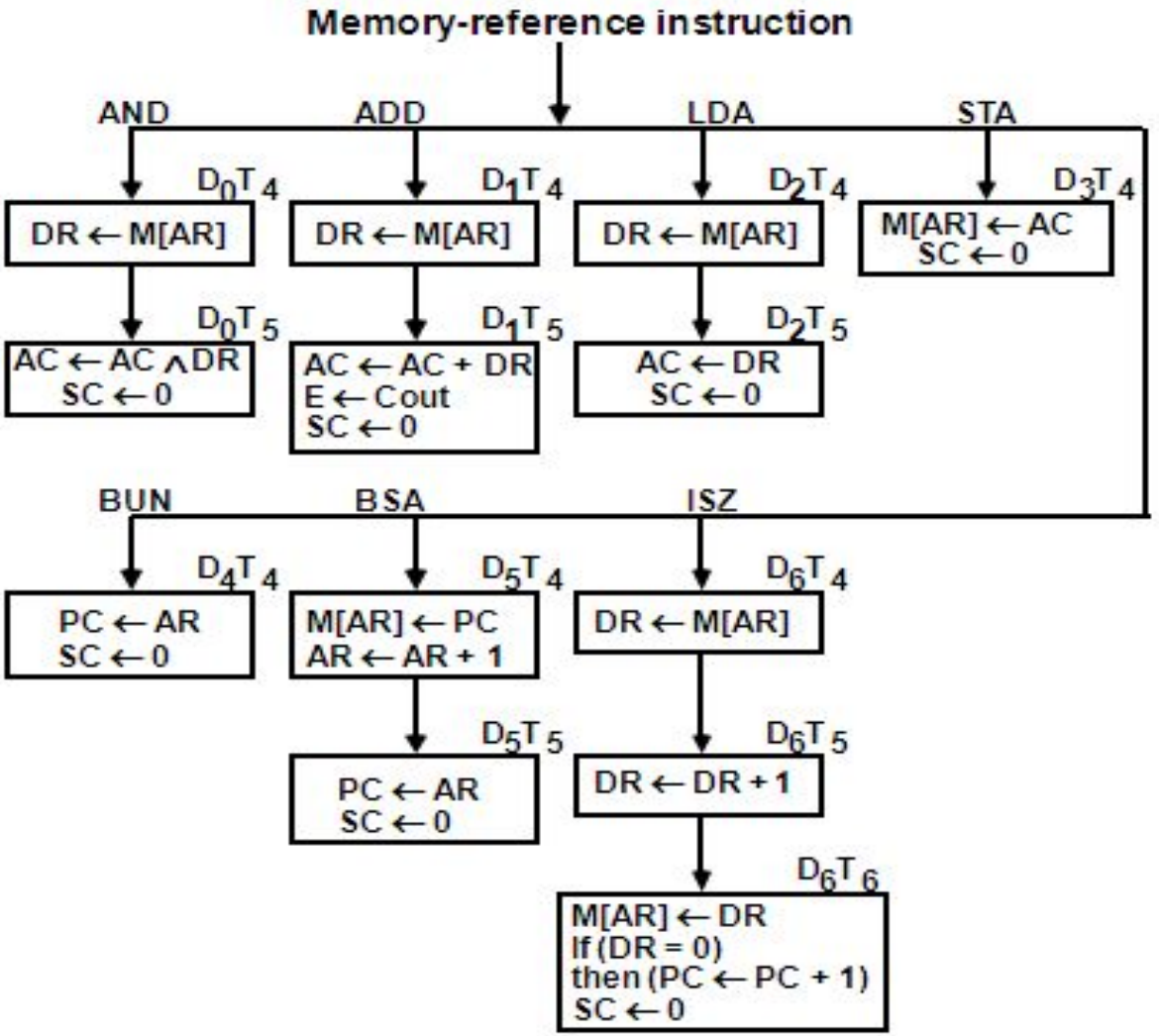
ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR + 1

