

# Systems I: Computer Organization and Architecture

## Lecture 10: Basic Computer Organization and Design

### Instruction Codes

- An instruction code is a group of bits that instruct the computer to perform a specific operation.
- The operation code of an instruction is a group of bits that define operations such as addition, subtraction, shift, complement, etc.
- An instruction must also include one or more operands, which indicate the registers and/or memory addresses from which data is taken or to which data is deposited.

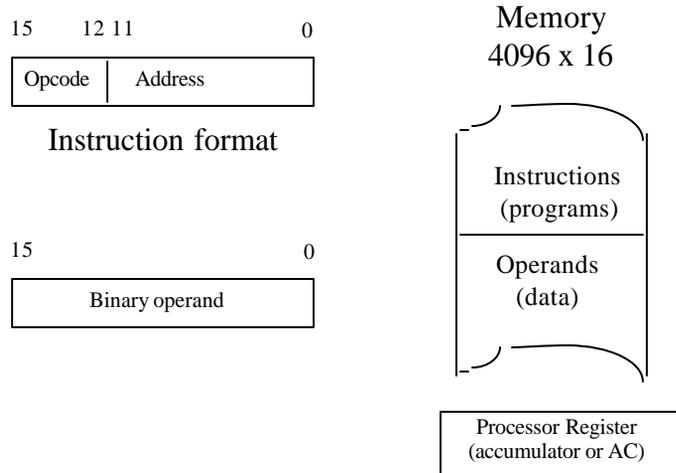
## Microoperations

- The instructions are stored in computer memory in the same manner that data is stored.
- The control unit interprets these instructions and uses the operations code to determine the sequences of microoperations that must be performed to execute the instruction.

## Stored Program Organization

- The operands are specified by indicating the registers and/or memory locations in which they are stored.
  - $k$  bits can be used to specify which of  $2^k$  registers (or memory locations) are to be used.
- The simplest design is to have one processor register (called the *accumulator*) and two fields in the instruction, one for the opcode and one for the operand.
- Any operation that does not need a memory operand frees the other bits to be used for other purposes, such as specifying different operations.

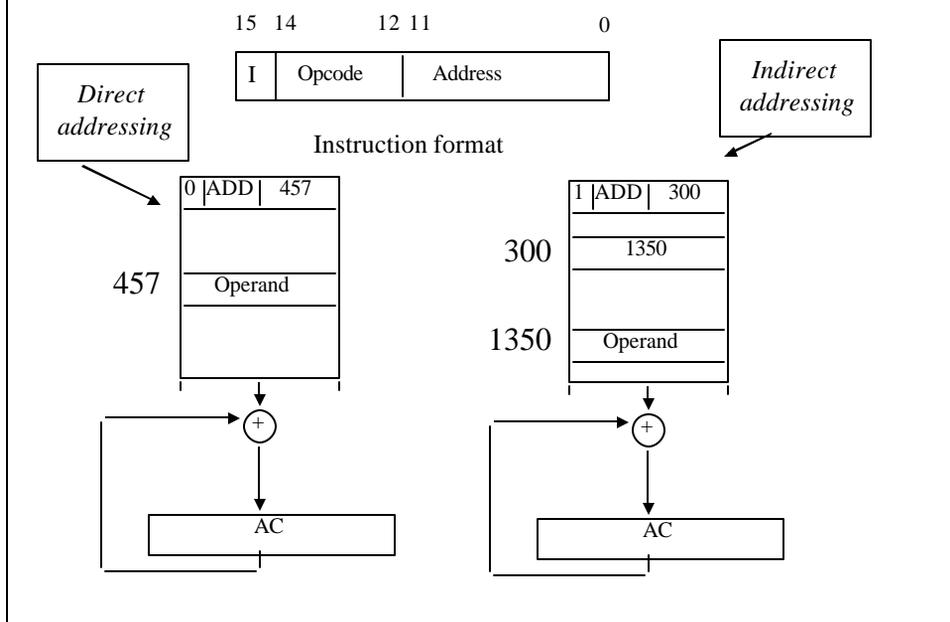
## Stored Program Organization



## Addressing Modes

- There are four different types of operands that can appear in an instruction:
  - ***Direct operand*** - an operand stored in the register or in the memory location specified.
  - ***Indirect operand*** - an operand whose address is stored in the register or in the memory location specified.
  - ***Immediate operand*** - an operand whose *value* is specified in the instruction.

