Chapter 5:

Basic Computer Organization and Design

5-1 Instruction Codes

- The Internal organization of a digital system is defined by the sequence of microoperations it performs on data stored in its registers
- The user of a computer can control the process by means of a program
- A *program* is a set of instructions that specify the operations, operands, and the processing sequence

5-1 Instruction Codes cont.

- 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
- The computer reads each instruction from memory and places it in a control register
- The control unit interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations

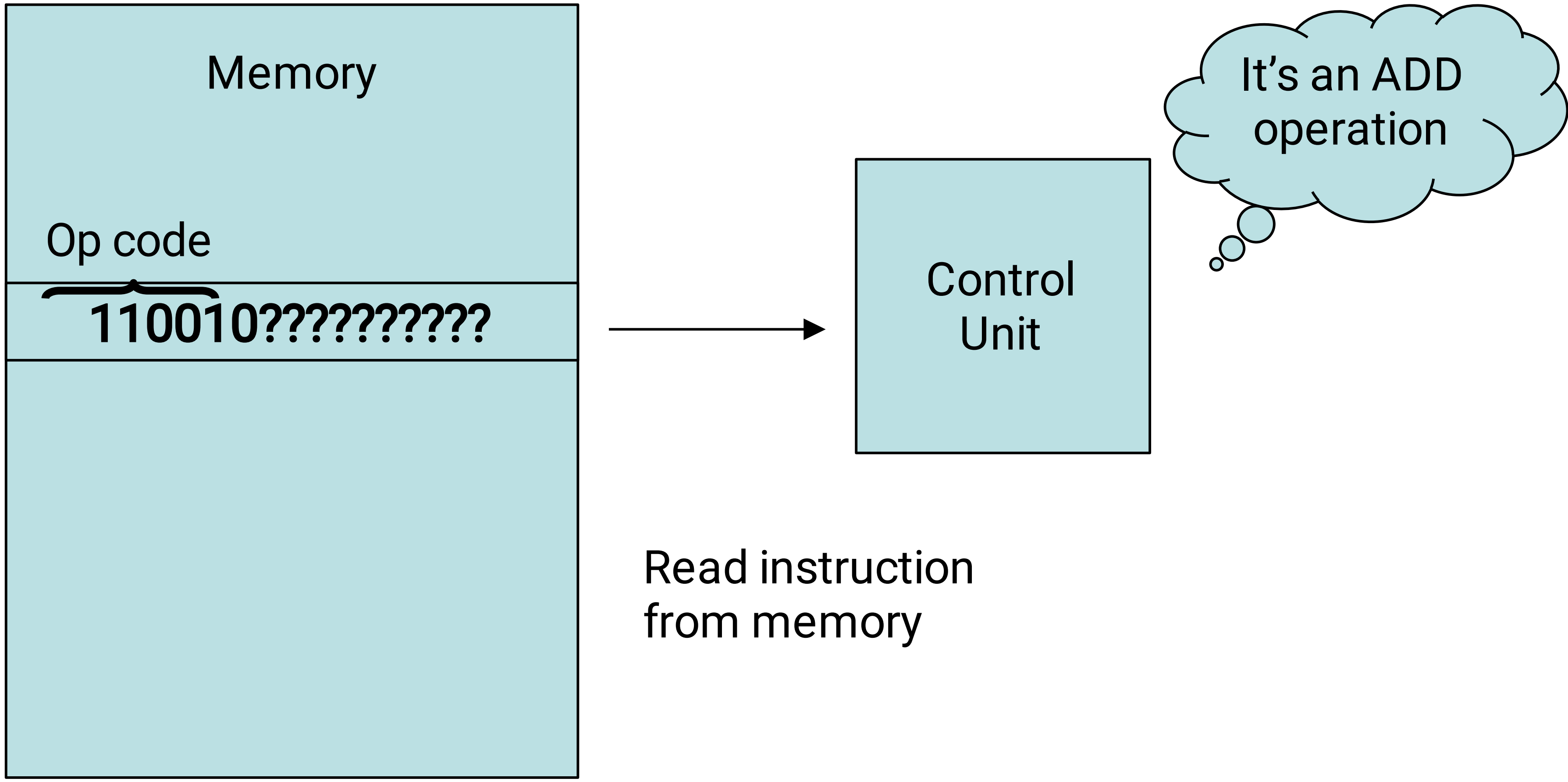
5-1 Instruction Codes cont.

- An Instruction code is a group of bits that instructs the computer to perform a specific operation (sequence of microoperations). It is divided into parts (basic part is the operation part)
- The operation code of an instruction is a group of bits that defines certain operations such as add, subtract, shift, and complement

5-1 Instruction Codes $^{\text{cont.}}$

- The 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

5-1 Instruction Codes cont.



5-1 Instruction Codes cont.

- An operation must be performed on some data stored in processor registers or in memory
- An instruction code must therefore specify not only the operation, but also the location of the operands (in registers or in the memory), and where the result will be stored (registers/memory)

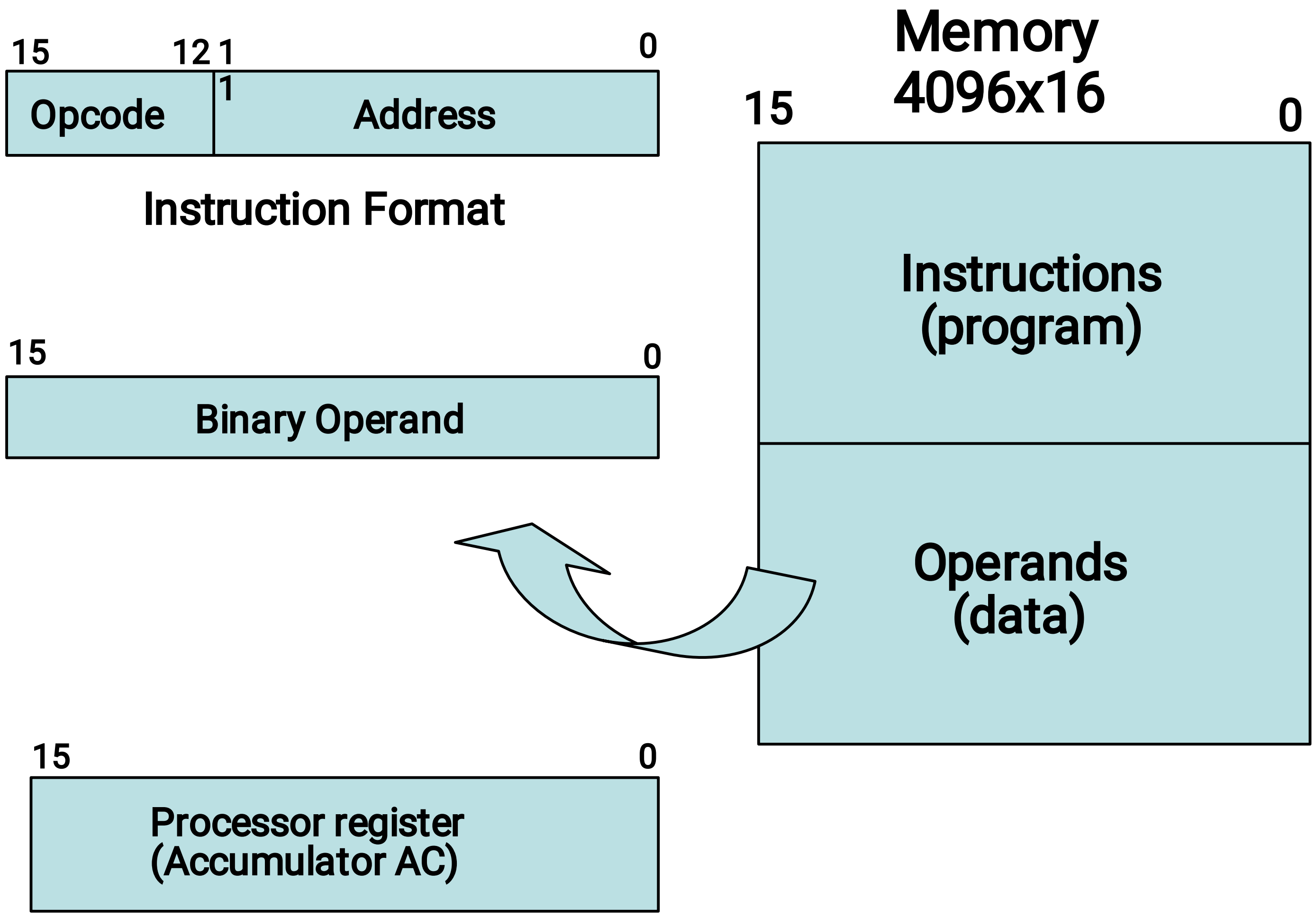
5-1 Instruction Codes cont.

- Memory words can be specified in instruction codes by their address
- Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers
- Each computer has its own particular instruction code format
- Instruction code formats are conceived by computer designers who specify the architecture of the computer

5-1 Instruction Codes cont. Stored Program Organization

- An instruction code is usually divided into *operation code*, *operand address*, *addressing mode*, etc.
- The 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)

