BASIC COMPUTER ORGANIZATION AND DESIGN

- **Instruction Codes**
- **Computer Registers**
- **Computer Instructions**
- **Timing and Control**
- **Instruction Cycle**
- **Memory Reference Instructions**
- **Input-Output and Interrupt**
- **Complete Computer Description**
- **Design of Basic Computer**
- **Design of Accumulator Logic**

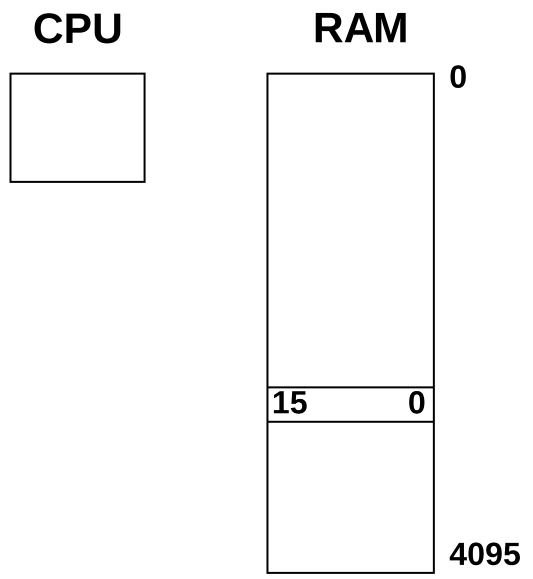
INTRODUCTION

- **Every different processor has its own design**

  (different registers, buses, micro-operations, machine instructions, etc)
- **Modern processor is a very complex device**
- **It contains**
  - **Many registers**
  - **Multiple arithmetic units, for both integer and floating point calculations**
  - **The ability to pipeline several consecutive instructions to speed execution**
  - **Etc.**
- **However, to understand how processors work, use a simplified processor model**
- **This is similar to what real processors were like ~25 years ago**

THE BASIC COMPUTER

- **The Basic Computer has two components, a processor and memory**
- **The memory has 4096 words in it**
  - **4096 = 2^{12}, so it takes 12 bits to select a word in memory**
- **Each word is 16 bits long**



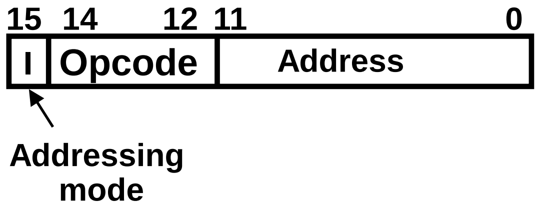
INSTRUCTIONS

- **Program**
  - **A sequence of (machine) instructions**
- **(Machine) Instruction**
  - **A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)**
- **The instructions of a program, along with any needed data are stored in memory**
- **The CPU reads the next instruction from memory**
- **It is placed in an *Instruction Register* (IR)**
- **Control circuitry in control unit then translates the instruction into the sequence of microoperations necessary to implement it**

INSTRUCTION FORMAT

- **A computer instruction is often divided into two parts**
  - **An *opcode* (Operation Code) that specifies the operation for that instruction**
  - **An *address* that specifies the registers and/or locations in memory to use for that operation**
- **In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use**
- **In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)**
- **Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode**

Instruction Format



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 specifies an address.
- 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.
- The following figure 2.1 shows this type of organization.

