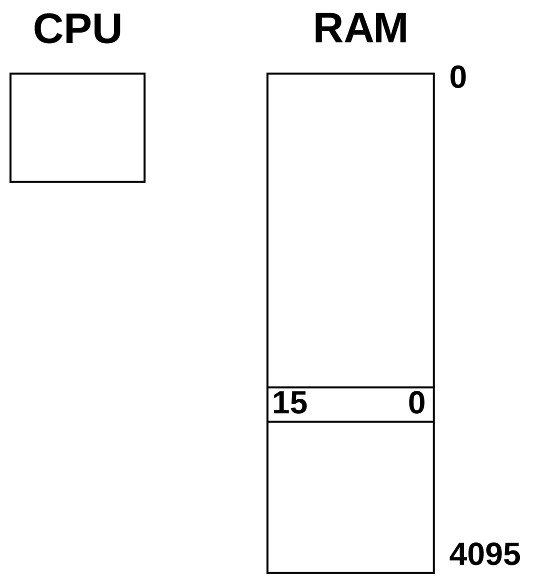
INTRODUCTION

- **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



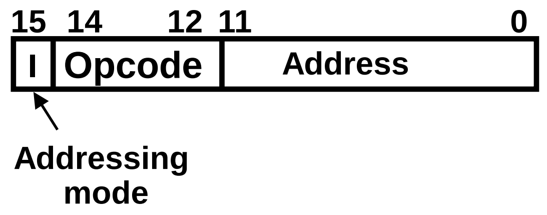
INSTRUCTIONS

- **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

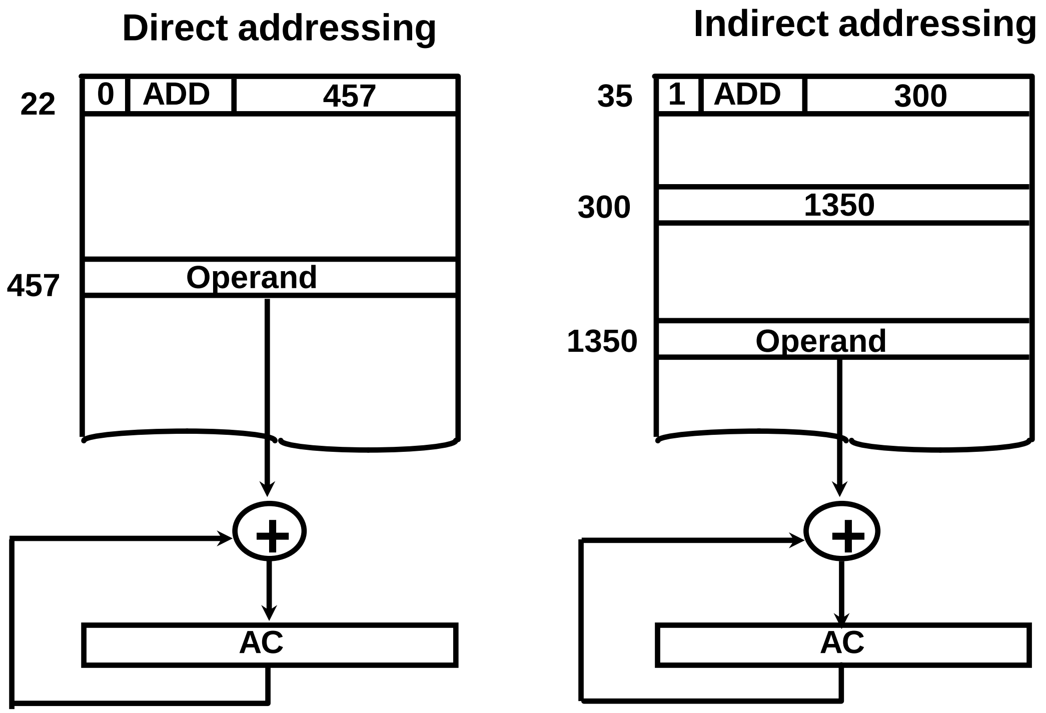
- A computer instruction is often divided into two parts
  - An *opcode* (Operation Code) that specifies the operation for that instruction
  - An *address* that specifies the registers and/or locations in memory to use for that operation
- In the Basic Computer, since the memory contains 4096 (= 2^{12}) 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



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. EA is 457 in first and 1350 in second figure.

PROCESSOR REGISTERS

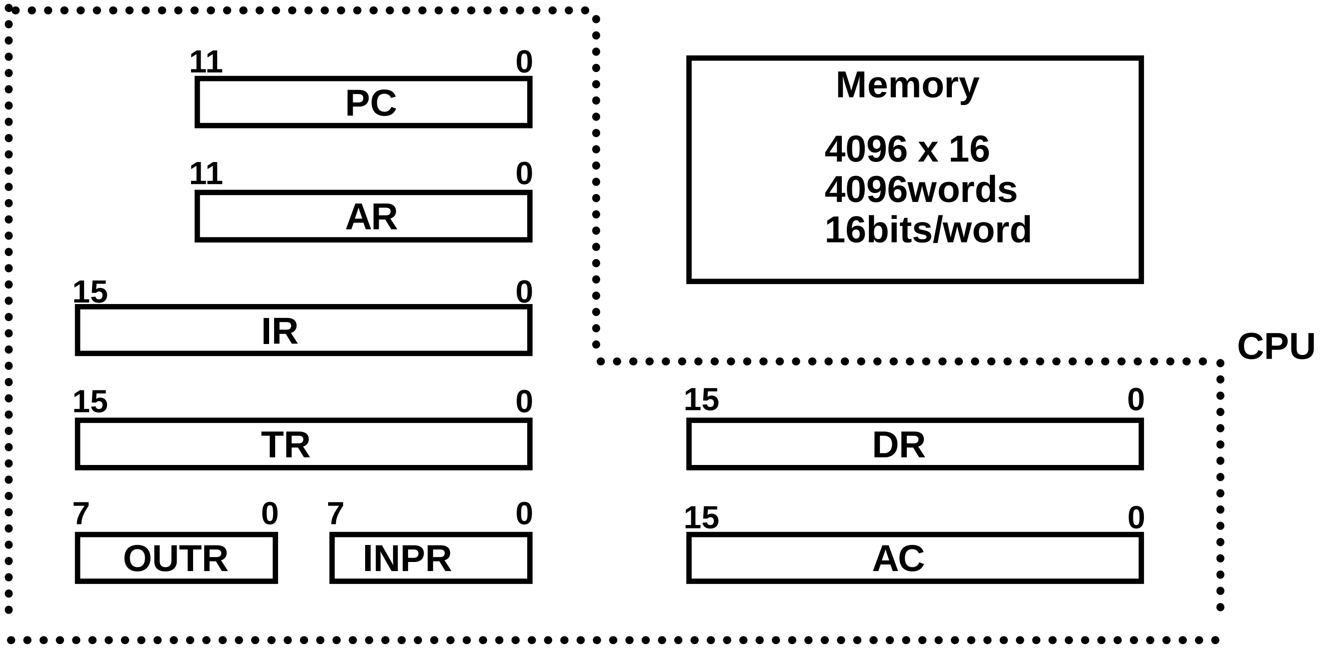
- **A processor has many registers to hold instructions, addresses, data, etc**
- **The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction**
  - **Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits**
- **In 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 processing 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



