Unit 2

Basic Computer Organization and Design

SANTOSH MEENA

Instruction Codes

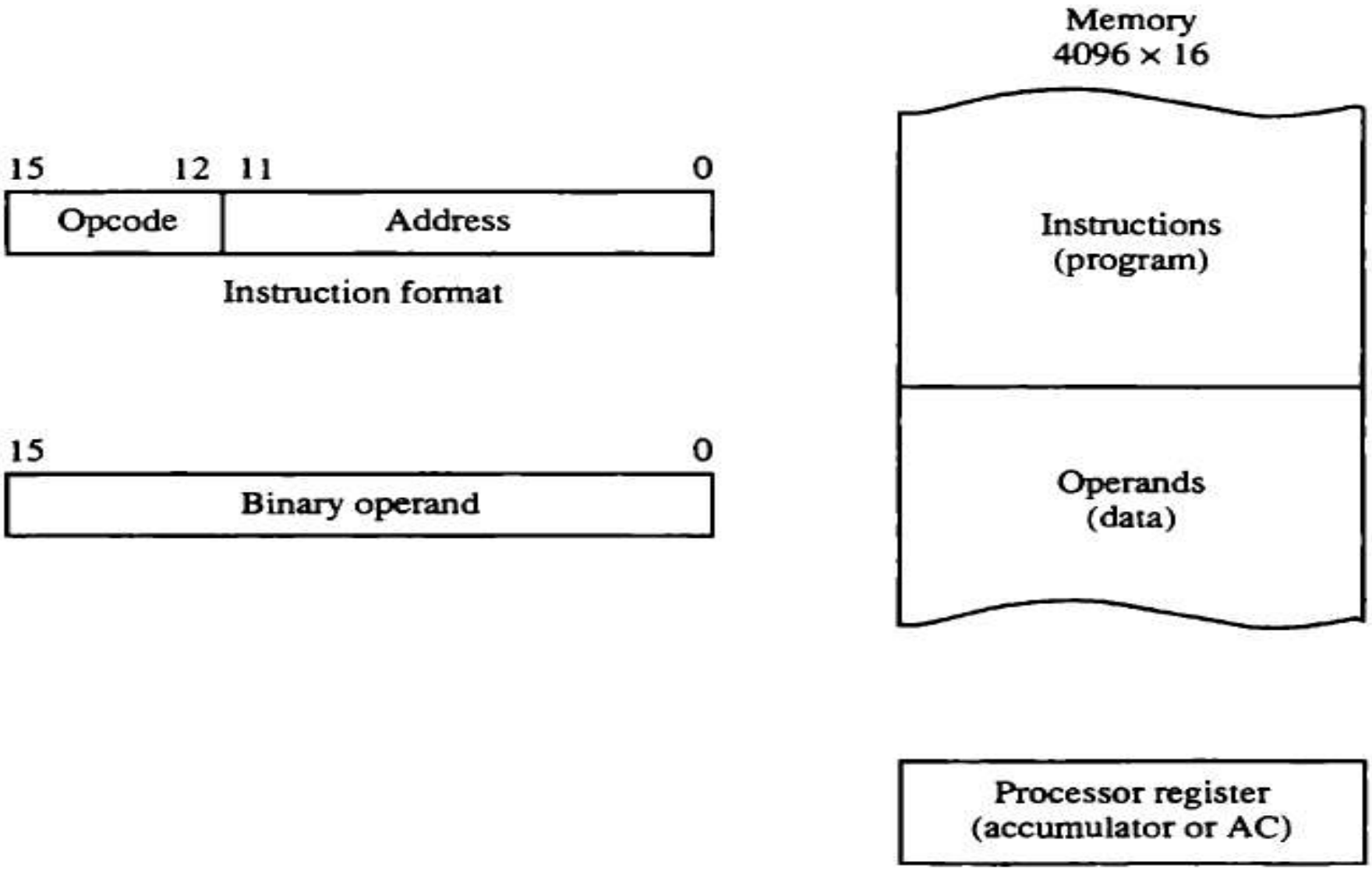
- **A computer instruction is a binary code** that specifies a sequence of micro-operations for the computer. The computer reads each instruction from memory and places it in a control register.
- The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. Every computer has its own unique instruction set.
- **Instruction Code:** An instruction code is a group of bits that instruct the computer to perform a specific operation.
- **Operation Code (Opcode):** The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. **The operation is specified by a binary code, known as the operation code, or opcode**. An operation code is sometimes called a **macro-operation** because it specifies a set of micro-operations
- **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.
- As an illustration, consider a computer with 64 (2^6) distinct operations, one of them being an ADD operation. The operation code consists of six bits, with a bit configuration 110010 assigned to the ADD operation.

Continue…

- **Stored Program Organization:** The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second part specifies an address. Instructions are stored in one section of memory and data in another memory section.
- For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$. If each instruction is stored in one 16-bit memory word, four bits are available for the operation code (opcode) to specify one out of 16 (2^4) possible different operations, and 12 bits to specify the address of an operand.
- 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.
- **Accumulator (AC):** Computers that have a single-processor register usually assign to it the name accumulator (AC). The operation is performed with the memory operand and the content of accumulator (AC).

Continue…

Figure Stored program organization



Continue…

- **Immediate instruction:** When second part of an instruction code does not specify an address but specifies an operand, the instruction is said to have an immediate operand.
- **Direct address:** When the second part specifies the address of an operand, the instruction is said to have a direct address.
- **Indirect address:** If 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, the instruction is said to have an indirect address.
- **One bit (mode bit-I)** of the instruction code can be used to distinguish between a direct and an indirect address. The mode bit (I) is 0 for a direct address and 1 for an indirect address.
- **Effective address:** We define the effective address to be the address of the operand in a computation-type instruction or the target address in a branch-type instruction. Thus the effective address in the instruction is 457 and 1350.