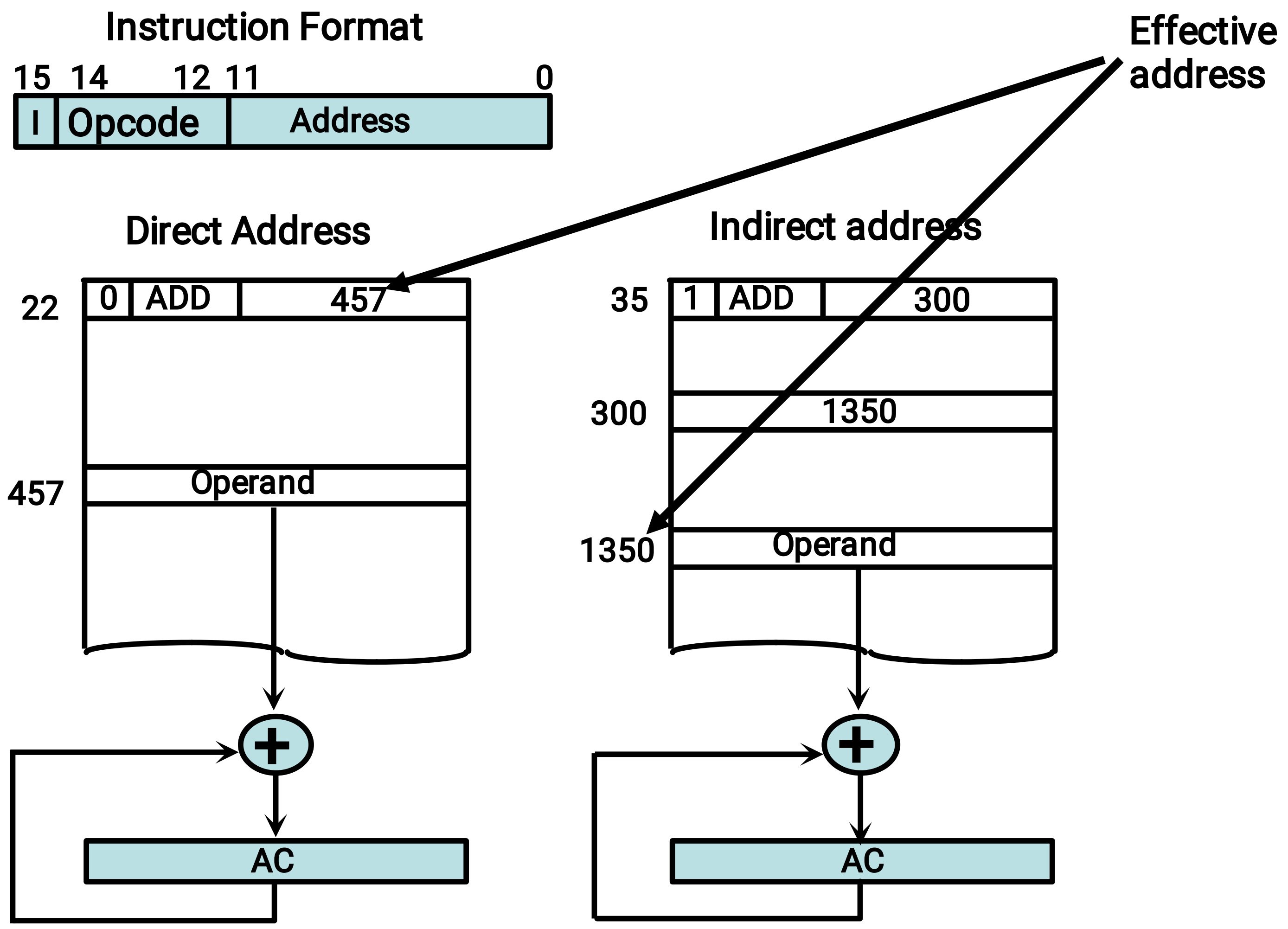
5-1 Instruction Codes Stored Program Organization cont.



5-1 Instruction Codes Indirect Address

- There are three **Addressing Modes** used for address portion of the instruction code:
  - Immediate: the operand is given in the address portion (constant)
  - Direct: the address points to the operand stored in the memory
  - Indirect: the address points to the pointer (another address) stored in the memory that references the operand in memory
- One bit of the instruction code can be used to distinguish between direct & indirect addresses

5-1 Instruction Codes Indirect Address cont.



5-1 Instruction Codes Indirect Address cont.

- Effective address: the address of the operand in a computation-type instruction or the target address in a branch-type instruction
- The pointer can be placed in a processor register instead of memory as done in commercial computers

5-2 Computer Registers

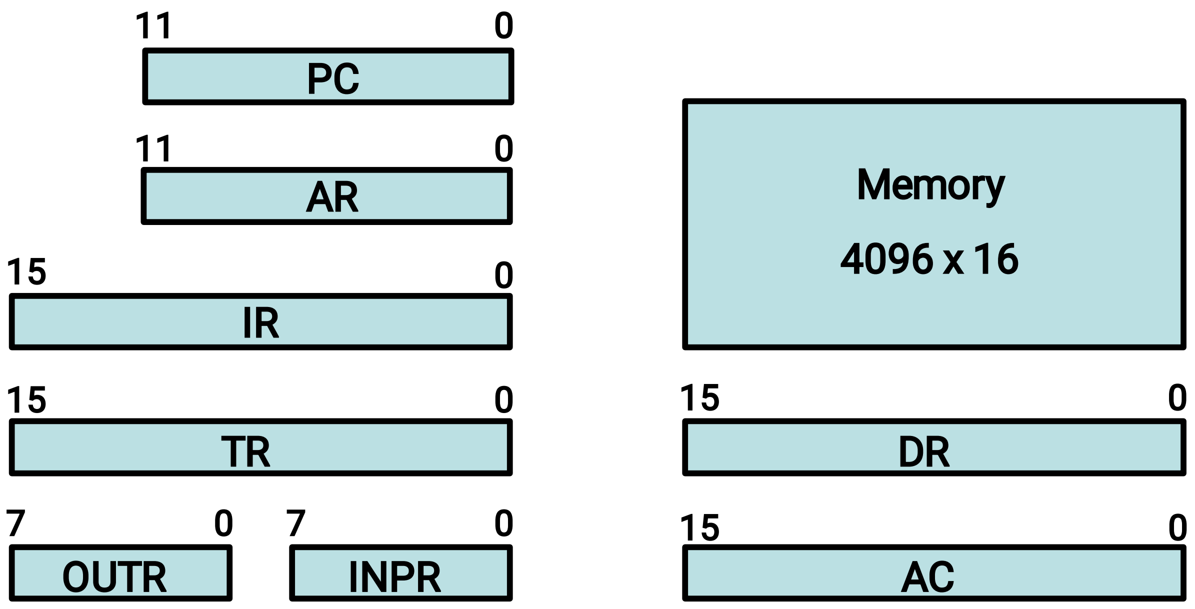
- Computer instructions are normally stored in consecutive memory locations and executed sequentially one at a time
- The control reads an instruction from a specific address in memory and executes it, and so on
- This type of sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed

5-2 Computer Registers $^{\text{cont.}}$

- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory
- The computer needs processor registers for manipulating data and a register for holding a memory address

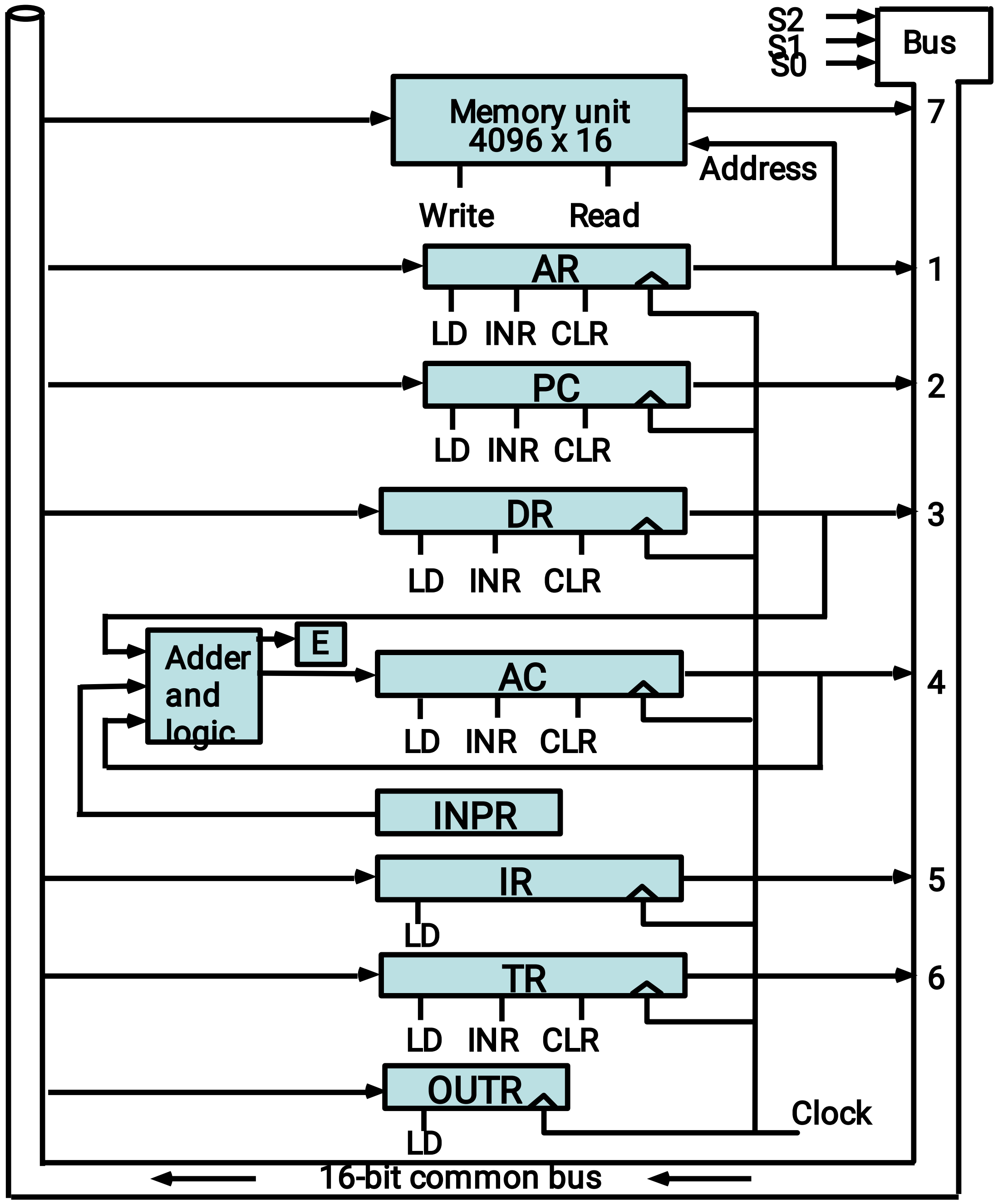
In order to cover the basic concepts behind designing a computer, a model (an imaginary system) will be presented to you throughout this chapter. This model will be called the “Basic Computer”

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



Computer Registers Common Bus System

5-2 Computer Registers Common Bus System cont.

- $S_2S_1S_0$: Selects the register/memory that would use the bus
- LD (load): When enabled, the particular register receives the data from the bus during the next clock pulse transition
- E (extended AC bit): flip-flop holds the carry
- DR, AC, IR, and TR: have 16 bits each
- AR and PC: have 12 bits each since they hold a memory address

5-2 Computer Registers Common Bus System cont.

- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to zeros
- When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register
- INPR and OUTR: communicate with the eight least significant bits in the bus

5-2 Computer Registers Common Bus System cont.