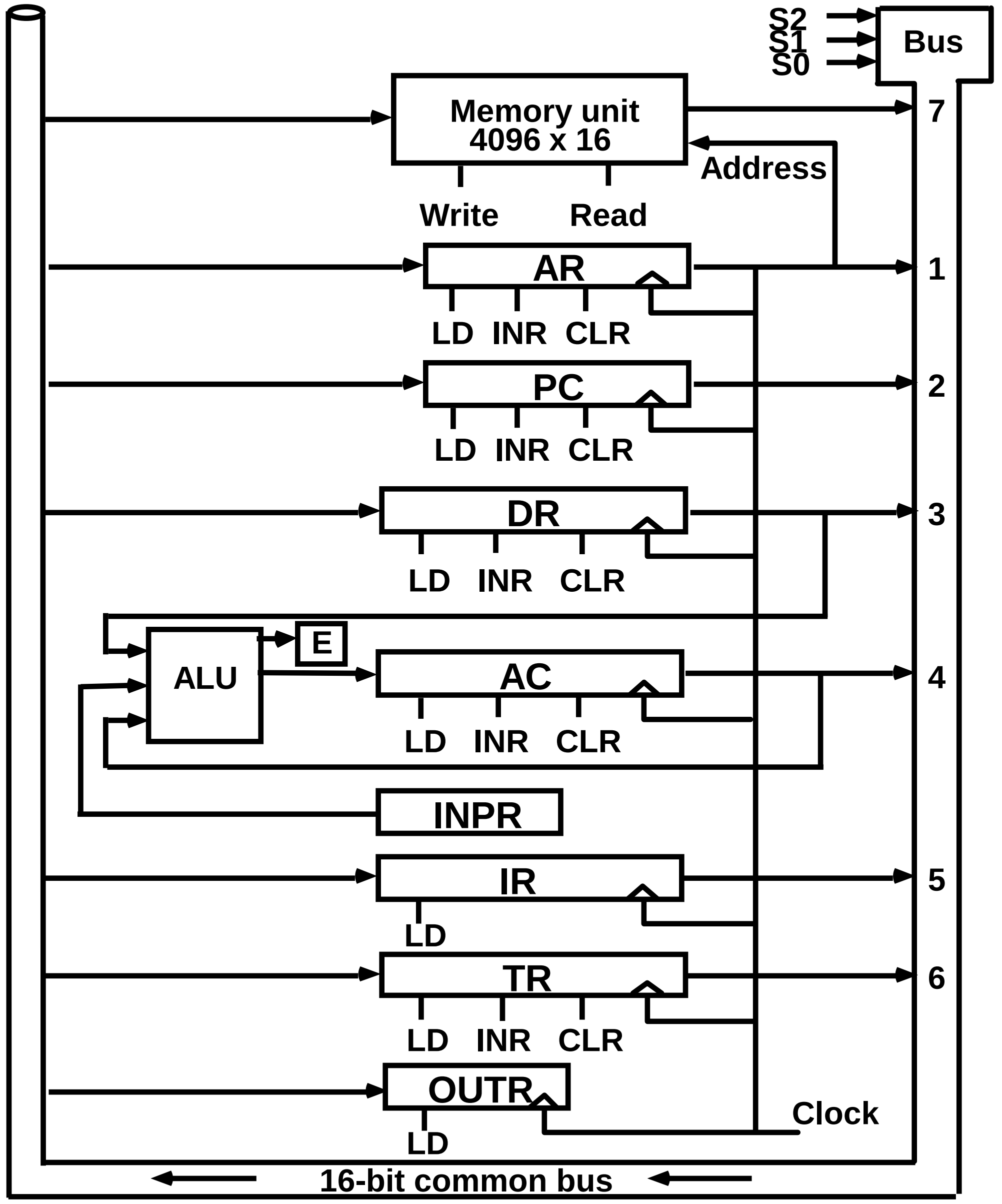
List of Registers

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

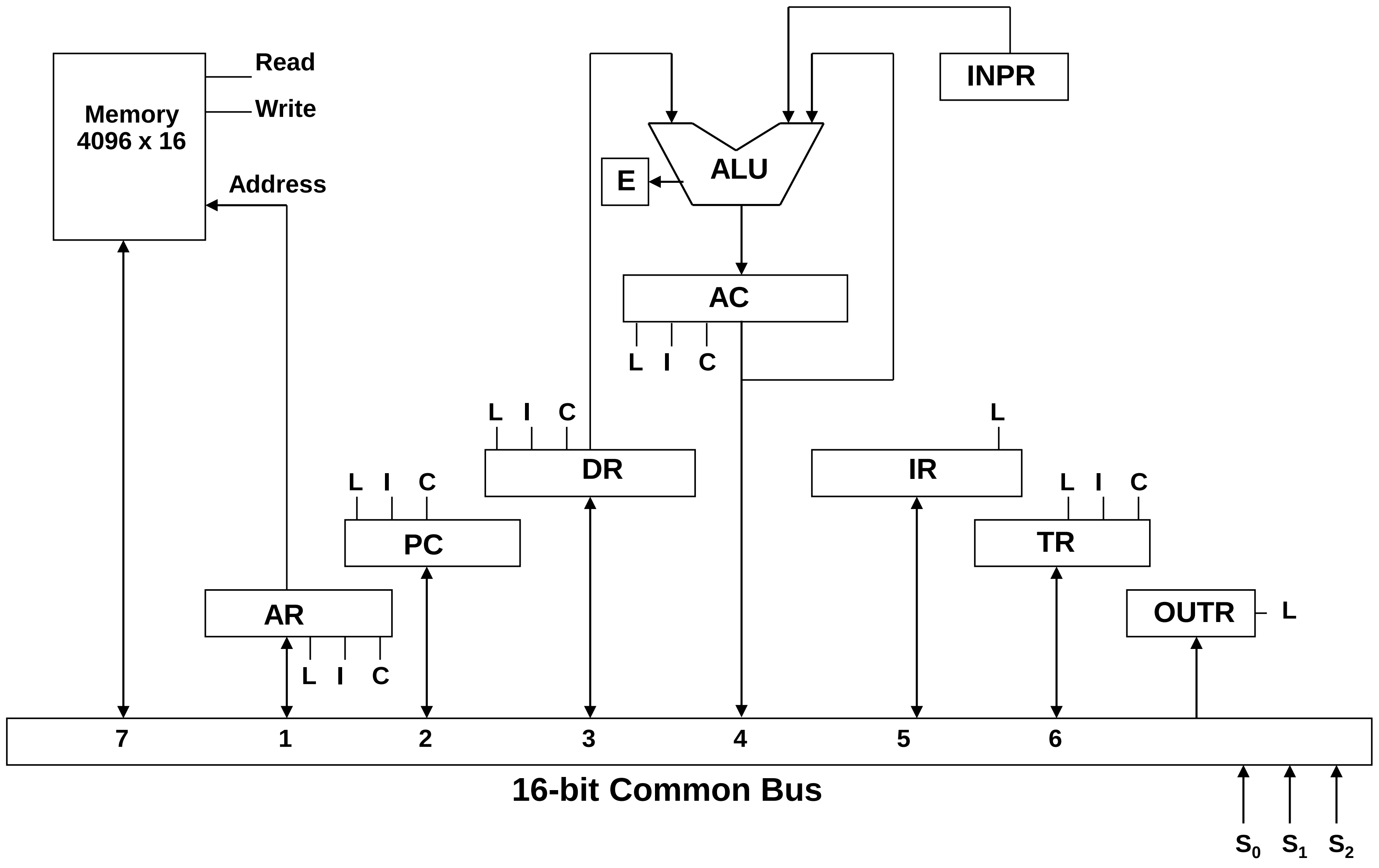
COMMON BUS SYSTEM

- **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**

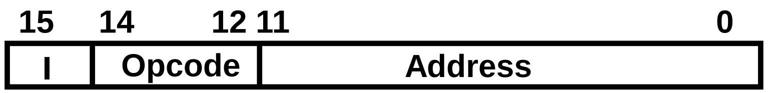
S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

- **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**

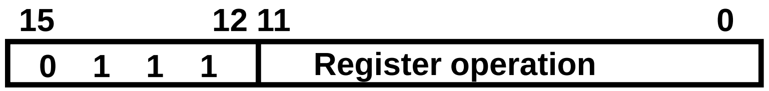
BASIC COMPUTER INSTRUCTIONS

- **Basic Computer Instruction Format**

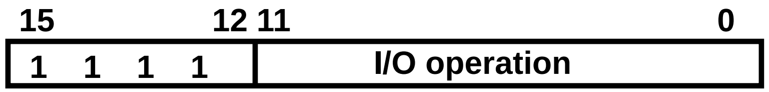
Memory-Reference Instructions (OP-code = 000 ~ 110)



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



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



In Memory-Reference Instructions, I is equal to 0 for direct address and to 1 for indirect address.