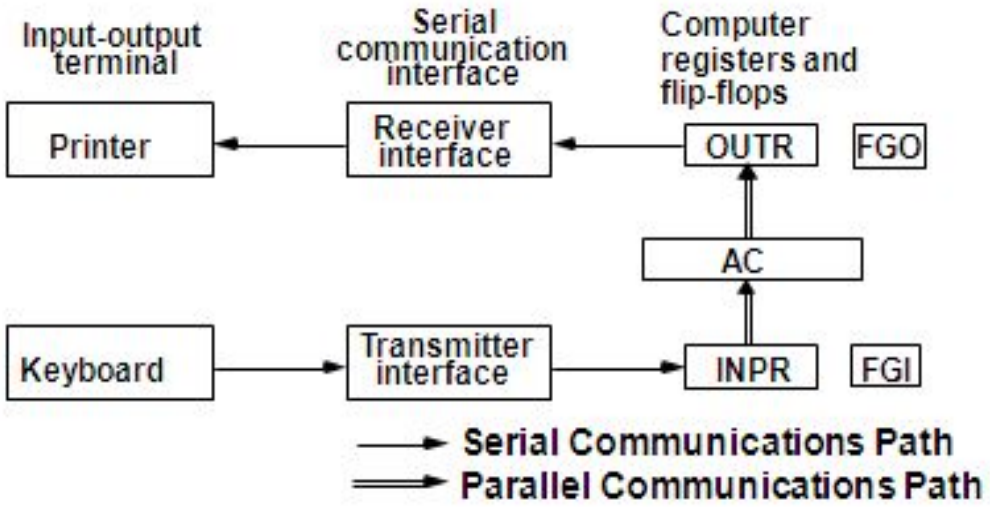
D_6T_4: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS



INPUT-OUTPUT AND INTERRUPT

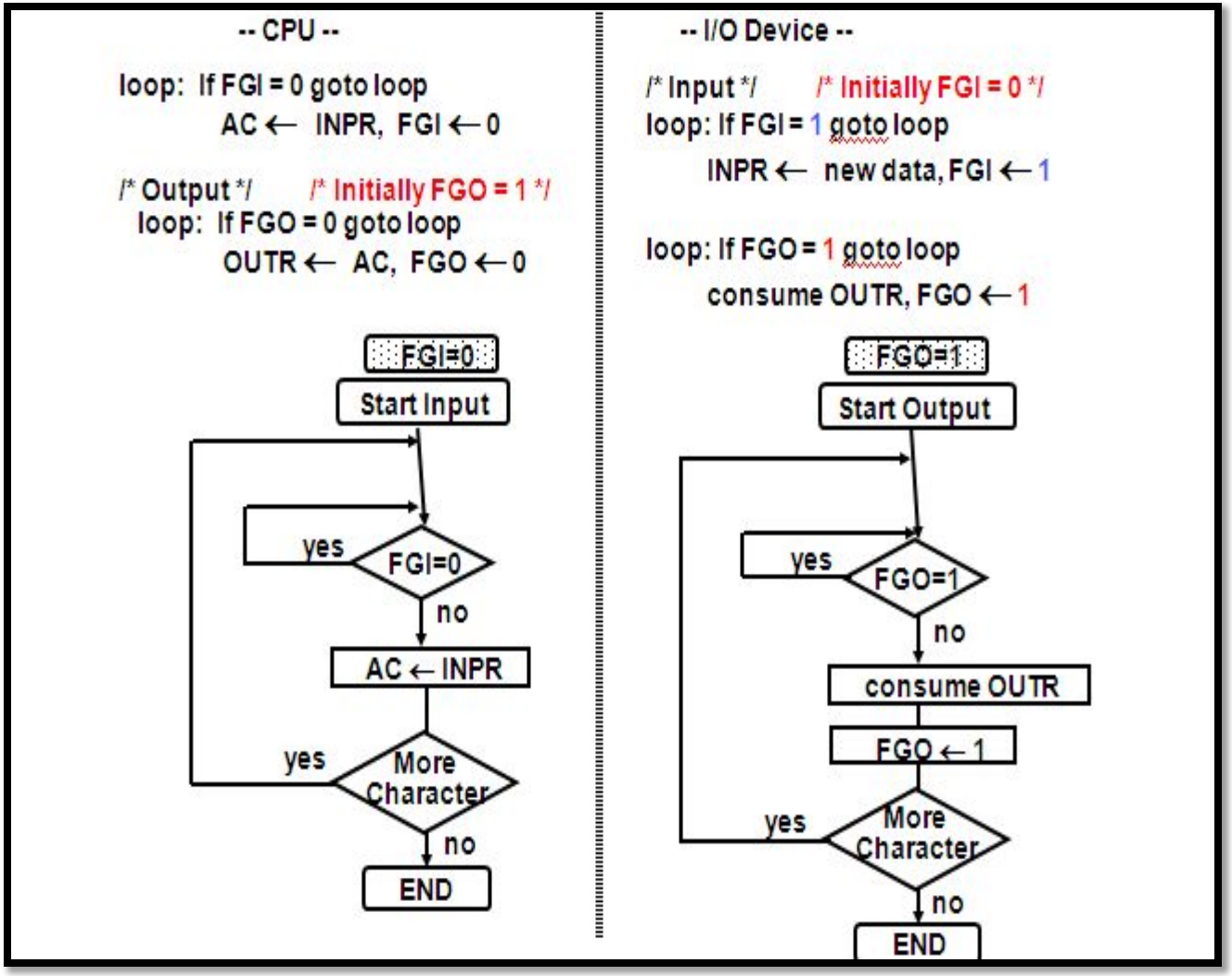
- Input-Output Configuration



INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing difference between I/O device and the computer

PROGRAM CONTROLLED DATA TRANSFER



INPUT-OUTPUT INSTRUCTIONS

$D_7IT_3 = p$
$IR(i) = B_i$, $i = 6, \ldots, 11$

	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

PROGRAM-CONTROLLED INPUT/OUTPUT

- Program-controlled I/O

  - Continuous CPU involvement
  - I/O takes valuable CPU time
  - CPU slowed down to I/O speed
  - Simple
  - Least hardware

Input

```
LOOP    SKI   DEV
        BUN   LOOP
        INP   DEV
```

Output

```
        LDA   DATA
LOOP    SKO   DEV
        BUN   LOOP
        OUT   DEV
```

INTERRUPT INITIATED INPUT/OUTPUT

- Open communication only when some data has to be passed --> *interrupt*.

- The I/O interface, instead of the CPU, monitors the I/O device.

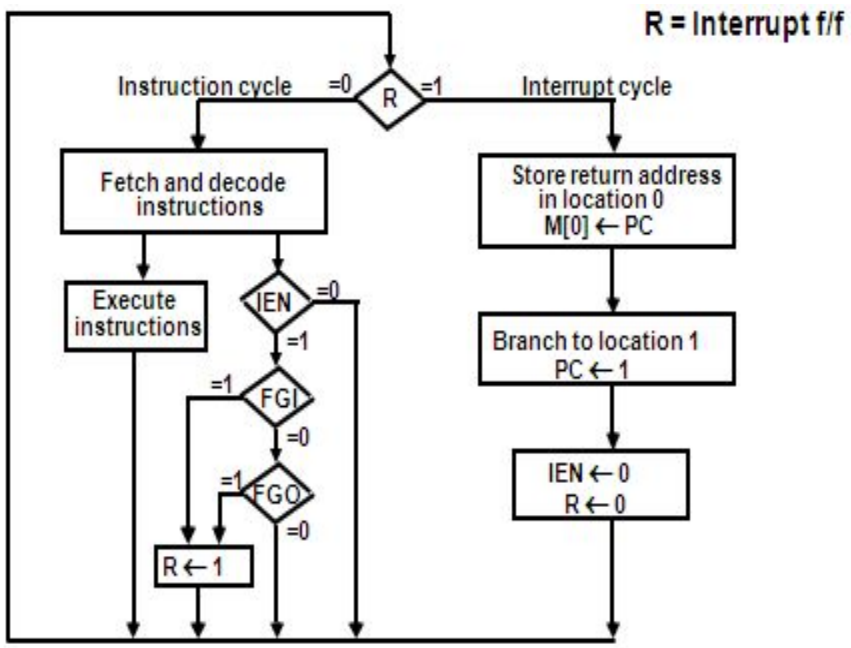
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

* IEN (Interrupt-enable flip-flop)

  - can be set and cleared by instructions
  - when cleared, the computer cannot be interrupted