Continue…

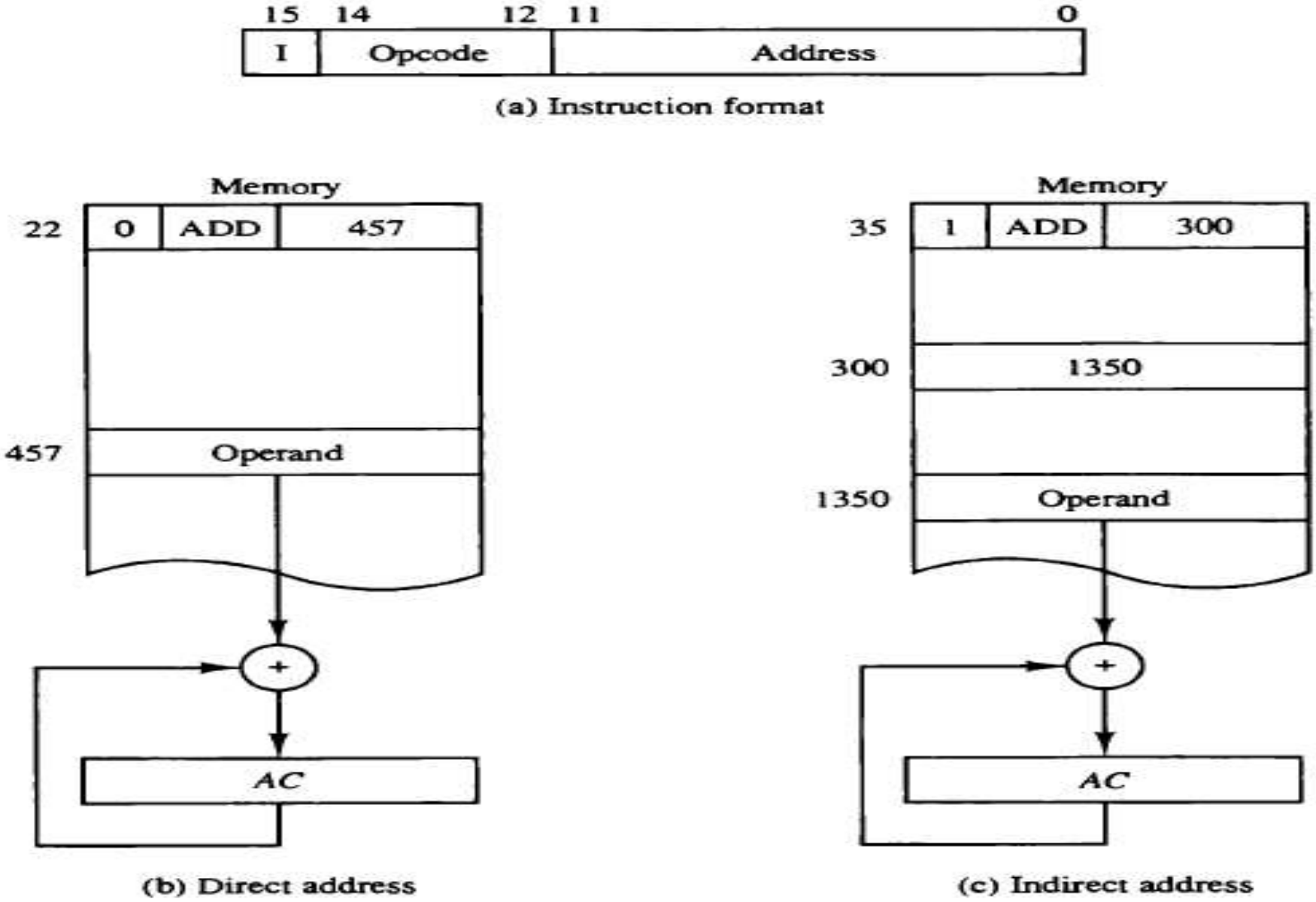


Figure Demonstration of direct and indirect address

Computer Registers

- Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.
- The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on.
- It is 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.
- The memory unit has a capacity of 4096 words and each word contains 16 bits. 12 bits of an instruction word are needed to specify the address of an operand. This leaves three bits for the operation, part of the instruction and a bit to specify a direct or indirect address.

Continue…

TABLE List of Registers for the Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Continue…

- **DR:** The data register (DR) holds the operand read from memory.
- **AC:** The accumulator (AC) register is a general-purpose processing register.
- **IR:** The instruction read from memory is placed in the instruction register (IR).
- **TR:** The Temporary register (TR) is used for holding temporary data during the processing.
- **AR:** The memory address register (AR) has 12 bits since this is the width of a memory address.
- **PC:** The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed. To read an instruction, memory read cycle is initiated. PC is then incremented by one, so it holds the address of the next instruction in sequence.
- **INPR:** The input register (INPR) receives an 8-bit character from an input device.
- **OUTR:** The output register (OUTR) holds an 8-bit character for an output device.

