Unit-2
Basic Computer Organization and Design

Marwadi University

Department of Computer Engineering

Computer Organization and Architecture-01CE1402

Prof. Kishan Makadiya

Outline

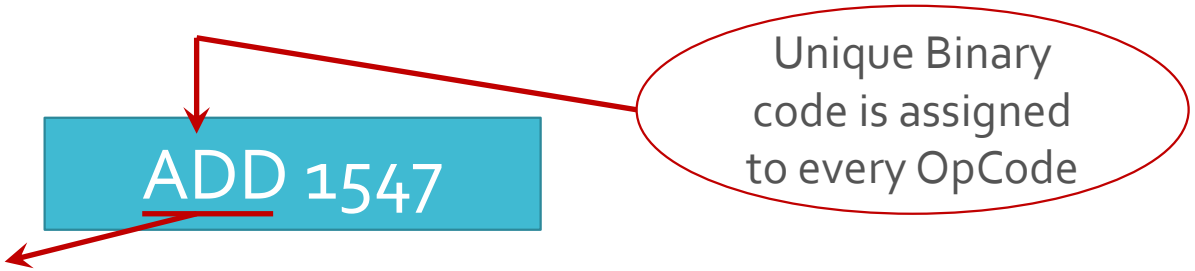
- Instruction codes
- Computer registers
- Computer instructions
- Timing and Control
- Instruction cycle
- Memory-Reference Instructions
- Input- output and interrupt
- Complete description
- Design of Basic Computer
- Design of Accumulator Logic

Instruction Codes

- Program
  - A program is a set of instructions that specify the operations, operands and the sequence by which processing has to occur.
- Computer Instruction
  - A computer instruction is a binary code that specifies a sequence of microoperations for the computer.
  - The computer reads each instruction from memory and places it in a control register.
  - The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.

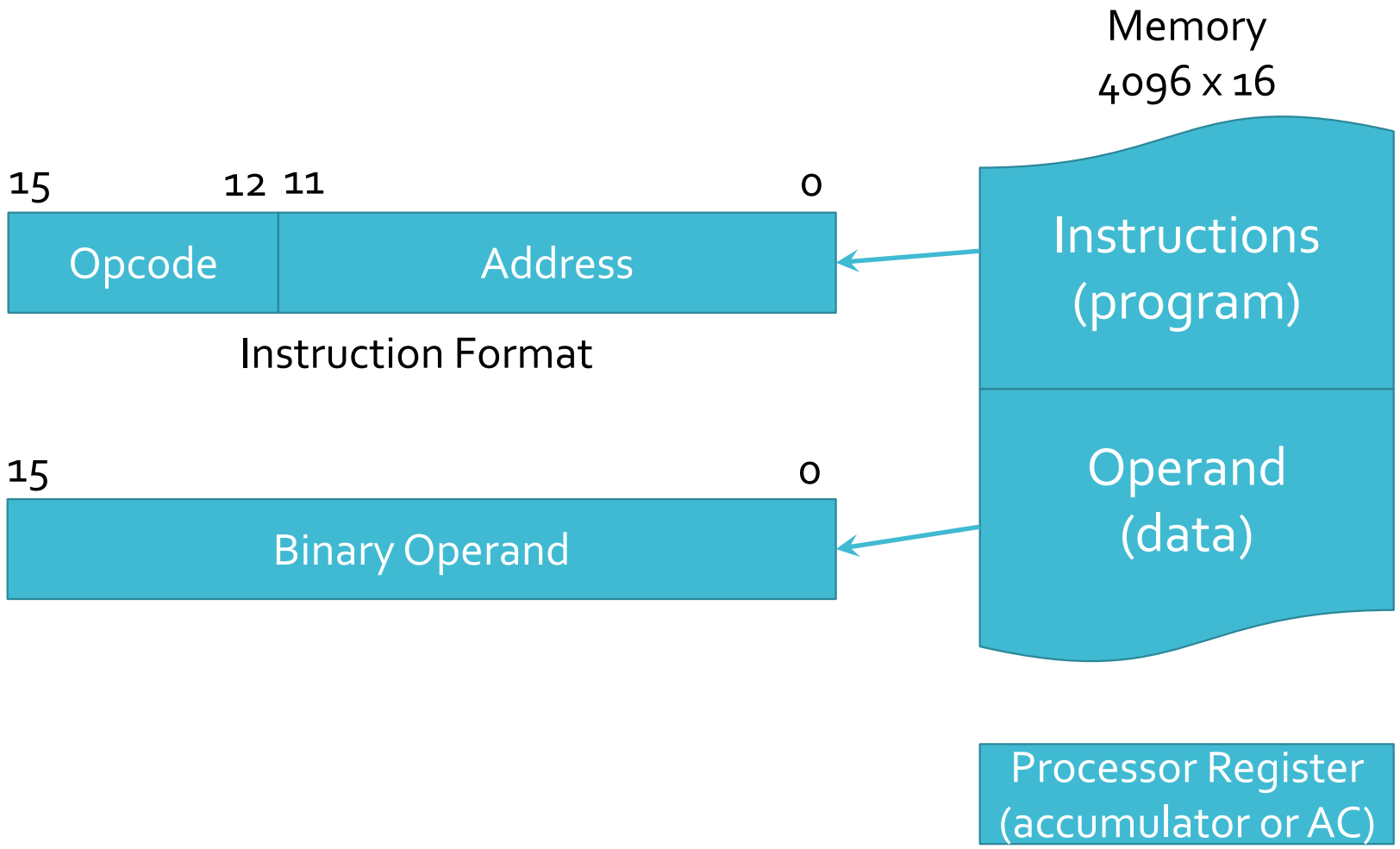
Instruction Codes

- Instruction Code
  - An instruction code is a group of bits that instruct the computer to perform a specific operation.
  - Example



- Operation Code (Opcode)
  - The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
  - The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.
  - The operation code must consist of at least n bits for a given 2^n (or less) distinct operations.

Stored Program Organization



Stored Program Organization

- The simplest way to organize a computer is to have one processor register(AC) and an instruction code format with two parts.
- The first part specifies the operation (opcode) to be performed and the second specifies an address (operand).
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.

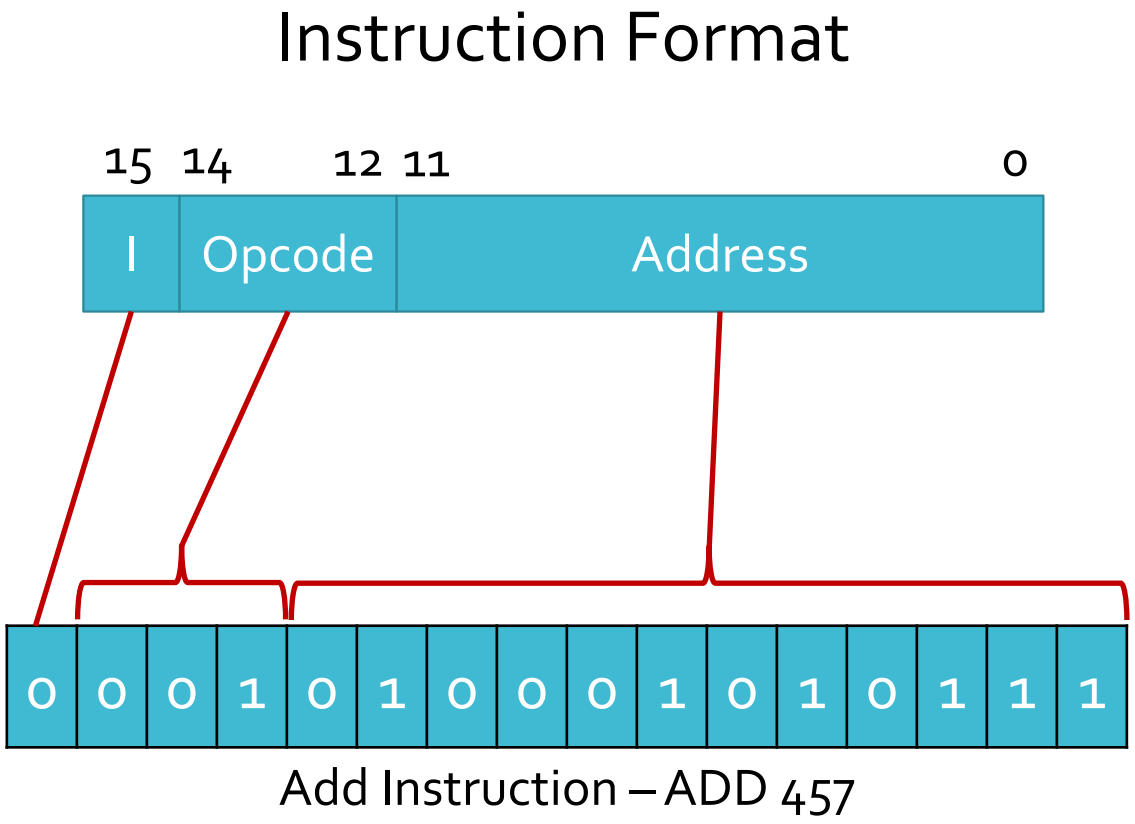
Stored Program Organization

- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words, we need 12 bits to specify an address since $2^{12} = 4096$.
- If we store each instruction code in one 16-bit memory word, we have available four bits for operation code (opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

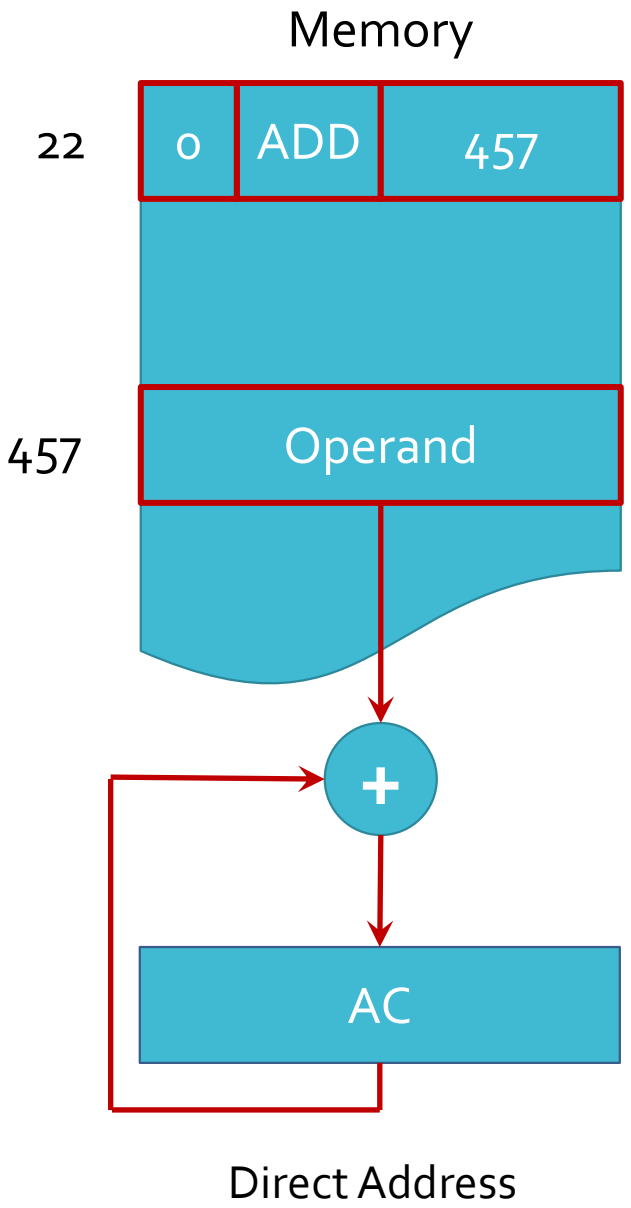
Stored Program Organization

- The control reads a 16-bit instruction from the program portion of memory.
- It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
- It then executes the operation specified by the operation code.

