Computer System Architectures

BASIC COMPUTER ORGANIZATION AND DESIGN

System Buses:

- The major computer system components (processor, main memory, I/O modules) need to be interconnected in order to exchange data and control signals
- A bus is a communication pathway connecting two or more devices
- A bus that connects major computer components (processor, memory, I/O) is called a system bus.
- Bus = a shared transmission medium. Only one device at a time Can successfully transmit.
    - shared system bus consisting of multiple lines
    - A hierarchy of buses to improve performance.
- Key design elements for buses include: Arbitration, Timing, width

Multiple Bus Hierarchies:

- In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay.
- The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.
-

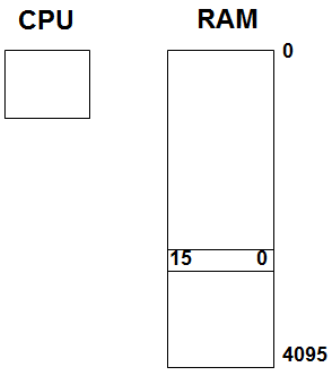
Synchronous Buses:

- Synchronous buses include a clock line between the control lines, line that is controlled by a clock quartz oscillator, usually between 5 - 133 MHz
- All the transfers on the system bus has a fixed protocol related to the clock signal, and it is developed along an integer number of cycles, called bus cycles.
- The advantages of a synchronous bus are a high speed of transfer, the very simple implied logic
- The disadvantage comes from transfers that can be shorter than the time corresponding to the integer number of bus cycles.
-

Introduction of Basic Computer:

- Every different processor type has its own design (different registers, buses, Microoperations, 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, we will start with a simplified processor model
    - This is similar to what real processors were like ~25 years ago
    - M. Morris Mano introduces a simple processor model he calls the *Basic Computer*
    - We will use this to introduce processor organization and the relationship of the RTL model to the higher level computer processor.

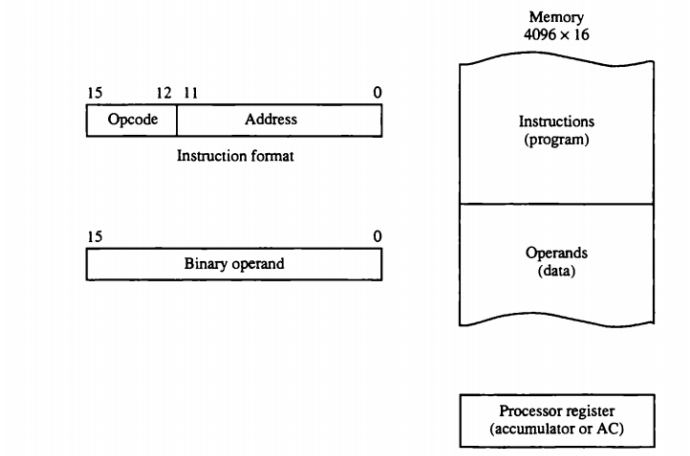
- The Basic Computer has two components, a processor and memory
- The memory has 4096 words in it
  - $4096 = 2^{12}$, so it takes 12 bits to select a word in memory
- Each word is 16 bits long.



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.

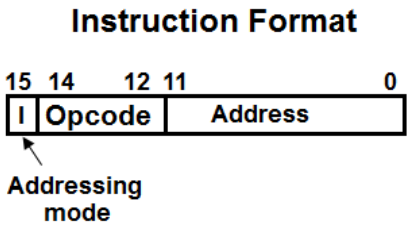
Stored program organization



Instruction Format:

- A computer instruction is often divided into two parts
  - An *opcode* (Operation Code) that specifies the operation for that instruction

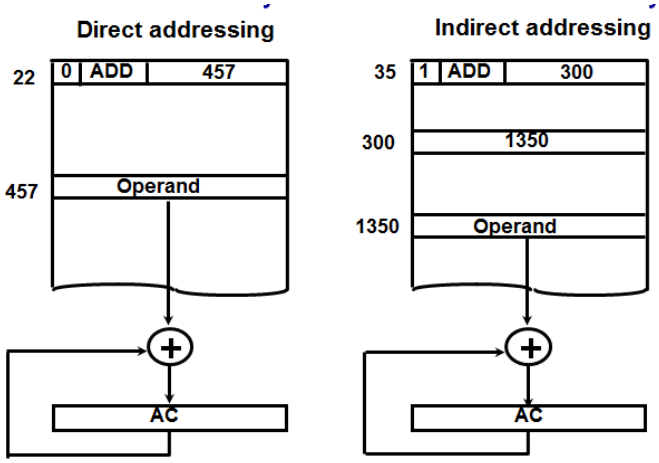
  - An *address* that specifies the registers and/or locations in memory to use for that operation
- In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use
- In the Basic Computer, bit 15 of the instruction specifies 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.
-



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.