Continue…

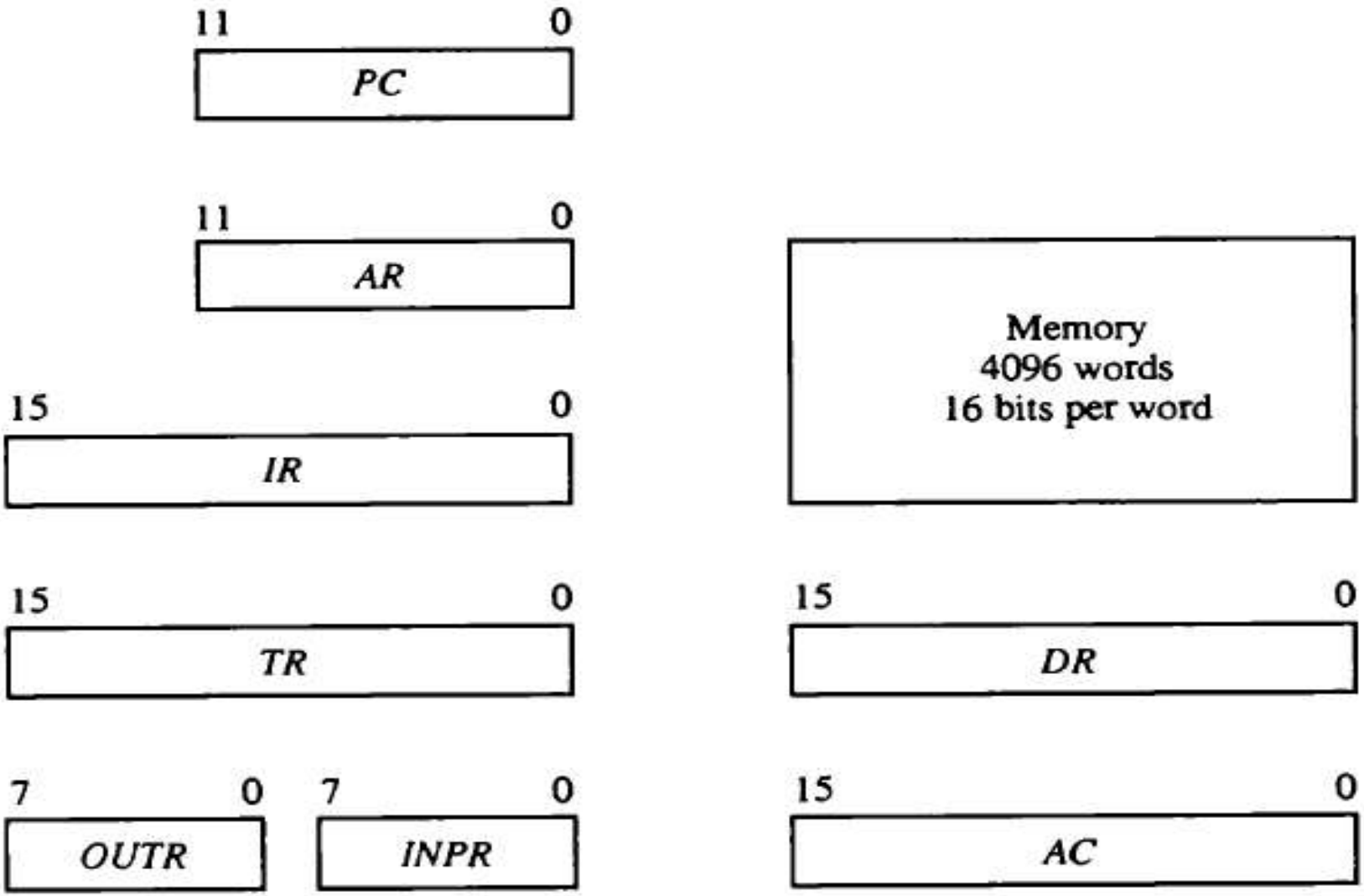


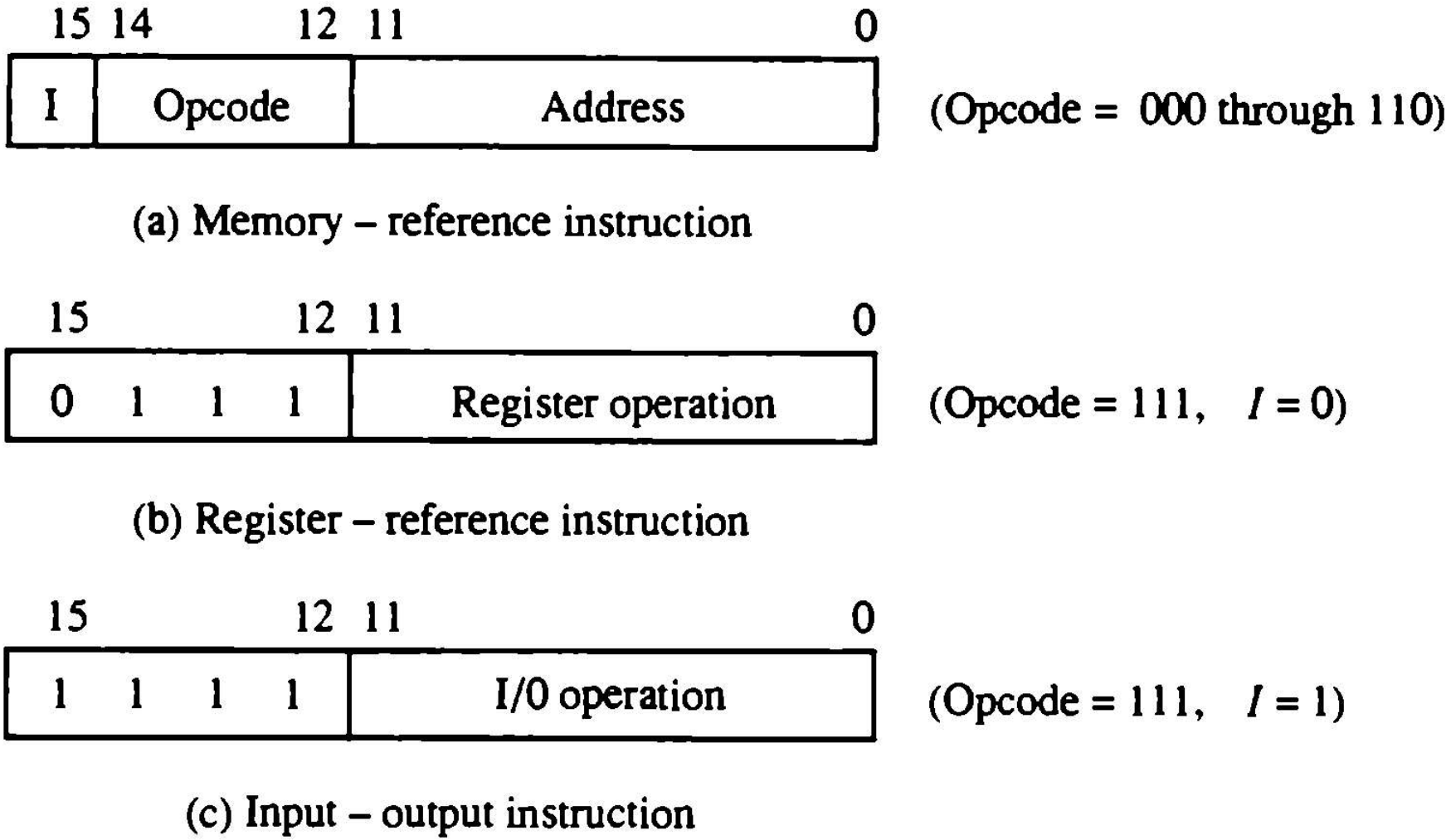
Figure Basic computer registers and memory