## Direct and Indirect Addressing



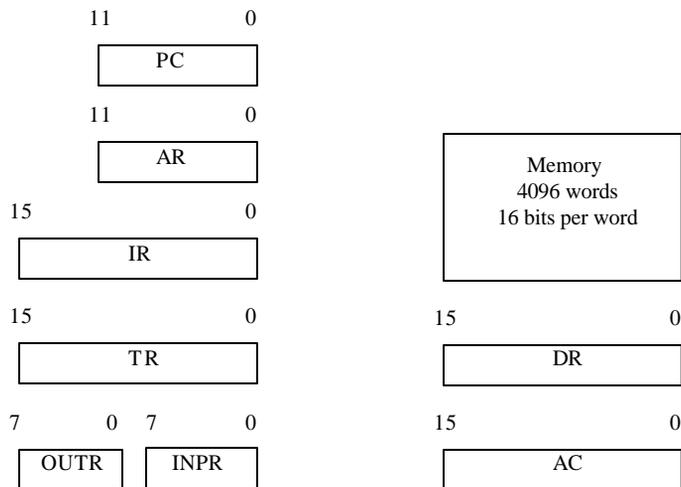
## Registers

- Computer instructions are stored in consecutive locations and are executed sequentially; this requires a register which can store the address of the next instruction; we call it the ***Program Counter***.
- We need registers which can hold the address at which a memory operand is stored as well as the value itself.
- We need a place where we can store
  - temporary data
  - the instruction being executed,
  - a character being read in
  - a character being written out.

## List of Registers for the Basic Computer

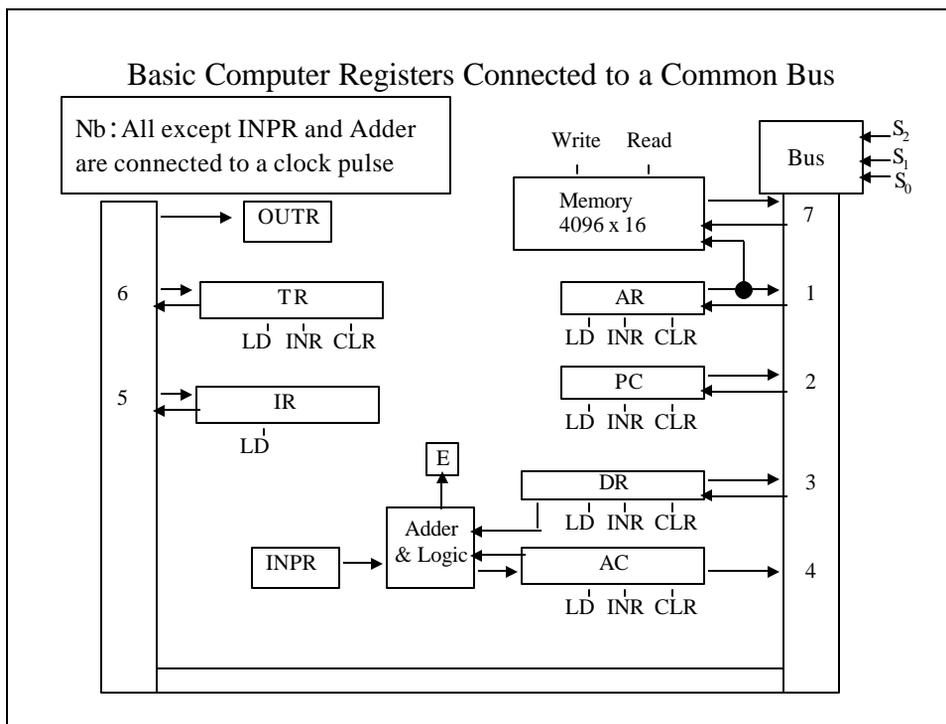
<u>Register Symbol</u>	<u># of Bits</u>	<u>Register Name</u>	<u>Function</u>
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds mem. address
AC	16	Accumulator	Processor Reg.
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds instruction address
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