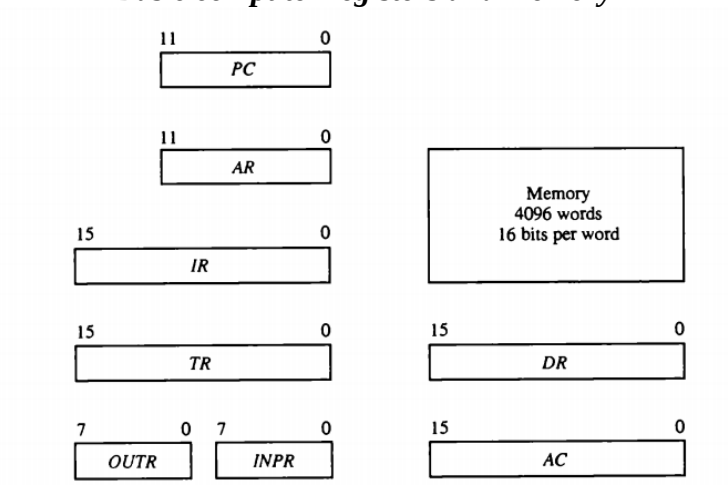
Processor Registers:

- 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 to get
  - 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).

- The significance of a general purpose register is that it can be referred to in instructions
  - 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 sent to an output device.