BASIC COMPUTER INSTRUCTIONS

Symbol	*Hex Code*		*Description*
	I = 0	*I = 1*	
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**

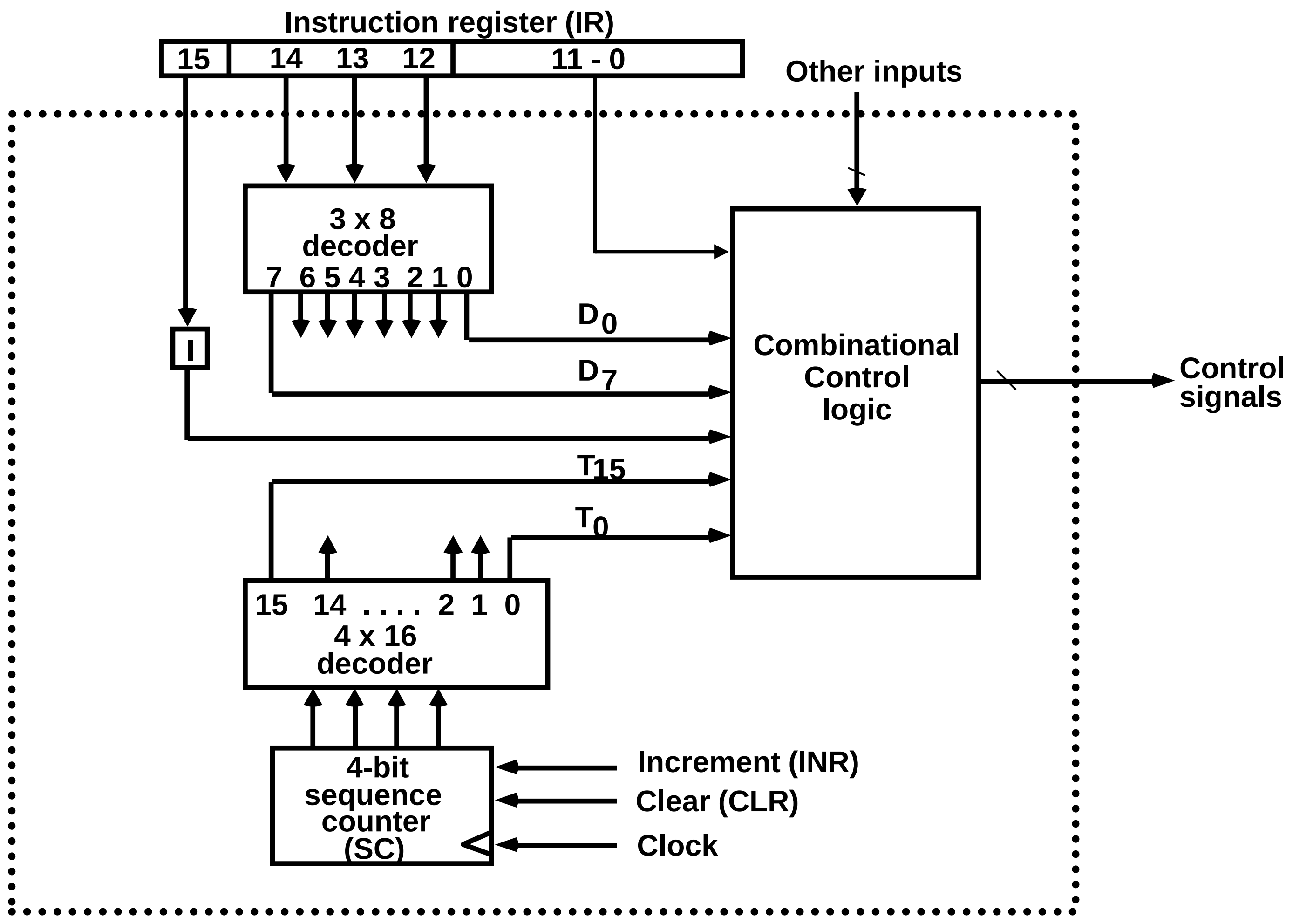
  Transfer Instructions
  - **Data transfers between the main memory and the processor registers**
  - **LDA, STA**

  Control Instructions
  - **Program sequencing and control**
  - **BUN, BSA, ISZ**