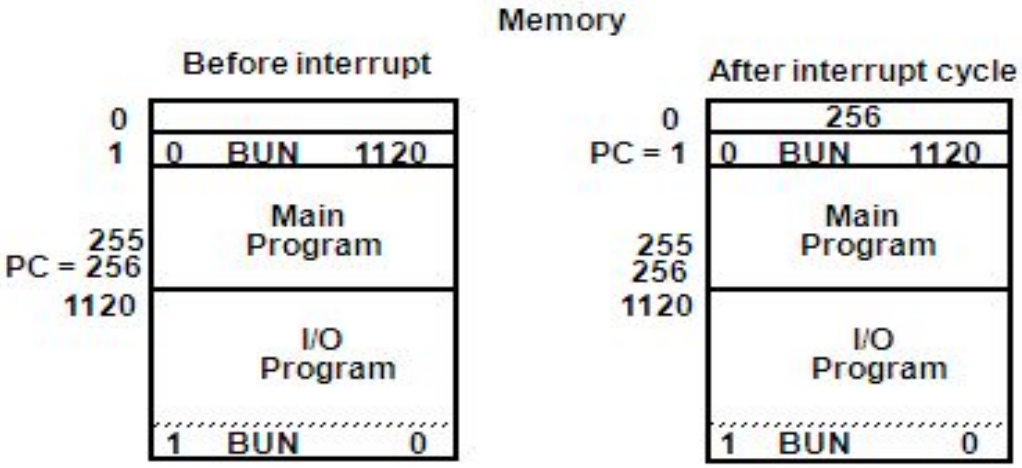
FLOWCHART FOR INTERRUPT CYCLE



The interrupt cycle is a HW implementation of a branch and save return address operation.

- At the beginning of the next 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"

REGISTER TRANSFER OPERATIONS IN INTERRUPT CYCLE



Register Transfer Statements for Interrupt Cycle

- R F/F ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$

  ⇔ $T_0'T_1'T_2'$ (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 :