- INPR: Receives a character from the input device (keyboard,…etc) which is then transferred to AC
- OUTR: Receives a character from AC and delivers it to an output device (say a Monitor)
- Five registers have three control inputs: LD (load) , INR (increment), and CLR (clear)
- Register ≡ binary counter with parallel load and synchronous clear

5-2 Computer Registers Memory Address

- The input data and output data of the memory are connected to the common bus
- But the memory address is connected to AR
- Therefore, AR must always be used to specify a memory address
- By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise

5-2 Computer Registers Memory Address cont.

- Register → Memory: Write operation
- Memory → Register: Read operation (note that AC cannot directly read from memory!!)
- Note that the content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle

5-2 Computer Registers Memory Address cont.

- The transition at the end of the cycle transfers the content of the bus into the destination register, and the output of the adder and logic circuit into the AC
- For example, the two microoperations

  $DR \leftarrow AC$ and $AC \leftarrow DR$ (Exchange)

  can be executed at the same time
- This is done by:

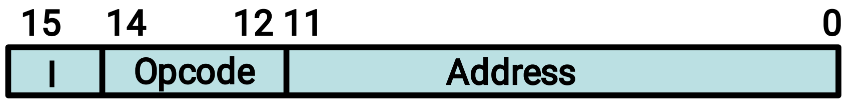
5-2 Computer Registers Memory Address cont.

- 1- place the contents of AC on the bus ($S_2S_1S_0$💾100)
- 2- enabling the LD (load) input of DR
- 3- Transferring the contents of the DR through the adder and logic circuit into AC
- 4- enabling the LD (load) input of AC
- All during the same clock cycle
- The two transfers occur upon the arrival of the clock pulse transition at the end of the clock cycle

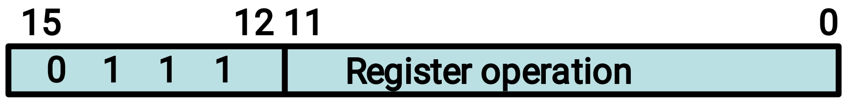
5-3 Computer Instructions

Basic Computer Instruction code format

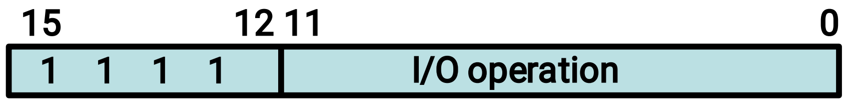
Memory-Reference Instructions (OP-code = 000 ~ 110)



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



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



BASIC COMPUTER INSTRUCTIONS

Symbol	*Hex Code* *I = 0*	*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

5-3 Computer Instructions Instruction Set Completeness

- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
  - Arithmetic, logical, and shift instructions
  - Instructions for moving information to and from memory and processor registers
  - Program control instructions together with instructions that check status conditions
  - Input & output instructions

5-4 Timing & Control

- The timing for all registers in the basic computer is controlled by a master clock generator
- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit
- The clock pulses do not change the state of a register unless the register is enabled by a control signal (i.e., Load)

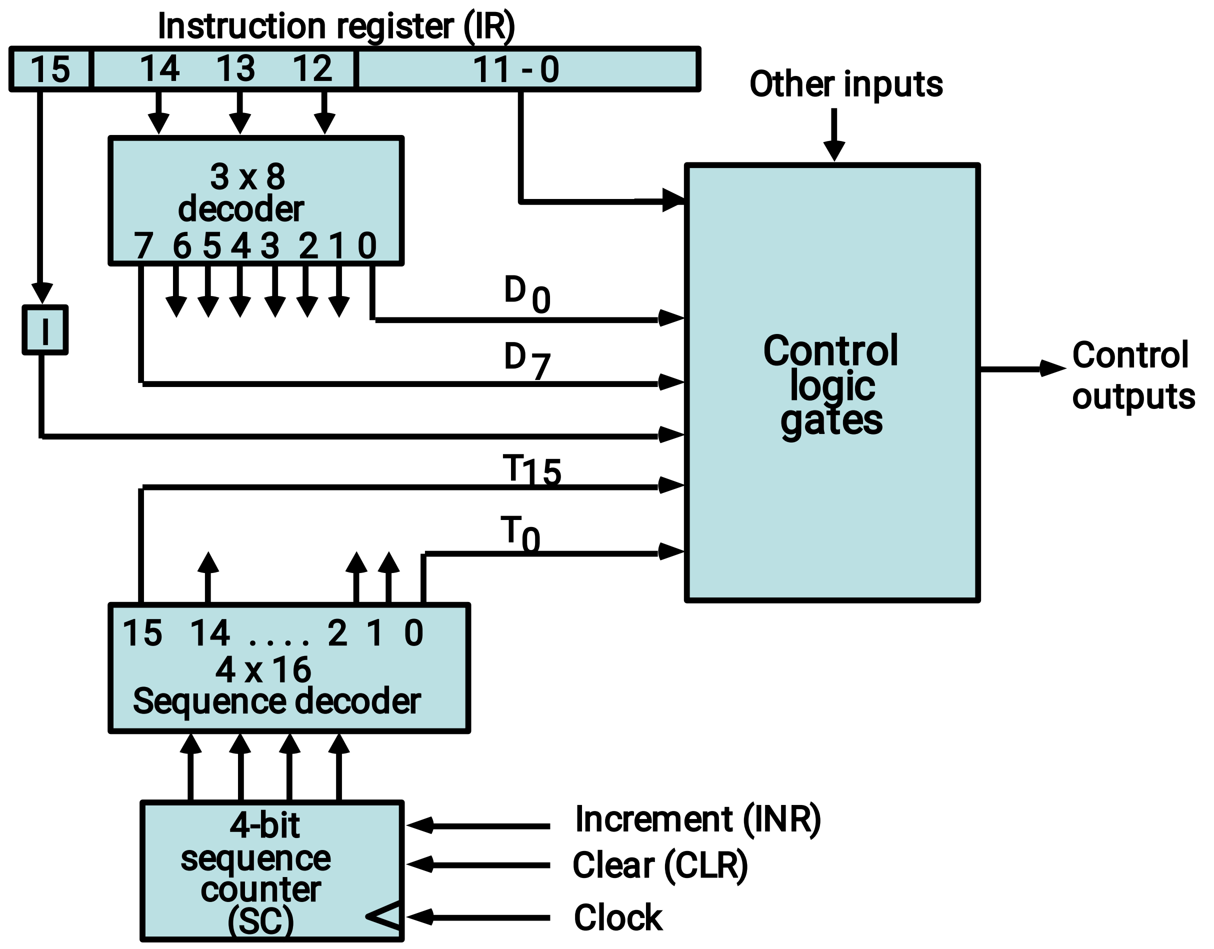
5-4 Timing & Control $^{\text{cont.}}$

- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator
- There are two major types of control organization:
  - Hardwired control
  - Microprogrammed control

5-4 Timing & Control $^{\text{cont.}}$

- In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- In the microprogrammed organization, the control information is stored in a control memory (if the design is modified, the microprogram in control memory has to be updated)
- D_3T_4: $SC \leftarrow 0$

The Control Unit for the basic computer



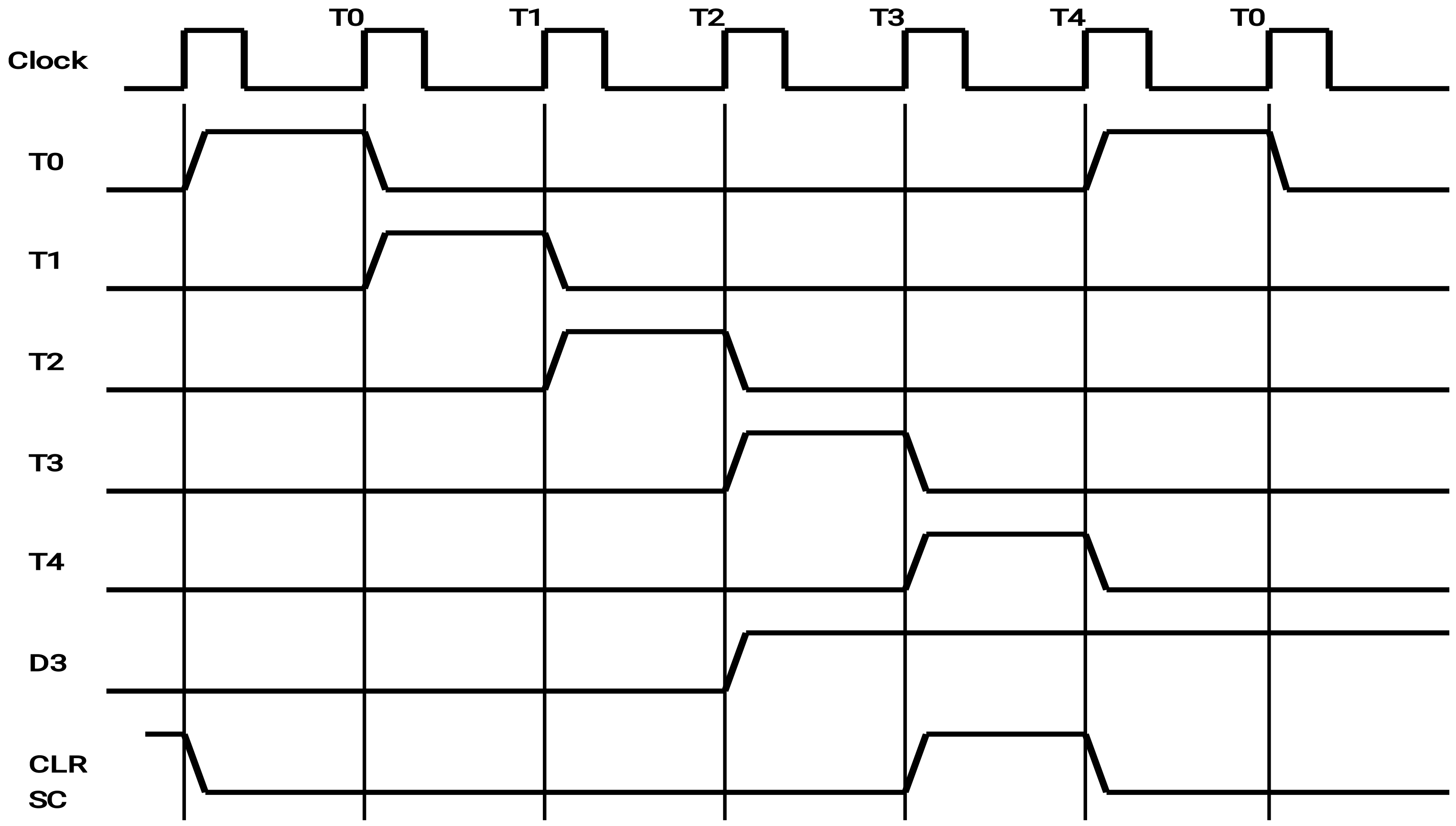
Hardwired Control Organization

- Generated by 4-bit sequence counter and 4x16 decoder
- The SC can be incremented or cleared.