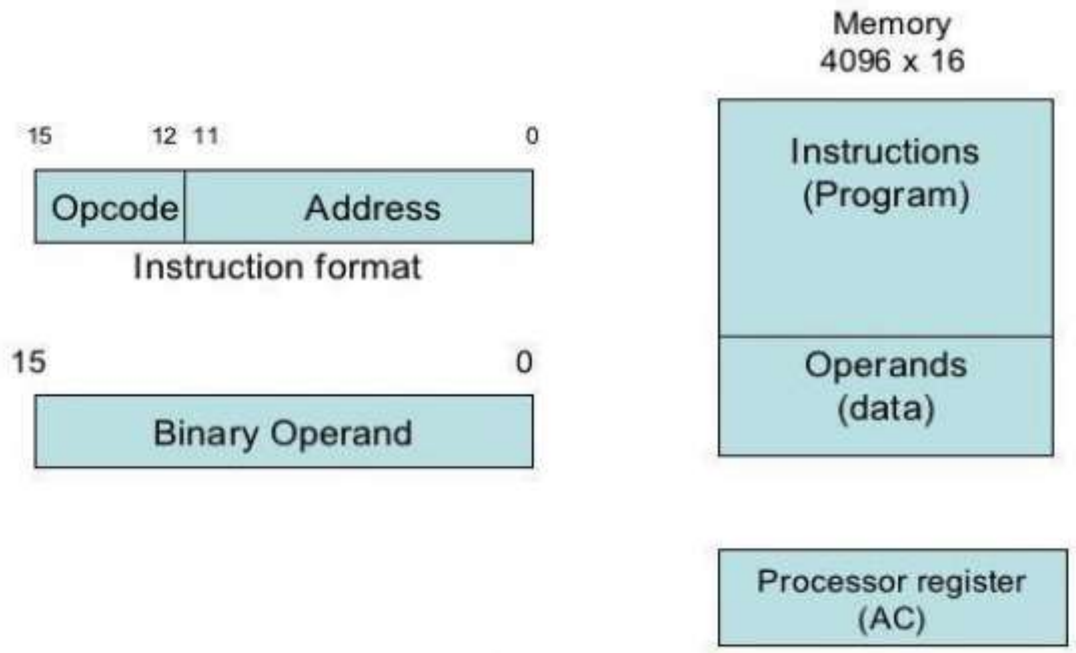
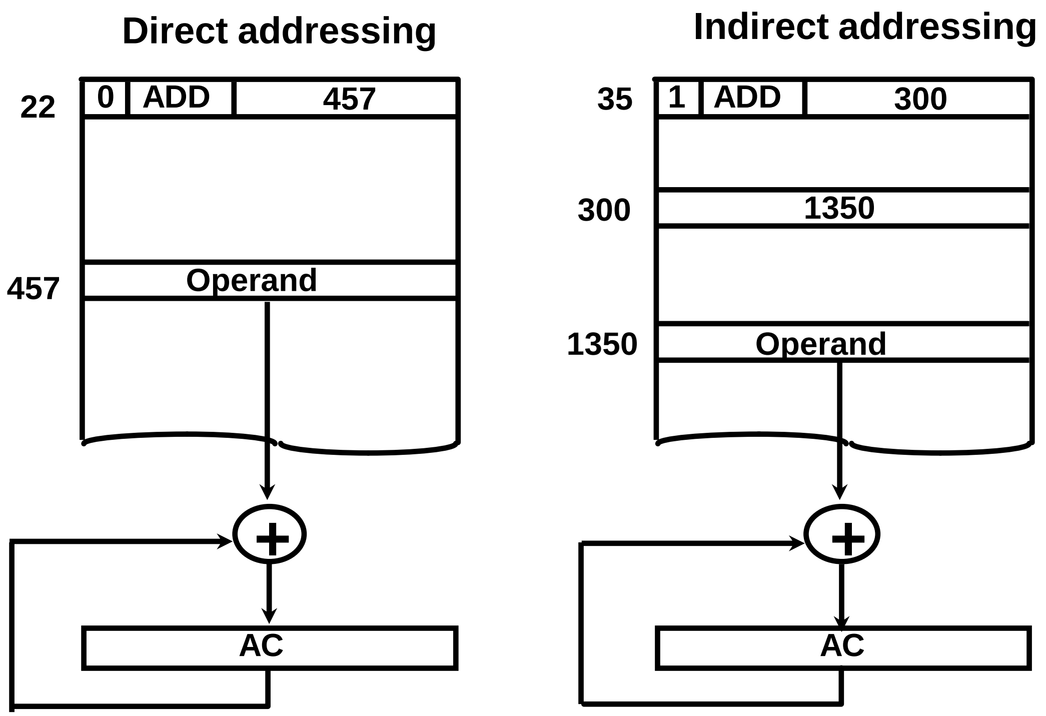


Figure 2.1: Stored Program Organization

- 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.

ADDRESSING MODES

- **The address field of an instruction can represent either**
  - **Direct address: the address in memory of the data to use (the address of the operand), or**
  - **Indirect address: the address in memory of the address in memory of the data to use**



- **Effective Address (EA)**
  - **The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction**

PROCESSOR REGISTERS

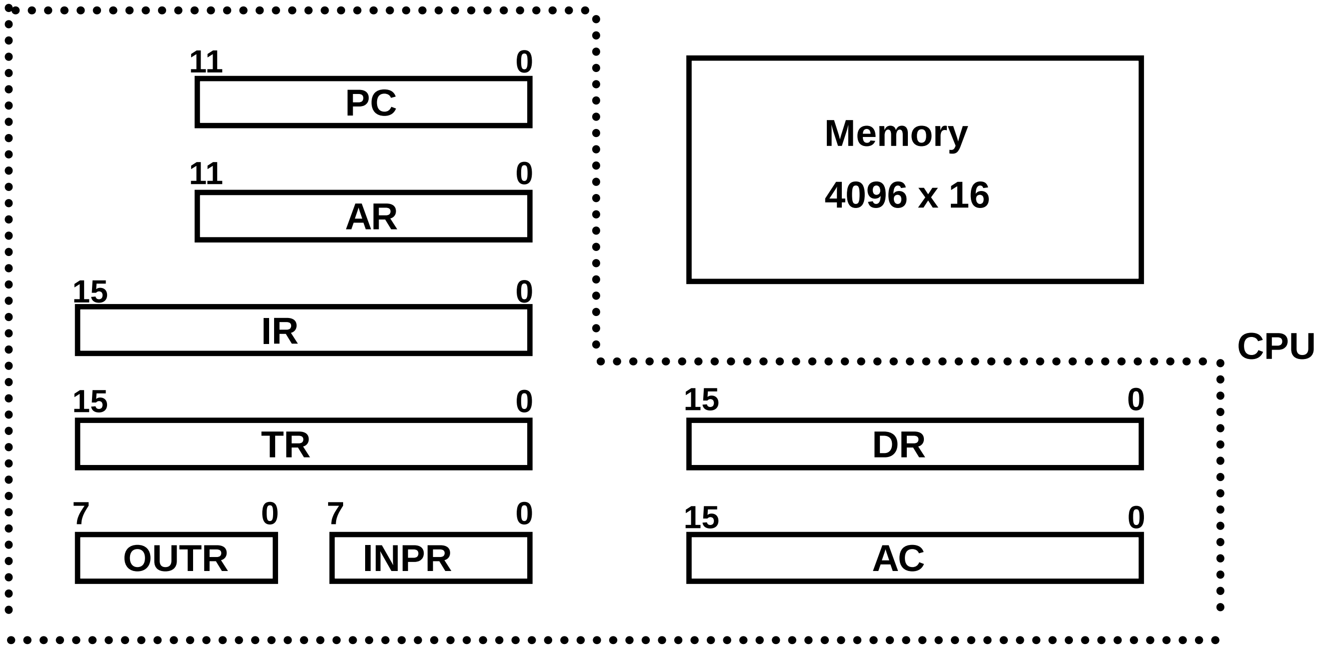
- **A processor has many registers to hold instructions, addresses, data, etc**
- **The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction**
  - **Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits**
- **In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