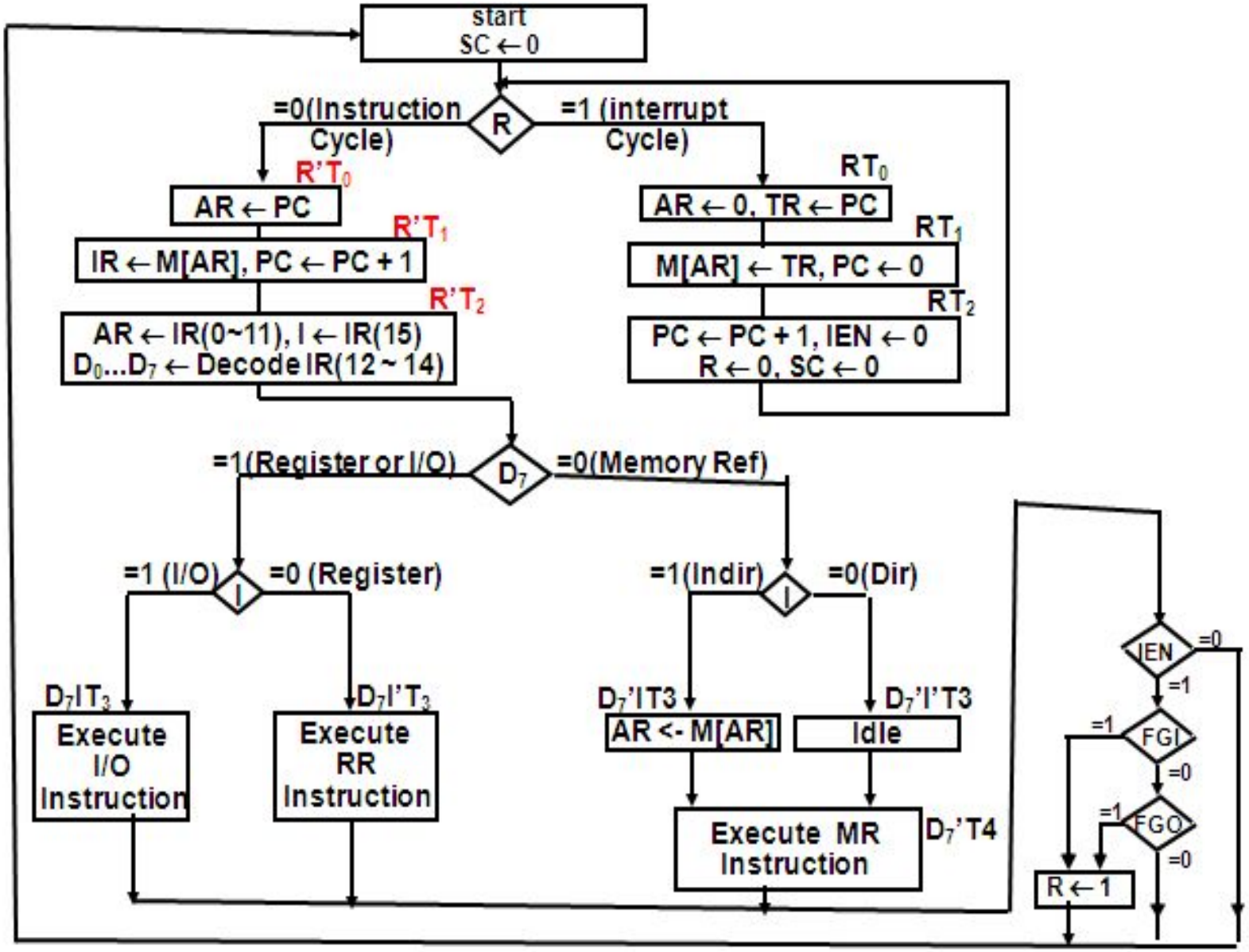
  RT_0: AR ← 0, TR ← PC

  RT_1: M[AR] ← TR, PC ← 0

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

COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations



COMPLETE COMPUTER DESCRIPTION

Microoperations

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D0, ..., D7 \leftarrow$ Decode $IR(12 \sim 14)$, $AR \leftarrow IR(0 \sim 11), I \leftarrow IR(15)$
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 ($PC \leftarrow PC + 1$), $SC \leftarrow 0$

COMPLETE COMPUTER DESCRIPTION

Microoperations

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

DESIGN OF BASIC COMPUTER(BC)

Hardware Components of BC

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

Control Logic Gates

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

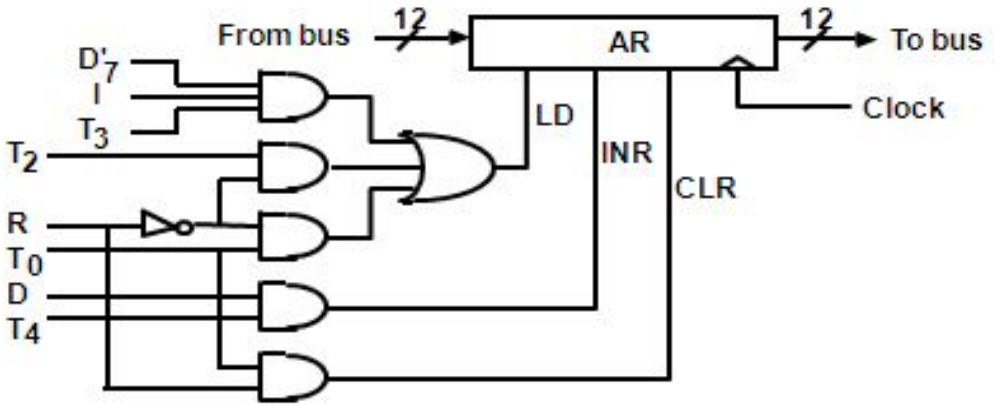
CONTROL OF REGISTERS AND MEMORY

Address Register; AR

Scan all of the register transfer statements that change the content 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:	$AR \leftarrow AR + 1$	$INR(AR)$