Computer Instructions

- The basic computer has three instruction code formats. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits.
- **Memory Reference Instruction–** These instructions refer to memory address as an operand. These instructions specifies 12-bit (0 to 11) address, 3-bit (12 to 14) opcode (000 to 110) and 1-bit (15) addressing mode (I) for direct address (0) and indirect address (1).
- **Register Reference Instruction–** These instructions perform operations on registers rather than memory addresses. The 3-bit (12 to 14) opcode is 111 (differentiates it from memory reference) and bit no. 15 is 0 (differentiates it from input/output instructions). The rest 12 bits (0 to 11) specify register operation to be executed.
- **Input/Output Reference Instruction–** These instructions are for communication between computer and outside environment and does not need a reference to memory. The 3-bit (12 to 14) is 111 (differentiates it from memory reference) and bit no. 15 is 1 (differentiates it from register reference instructions). The rest 12 bits (0 to 11) are used to specify the type of I/O operation.
- The type of instruction is recognized by the computer control from the four bits in position 12 through 15 of the instruction.

Continue…

Figure Basic computer instruction formats



Continue…

TABLE Basic Computer Instructions

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
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

Continue…

- The hexadecimal code is equal to the equivalent hexadecimal number of the binary code for the instruction.
- A memory reference instruction has an address part of 12 bits and this part is denoted by three x's and stand for three hexadecimal digits corresponding to the 12-bit address.
- When the last bit (I) of the instruction is 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0.
- When I = 1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is 1.
- Register-reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12 bits.
- The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Continue…

- The computer includes a sufficient number of instruction in each of the following categories:
  - **-Data processing:** Arithmetic, logical, and shift instructions.
  - **-Data storage:** Instruction for moving information to and from memory and processor registers.
  - **-Data movement:** I/O instructions.
  - **-Control:** Program control instructions together with instructions that check status conditions.
- Arithmetic, logical, and shift instructions provide computational capabilities for processing the type of data.
- The bulk of the binary information in a digital computer is stored in memory, but all computations are done in processor registers. Therefore, the user must have the capability of moving information between these two units (memory and registers).
- Input and output instructions are needed for communication between the computer and the user. These instructions are needed to transfer programs and data into memory and the results of computations back to the user.
- Program control instructions such as branch instructions are used to change the sequence in which the program is executed.

Timing and Control

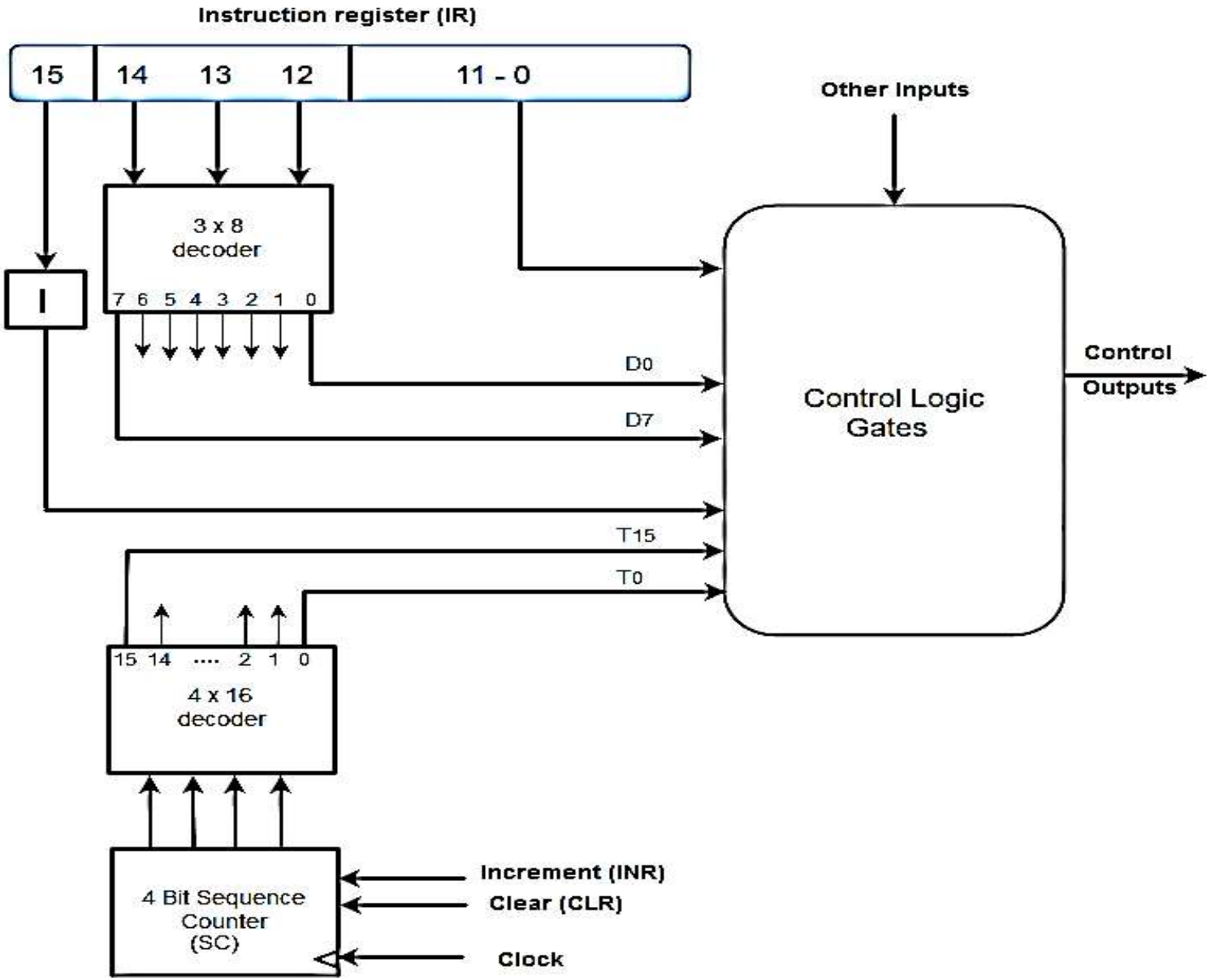
- The timing for all registers in the basic computer is controlled by a master clock generator.
- The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit.
- There are two major types of control organization: hardwired control and microprogrammed control.