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

TIMING AND CONTROL

Control unit of Basic Computer

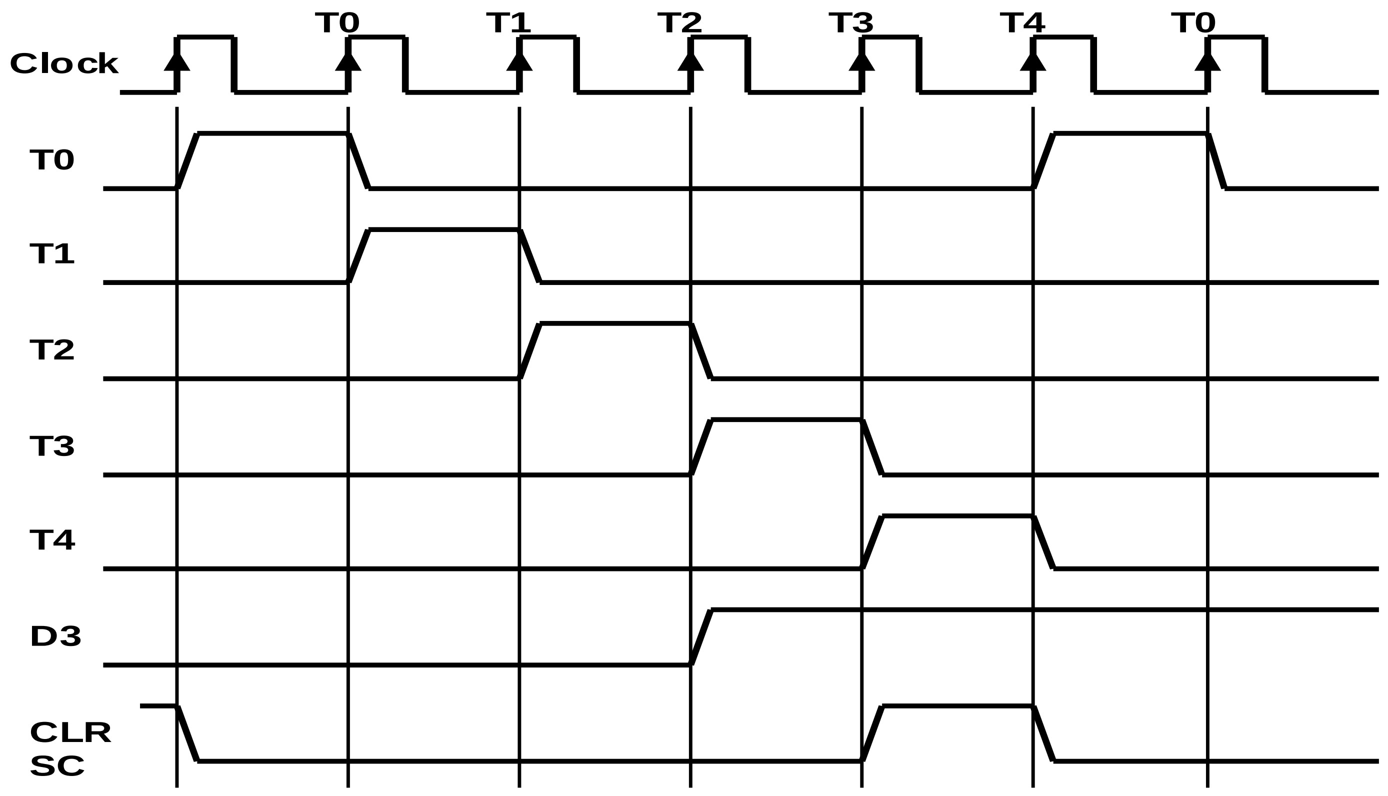


TIMING SIGNALS

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

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

$$D_3T_4\text{: } SC \leftarrow 0$$



INSTRUCTION CYCLE

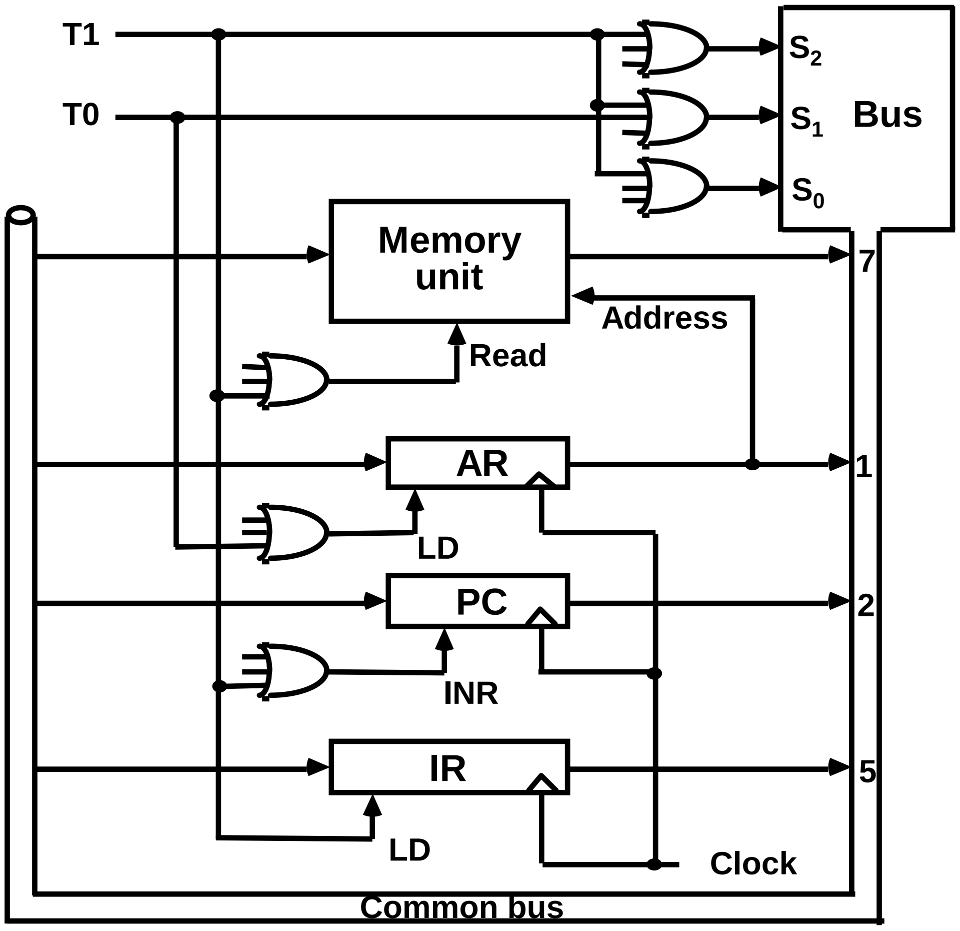
- In Basic Computer, a machine instruction is executed in the following cycle:
  1. Fetch an instruction from memory
  2. Decode the instruction and calculate effective address (EA)
  3. Read the EA from memory if the instruction has an indirect address (Fetch operand)
  4. Execute the instruction

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

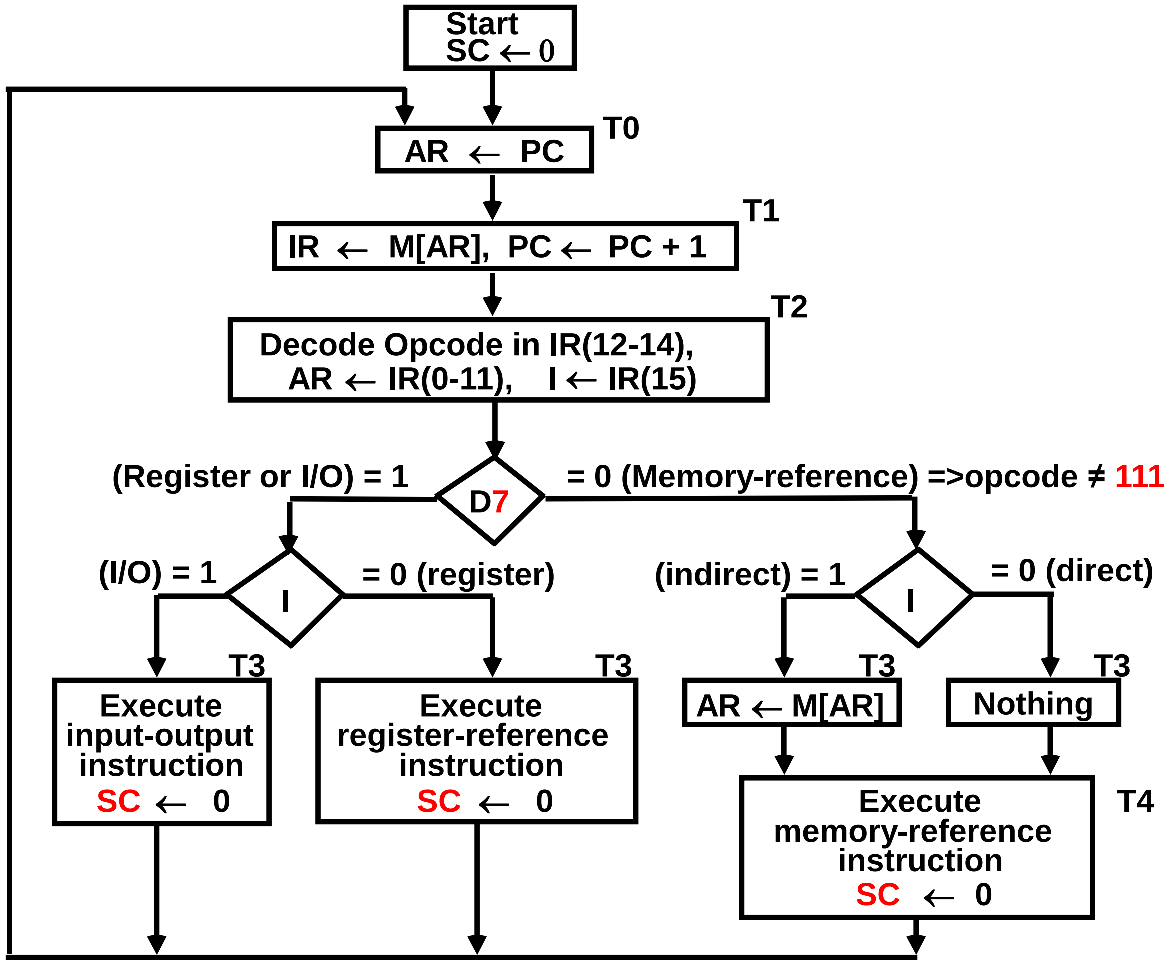
FETCH and DECODE

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=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
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.