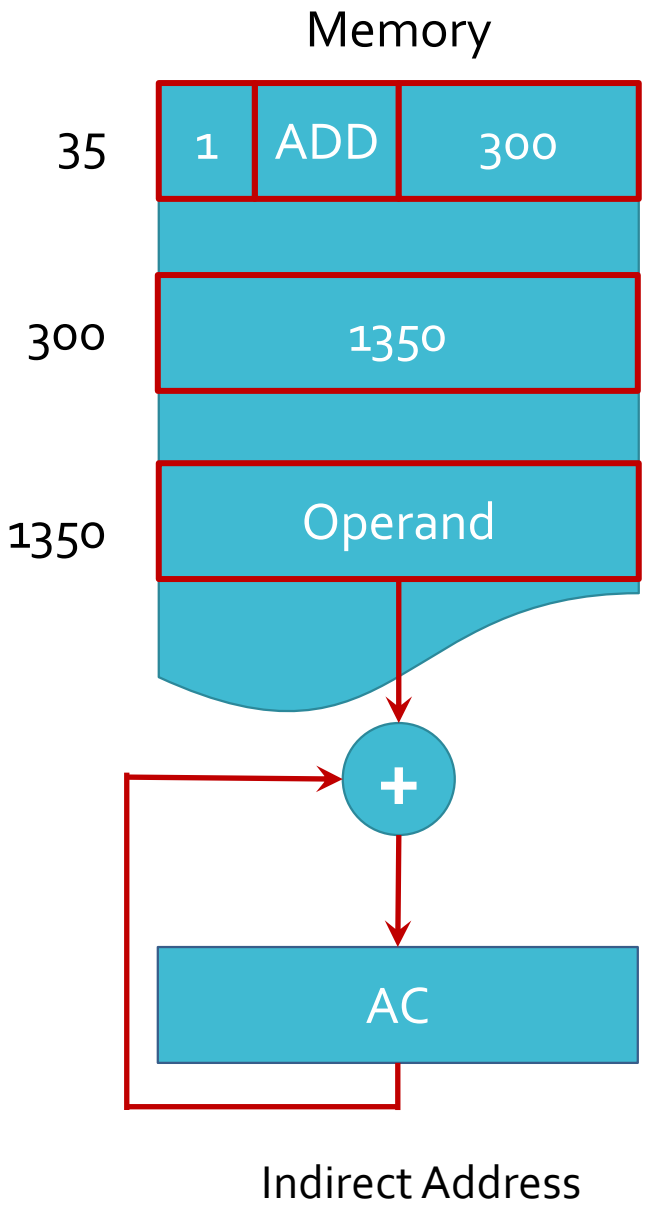
Instruction format of basic computer



Direct & Indirect Addressing of Memory



Direct Address



Indirect Address

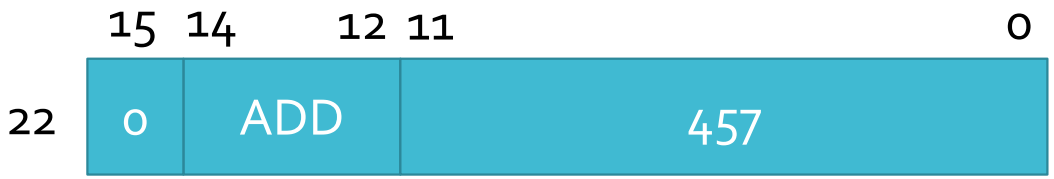
Direct & Indirect Addressing of Memory

- If the second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.

Direct & Indirect Addressing of Memory

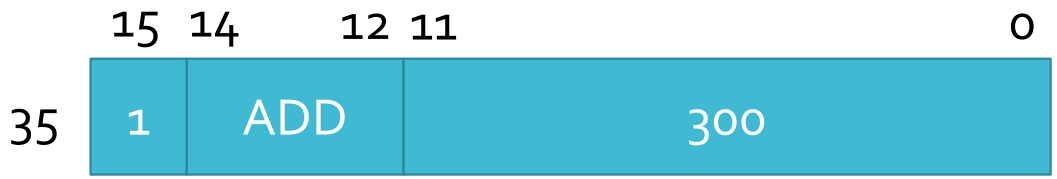
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The mode bit is 0 for a direct address and 1 for an indirect address.

Direct & Indirect Addressing of Memory



- A direct address instruction is placed at address 22 in memory.
- The I bit is 0, so the instruction is recognized as a direct address instruction.
- The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.

Direct & Indirect Addressing of Memory

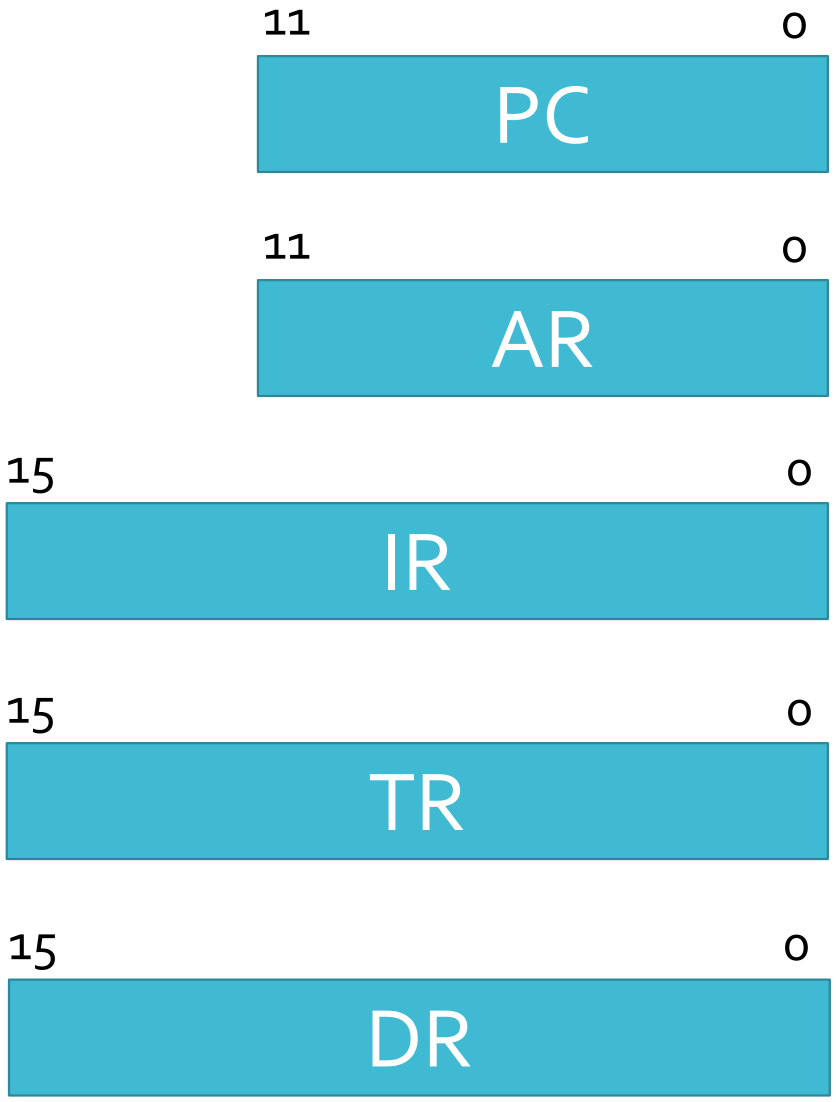


- The instruction in address 35 has a mode bit I = 1, recognized as an indirect address instruction.
- The address part is the binary equivalent of 300.
- The control goes to address 300 to find the address of the operand.
- The address of the operand in this case is 1350.
- The operand found in address 1350 is then added to the content of AC.

Direct & Indirect Addressing of Memory

- The indirect address instruction needs two references to memory to fetch an operand.
- The first reference is needed to read the address of the operand.
- Second reference is for the operand itself.
- The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data.

Computer Registers



Program Counter(12)
Holds address of instruction

Address Register(12)
Holds address for memory

Instruction Register(16)
Holds instruction code

Temporary Register(16)
Holds temporary data

Data Register(16)
Holds memory operand

Computer Registers

Accumulator(16)
Processor Register

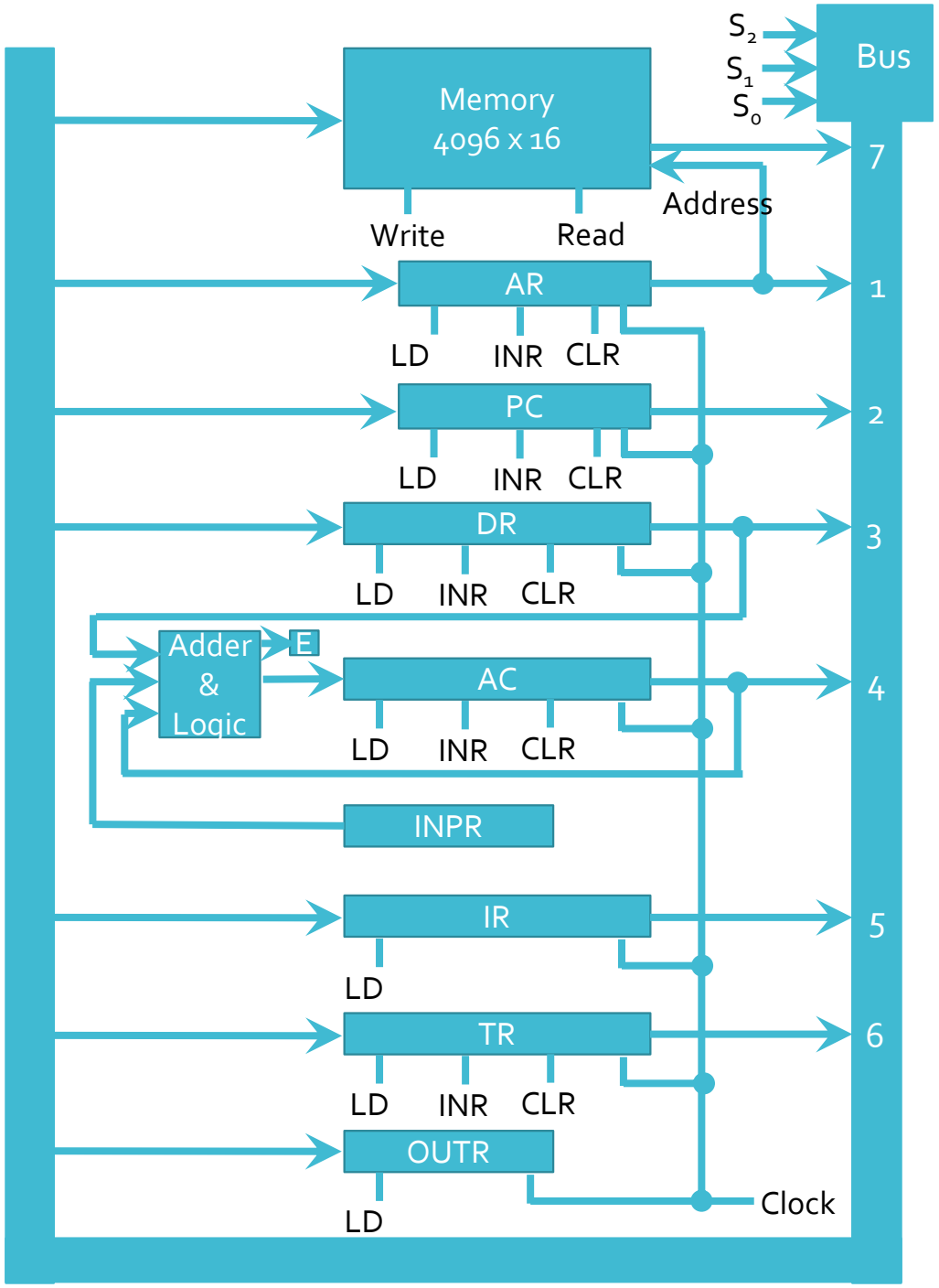
Output Register(8)
Holds output character