Basic computer registers and memory

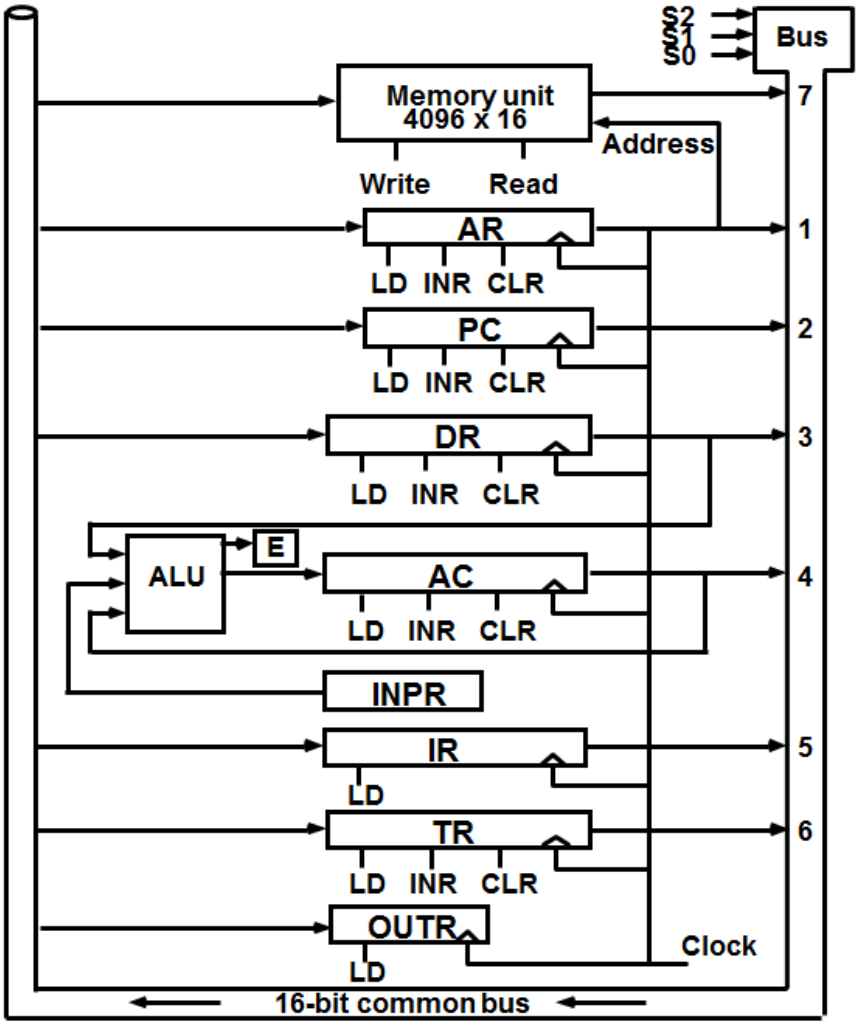


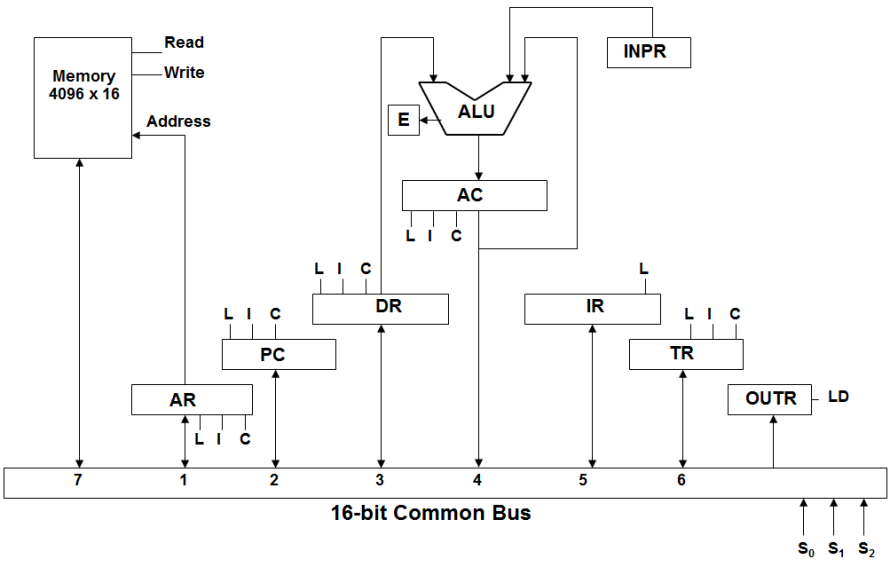
List of Register for the basic computer

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

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.





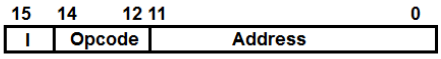
- Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input:

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.

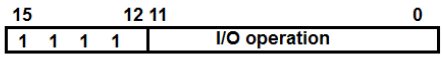
Basic Computer Instruction Format:

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



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

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



A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable.

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

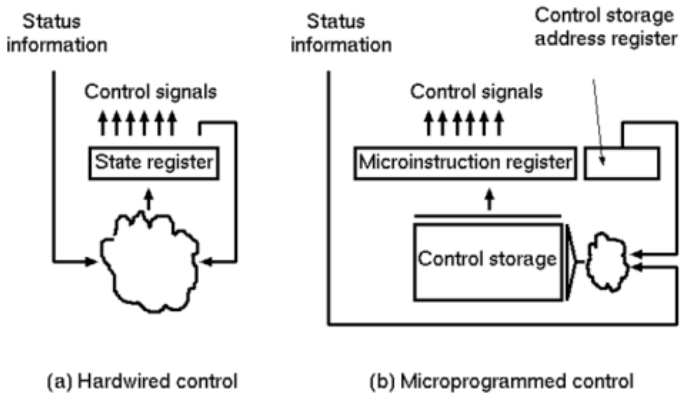
Input/Output Instructions
- Input and output
- INP, OUT.

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

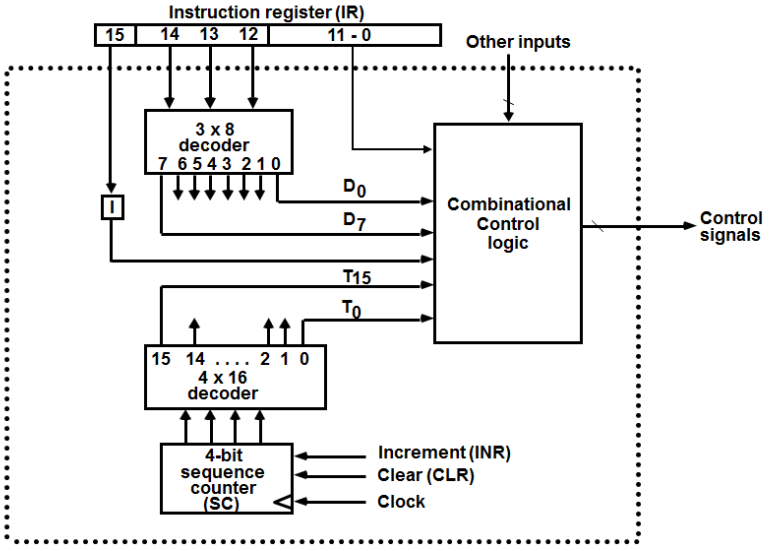
- Control unit (CU) of a processor translates from machine instructions to the control signals for the Microoperations that implement them
- Control units are implemented in one of two ways
- *Hardwired* Control
  - CU is made up of sequential and combinational circuits to generate the control signals
- *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