Continue…

- **In the hardwired control organization**, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- A hardwired control requires changes in the wiring among the various components if the design has to be modified or changed.
- **In the microprogrammed control organization**, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro-operations.
- In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.

Continue…

Control Unit of a Basic Computer:



Continue…

- The control unit consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction fetched (read) from the memory unit is placed in the instruction register (IR). The component of an instruction register includes; I bit (bit-15), the operation code (bits from 12 to 14), and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D0 through D7.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates.
- The four bit sequence counter (SC) can count in binary from 0 through15. It can be incremented or cleared synchronously. The outputs of the counter are decoded into 16 timing signals (using 4 x 16 decoder) from T_0 through T_{15} .
- The positive clock transition T_0 will trigger only those registers whose control inputs are connected to timing signal T_0 .
- The clock portion of the control unit issues a repetitive sequence of pulses. This is useful for measuring the duration of micro-operations.

Continue…

- 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 the 4 x 16 decoder.
- Once in a while, the counter is cleared to 0, causing the next active timing signal to be T0.
- The SC responds to the positive transition of the clock. Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to 0, which in turn activates the timing signal T0 out of the decoder. T0 is active during one clock cycle.
- SC is incremented (from T0 to T15 and back to T0) with every positive clock transition, unless its CLR input is active.
- As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3, and T4 in sequence.
- At time T4, and if 3 x 8 decoder output D3 is active, CLR input of SC is active i.e., SC is cleared to 0.