Input Register(8)
Holds input character

Memory
4096 words
16 bits per word

Common bus system of basic computer

- Common bus


Bus
S_2
S_1
S_0
Memory
4096 x 16
Address
Write
Read
AR
LD
INR
CLR
PC
LD
INR
CLR
DR
LD
INR
CLR
Adder
&
Logic
E
AC
LD
INR
CLR
INPR
IR
LD
TR
LD
INR
CLR
OUTR
LD
Clock
7
1
2
3
4
5
6

Types of Computer Instructions

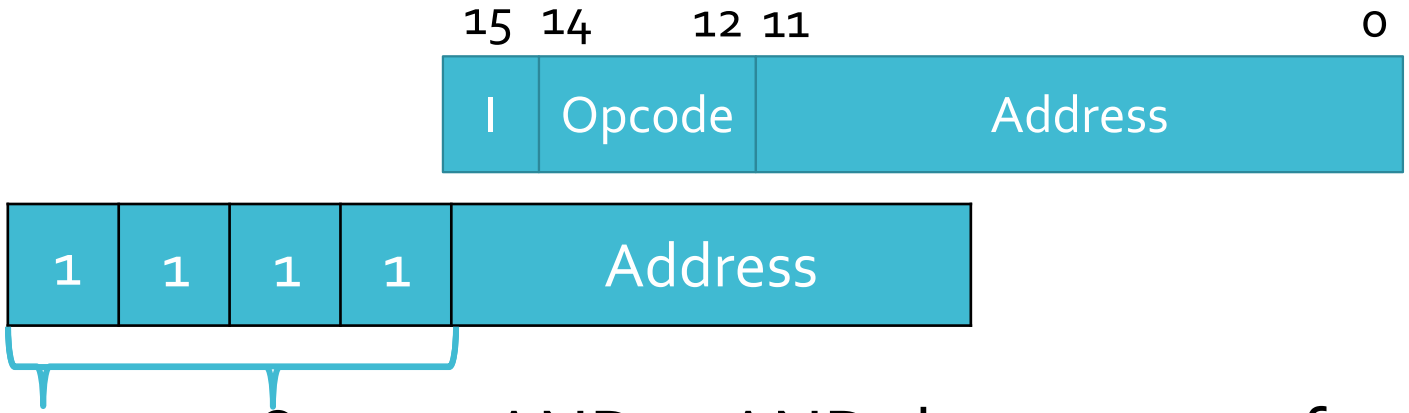
1. Memory Reference Instruction

2. Register Reference Instruction

3. Input – Output Instruction

Types of Computer Instructions

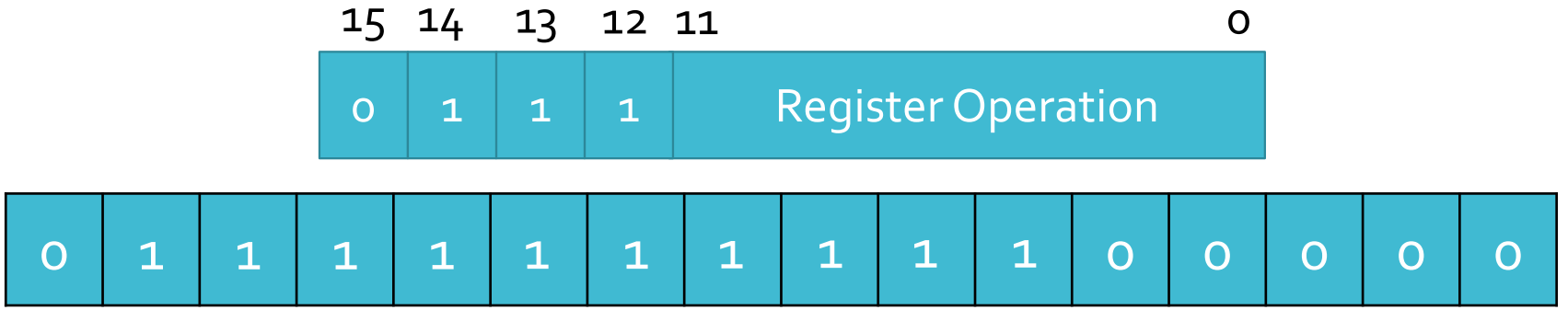
1. Memory Reference Instruction



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

Types of Computer Instructions

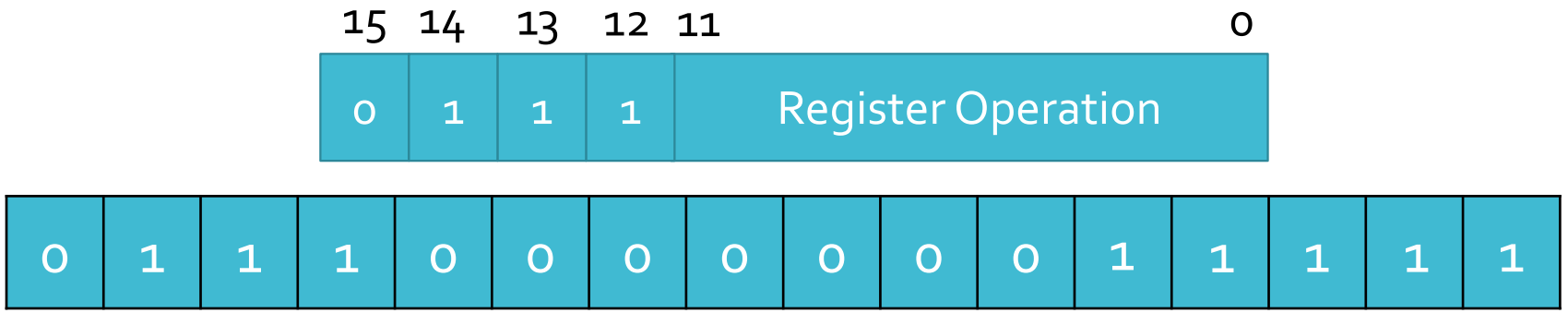
2. Register Reference Instruction



7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC
7100	CME	Complement E
7080	CIR	Circulate right AC and E
7040	CIL	Circulate left AC and E
7020	INC	Increment AC

Types of Computer Instructions

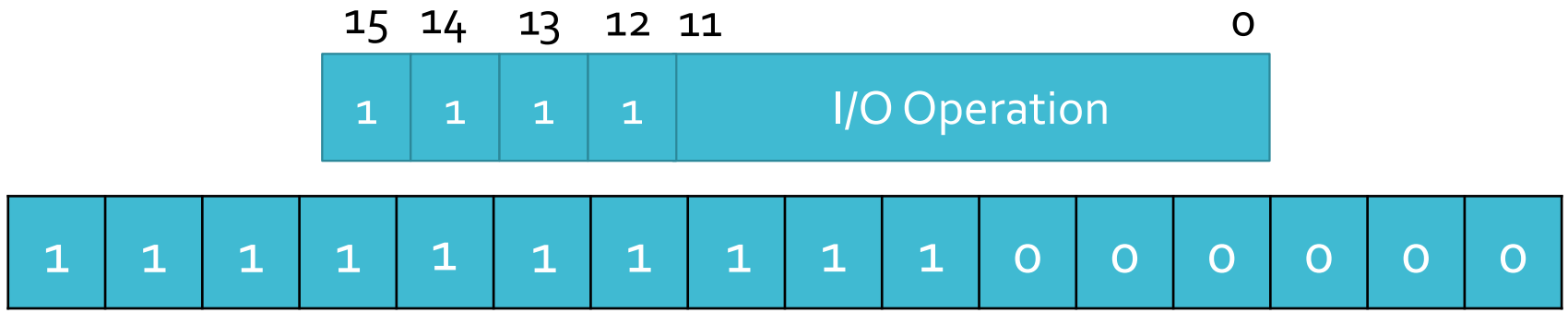
2. Register Reference Instruction



7010	SPA	Skip next instruction if AC is positive
7008	SNA	Skip next instruction if AC is negative
7004	SZA	Skip next instruction if AC is zero
7002	SZE	Skip next instruction if E is zero
7001	HLT	Halt computer