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**

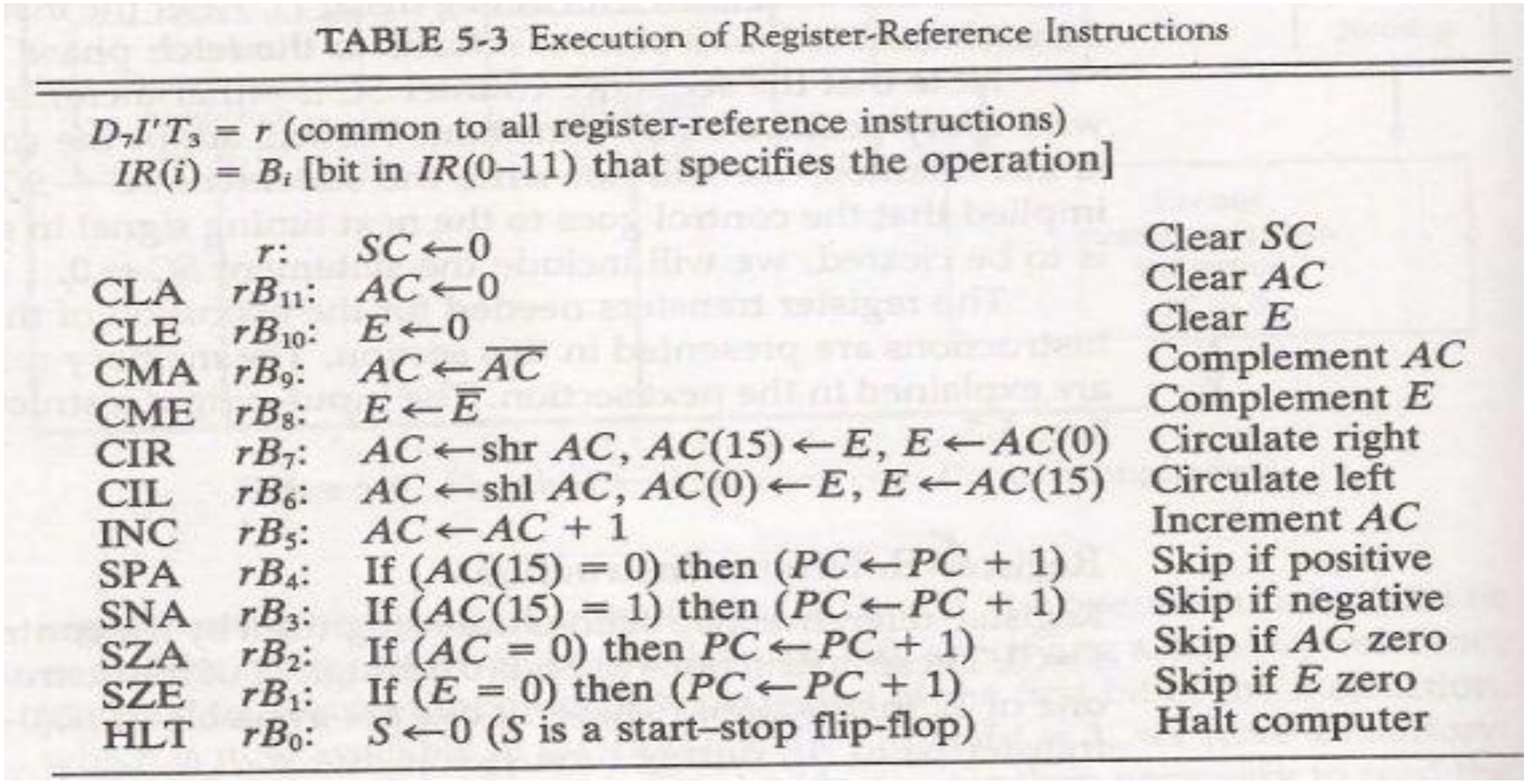


TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (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

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 T_3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

D_0T_4:	$DR \leftarrow M[AR]$	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 (Extended Accumulator)