$$D3T4: \quad SC \leftarrow 0$$

Continue…

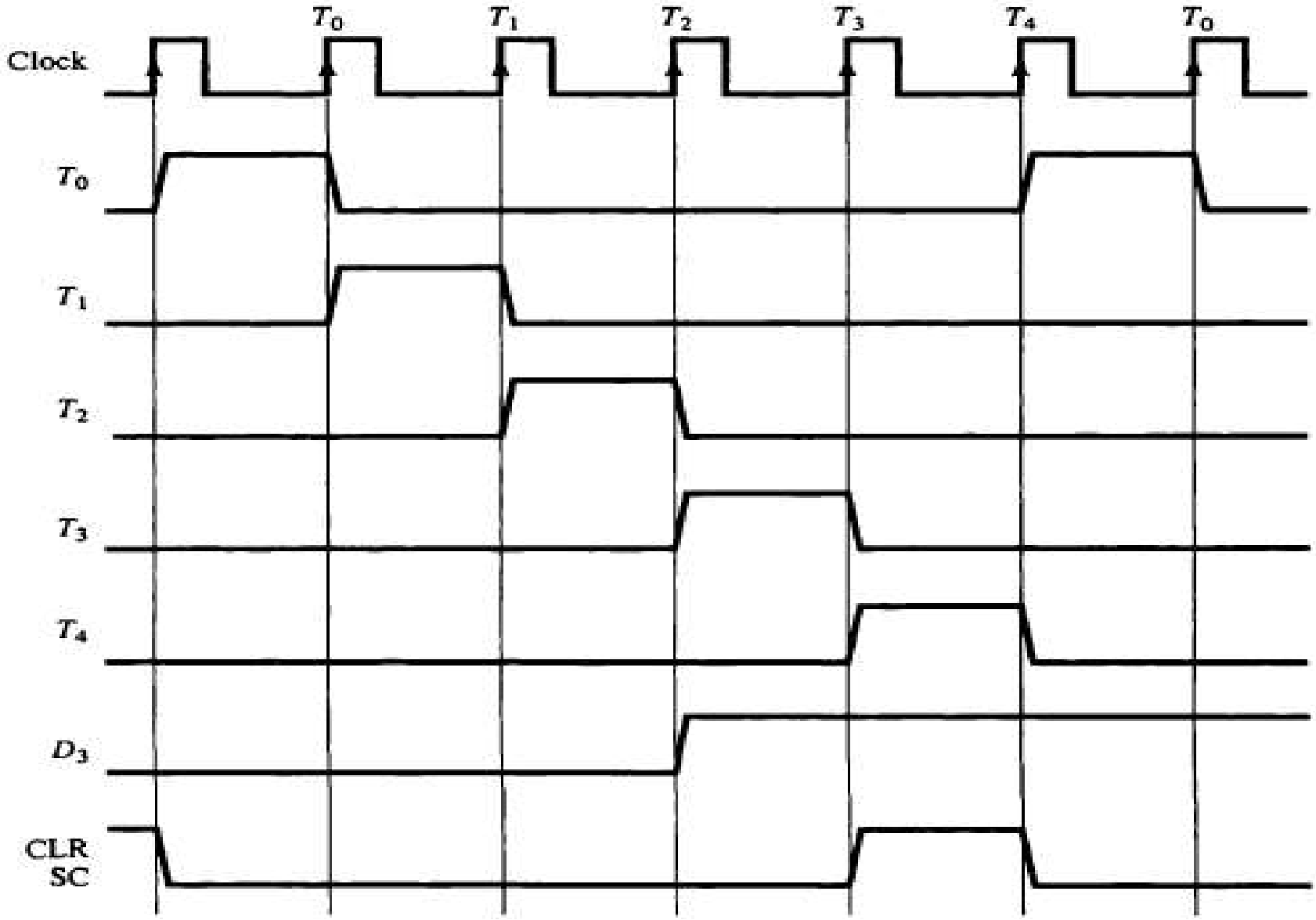


Figure Example of control timing signals

Instruction Cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions.
- The program is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle in turn is subdivided into a sequence of sub-cycles or phases. 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.
- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. The process continues indefinitely unless a HALT instruction is encountered.

Continue…

- **Fetch and Decode:** Initially, the program counter PC is loaded with the address of the first instruction in the program.
- The SC is cleared to 0, providing a decoded timing signal T0.
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on.
- The micro-operations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: $AR \leftarrow PC$

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

T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

Continue…

- Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T0.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.
- At the same time (timing signal T1), PC is incremented by one to prepare it for the address for the next instruction in the program.
- At timing signal T2, the operation code in IR (bits from 12 to 14) is decoded through 3 x 8 decoder, the indirect bit (bit 15) is transferred to flip flop I, and the address part of the instruction (bits from 0 to 11) is transferred to AR.
- Note that SC is incremented after each clock pulse to produce the sequence T0, T1, and T2 (i.e., timing signal T0, T1, and T2).

Continue…

- **Determine the Type of Instruction:** The timing signal that is active after the decoding is T3. During timing signal T3, the control unit determines the type of instruction that was just read from memory
- 3 x 8 Decoder output D7 = 1 (instruction must be a register-reference or input-output type) if the operation code is equal to binary 111.
- If D7 = 0, the operation code must be one of the other seven values 000 through 011, specifying a memory-reference instruction.
- Control then inspects the value of the bit I of the instruction, i.e., available in flip-flop.
- If D7 = 0 and I = 1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory and store in the AR register.

$$AR \leftarrow M[AR]$$

Continue…

- The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

  D7'IT3: AR←M[AR]

  D7'I'T3: Nothing

  D7I'T3: Execute a register-reference instruction

  D7IT3: Execute an input-output instruction

- A memory-reference instruction with I = 0, it is not necessary to do anything since the effective address is already available in AR.
- After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T0 = 1.
- Note that the SC is either incremented or cleared to zero with every positive clock transition. When SC is to be cleared, we will include the statement SC←0.