Types of Computer Instructions

3. Input – Output Instruction

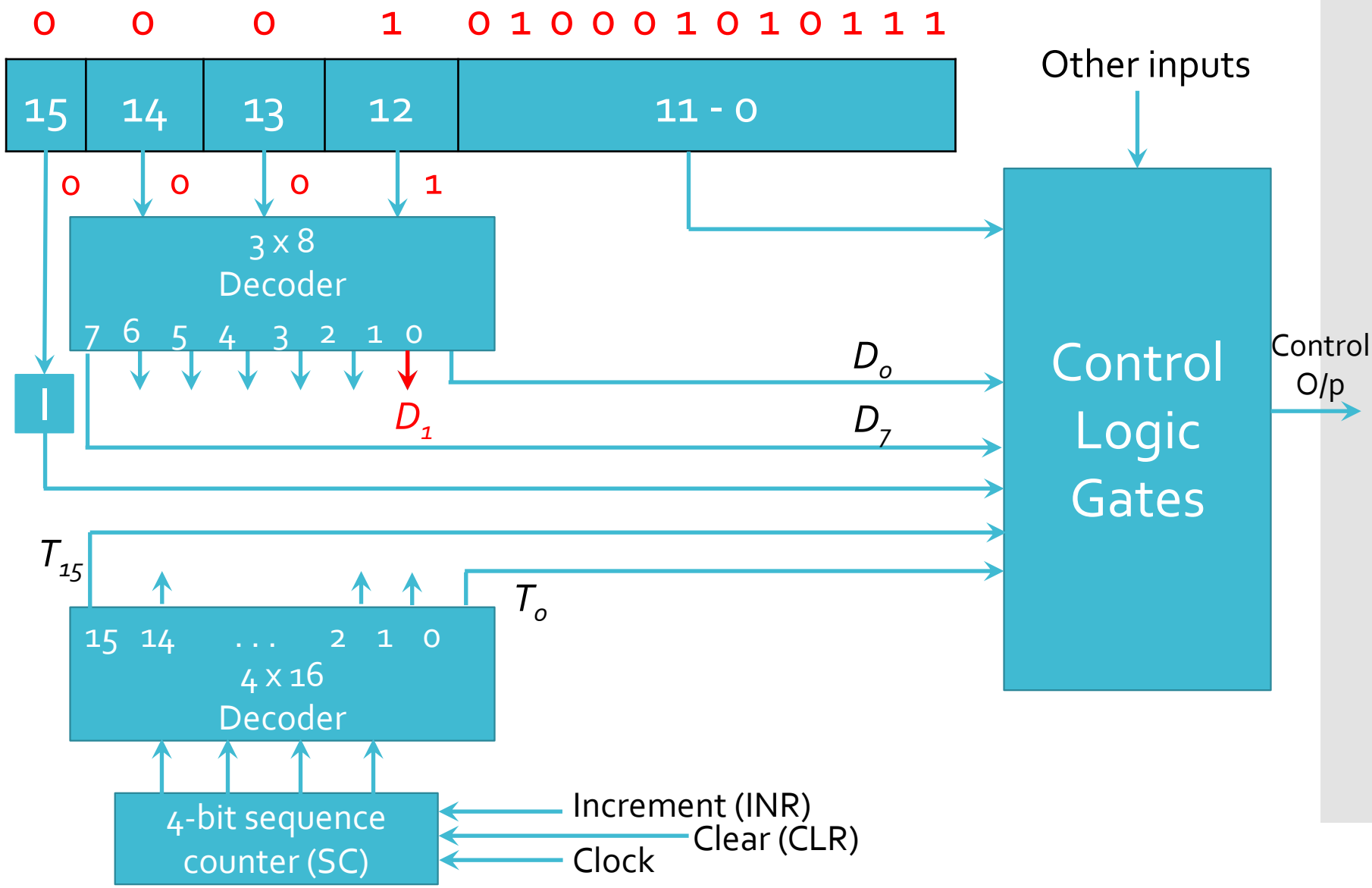


F800	INP	Input character to AC
F400	OUT	Output character from AC
F200	SKI	Skip on input flag
F100	SKO	Skip on output flag
F080	ION	Interrupt on
F040	IOF	Interrupt off

Instruction Set Completeness

- Instruction set is said to be complete if it includes sufficient number of instructions in each of the following categories:
  1. Arithmetic, logical and shift instructions
  2. Instructions for moving information to and from memory and processor registers
  3. Program control instructions together with instructions that check status conditions
  4. Input and output instructions

Control Unit of Basic Computer



Control Unit

- Components of Control unit are
  1. Two decoders
  2. A sequence counter
  3. Control logic gates
- An instruction read from memory is placed in the instruction register (IR).
- In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder.
- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.

Control Unit

- The eight outputs of the decoder are designated by the symbols D_0 through D_7.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit sequence counter can count in binary from 0 through 15. The outputs of counter are decoded into 16 timing signals T_0 through T_{15}.
- The sequence counter SC can be incremented or cleared synchronously.
- Most of the time, the counter is incremented to provide the sequence of timing signals out of 4 X 16 decoder.
- Once in awhile, the counter is cleared to 0, causing the next timing signal to be T_0.

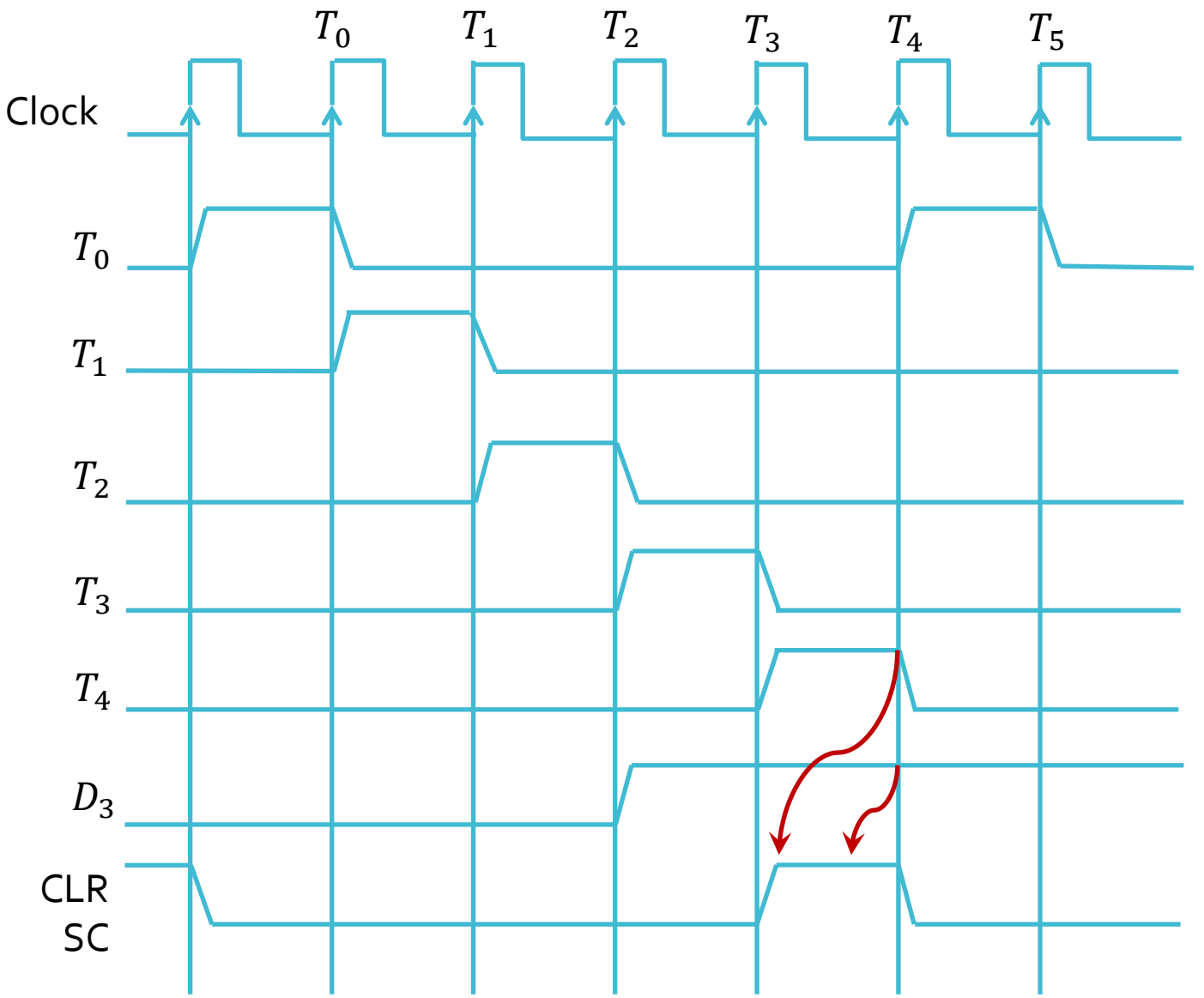
Control Unit

- As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, SC is cleared to o if decoder output D_3 is active. This is expressed symbolically by the statement

$$D_3T_4: SC \leftarrow o$$

- Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to o, which in turn activates the timing T_0 out of the decoder.
- T_0 is active during one clock cycle.
- The positive clock transition labeled T_0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T_0.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This procedures the sequence of timing signals T_0, T_1, T_2, T_3 and T_4, and so on. If SC is not cleared, the timing signals will continue with T_5, T_6, up to T_{15} and back to T_0.

Timing Cycle for D_3T_4: SC ← 0



Control Unit

- The last three waveforms shows how SC is cleared when D_3T_4 = 1.
- Output D_3 from the operation decoder becomes active at the end of timing signal T_2.
- When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active.
- This signal is applied to the CLR input of SC.
- On the next positive clock transition the counter is cleared to o.
- This causes the timing signal T_o to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Control Organization

- Hardwired Control
  - The control logic is implemented with gates, flips-flops, decoders and other digital circuits.
  - It can be optimized to produce a fast mode of operation.
  - It requires changes in the wiring among the various components if the design has to be modified or changed.
- Microprogrammed Control
  - The control information is stored in a control memory.
  - The control memory is programmed to initiate the required sequence of micro-operations.
  - Any required changes or modifications can be done by updating the microprogram in control memory.

Instruction Cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
- After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.
- This process continues unless a HALT instruction is encountered.

Instruction Cycle

- Fetch & Decode
  - PC is loaded with the address of the first instruction in the program.
  - The micro-operations for fetch and decode phases are as follows:

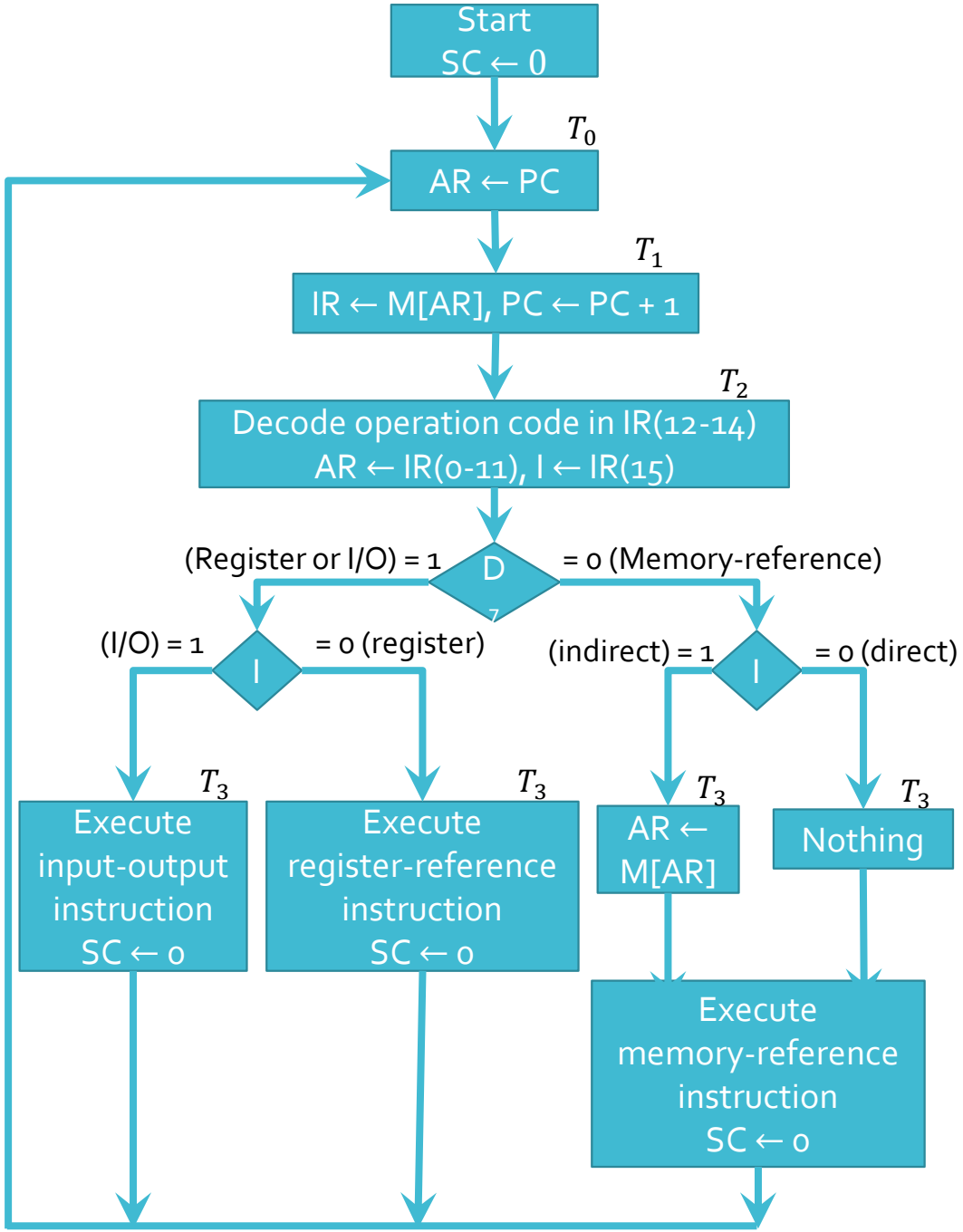
$$T_0 : AR \leftarrow PC$$

$$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$$

$$T_2 : D_0, \dots, D_7 \leftarrow Decode\ IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$$

Instruction Cycle

- Determine the type of instruction
  - During time T_3, the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-Output instruction.
  - If $D_7 = 1$ then instruction must be register reference or input-output else memory reference instruction.
- Instruction Cycle Flowchart


Start
SC ← 0
T_0
AR ← PC
T_1
IR ← M[AR], PC ← PC + 1
T_2
Decode operation code in IR(12-14)
AR ← IR(0-11), I ← IR(15)
D
(Register or I/O) = 1
= 0 (Memory-reference)
I
(I/O) = 1
= 0 (register)
I
(indirect) = 1
= 0 (direct)
T_3
Execute input-output instruction
SC ← 0
T_3
Execute register-reference instruction
SC ← 0
T_3
AR ← M[AR]
T_3
Nothing
Execute memory-reference instruction
SC ← 0

Register Reference Instruction

$D_7 I' T_3 = r$ (common to all register reference instructions)

$IR(i) = B_i$ [bit in IR(0-11) that specifies the operation]

CLA	rB_{11}	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}	$E \leftarrow 0$	Clear E
CMA	rB_9	$AC \leftarrow AC'$	Complement AC
CME	rB_8	$E \leftarrow E'$	Complement E
CIR	rB_7	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is positive
SNA	rB_3	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if AC is negative
SZA	rB_2	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	Skip if AC is zero
SZE	rB_1	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E is zero
HLT	rB_0	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt Computer

Memory Reference Instructions

1. AND: AND to AC

   This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC.

   D_0T_4: DR$\leftarrow$M[AR]

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

Memory Reference Instructions

2. ADD: ADD to AC

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

   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

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

D_2T_4: DR←M[AR]

D_2T_5: AC ← DR, SC ← 0

Memory Reference Instructions

4. STA: Store AC

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

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

Memory Reference Instructions

5. BUN: Branch Unconditionally

   This instruction transfers the program to instruction specified by the effective address. The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.

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

Memory Reference Instructions

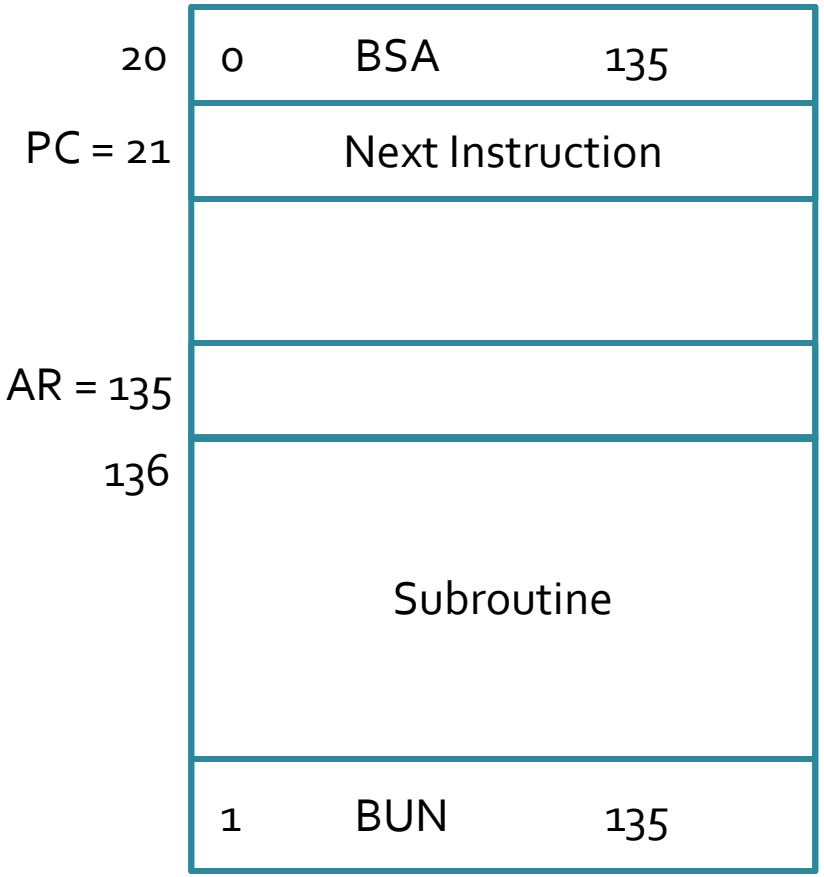
6. BSA: Branch and Save Return Address

   This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

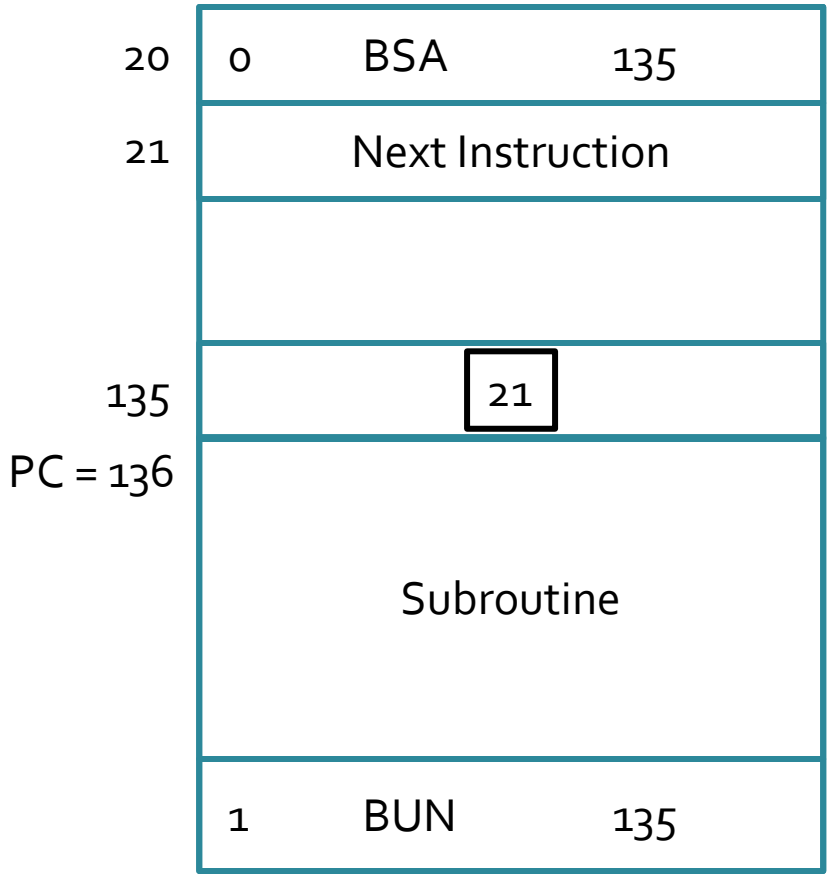
   D_5T_4: $M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$