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

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

$$D_3T_4: SC \leftarrow 0$$



5-4 Timing & Control $^{\text{cont.}}$

- A memory read or write cycle will be initiated with the rising edge of a timing signal
- Assume: memory cycle time **<** clock cycle time!
- So, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive edge
- The clock transition will then be used to load the memory word into a register
- The memory cycle time is usually longer than the processor clock cycle → wait cycles

5-4 Timing & Control cont.

- T_0: AR ← PC
  - Transfers the content of PC into AR if timing signal T_0 is active
  - T_0 is active during an entire clock cycle interval
  - During this time, the content of PC is placed onto the bus (with $S_2S_1S_0$=010) and the LD (load) input of AR is enabled
  - The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition
  - This same positive clock transition increments the sequence counter SC from 0000 to 0001
  - The next clock cycle has T_1 active and T_0 inactive

5-5 Instruction Cycle

- A program is a sequence of instructions stored in memory
- The program is executed in the computer by going through a cycle for each instruction (in most cases)
- Each instruction in turn is subdivided into a sequence of sub-cycles or phases

5-5 Instruction Cycle $^{\text{cont.}}$

- Instruction Cycle Phases:
  - 1- Fetch an instruction from memory
  - 2- Decode the instruction
  - 3- Read the effective address from memory if the instruction has an indirect address
  - 4- Execute the instruction
- This cycle repeats indefinitely unless a HALT instruction is encountered

5-5 Instruction Cycle Fetch and Decode

- Initially, the Program Counter (PC) is loaded with the address of the first instruction in the program
- The sequence counter SC is cleared to 0, providing a decoded timing signal T_0
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on

5-5 Instruction Cycle Fetch and Decode $^{\text{cont.}}$

– T_0: AR ← PC   (this is essential!!)

The address of the instruction is moved to AR.

– T_1: IR ← M[AR], PC ← PC+1

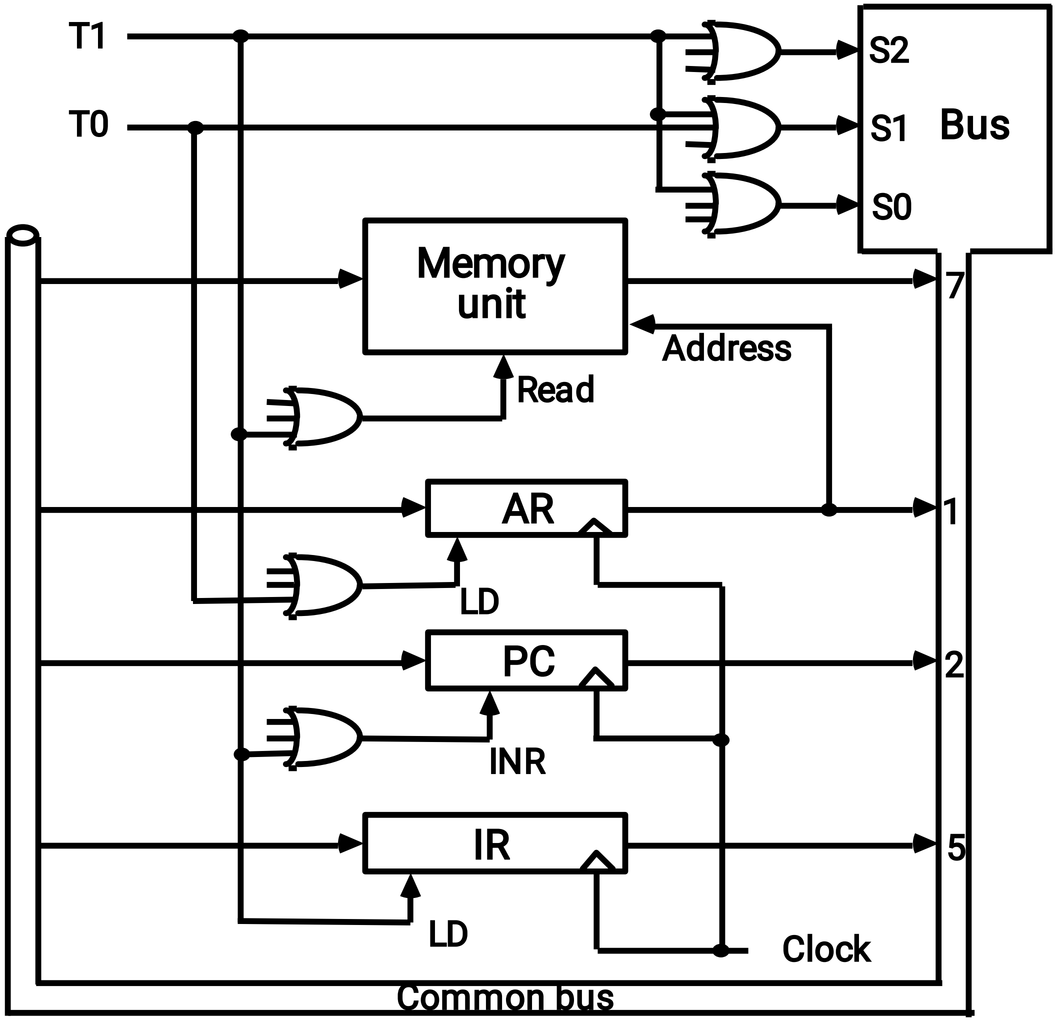
The instruction is fetched from the memory to IR , and the PC is incremented.

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

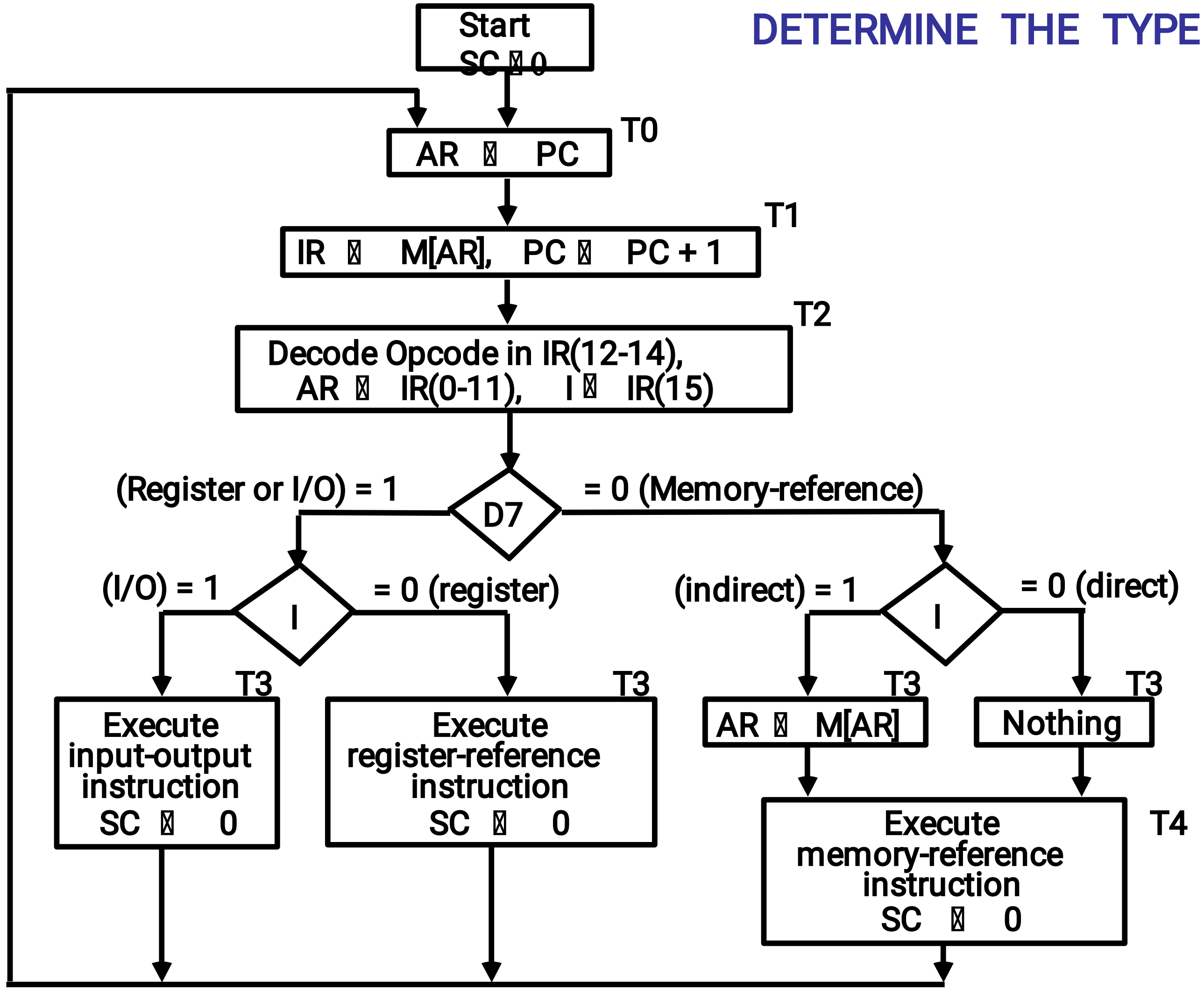
Instruction cycle: [Fetch Decode [Indirect] Execute]*

- **Fetch and Decode**

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)



DETERMINE THE TYPE OF INSTRUCTION


Start
SC ← 0
AR ← PC
T0
IR ← M[AR], PC ← PC + 1
T1
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
T2
(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
T4

D'_7IT_3: AR ← 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, the ith bit of IR.