PROCESSOR REGISTERS

- **The significance of a general purpose register is that it can be used for loading operands and storing results**
  - **e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location**
- **Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR)**
- **The Basic Computer uses a very simple model of input/output (I/O) operations**
  - **Input devices are considered to send 8 bits of character data to the processor**
  - **The processor can send 8 bits of character data to output devices**
- **The *Input Register* (INPR) holds an 8 bit character gotten from an input device**
- **The *Output Register* (OUTR) holds an 8 bit character to be send to an output device**

BASIC COMPUTER REGISTERS

Registers in the Basic Computer



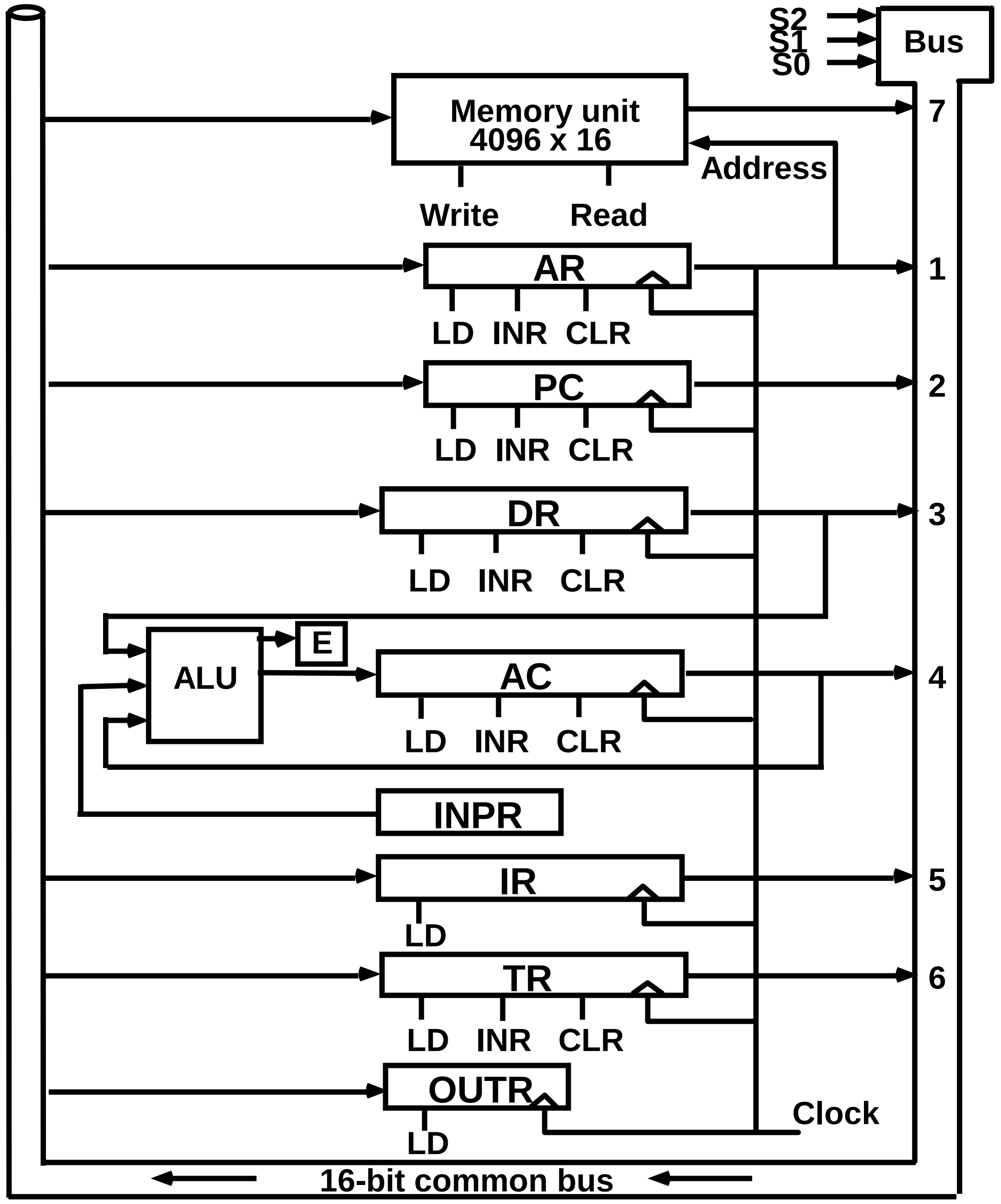
List of Registers

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

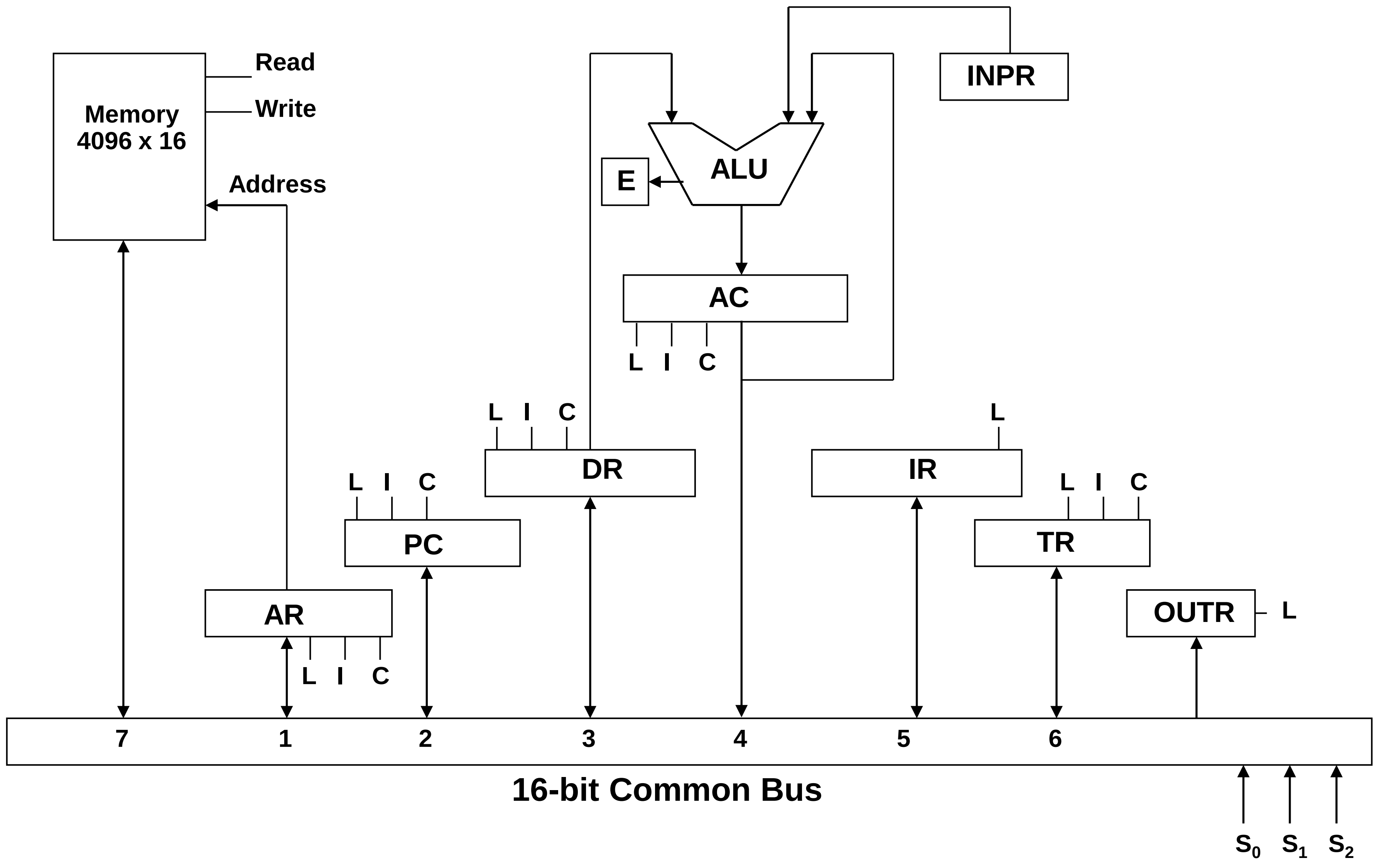
COMMON BUS SYSTEM

- **The registers in the Basic Computer are connected using a bus**
- **This gives a savings in circuitry over complete connections between registers**

COMMON BUS SYSTEM