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

BSA



Memory, PC and AR at Time T_4



Memory and PC after execution

D_5T_4: $M[135] \leftarrow 21,\ AR \leftarrow 135 + 1$

D_5T_5: $PC \leftarrow 136,\ SC \leftarrow 0$

Memory Reference Instructions

7. ISZ: Increment and Skip if Zero

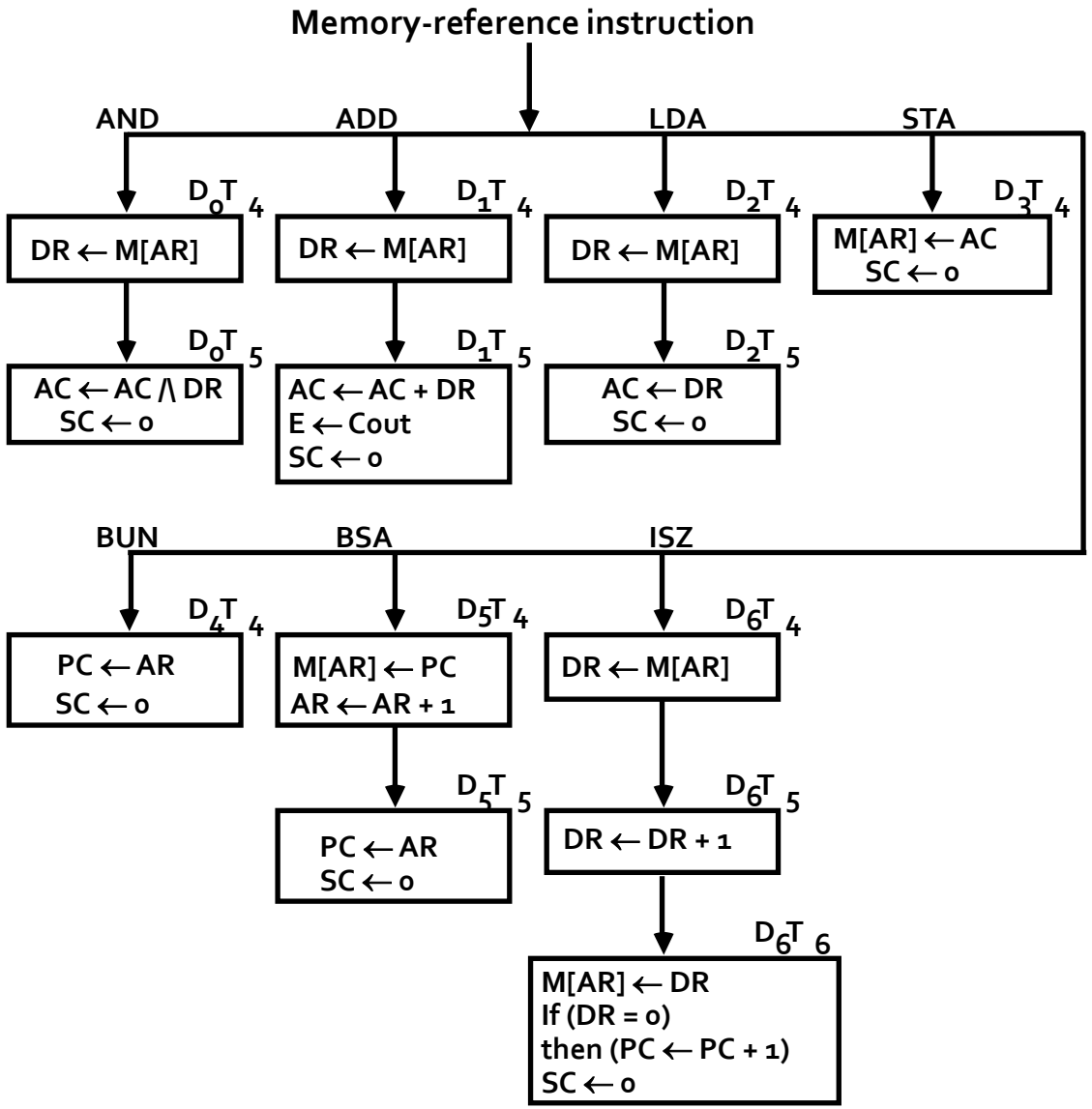
   These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.

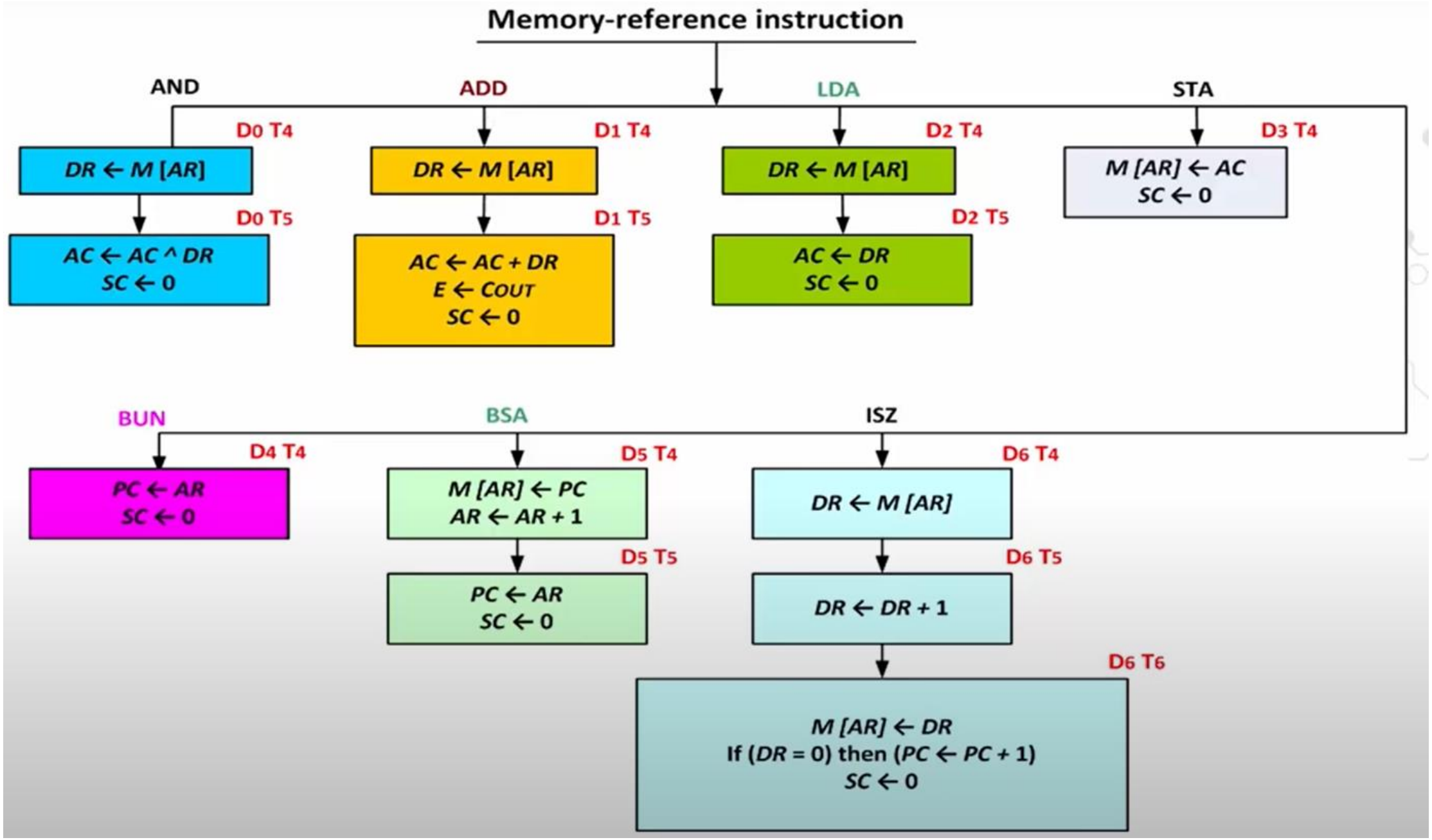
D_6T_4: $DR \leftarrow M[AR]$

D_6T_5: $DR \leftarrow DR + 1$

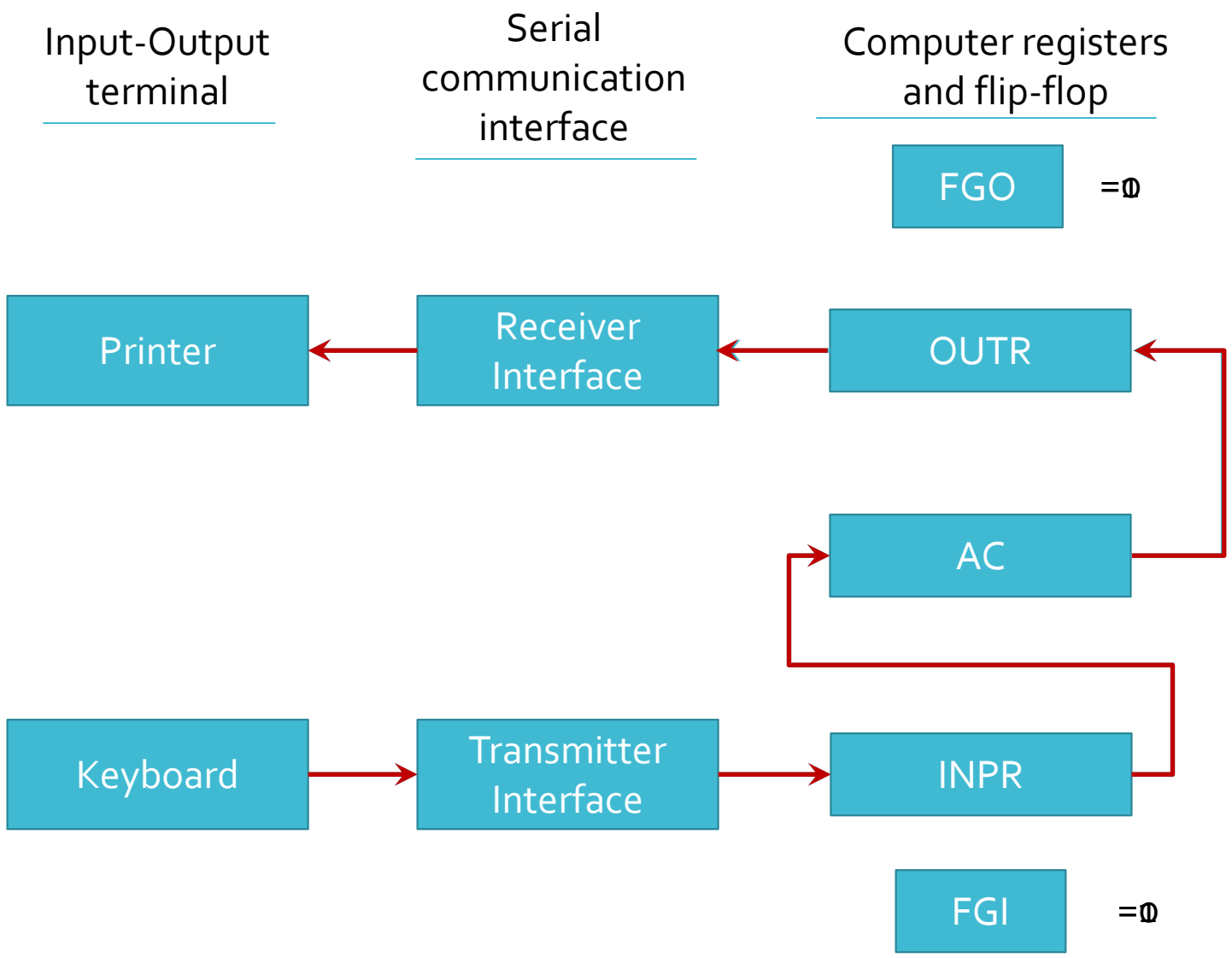
D_6T_6: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

Flowchart for Memory Reference Instructions




Memory-reference instruction
AND
ADD
LDA
STA
D0 T4
D1 T4
D2 T4
D3 T4
DR ← M [AR]
DR ← M [AR]
DR ← M [AR]
M [AR] ← AC
SC ← 0
D0 T5
D1 T5
D2 T5
AC ← AC ^ DR
SC ← 0
AC ← AC + DR
E ← COUT
SC ← 0
AC ← DR
SC ← 0
BUN
BSA
ISZ
D4 T4
D5 T4
D6 T4
PC ← AR
SC ← 0
M [AR] ← PC
AR ← AR + 1
DR ← M [AR]
D5 T5
D6 T5
PC ← AR
SC ← 0
DR ← DR + 1
D6 T6
M [AR] ← DR
If (DR = 0) then (PC ← PC + 1)
SC ← 0

Input-Output of basic computer



Input-Output of basic computer

- A computer can serve no useful purpose unless it communicates with the external environment.
- To exhibit the most basic requirements for input and output communication, we will use a terminal unit with a keyboard and printer.
- The terminal sends and receives serial information and each quantity of information has eight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with the AC in parallel.

Process of input information transfer

- Initially, the input flag FGI is cleared to 0. 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 cannot be changed by striking another key. 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.
- Once the flag is cleared, new information can be shifted into INPR by striking another key.