MEMORY REFERENCE INSTRUCTIONS

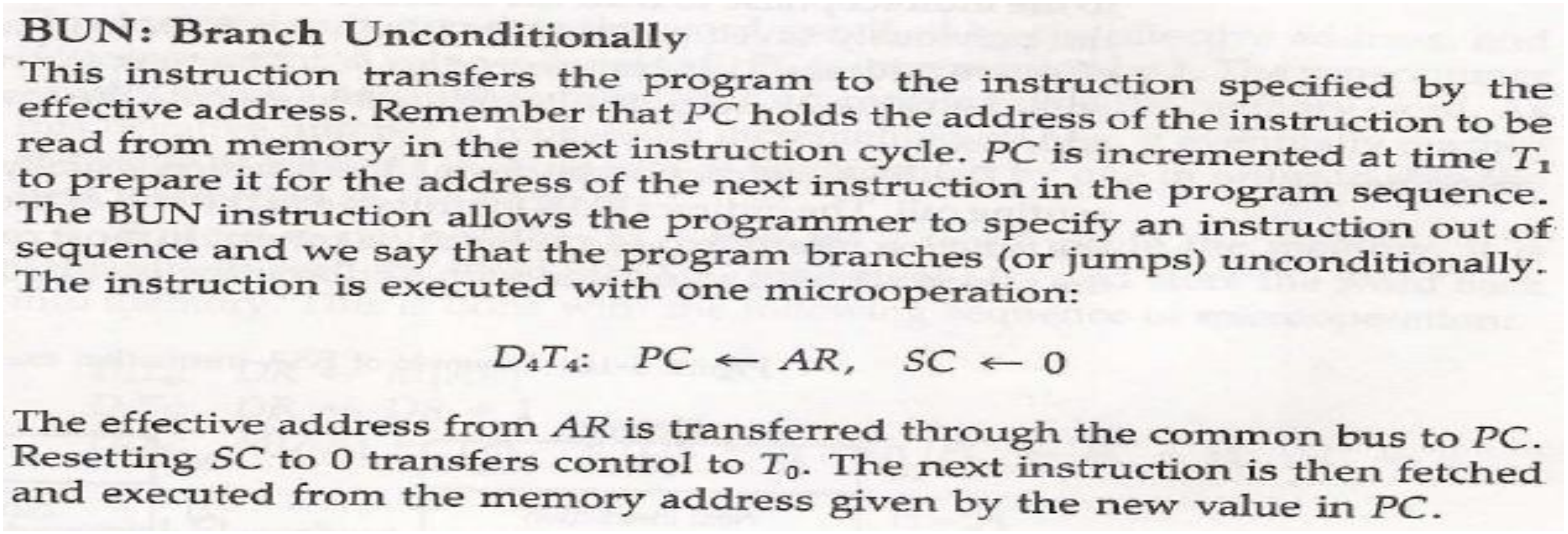
LDA: Load to AC

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

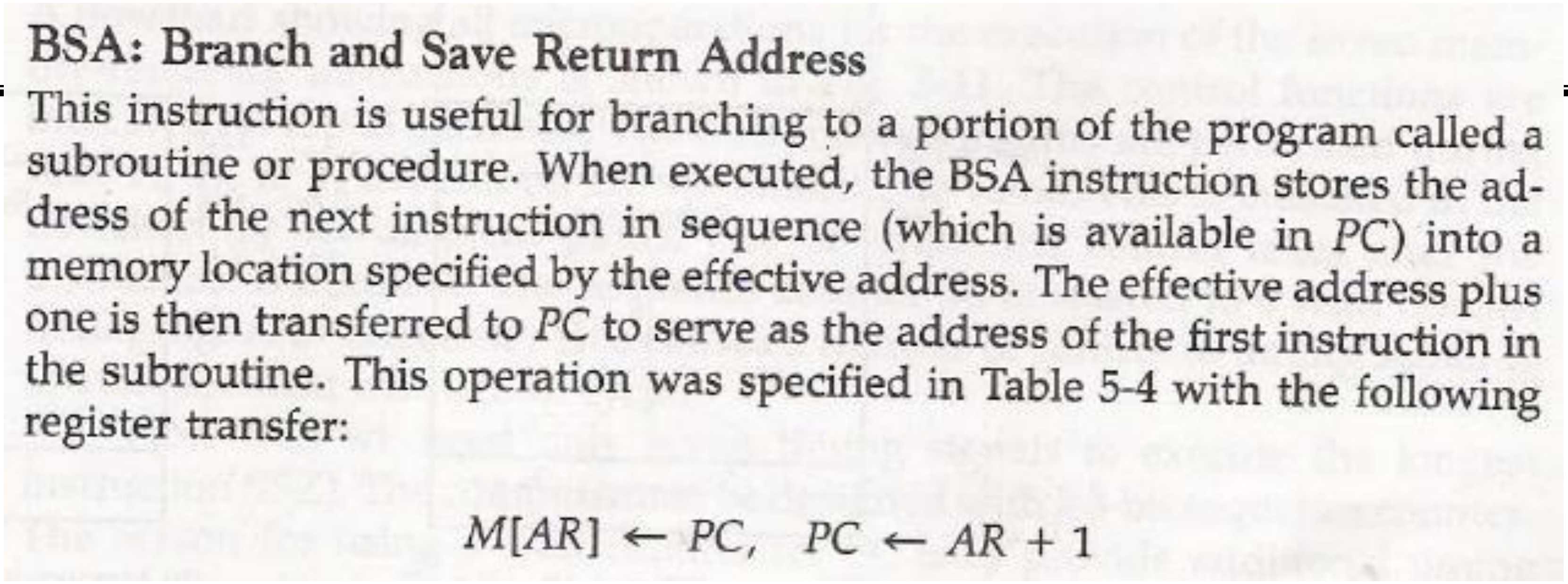
This instruction transfers the program to the instruction specified by the effective address. Remember that *PC* holds the address of the instruction to be read from memory in the next instruction cycle. *PC* is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

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

The effective address from *AR* is transferred through the common bus to *PC*. Resetting *SC* to 0 transfers control to T_0. The next instruction is then fetched and executed from the memory address given by the new value in *PC*.

BUN: Branch Unconditionally

D_4T_4:PC ← AR, SC ← 0



BSA: Branch and Save Return Address

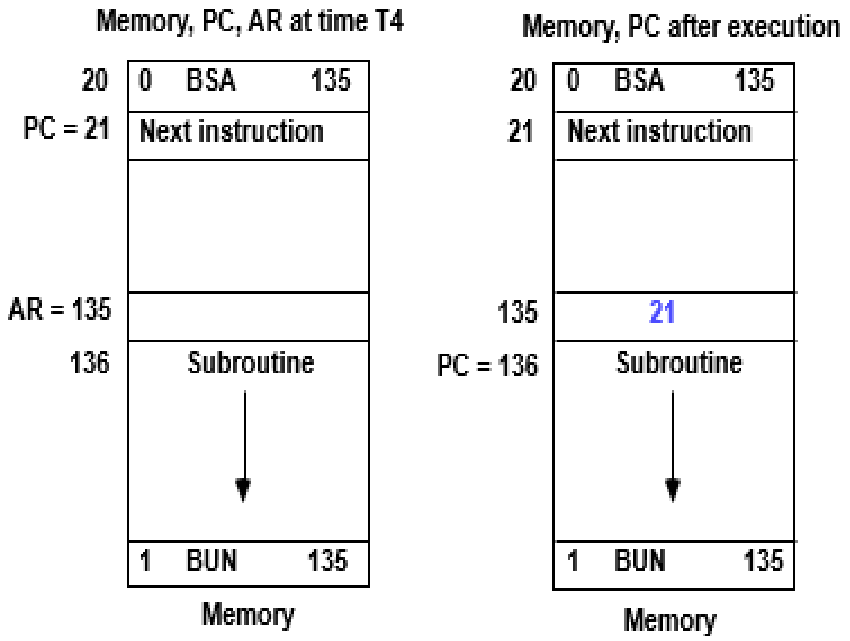
This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in *PC*) into a memory location specified by the effective address. The effective address plus one is then transferred to *PC* to serve as the address of the first instruction in the subroutine. This operation was specified in Table 5-4 with the following register transfer:

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

BSA:

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

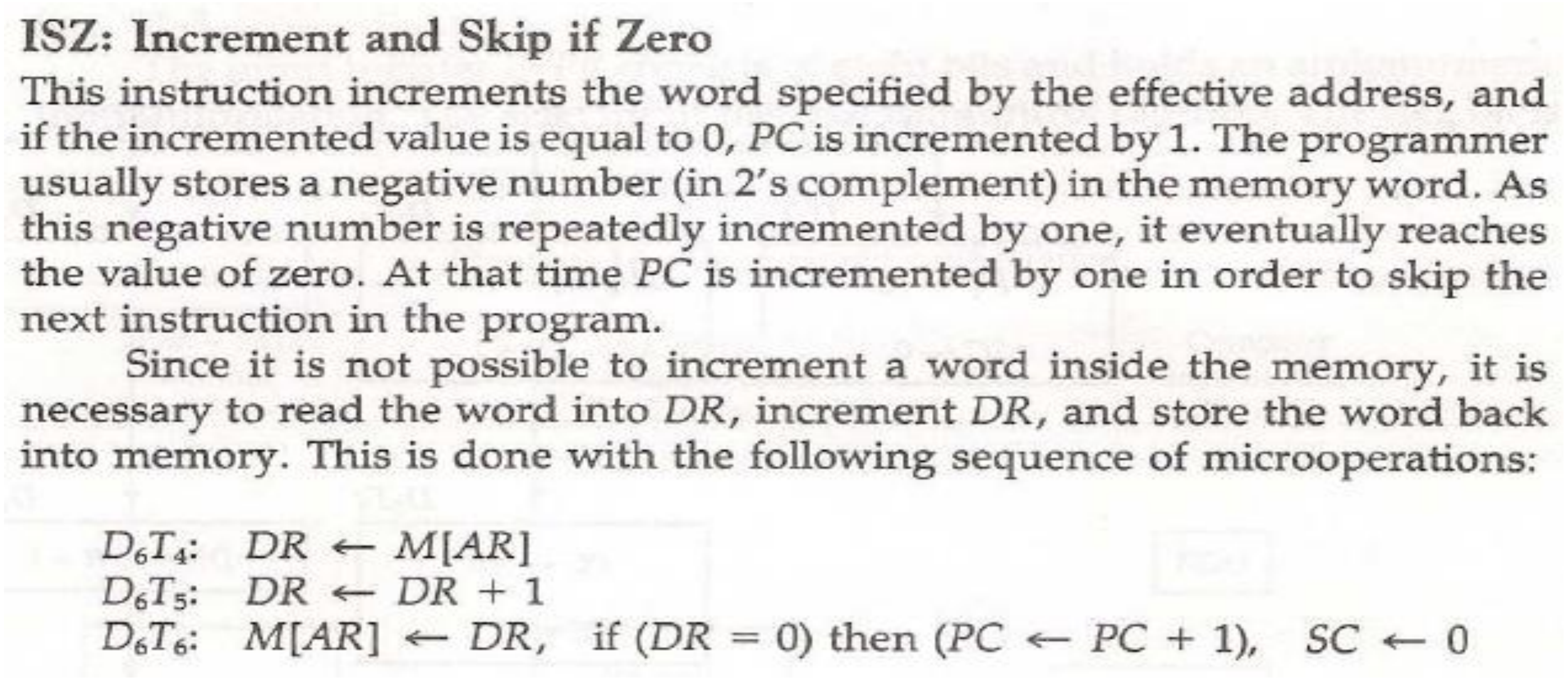
D_5T_5: $PC \leftarrow AR,\ SC \leftarrow 0$