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: $S \leftarrow 0$ (S is a start-stop flip-flop)

5.6 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 $PC \leftarrow PC+1$

- The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T3 when I = 1
- Memory cycle is assumed to be short enough to be completed in a CPU cycle
- The execution of MR Instruction starts with T_4

AND to AC

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

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

ADD to AC

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

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

MEMORY REFERENCE INSTRUCTIONS$^{\text{cont.}}$

LDA: Load to AC

D_2T_4: DR $\leftarrow$ M[AR]

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

STA: Store AC

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

BUN: Branch Unconditionally

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

BSA: Branch and Save Return Address

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

Memory Reference Instructions$^{\text{cont.}}$

BSA: executed in a sequence of two micro-operations:

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

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

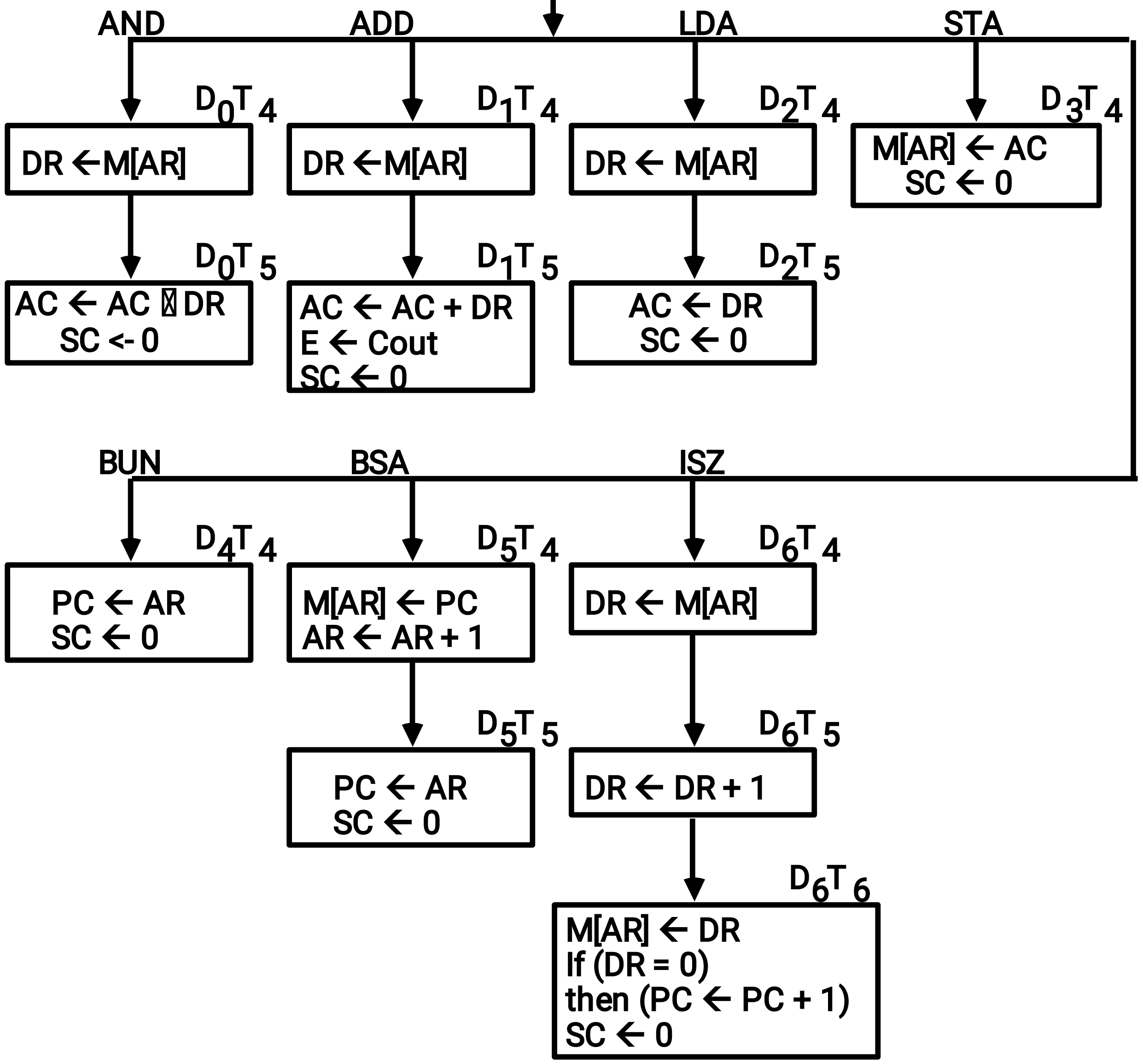
ISZ: Increment and Skip-if-Zero

D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR + 1$

D_6T6: $M[AR] \leftarrow DR,\ \ \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1),\ \ SC \leftarrow 0$

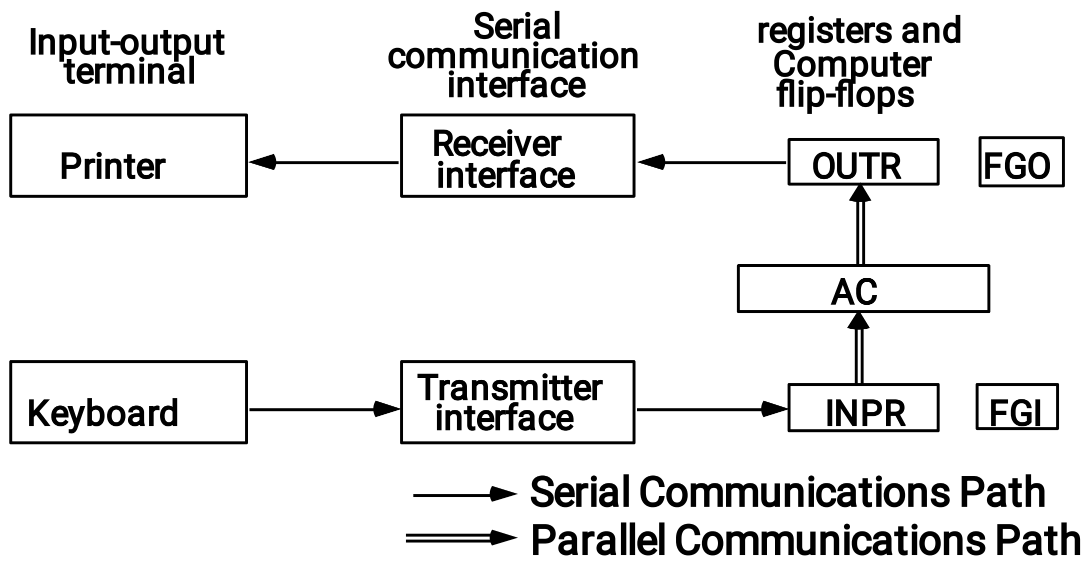
Memory-reference instruction



5-7 Input-Output and Interrupt

- Instructions and data stored in memory must come from some input device
- Computational results must be transmitted to the user through some output device
- For the system to communicate with an input device, serial information is shifted into the input register INPR
- To output information, it is stored in the output register OUTR

5-7 Input-Output and Interrupt$^{\text{cont.}}$


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

5-7 Input-Output and Interrupt$^{cont.}$

- INPR and OUTR communicate with a communication interface serially and with the AC in parallel. They hold an 8-bit alphanumeric information
- I/O devices are slower than a computer system → we need to synchronize the timing rate difference between the input/ output device and the computer.
- FGI: 1-bit input flag (Flip-Flop) aimed to control the input operation

5-7 Input-Output and Interrupt cont.

- FGI is set to 1 when a new information is available in the input device and is cleared to 0 when the information is accepted by the computer
- FGO: 1-bit output flag used as a control flip-flop to control the output operation
- If FGO is set to 1, then this means that the computer can send out the information from AC. If it is 0, then the output device is busy and the computer has to wait!

5-7 Input-Output and Interrupt$^{cont.}$

- The process of input information transfer:
  - Initially, FGI is cleared to 0
  - An 8-bit alphanumeric code is shifted into INPR (Keyboard key strike) and the input flag FGI is set to 1
  - As long as the flag is set, the information in INPR cannot be changed by another data entry
  - The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0

5-7 Input-Output and Interrupt$^{\text{cont.}}$

  - Once the flag is cleared, new information can be shifted into INPR by the input device (striking another key)

- The process of outputting information:
  - Initially, the output flag FGO is set to 1
  - The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0
  - The output accepts the coded information (prints the corresponding character)

5-7 Input-Output and Interrupt$^{cont.}$

– When the operation is completed, the output device sets FGO back to 1

– The computer does not load a new data information into OUTR when FGO is 0 because this condition indicates that the output device is busy to receive another information at the moment!!

Input-Output Instructions

- Needed for:
  - Transferring information to and from AC register
  - Checking the flag bits
  - Controlling the interrupt facility
- The control unit recognize it when D_7=1 and I = 1
- The remaining bits of the instruction specify the particular operation
- Executed with the clock transition associated with timing signal T_3
- Input-Output instructions are summarized next

Input-Output Instructions

INP pB_{11}: AC(0-7) $\leftarrow$ INPR, FGI $\leftarrow$ 0 Input char. to AC
OUT pB_{10}: OUTR $\leftarrow$ AC(0-7), FGO $\leftarrow$ 0 Output char. from AC
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

- The process of communication just described is referred to as ***Programmed Control Transfer***
- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transform (this is sometimes called ***Polling***)
- This type of transfer is in-efficient due to the difference of information flow rate between the computer and the I/O device

Program Interrupt$^{\text{cont.}}$

- The computer is wasting time while checking the flag instead of doing some other useful processing task
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer
- This type of transfer uses the interrupt facility

Program Interrupt$^{\text{cont.}}$

- While the computer is running a program, it does not check the flags
- Instead:
  - When a flag is set, the computer is immediately interrupted from proceeding with the current program

Program Interrupt$^{\text{cont.}}$

  - The computer stops what it is doing to take care of the input or output transfer
  - Then, it returns to the current program to continue what it was doing before the interrupt
- The interrupt facility can be enabled or disabled via a flip-flop called IEN
- The interrupt enable flip-flop IEN can be set and cleared with two instructions (IOF, ION):
  - IOF: IEN $\leftarrow$ 0 (the computer cannot be interrupted)
  - ION: IEN $\leftarrow$ 1 (the computer can be interrupted)