Continue…

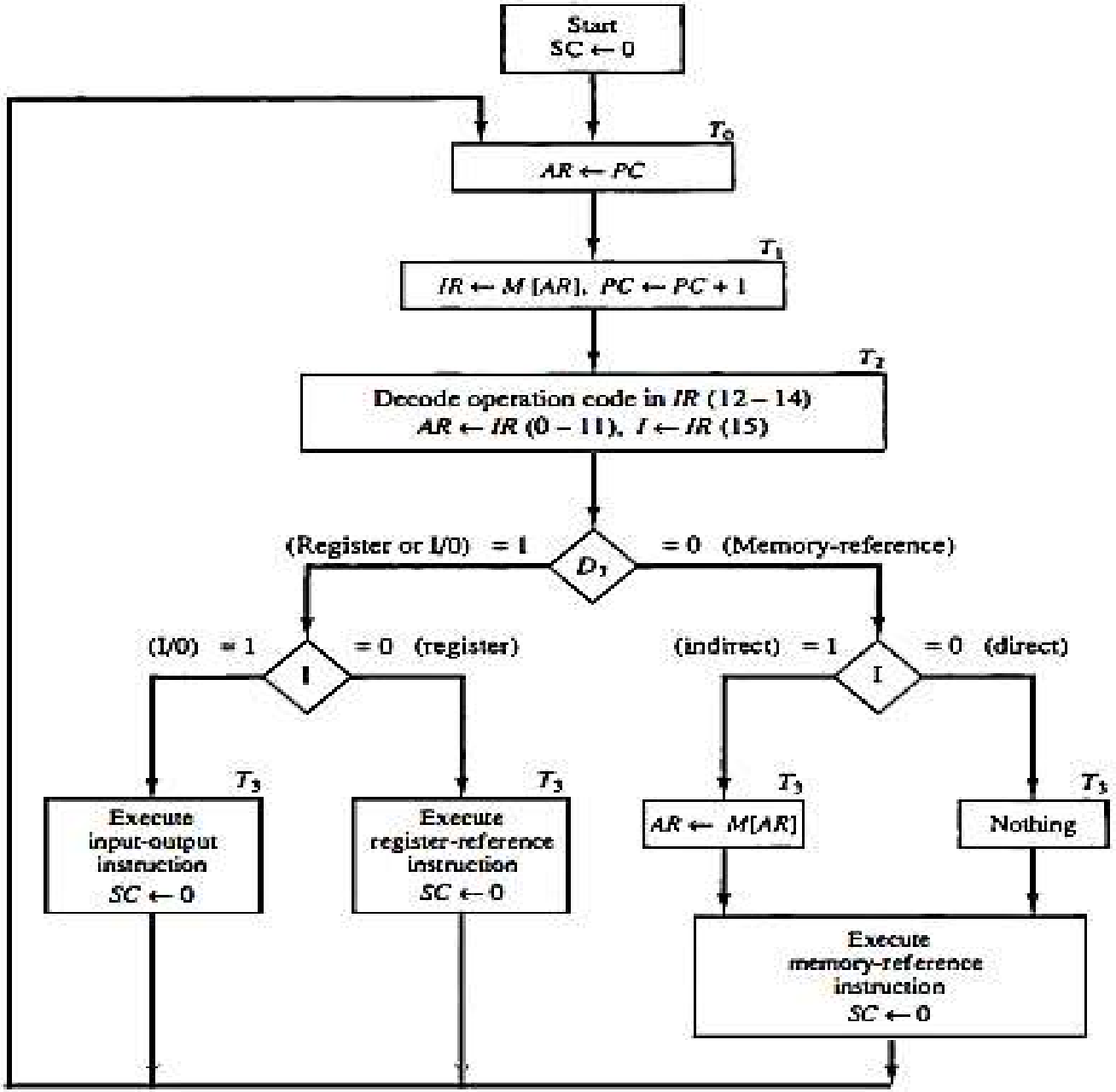


Figure Flowchart for instruction cycle (initial configuration)

Register-Reference Instructions

- Register-reference instructions are recognized by the control when D7 = 1 and I = 0.
- These instructions are executed with the clock transition associated with timing variable T3.
- After the instruction is executed, SC is cleared to 0 and control returns to the fetch the next instruction with T0 = 1.
- The HLT instruction clears a start-stop flip-flop S and stops the sequence counter SC from counting.
- To restore the operation of the computer, the start-stop flip-flop must be set manually.

Continue…

TABLE Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	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 positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Memory-Reference Instructions

- The decoded output Di for i = 0,1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in the table.

TABLE Memory-Reference Instructions

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

Continue…

- The effective address of the instruction is placed in the address register AR during timing signal T2 when I = 0 (direct address), or during T3 when I = 1 (indirect address).
- The data must be read from memory to a register where they can be operated on with logic circuits.
- The execution of the memory-reference instructions starts with timing signal T4.

Continue…

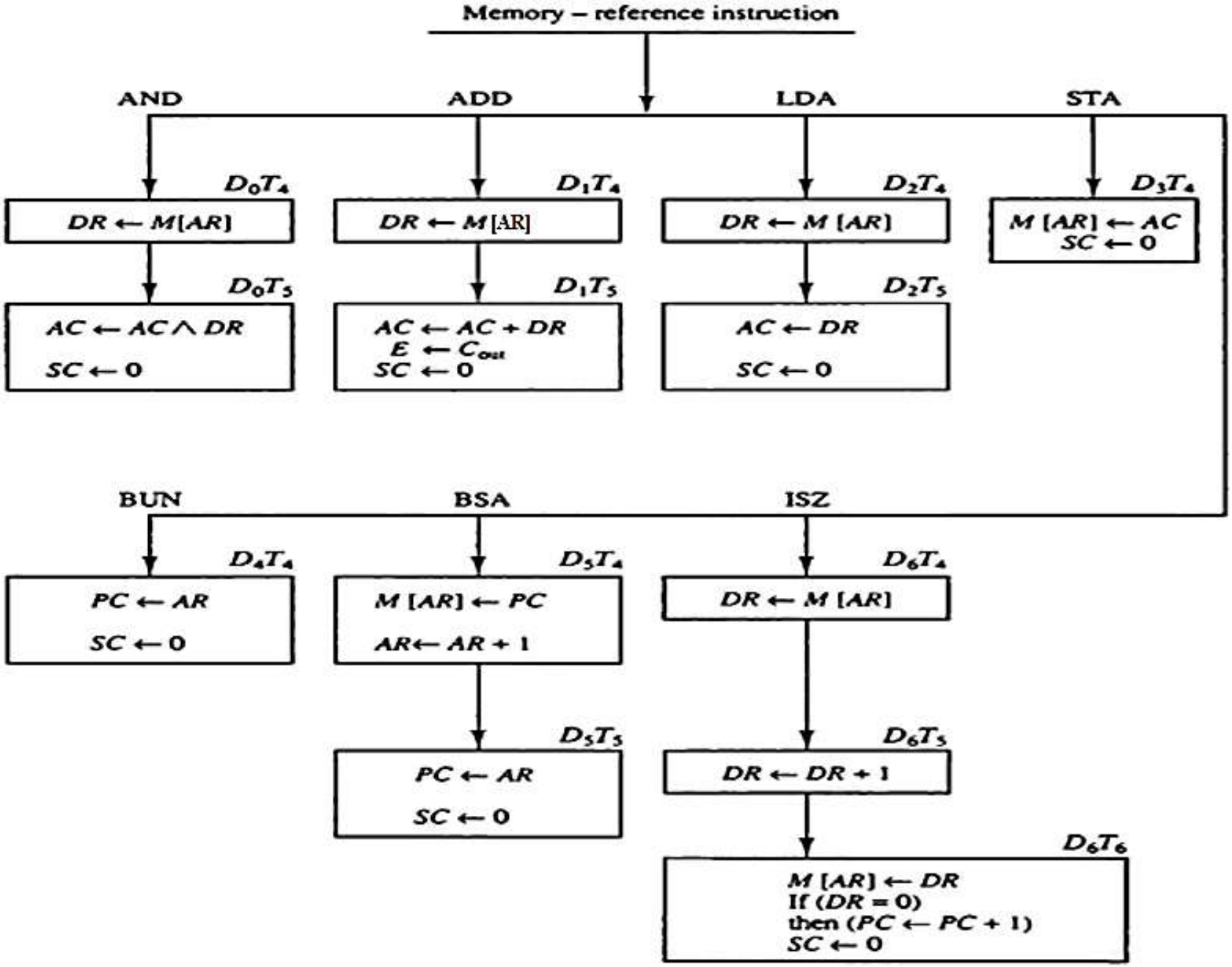


Figure Flowchart for memory-reference instructions.