Hardwired/Microprogrammed:



Timing and Control:

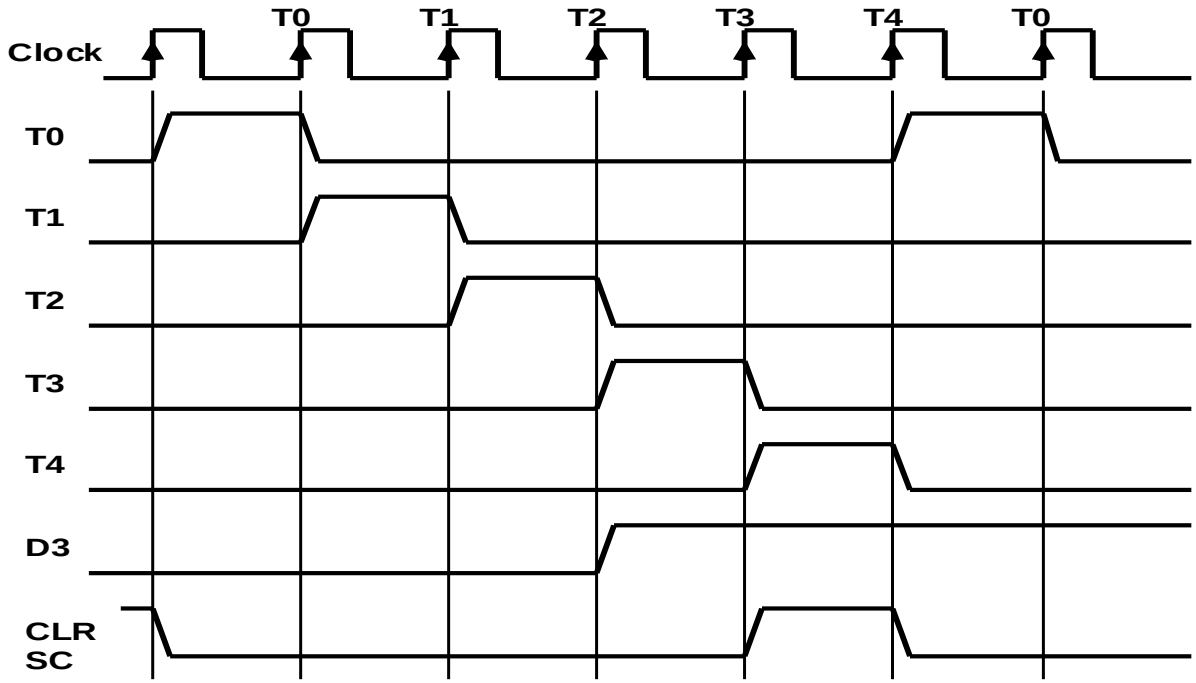
Control unit of basic computer



Timing Signals:

Generated by 4-bit sequence counter and 4×16 decoder
- The SC can be incremented or cleared.
- Example: T0, T1, T2, T3, T4, T0, T1, . . .
  Assume: At time T4, SC is cleared to 0 if decoder output D3 is active.

D_3T_4: SC ← 0

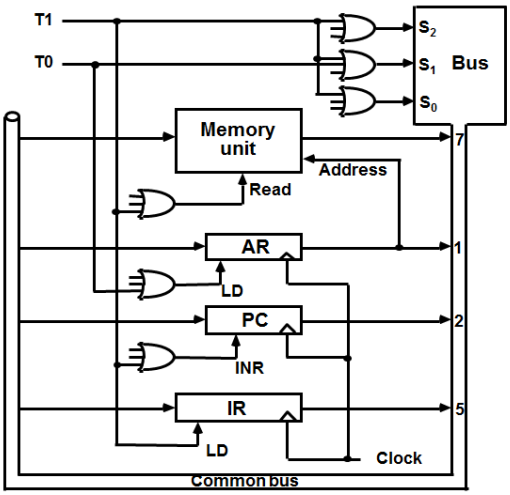


Instruction Cycle:

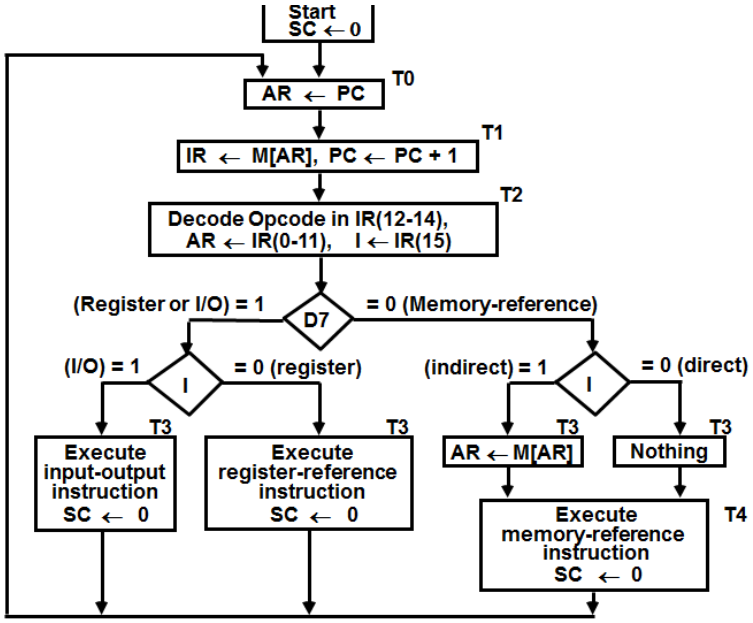
- In Basic Computer, a machine instruction is executed in the following cycle:
  - Fetch an instruction from memory
  - Decode the instruction
  - Read the effective address from memory if the instruction has an indirect address
  - Execute the instruction
  - After an instruction is executed, the cycle starts again at step 1, for the next instruction
- *Note*: Every different processor has its own (different) instruction cycle

Fetch and Decode:

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:



D'7IT3: AR ¬ M[AR]
D'7I'T3: Nothing
D7I'T3: Execute a register-reference instr.
D7IT3: Execute an input-output instr.

Register Reference Instructions:

Register Reference Instructions are identified when
- **$D_7 = 1$, $I = 0$**
- **Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR**
- **Execution starts with timing signal T_3**

$r = D_7 I'T_3$ => Register Reference Instruction
$B_i = IR(i)$, i=0,1,2,...,11

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in IR(0–11) 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:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ 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 ← M[AR] **Read operand**

D_0T_5: AC ← AC ∧ DR, SC ← 0 **AND with AC**

ADD to AC

D_1T_4: DR ← M[AR] **Read operand**

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0 **Add to AC and store carry in E**

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

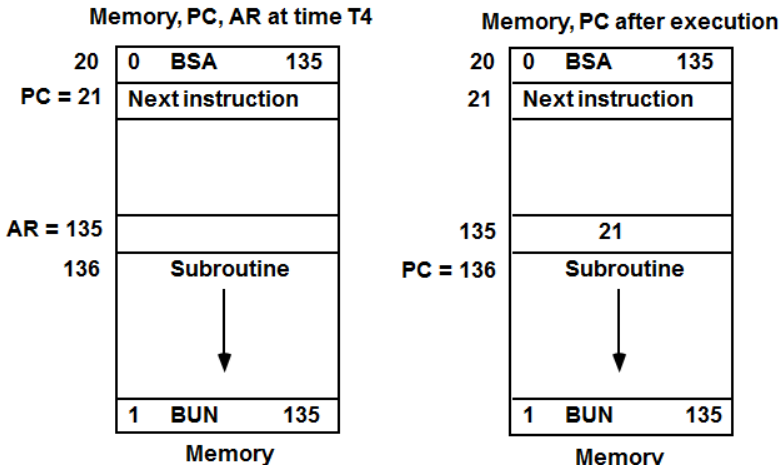
D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1



BSA:

D_5T_4: M[AR] ← PC, AR ← AR + 1

D_5T_5: PC ← AR, SC ← 0

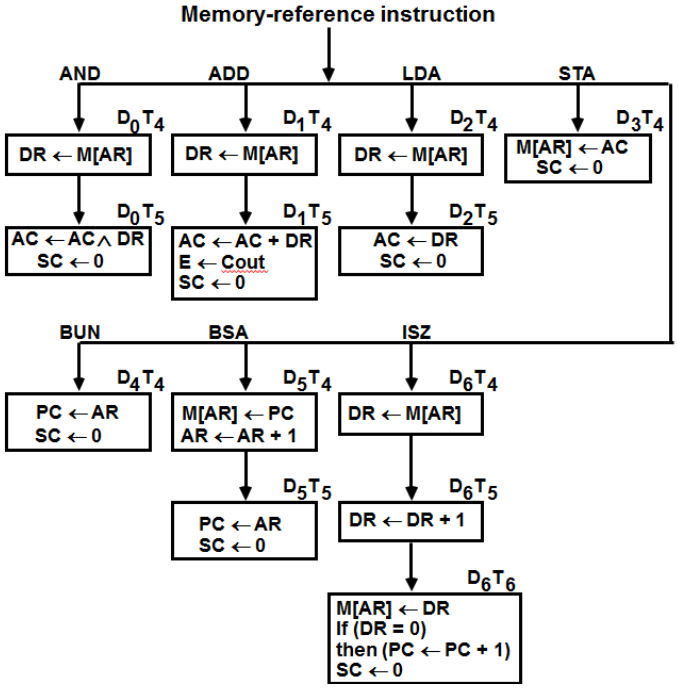
ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR]

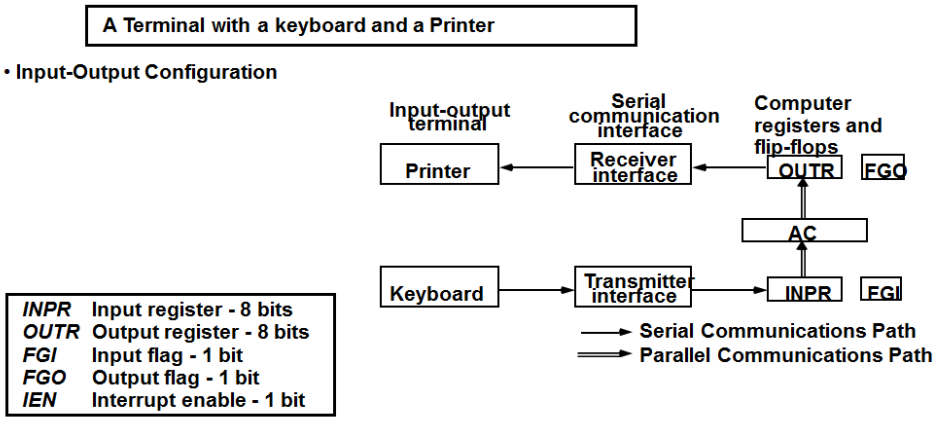
D_6T_5: DR ← DR + 1

D_6T_4: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

Flowchart for Memory Reference Instructions:

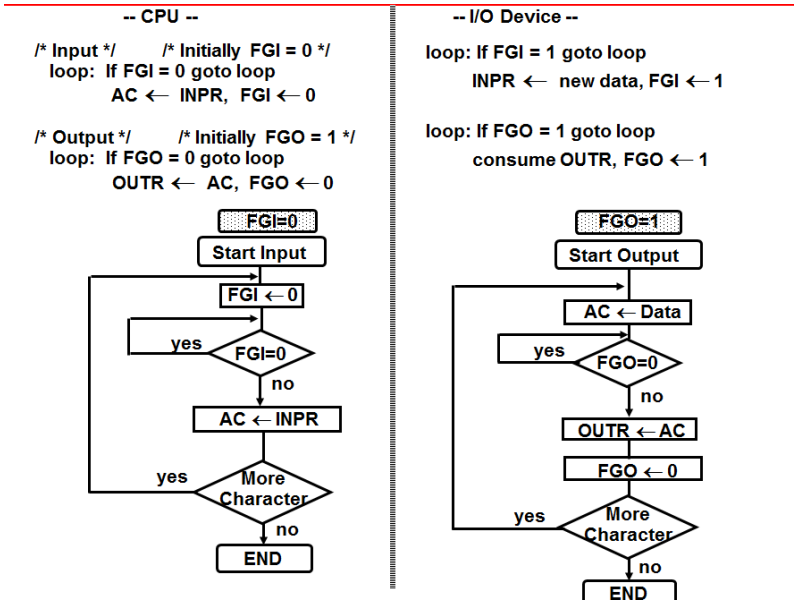


Input-Output and Interrupt:



- 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:



Input-Output Instructions:

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6\text{–}11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
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

Program-Controlled Input/output:

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

```
        Input
LOOP,   SKI   DEV
        BUN   LOOP
        INP   DEV
Output
        LOOP, LDA   DATA
        LOP,  SKO   DEV
              BUN   LOP
              OUT   DEV
```