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

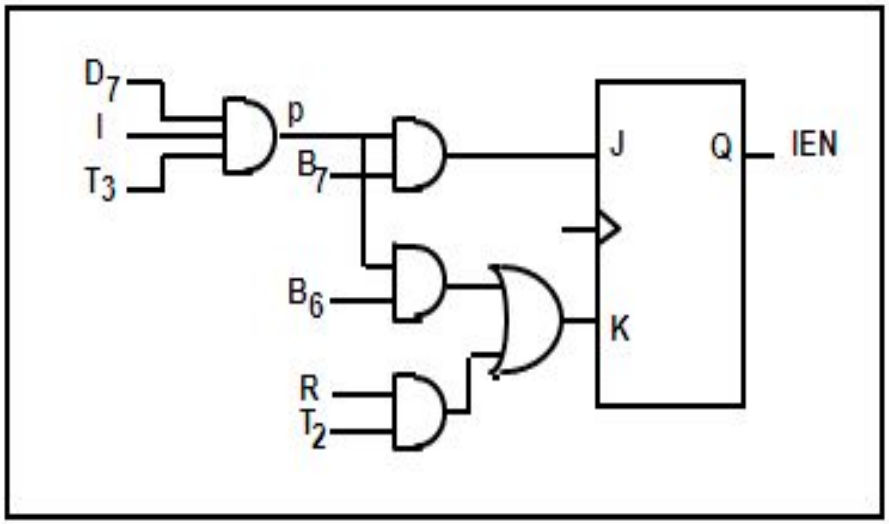


CONTROL OF FLAGS

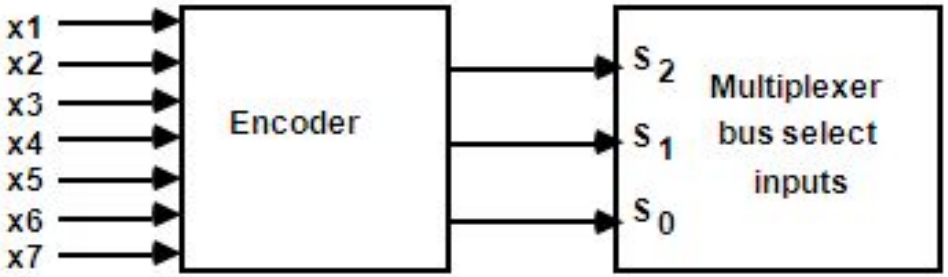
IEN: Interrupt Enable Flag

pB_7: IEN ← 1 (I/O Instruction)
pB_6: IEN ← 0 (I/O Instruction)
RT_2: IEN ← 0 (Interrupt)

$p = D_7IT_3$ (Input/Output Instruction)



CONTROL OF COMMON BUS



x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	selected register
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