Continue…

- **BUN: Branch Unconditionally**

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

- This instruction transfers the program to the instruction specified by the effective address.
- The effective address from AR is transferred through the common bus to PC.
- The **BUN instruction** allows the programmer to specify an instruction out of sequence and we say that the program branches (or jump) unconditionally.
- **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.

$$D_5T_4: \quad M[AR] \leftarrow PC$$
$$M[135] \leftarrow 21$$

- The effective address plus one (AR + 1) is then transferred to PC to serve as the address of the first instruction in the subroutine.

$$D_5T_4: \quad AR \leftarrow AR + 1$$
$$PC \leftarrow 135 + 1 = 136 \text{ (AR)}$$

Continue…

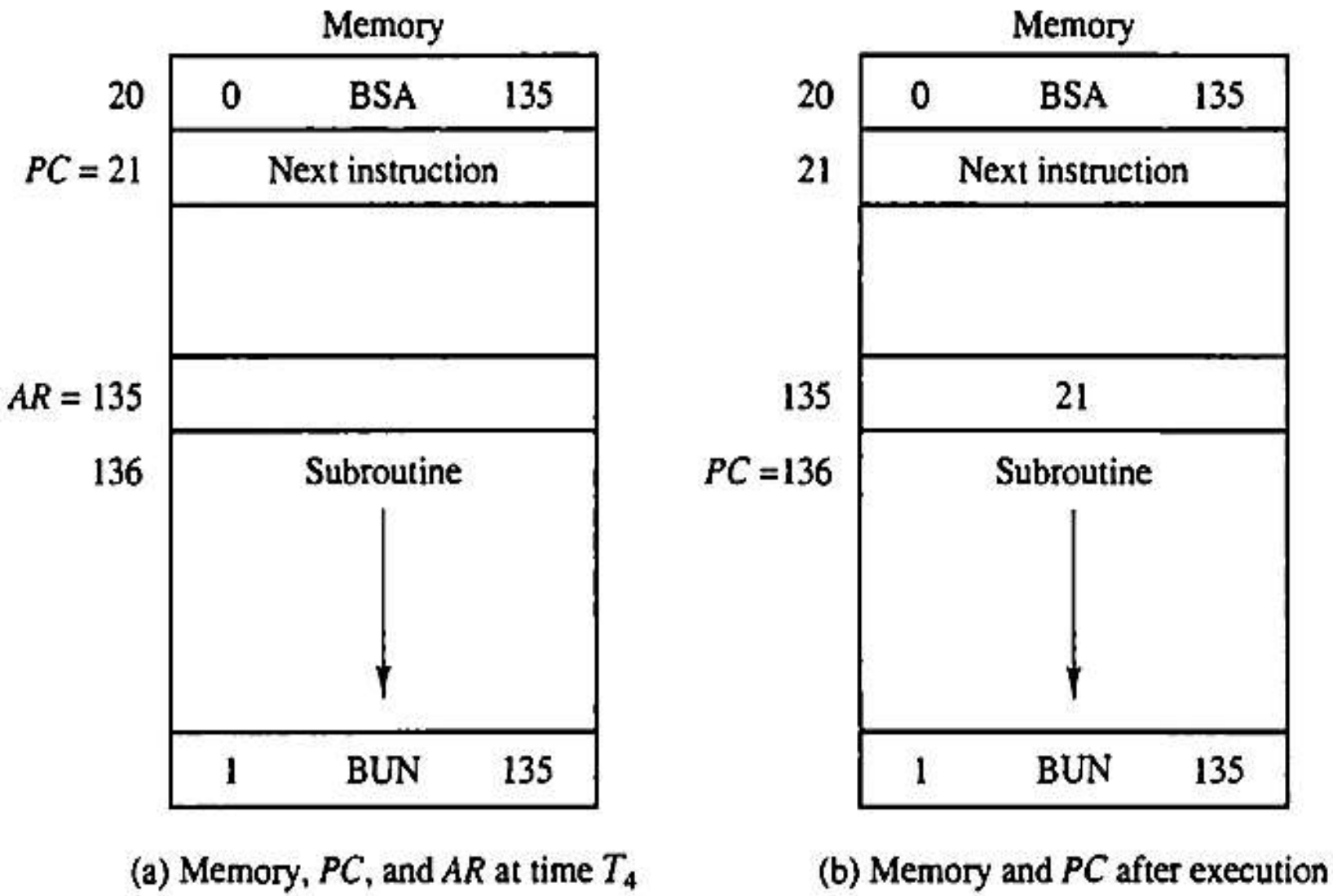
- The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.
- The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.
- When BUN instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21. The effective address 21 is transferred to PC, now control continues to execute the instruction at the return address (21).

$$D_5T_5: \quad PC \leftarrow AR\ (135), \qquad SC \leftarrow 0$$
$$PC \leftarrow 21$$

- The **BSA instruction** performs the function usually referred to as a subroutine call.
- The indirect **BUN instruction** at the end of the subroutine performs the function referred to as a subroutine return.

Continue…

Figure Example of BSA instruction execution.



(a) Memory, *PC*, and *AR* at time T_4

(b) Memory and *PC* after execution

Continue…

- **ISZ: Increment and Skip if Zero**
- This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1 in order to skip the next instruction in the program.
- 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 ← M[AR]

D_6T_5: DR ← DR + 1

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