COMMON BUS SYSTEM



COMMON BUS SYSTEM

- **Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input**

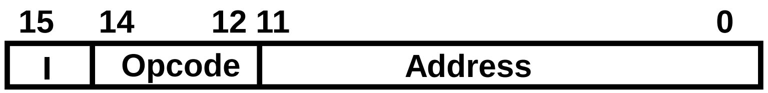
S_2 S_1 S_0	Register
0 0 0	x
0 0 1	AR
0 1 0	PC
0 1 1	DR
1 0 0	AC
1 0 1	IR
1 1 0	TR
1 1 1	Memory

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

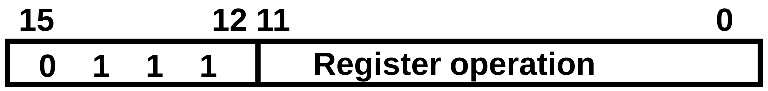
BASIC COMPUTER INSTRUCTIONS

• **Basic Computer Instruction Format**

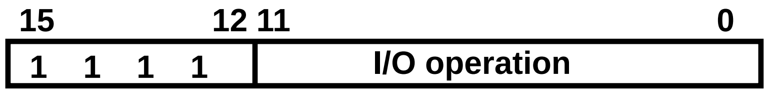
Memory-Reference Instructions (OP-code = 000 ~ 110)



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



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



BASIC COMPUTER INSTRUCTIONS

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

INSTRUCTION SET COMPLETENESS

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

- **Instruction Types**

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

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

Control Instructions
- **Program sequencing and control**
- **BUN(branch unconditionally), BSA(branch nd save return address), ISZ(**Increment and skip if Zero**)**