Program Interrupt$^{\text{cont.}}$

- Another flip-flop (called the interrupt flip-flop **R**) is used in the computer's interrupt facility to decide when to go through the interrupt cycle
- **FGI** and **FGO** are different here compared to the way they acted in an earlier discussion!!
- So, the computer is either in an <u>**Instruction Cycle**</u> or in an <u>**Interrupt Cycle**</u>

Program Interrupt$^{\text{cont.}}$

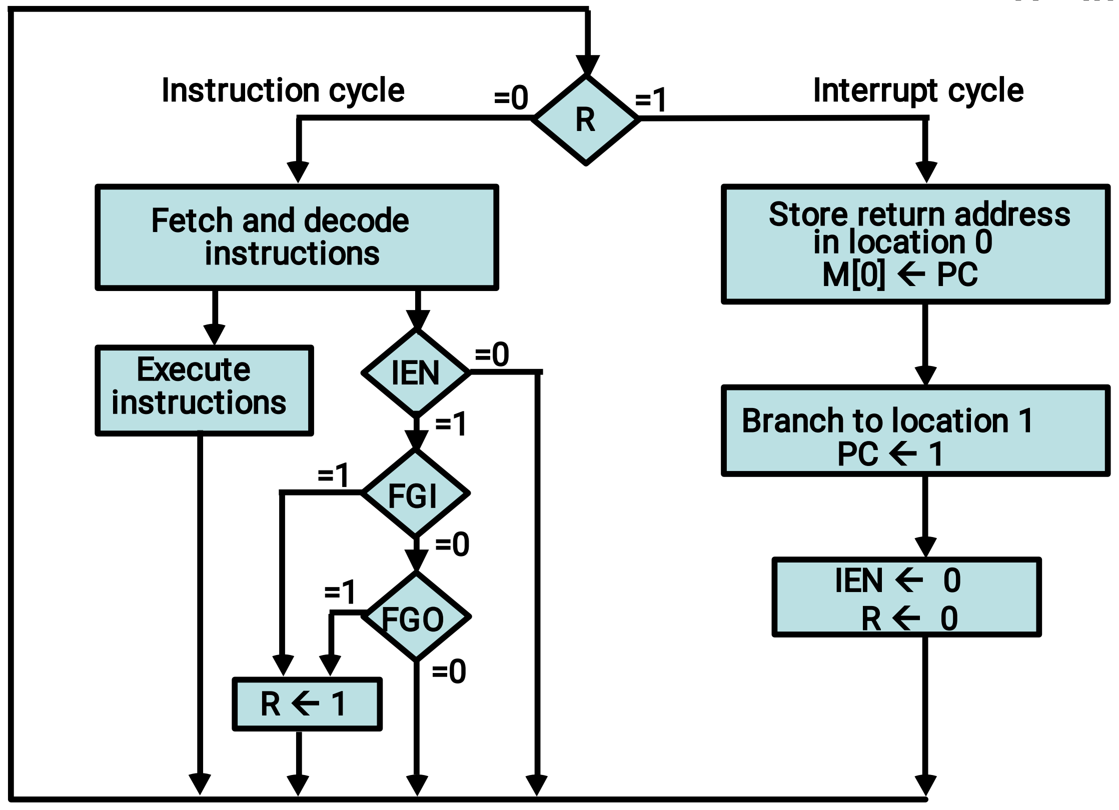
- The interrupt cycle is a hardware implementation of a branch and save return address operation (BSA)
- The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted
- This location may be a processor register, a memory stack, or a specific memory location

Program Interrupt$^{\text{cont.}}$

- For our computer, we choose the memory location at address 0 as a place for storing the return address
- Control then inserts address 1 into PC: this means that the first instruction of the interrupt service routine should be stored in memory at address 1, or, the programmer must store a branch instruction that sends the control to an interrupt service routine!!

Program Interrupt$^{\text{cont.}}$

R = Interrupt flip-flop



Instruction cycle
=0
R
=1
Interrupt cycle
Fetch and decode instructions
Store return address in location 0 M[0] ← PC
Execute instructions
IEN
=0
=1
Branch to location 1 PC ← 1
=1
FGI
=0
IEN ← 0 R ← 0
=1
FGO
=0
R ← 1

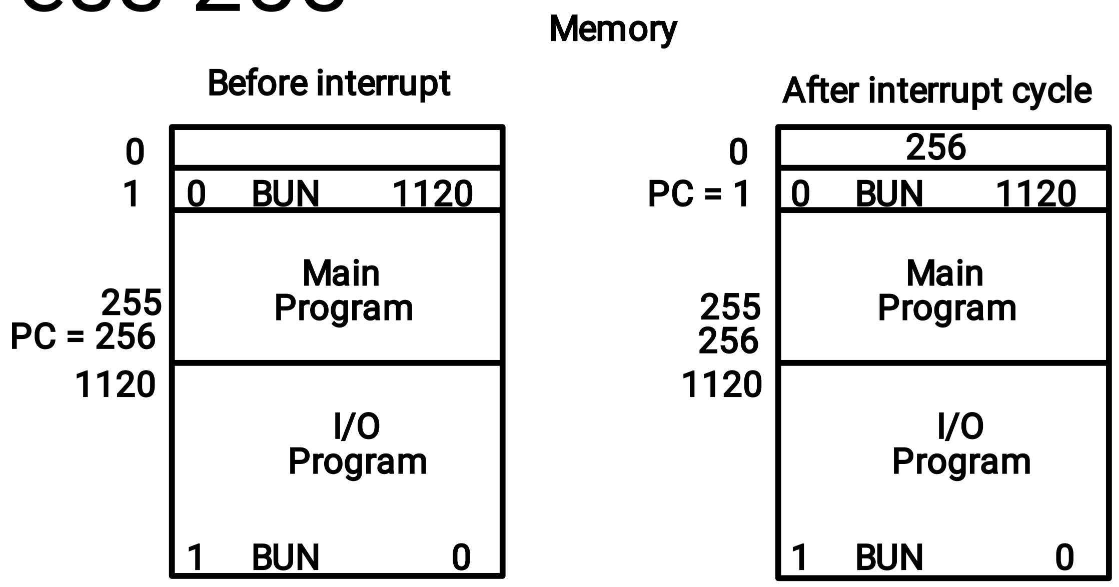
Flowchart for interrupt cycle

Program Interrupt$^{cont.}$

- IEN, R $\leftarrow$ 0: no more interruptions can occur until the interrupt request from the flag has been serviced
- The service routine must end with an instruction that re-enables the interrupt (IEN $\leftarrow$ 1) and an instruction to return to the instruction at which the interrupt occurred
- The instruction that returns the control to the original program is "indirect BUN 0"

Program Interrupt$^{\text{cont.}}$

- Example: the computer is interrupted during execution of the instruction at address 255



Interrupt Cycle

- The fetch and decode phases of the instruction cycle must be :

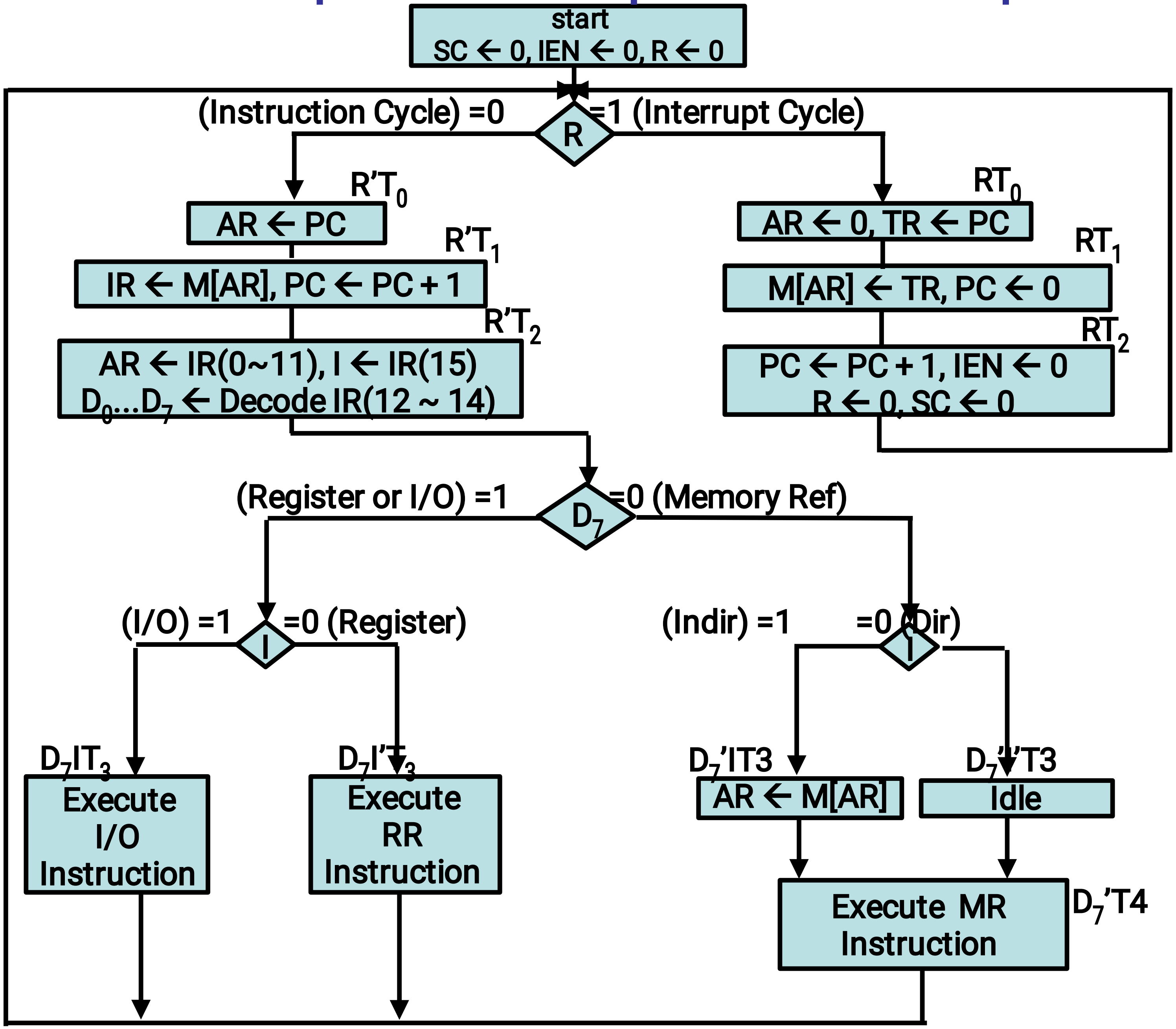
  (Replace T0, T1, T2 → R'T0, R'T1, R'T2 (fetch and decode phases occur at the instruction cycle when R = 0)

Interrupt cont.