MEMORY REFERENCE INSTRUCTIONS

ISZ: Increment and Skip-if-Zero

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 $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$



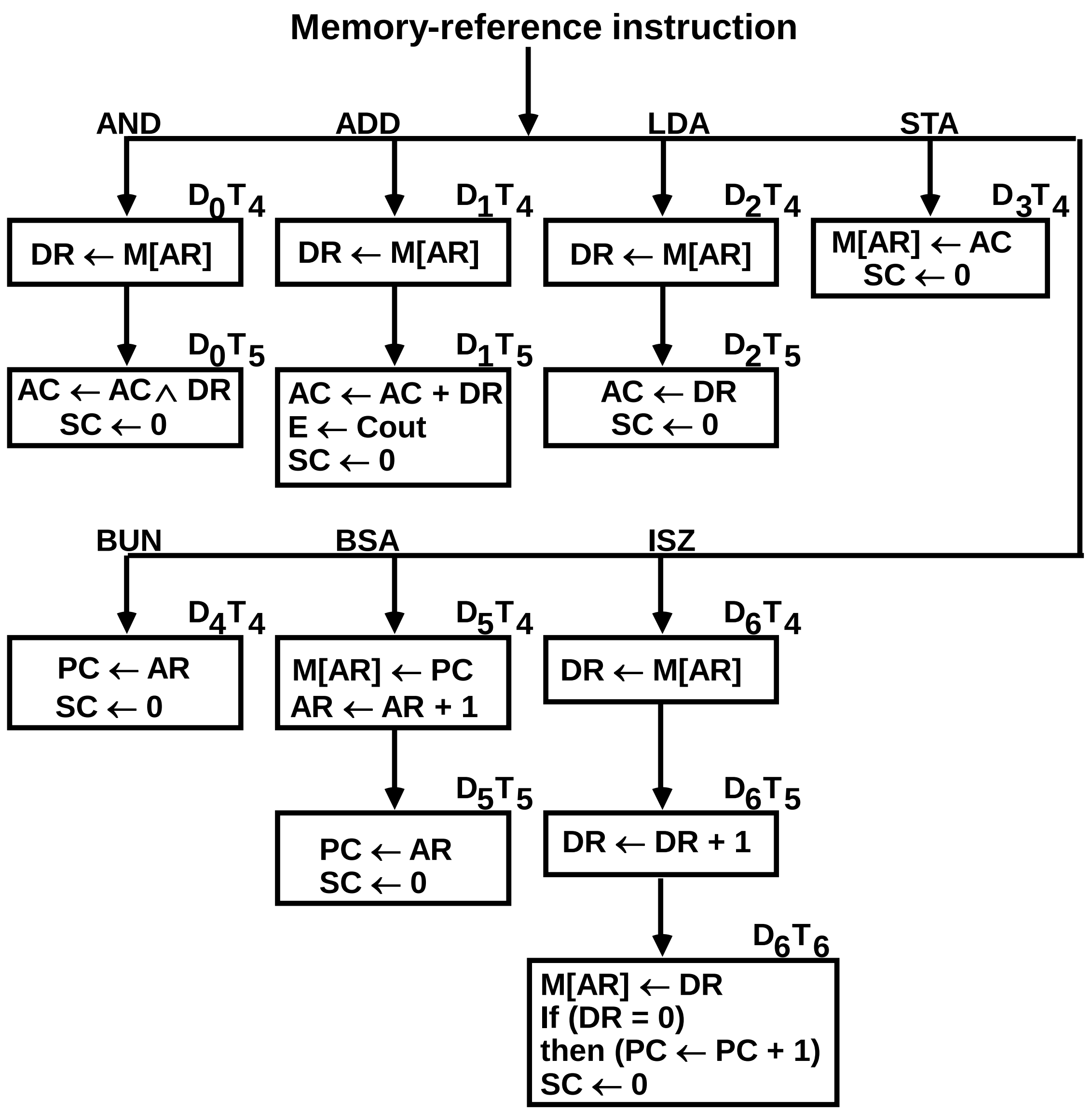
ISZ: Increment and Skip if Zero

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, *PC* is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time *PC* is incremented by one 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. This is done with the following sequence of microoperations:

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 $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