Input/Output Instructions
- **Input and output**
- **INP, OUT**

CONTROL UNIT

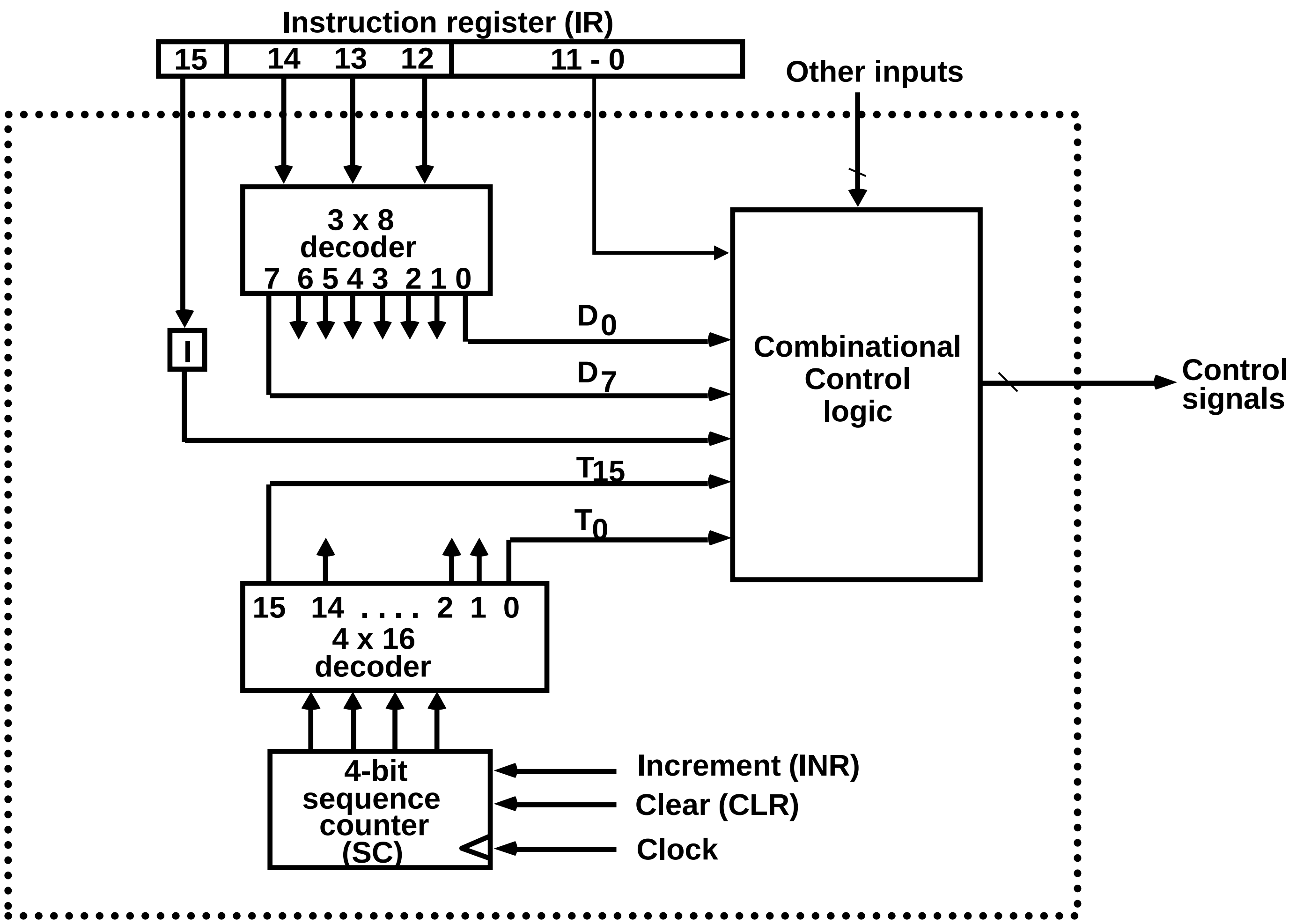
- **Control unit (CU) of a processor translates from machine instructions to the control signals (for the micro-operations) that implement them**

- **Control units are implemented in one of two ways**
- ***Hardwired* Control**
  - **CU is made up of sequential and combinational circuits to generate the control signals**
- ***Microprogrammed* Control**
  - **A control memory on the processor contains microprograms that activate the necessary control signals**