Process of outputting information

- The output register OUTR works similarly but the direction of information flow is reversed.
- Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
- The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

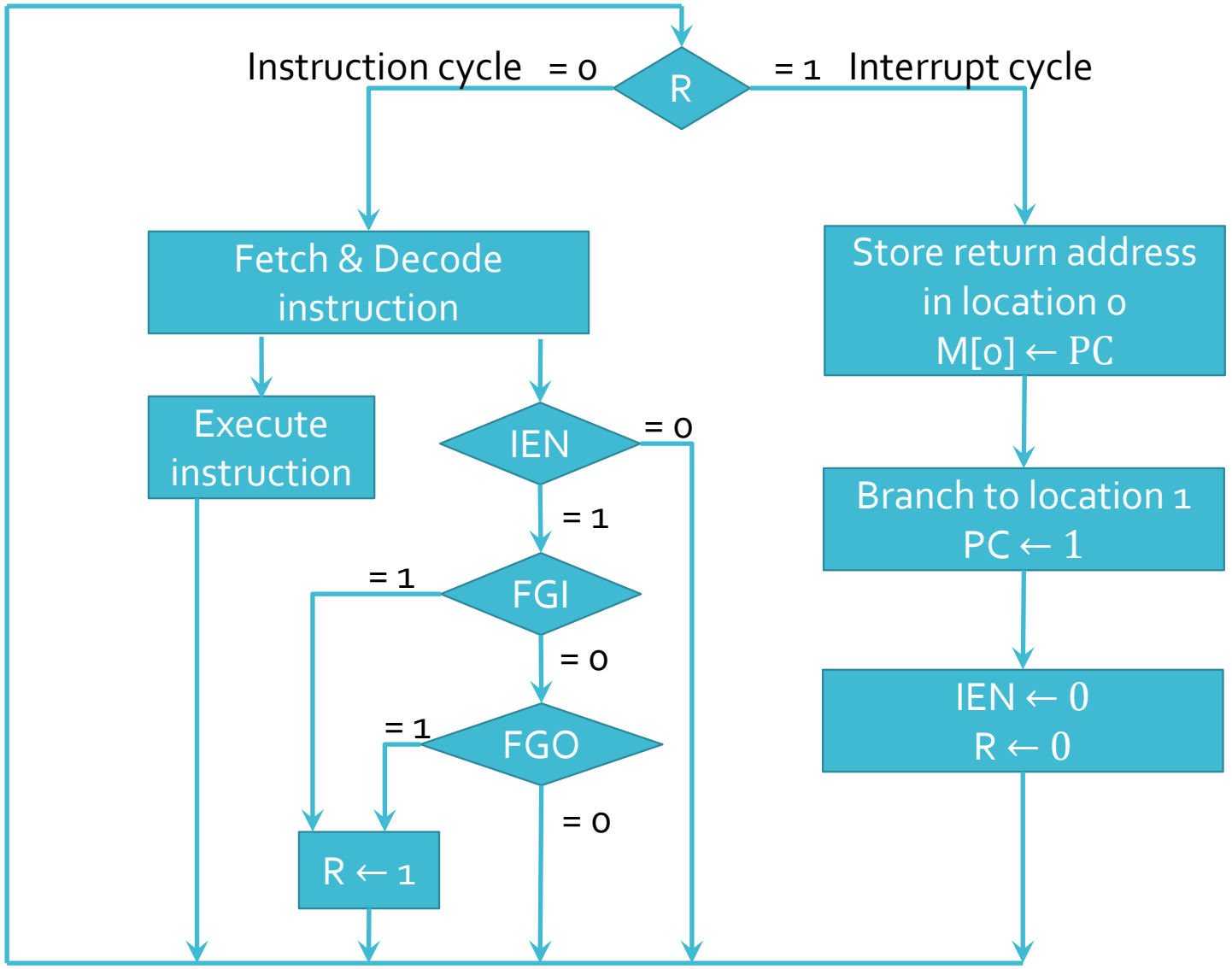
Input-Output Instruction

$D_7IT_3 = p$ (common to all input-output instructions)

$IR(i) = B_i$ [bit in IR(6-11) that specifies the operation]

INP	pB_{11}	$AC(0\text{-}7) \leftarrow INPR, FGI \leftarrow 0$	Input Character
OUT	pB_{10}	$OUTR \leftarrow AC(0\text{-}7), FGO \leftarrow 0$	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

Interrupt Cycle



Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- An interrupt flip-flop R is included in the computer.
- When R = 0, the computer goes through an instruction cycle.

Interrupt Cycle

- During the execute phase of the instruction cycle IEN is checked by the control.
- If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits.
- If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information.
- In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1.
- At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

Register transfer statements for Interrupt cycle

- The flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T_0, T_1 or T_2 are active.
- The condition for setting flip-flop R= 1 can be expressed with the following register transfer statement:

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

- The symbol + between FGI and FGO in the control function designates a logic OR operation. This is AND with IEN and $T_0'T_1'T_2'$.

Register transfer statements for Interrupt cycle

- The fetch and decode phases of the instruction cycle must be modified and Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- Therefore the interrupt cycle statements are :

$$RT_0 : AR \leftarrow 0,\ TR \leftarrow PC$$

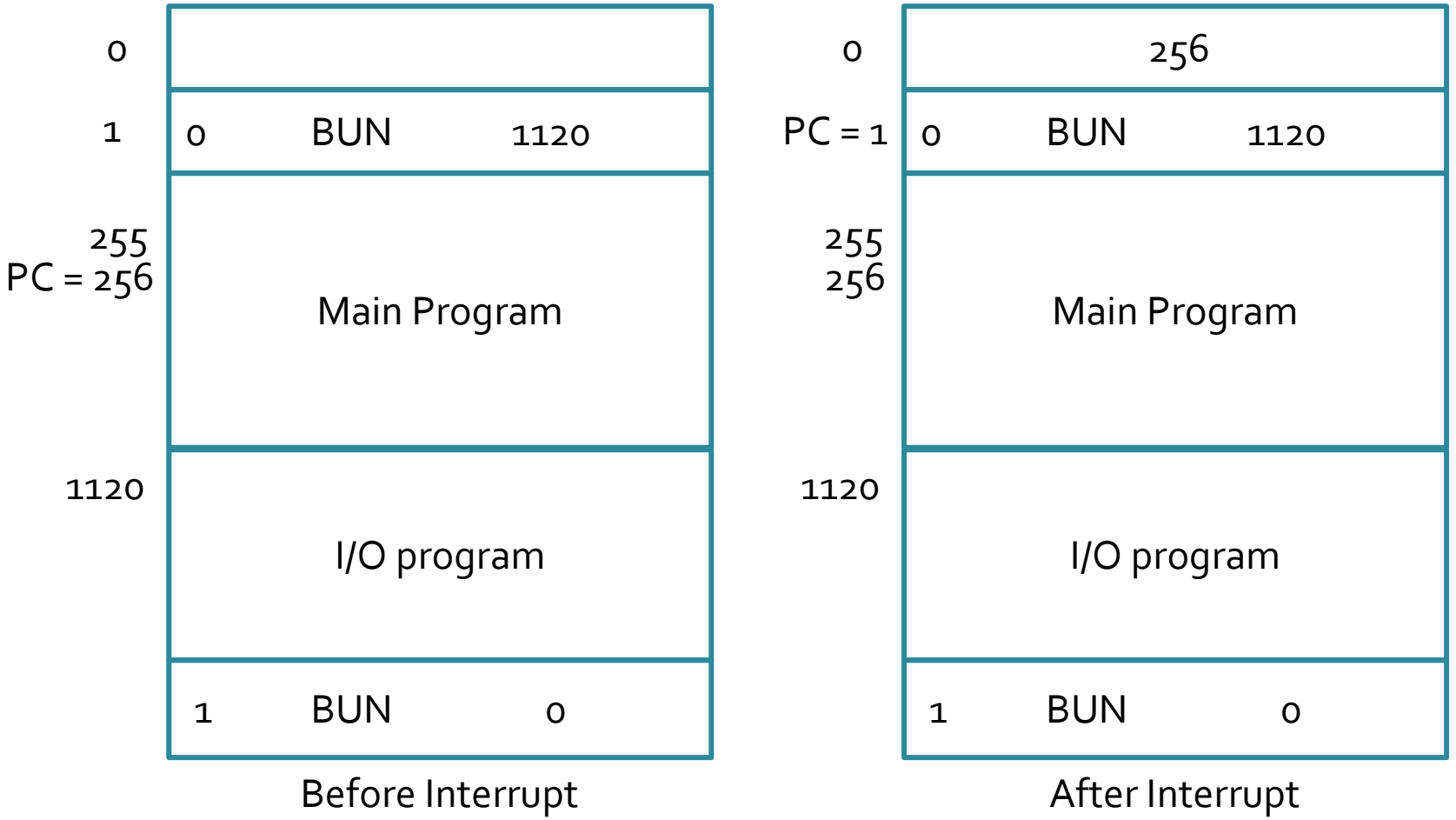
$$RT_1 : M[AR] \leftarrow TR,\ PC \leftarrow 0$$

$$RT_2 : PC \leftarrow PC + 1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$$

Register transfer statements for Interrupt cycle

- During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR.
- With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.
- The third timing signal increments PC to 1, clears IEN and R, and control goes back to T_0 by clearing SC to 0.
- The beginning of the next instruction cycle has the condition RT_0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

Demonstration of Interrupt Cycle



Complete Computer Description (Micro-operations)

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 (Micro-operations)

Register-Reference		
	$D_7I'T_3 = r$	(Common to all register-reference instr)
	$IR(i) = B_i$	(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
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 ← 0
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0
SKI	pB_9:	If(FGI=1) then (PC ← PC + 1)
SKO	pB_8:	If(FGO=1) then (PC ← PC + 1)
ION	pB_7:	IEN ← 1
IOF	pB_6:	IEN ← 0

Design of Basic Computer

- Hardware Components of Basic Computer

  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

  Adder and Logic circuit: Connected to AC

- Control Logic Gates:

  Input Controls of the nine registers

  Read and Write Controls of memory

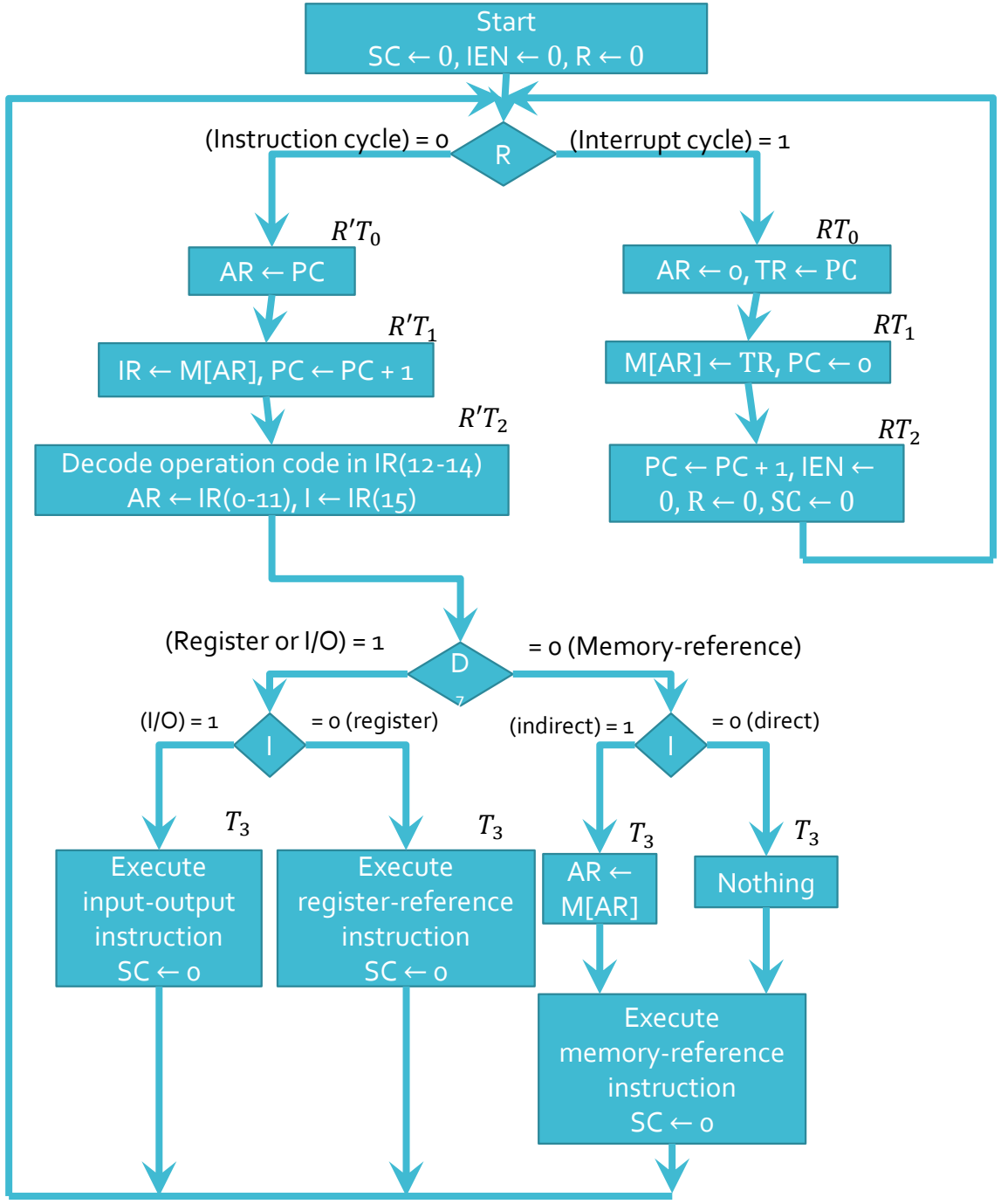
  Set, Clear, or Complement Controls of the flip-flops

  S_2, S_1, S_0 Controls to select a register for the bus

  AC, and Adder and Logic circuit

Design of Basic Computer

- Basic Computer

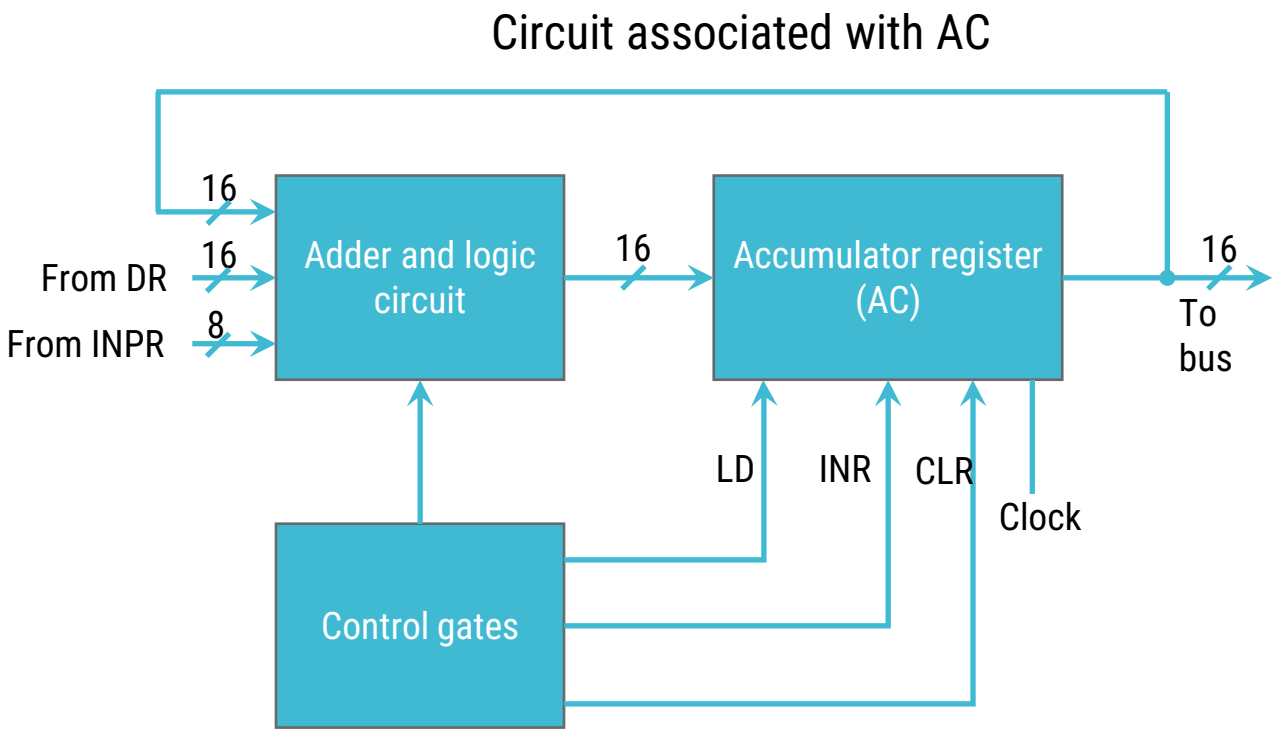

Start
SC ← 0, IEN ← 0, R ← 0
R
(Instruction cycle) = 0
(Interrupt cycle) = 1
$R'T_0$
AR ← PC
RT_0
AR ← 0, TR ← PC
$R'T_1$
IR ← M[AR], PC ← PC + 1
RT_1
M[AR] ← TR, PC ← 0
$R'T_2$
Decode operation code in IR(12-14)
AR ← IR(0-11), I ← IR(15)
RT_2
PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0
D
(Register or I/O) = 1
= 0 (Memory-reference)
I
(I/O) = 1
= 0 (register)
I
(indirect) = 1
= 0 (direct)
T_3
Execute input-output instruction
SC ← 0
T_3
Execute register-reference instruction
SC ← 0
T_3
AR ← M[AR]
T_3
Nothing
Execute memory-reference instruction
SC ← 0

Design of Accumulator Unit

- In order to design the logic associated with AC, it is necessary to extract all the statements that change the content of AC.

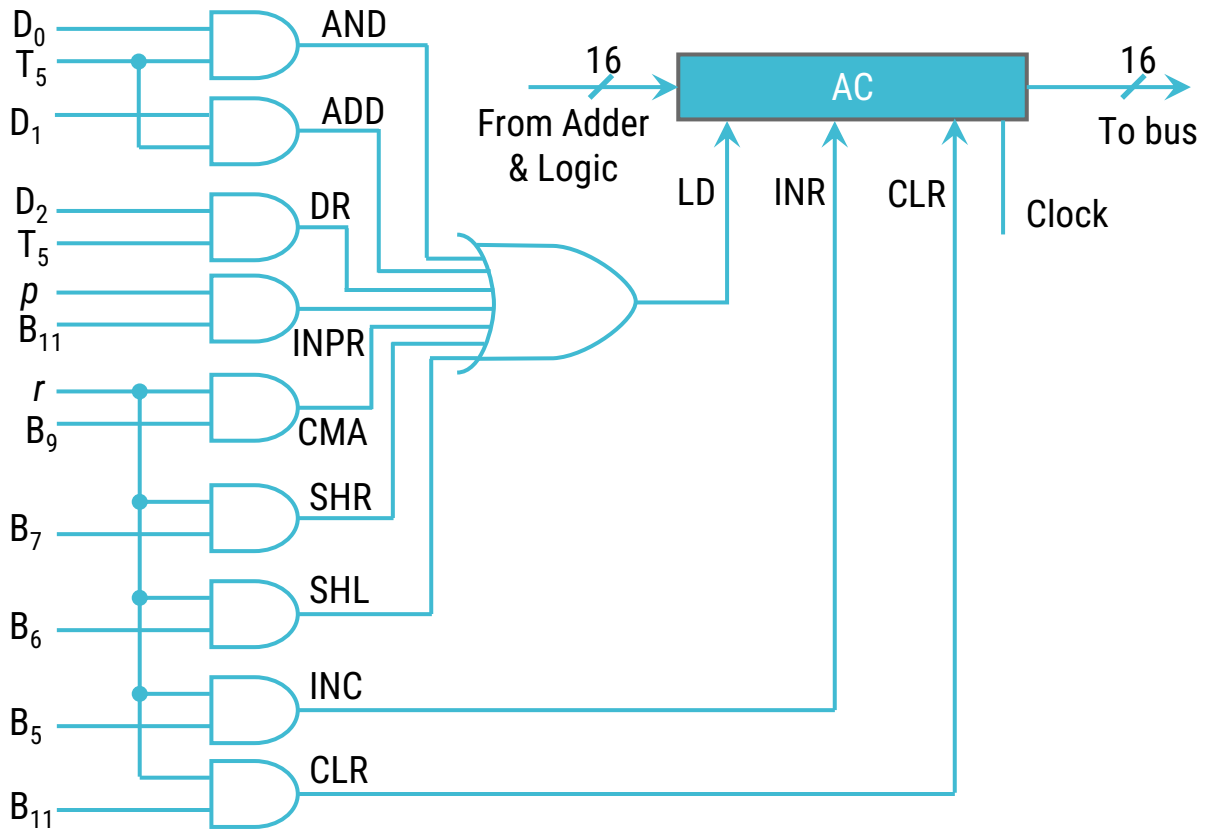
D_0T_5: AC ← AC ∧ DR, SC ← 0	AND with *DR*
D_1T_5: AC ← AC + DR, SC ← 0	ADD with *DR*
D_2T_5: AC ← DR	Transfer from *DR*
pB_{11}: AC(0-7) ← INPR, FGI ← 0	Transfer from *INPR*
rB_9: AC ← AC'	Complement
rB_7: AC ← shr AC, AC(15) ← E	Shift right
rB_6: AC ← shl AC, AC(0) ← E	Shift left
rB_{11}: AC ← 0	Clear
rB_5: AC ← AC + 1	Increment

Design of Accumulator Logic



Design of Accumulator Logic

Gate structure for controlling LD, INR and CLR of AC



Thank You