- Further Questions:
  - How can the CPU recognize the device requesting an interrupt?
  - Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case?
  - Should any device be allowed to interrupt the CPU while another interrupt is being serviced?
  - How can the situation be handled when two or more interrupt requests occur simultaneously?

5-8 Complete Computer Description

Fig 5-15



5-8 Complete Computer Description$^{\text{cont.}}$

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
	D0T4:	DR ← M[AR]
Memory-Reference:	D0T5:	AC ← AC . DR, SC ← 0
AND	D1T4:	DR ← M[AR]
	D1T5:	AC ← AC + DR, E ← Cout, SC ← 0
ADD	D2T4:	DR ← M[AR]
	D2T5:	AC ← DR, SC ← 0
LDA	D3T4:	M[AR] ← AC, SC ← 0
	D4T4:	PC ← AR, SC ← 0
STA	D5T4:	M[AR] ← PC, AR ← AR + 1
	D5T5:	PC ← AR, SC ← 0
BUN	D6T4:	DR ← M[AR]
BSA	D6T5:	DR ← DR + 1
	D6T6:	M[AR] ← DR, if(DR=0) then (PC ← PC + 1), SC ← 0
ISZ		

5-8 Complete Computer Description^cont.

Register-Reference:		
	D7I'T3 = r	(Common to all register-reference instructions)
	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

Table 5-6

5-9 Design of Basic Computer

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:
    1. a 3x8 Opcode decoder
    2. a 4x16 timing decoder
5. Common bus: 16 bits
6. Control logic gates
7. Adder and Logic circuit: Connected to AC

5-9 Design of Basic Computer$^{\text{cont.}}$

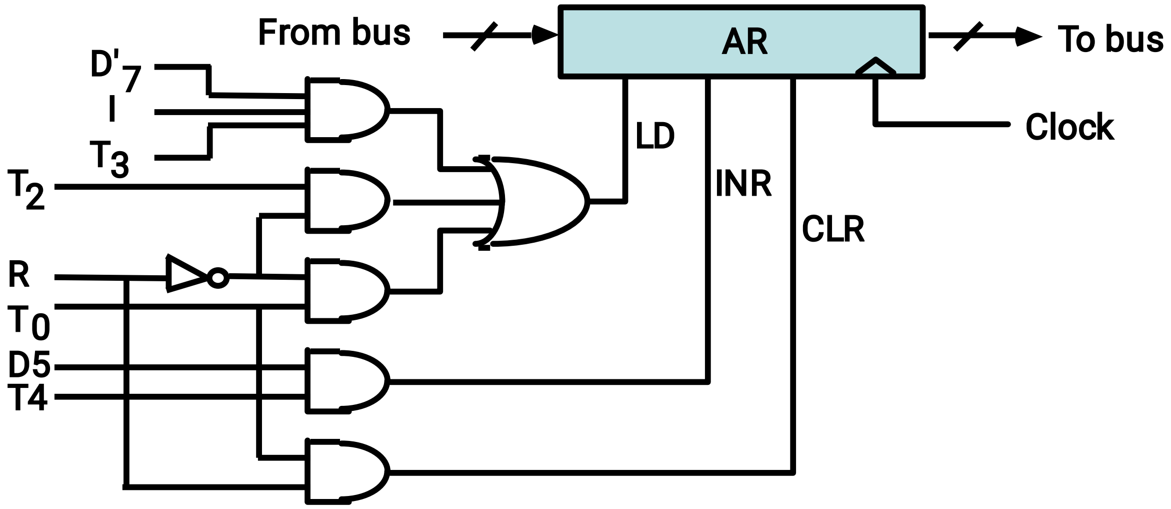
- The control logic gates are used to control:
  - Inputs of the nine registers
  - Read and Write inputs of memory
  - Set, Clear, or Complement inputs of the flip-flops
  - S2, S1, S0 that select a register for the bus
  - AC Adder and Logic circuit

5-9 Design of Basic Computer$^{\text{cont.}}$

- Control of registers and memory
  - The control inputs of the registers are LD (load), INR (increment), and CLR (clear)
  - To control AR We scan table 5-6 to find out all the statements that change the content of AR:
    - **R'T0: $AR \leftarrow PC$ LD(AR)**
    - **R'T2: $AR \leftarrow IR(0\text{-}11)$ LD(AR)**
    - **D'7IT3: $AR \leftarrow M[AR]$ LD(AR)**
    - **RT0: $AR \leftarrow 0$ CLR(AR)**
    - **D5T4: $AR \leftarrow AR + 1$ INR(AR)**

5-9 Design of Basic Computer$^{\text{cont.}}$

Control Gates associated with AR



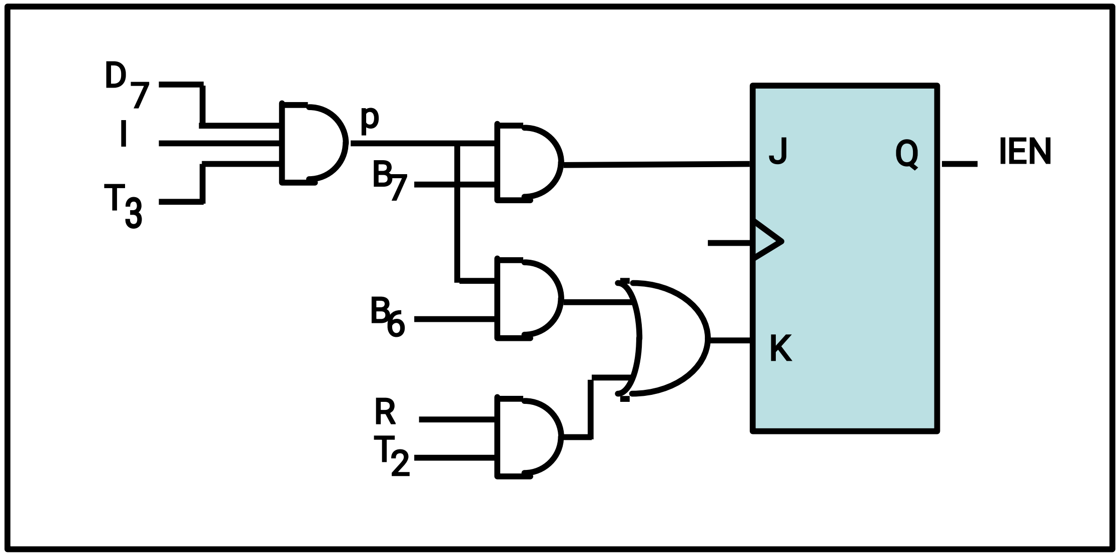
5-9 Design of Basic Computer$^{\text{cont.}}$

- To control the Read input of the memory we scan the table again to get these:
  - D_0T_4: $DR \leftarrow M[AR]$
  - D_1T_4: $DR \leftarrow M[AR]$
  - D_2T_4: $DR \leftarrow M[AR]$
  - D_6T_4: $DR \leftarrow M[AR]$
  - $D_7'IT_3$: $AR \leftarrow M[AR]$
  - $R'T_1$: $IR \leftarrow M[AR]$
- $\rightarrow$ Read 🖫 $R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$

5-9 Design of Basic Computer$^{\text{cont.}}$

- Control of Single Flip-flops (IEN for example)
  - **pB7: IEN ← 1 (I/O Instruction)**
  - **pB6: IEN ← 0 (I/O Instruction)**
  - **RT2: IEN ← 0 (Interrupt)**
    - **where p = D7IT3 (Input/Output Instruction)**
  - If we use a JK flip-flop for IEN, the control gate logic will be as shown in the following slide:

5-9 Design of Basic Computer$^{\text{cont.}}$

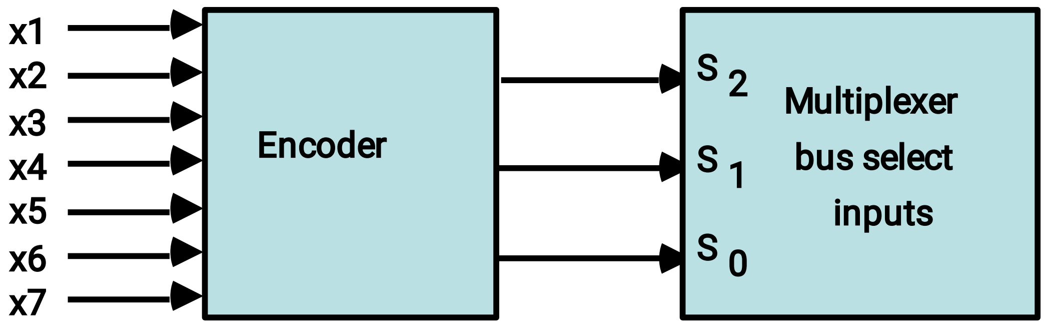


J	*K*	*Q(t+1)*
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

JK FF Characteristic Table

5-9 Design of Basic Computer$^{\text{cont.}}$

- Control of Common bus is accomplished by placing an encoder at the inputs of the bus selection logic and implementing the logic for each encoder input



5-9 Design of Basic Computer$^{cont.}$

- To select AR on the bus then x_1 must be 1. This is happen when:
  - **D_4T_4: PC ← AR**
  - **D_5T_5: PC ← AR**
- $\Rightarrow x_1 = D_4T_4 + D_5T_5$