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

TIMING AND CONTROL

Control unit of Basic Computer

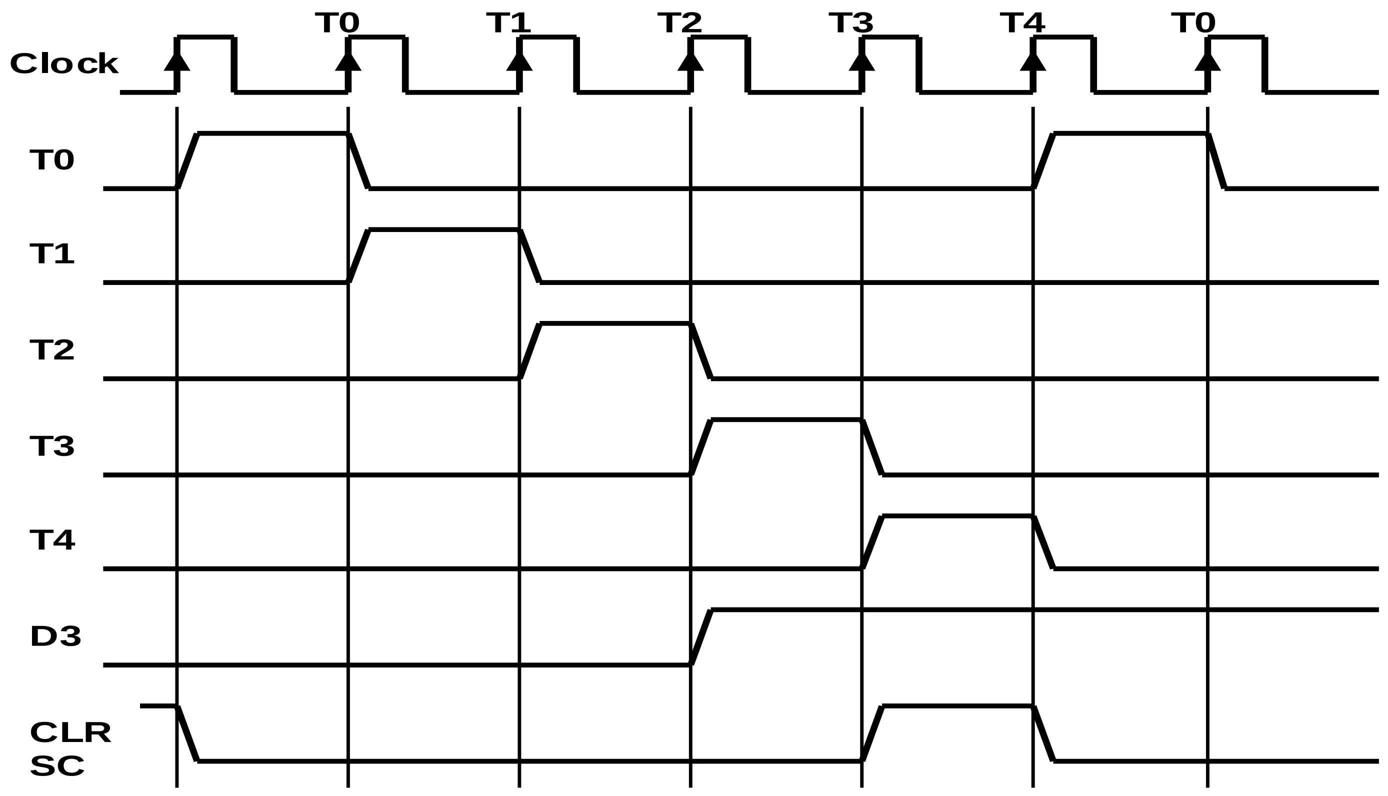


TIMING SIGNALS

- Generated by 4-bit sequence counter and 4×16 decoder
- The SC can be incremented or cleared.

- Example: T_0, T_1, T_2, T_3, T_4, T_0, T_1, . . .
  Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.

$$D_3T_4: SC \leftarrow 0$$



INSTRUCTION CYCLE

- **In Basic Computer, a machine instruction is executed in the following cycle:**
  1. **Fetch an instruction from memory**
  2. **Decode the instruction and calculate effective address (EA)**
  3. **Read the EA from memory if the instruction has an indirect address (Fetch operand)**
  1. **Execute the instruction**

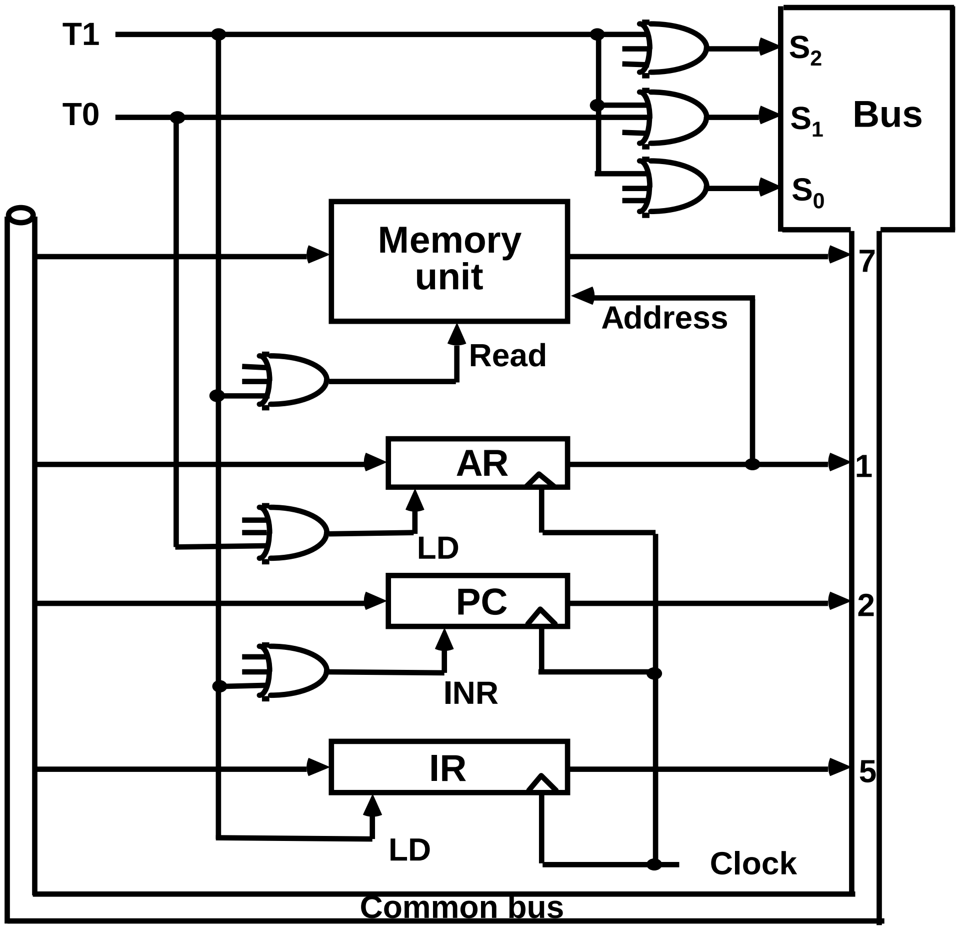
- **After an instruction is executed, the cycle starts again at step 1, for the next instruction**