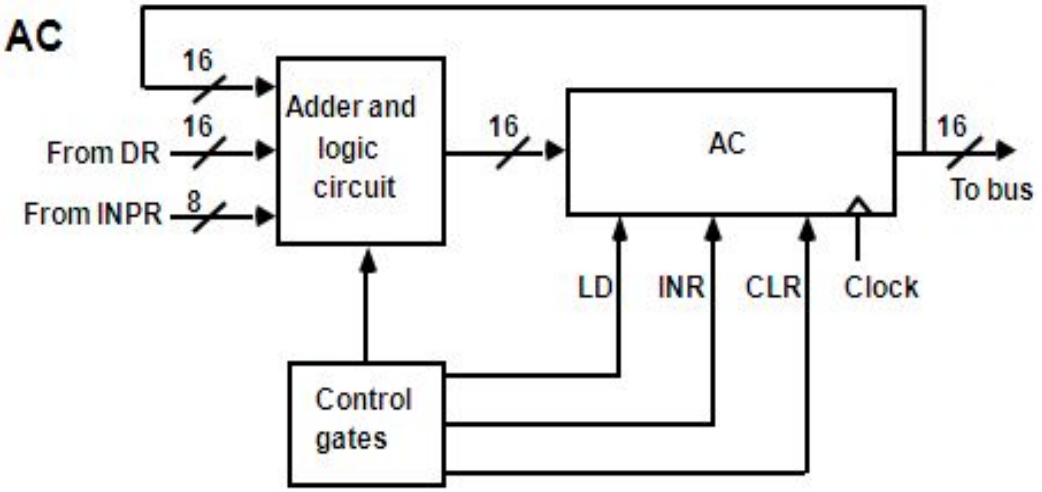
x1 for placing AR onto bus

D_4T_4: PC ← AR
D_5T_5: PC ← AR

$x1 = D_4T_4 + D_5T_5$

DESIGN OF ACCUMULATOR LOGIC

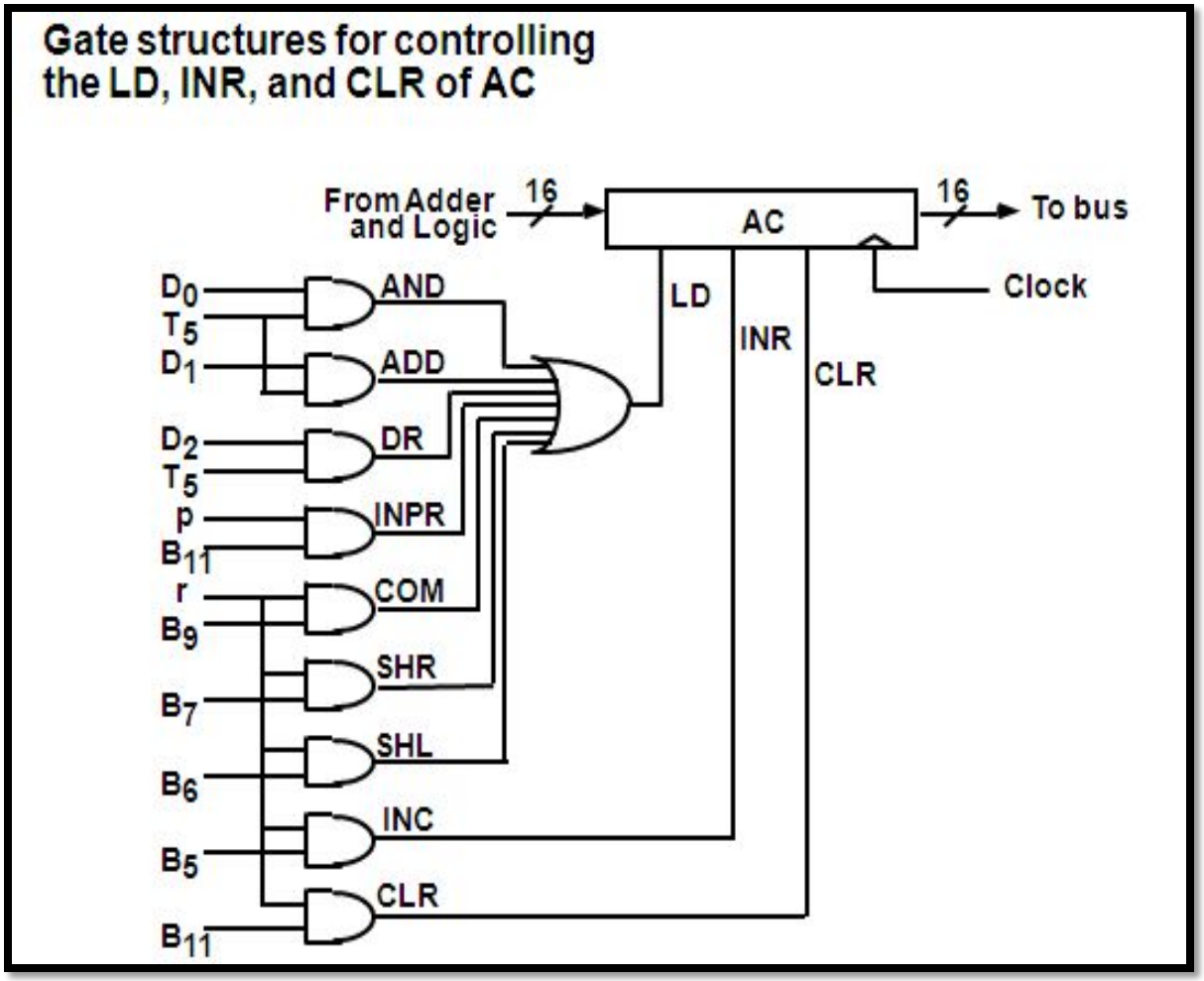
Circuits associated with AC



All the statements that change the content of AC

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:	$AC \leftarrow AC'$	Complement
rB_7:	$AC \leftarrow shr\,AC, AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow shl\,AC, AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

CONTROL OF AC REGISTER



ALU (Adder & Logic Circuits)