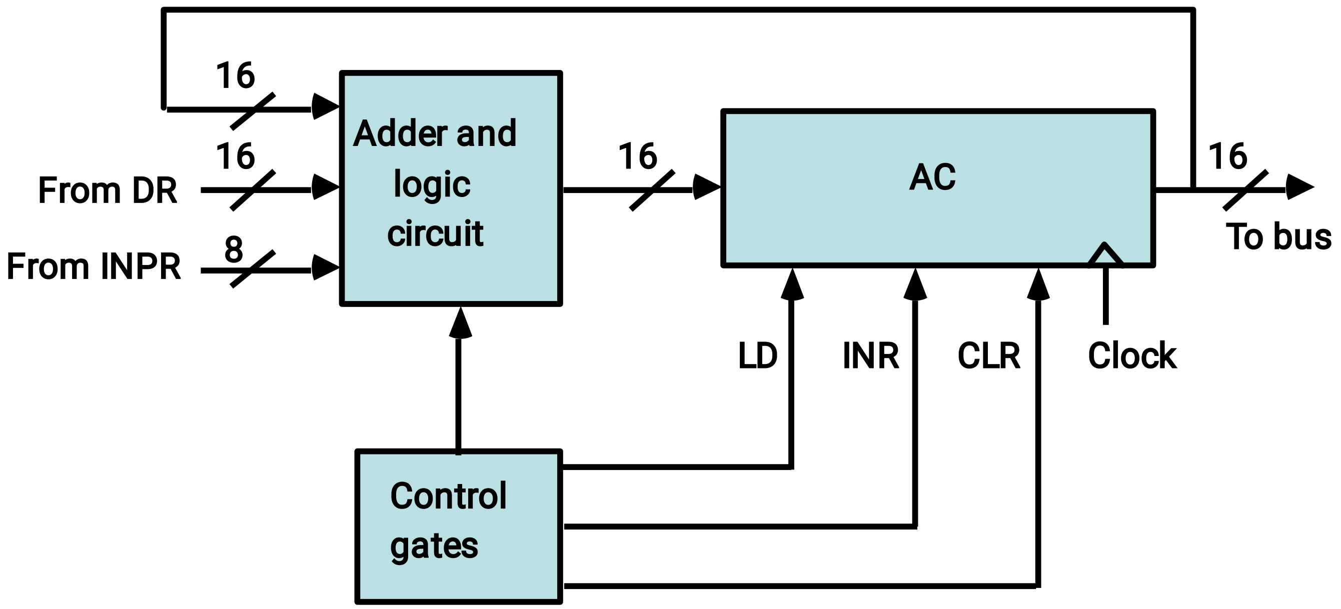
x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	selected register
0	0	0	0	0	0	0	0	0	0	none
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

5-9 Design of Basic Computer$^{\text{cont.}}$

- For x_7:
  - $X_7 = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$ where it is also applied to the read input

5-10 Design of Accumulator Logic

Circuits associated with AC

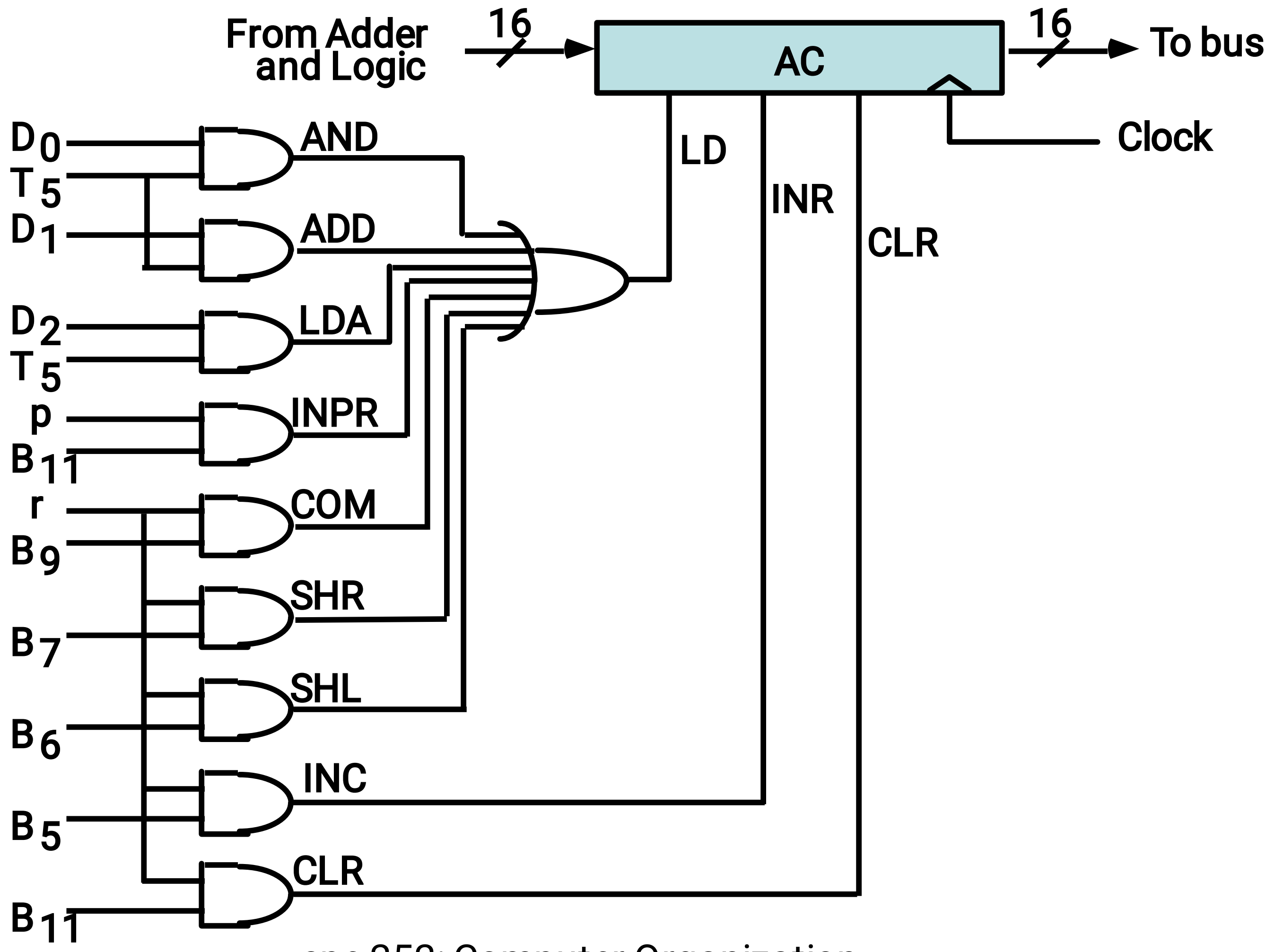


All the statements that change the content of AC

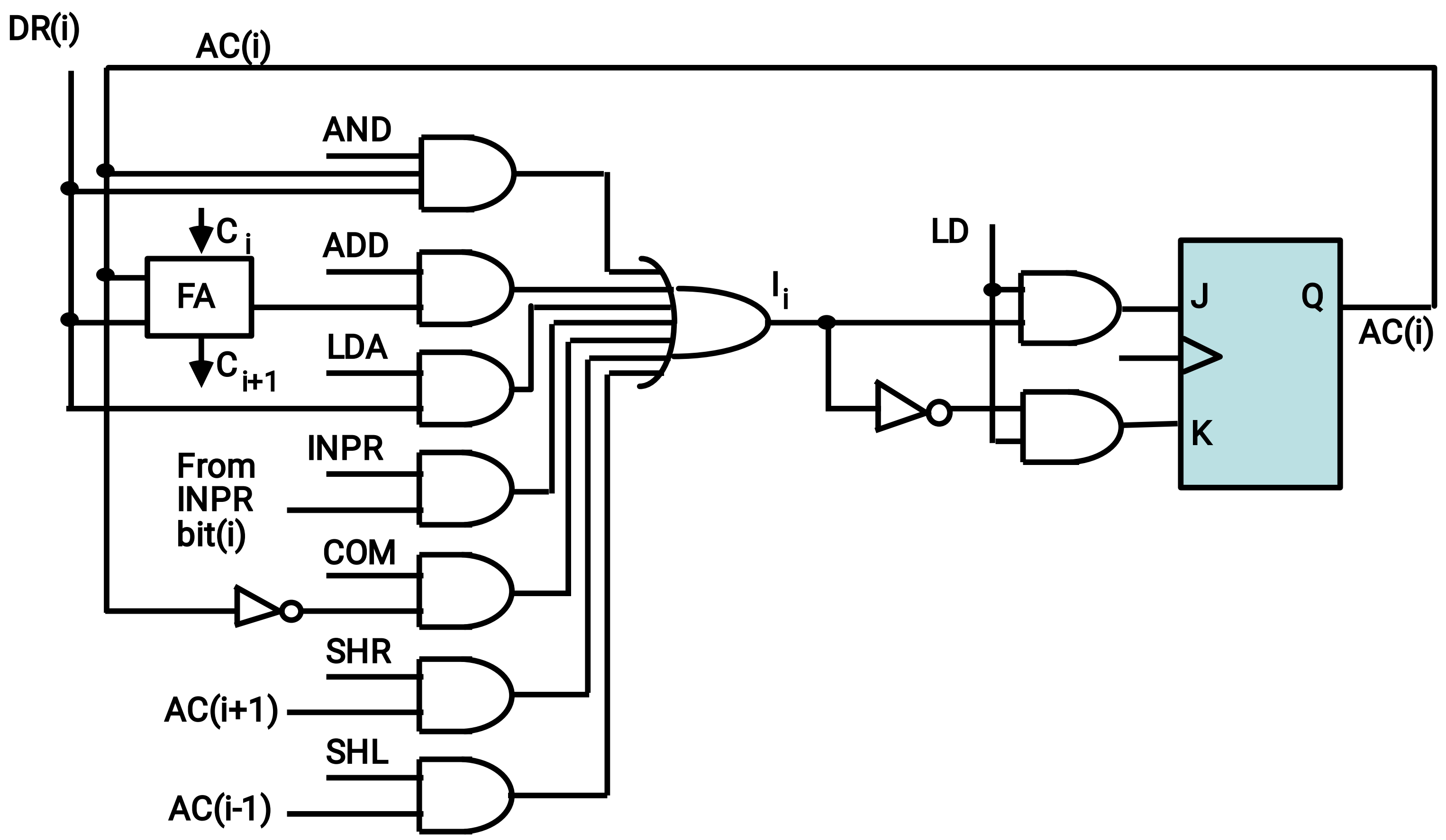
D_0T_5: $AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5: $AC \leftarrow AC + DR$	Add with DR
D_2T_5: $AC \leftarrow DR$	Transfer from DR
pB_{11}: $AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9: $AC \leftarrow AC'$	Complement
rB_7 : $AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$	Shift right
rB_6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

5-10 Design of Accumulator Logic$^{\text{cont.}}$

Gate structures for controlling the LD, INR, and CLR of AC



Adder and Logic Circuit


DR(i)
AC(i)
AND
ADD
LDA
INPR
COM
SHR
SHL
C_i
FA
C_{i+1}
From INPR bit(i)
AC(i+1)
AC(i-1)
I_i
LD
J
Q
K
AC(i)