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

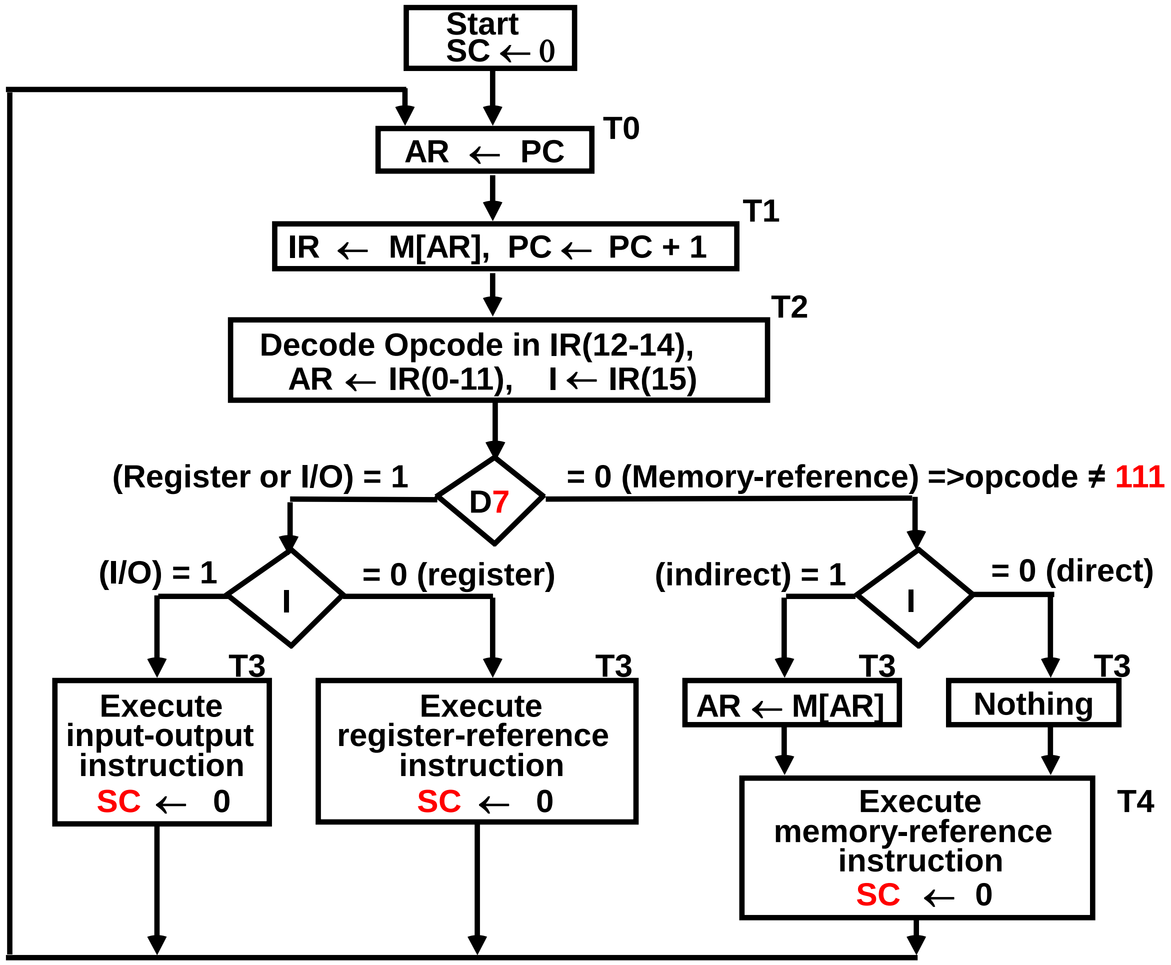
FETCH and DECODE

- **Fetch and Decode**

T0: AR $\leftarrow$ PC ($S_0S_1S_2$=010, T0=1)
T1: IR $\leftarrow$ M [AR], PC $\leftarrow$ PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 $\leftarrow$ Decode IR(12-14), AR $\leftarrow$ IR(0-11), I $\leftarrow$ IR(15)



DETERMINE THE TYPE OF INSTRUCTION


Start
SC ← 0
T0
AR ← PC
T1
IR ← M[AR], PC ← PC + 1
T2
Decode Opcode in IR(12-14),
AR ← IR(0-11), I ← IR(15)
(Register or I/O) = 1
D7
= 0 (Memory-reference) =>opcode ≠ 111
(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
T4
Execute memory-reference instruction SC ← 0

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.

INSTRUCTION SET

Register reference instruction.

- When the register-reference instruction is decoded, D7 bit is set to 1.
- Each control function needs the Boolean relation D7 I' T3

15 12	11 0
0 1 1 1	Register Operation

- There are 12 register-reference instructions listed below:

	r:	SC←0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_{9}:	AC ← AC'	Complement AC
CME	rB_{8}:	E ← E'	Complement E
CIR	rB_{7}:	AC ← shr AC, AC(15) ← E, E ← AC(0)	Circular Right
CIL	rB_{6}:	AC ← shl AC, AC(0) ← E, E ← AC(15)	Circular Left
INC	rB_{5}:	AC ← AC + 1	Increment AC
SPA	rB_{4}:	if (AC(15) = 0) then (PC ← PC+1)	Skip if positive
SNA	rB_{3}:	if (AC(15) = 1) then (PC ← PC+1	Skip if negative
SZA	rB_{2}:	if (AC = 0) then (PC ← PC+1)	Skip if AC is zero
SZE	rB_{1}:	if (E = 0) then (PC ← PC+1)	Skip if E is zero
HLT	rB_{0}:	S ← 0 (S is a start-stop flip-flop)	Halt computer

INSTRUCTION SET

- These 12 bits are available in IR (0-11). They were also transferred to AR during time T_2.
- These instructions are executed at timing cycle T_3.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers.
- The next four instructions cause a skip of the next instruction in sequence when

  condition is satisfied. The skipping of the instruction is achieved by incrementing PC.
- The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if all the flip-flops of the register are zero.
- The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

INSTRUCTION SET

Memory reference instructions

- When the memory-reference instruction is decoded, D7 bit is set to 0.

15	14 12	11 0
I	000~110	Address

- The following table lists seven 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 the address register AR and was placed there during timing signal T_2 when $I = 0$, or during timing signal T_3 when $I = 1$.
- The execution of the memory-reference instructions starts with timing signal T_4.