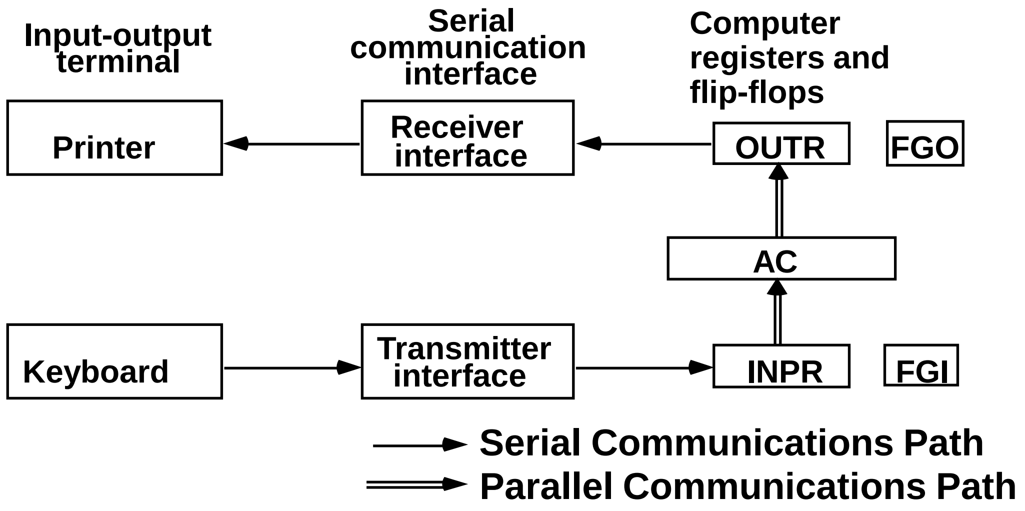
FLOWCHART FOR MEMORY REFERENCE INSTRUCTIONS



INPUT-OUTPUT AND INTERRUPT

A Terminal with a keyboard and a Printer

- **Input-Output Configuration**


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

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

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

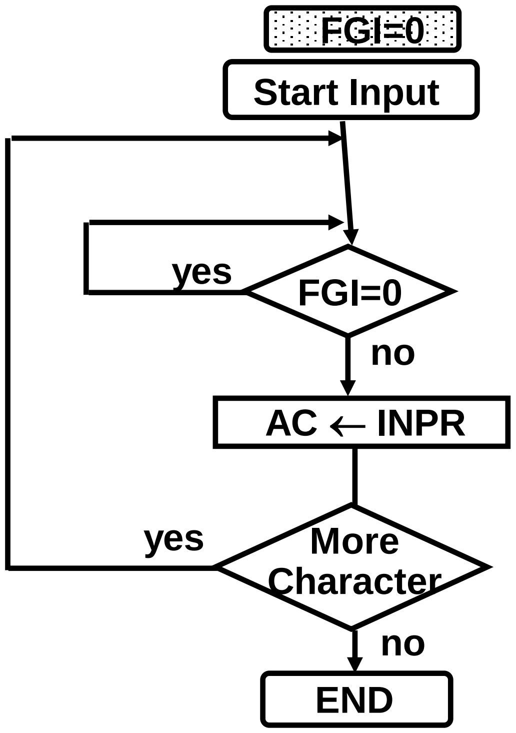
PROGRAM CONTROLLED DATA TRANSFER

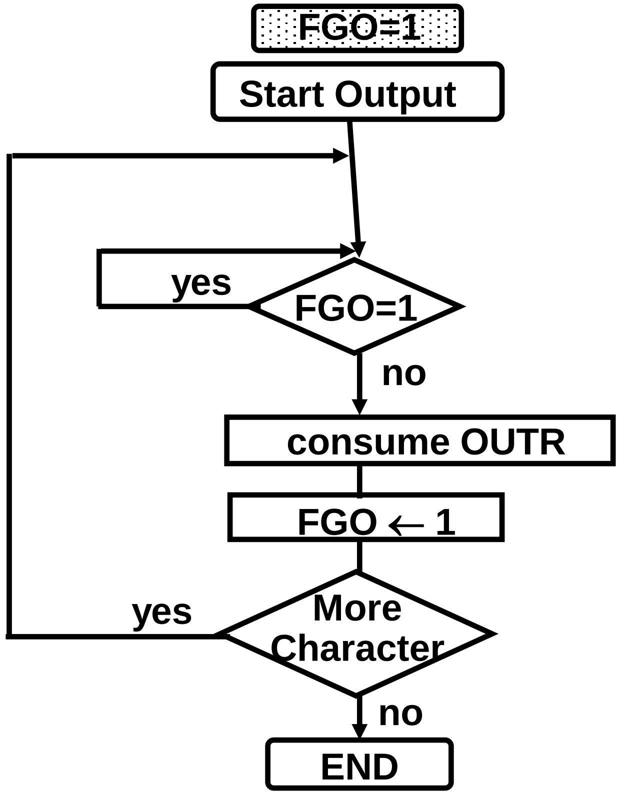
-- CPU --

```
loop:  If FGI = 0 goto loop
           AC ← INPR,  FGI ← 0

/* Output */         /* Initially FGO = 1 */
   loop:  If FGO = 0 goto loop
           OUTR ← AC,  FGO ← 0
```

-- I/O Device --

```
/* Input */        /* Initially FGI = 0 */
loop: If FGI = 1 goto loop
      INPR ← new data, FGI ← 1

loop: If FGO = 1 goto loop
      consume OUTR, FGO ← 1
```





INPUT-OUTPUT INSTRUCTIONS

CPU Side

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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input char. to AC
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$	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-CONTROLLED INPUT/OUTPUT

- **Program-controlled I/O**
  - **Continuous CPU involvement**

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

INTERRUPT INITIATED INPUT/OUTPUT

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

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

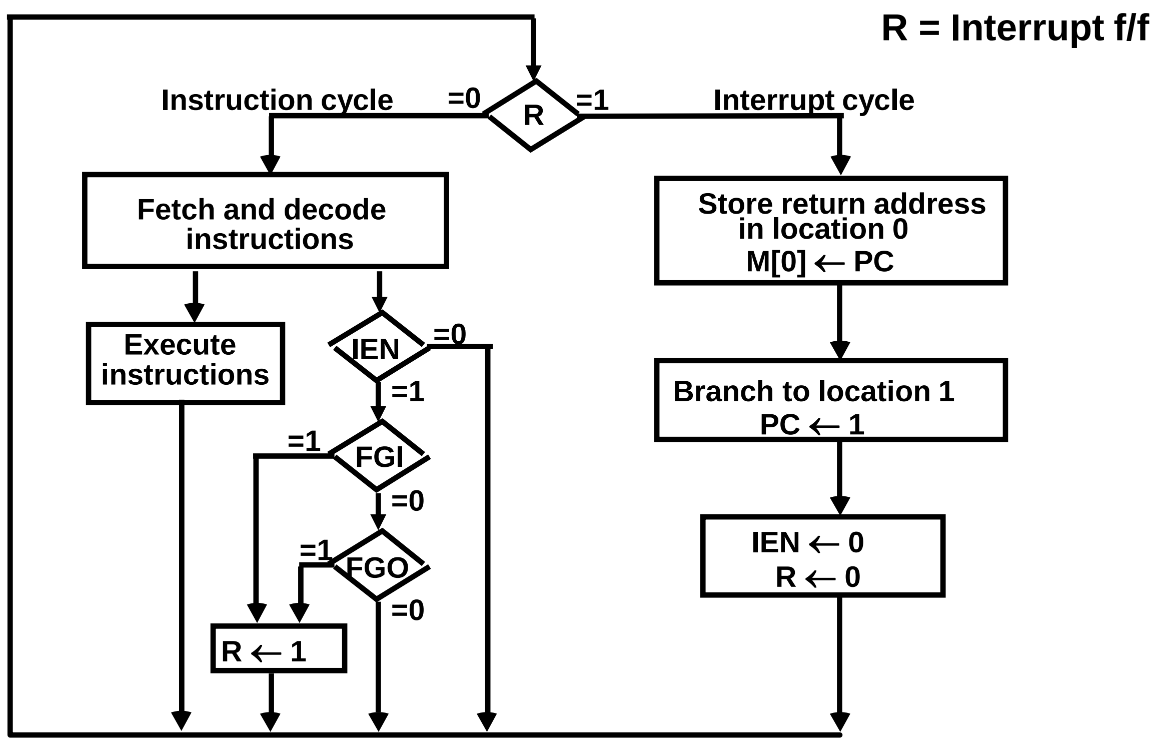
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

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

*** IEN (Interrupt-enable flip-flop)**

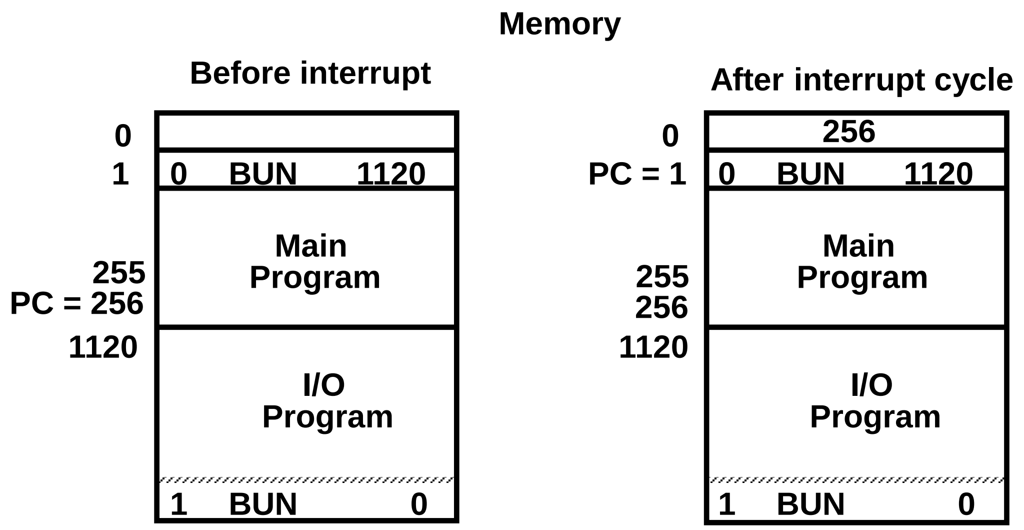
- can be set and cleared by instructions
- when cleared, the computer cannot be interrupted

FLOWCHART FOR INTERRUPT CYCLE



- The interrupt cycle is a HW implementation of a branch and save return address operation.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

REGISTER TRANSFER OPERATIONS IN INTERRUPT CYCLE



Register Transfer Statements for Interrupt Cycle

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

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

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

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

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

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