INSTRUCTION SET

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

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: $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$

LDA: Load to AC

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

D_2T_4: $DR \leftarrow M[AR]$

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

STA: Store AC

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

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

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

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.

$$M[AR] \leftarrow PC,\ PC \leftarrow AR + 1$$
$$M[135] \leftarrow 21,\ PC \leftarrow 135 + 1 = 136$$

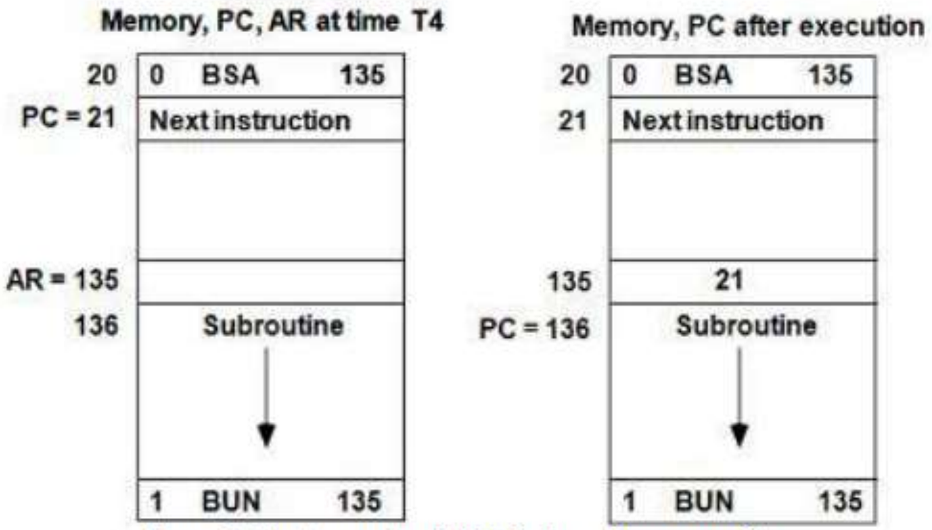


Figure2.10: Example of BSA instruction execution

It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two microoperations:

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

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

INSTRUCTION SET

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_4: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

INSTRUCTION SET

Control Flowchart

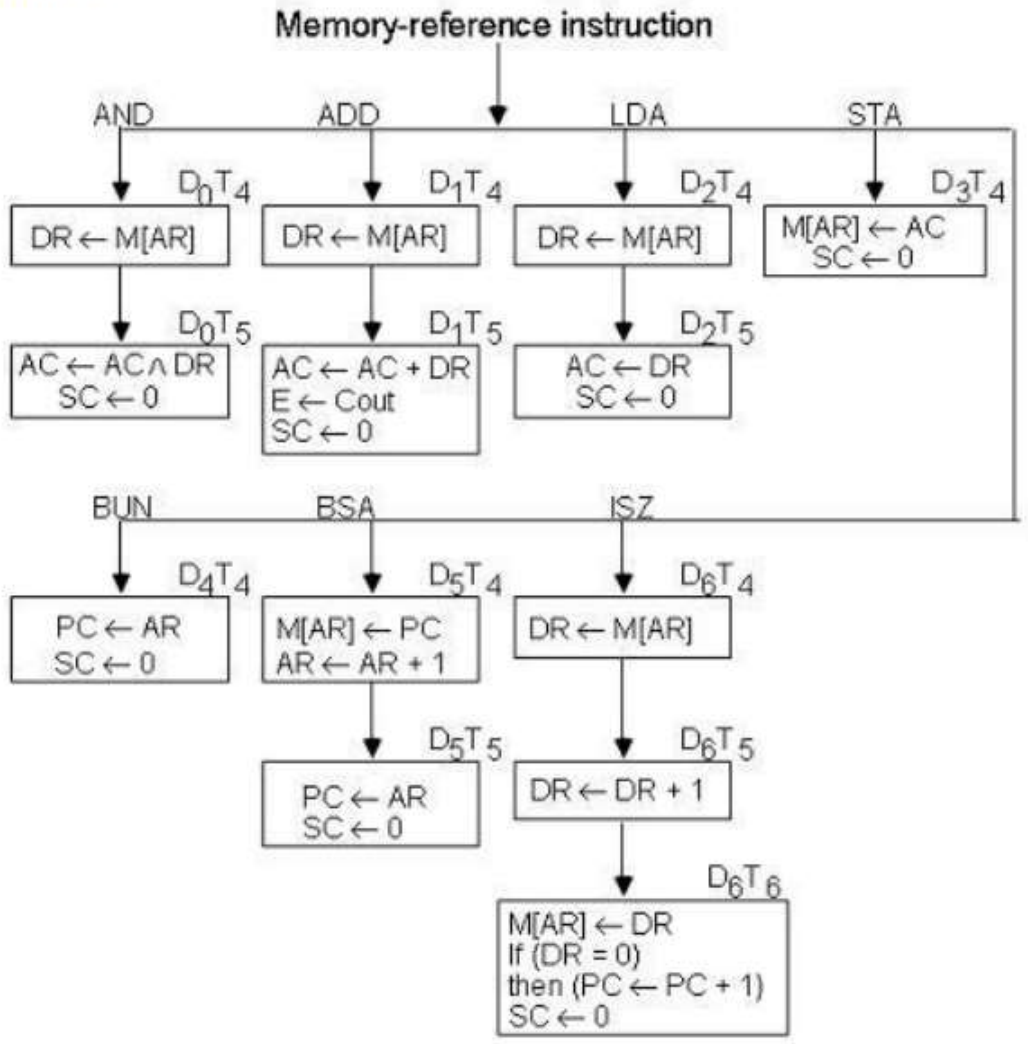


Figure 2.11: Flowchart for memory-reference instructions