## Basic Computer Registers and Memory



## The Common Bus

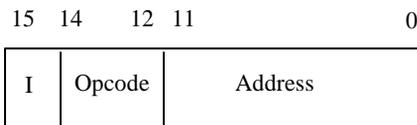
- To avoid excessive wiring, memory and all the register are connected via a common bus.
- The specific output that is selected for the bus is determined by  $S_2S_1S_0$ .
- The register whose LD (*Load*) is enable receives the data from the bus.
- Registers can be incremented by setting the INR control input and can be cleared by setting the CLR control input.
- The Accumulator's input must come via the Adder & Logic Circuit. This allows the Accumulator and Data Register to swap data simultaneously.
- The address of any memory location being accessed must be loaded in the Address Register.



## Computer Instructions

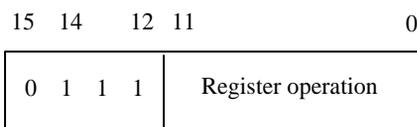
- The basic computer has three instruction code formats:
  - Memory-reference format – where seven 3-bit opcodes are followed by a 12-bit memory address and preceded by a bit which indicates whether direct or indirect addressing is being used.
  - Register-reference format – where  $0111_2$  is followed by 12 bits which indicate a register instruction.
  - Input-output format – where  $1111_2$  is followed by 12 bits which indicate an input-output instruction.
- In register-reference and I/O formats, only one of the lower 12 bits is set.

### Basic Computer Instruction Formats



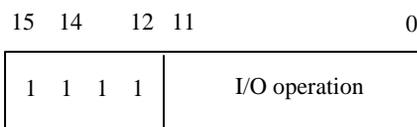
Opcode = 000 through 110

Memory-reference instruction



Opcode = 111, I = 0

Register-reference instruction



Opcode = 111, I = 1

Input-output instruction

## Instruction-Set Completeness

- A computer instruction set is said to be complete if the computer includes a sufficient number of instructions in each of these categories:
  - Arithmetic, logical and shift instructions
  - Instructions for moving data from registers to memory and memory to registers.
  - Program-control and status-checking instructions
  - Input and output instructions

## Arithmetic, Logic and Shifting Completeness

- We have instructions for adding, complementing and incrementing the accumulator. With these we can also subtract.
- AND and complement provide NAND, from which all other logical operations can be constructed.
- We can construct logical and arithmetic shifts from the circular shift operations.
- We can construct multiply and divide from adding, subtracting and shifting.
- While this is complete, it is not very efficient; it would be to our advantage to have subtract, multiply, OR and XOR.

## Instruction Set Completeness (continued)

- We can perform moves using the LDA and STA instructions.
- We have unconditional branches (BUN), subprogram calls (BSA) and conditional branches (ISZ).
- We also have all the instructions we need to perform input and output and handle the interrupt that they generate.

## Basic Memory-Reference Instructions

<u>Symbol</u>	<u>Hexadecimal code</u>		<u>Description</u>
	<u>I = 0</u>	<u>I = 1</u>	
AND	0xxx	8xxx	AND mem. Word to AC
ADD	1xxx	9xxx	ADD mem. Word to AC
LDA	2xxx	Axxx	Load mem. Word to AC
STA	3xxx	Bxxx	Store Content of AC in mem.
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero

## Basic Register-Reference Instructions

<u>Symbol</u>	<u>Hex. Code</u>	<u>Description</u>
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC & E
CIL	7040	Circulate left AC & E
INC	7020	Increment AC

## Basic Register-Reference Instructions (continued)

<u>Symbol</u>	<u>Hex. Code</u>	<u>Description</u>
SPA	7010	Skip next instruction if AC is positive
SNA	7008	Skip next instruction if AC is negative
SZA	7004	Skip next instruction if AC is zero
SZE	7002	Skip next instruction if E is zero
HLT	7001	Halt computer

## Basic Input-Output Instructions

<u>Symbol</u>	<u>Hex. Code</u>	<u>Description</u>
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

## Timing and Control

- The timings for all the registers is controlled a master clock generator.
  - Its pulses are applied to all flip-flops and registers, including in the control unit.
  - The control signals are generated in the control unit and provide control inputs for the bus's mutlitplexers and for the processor registers and provides microoperations for the accumulator.

## Control

- There are two types of control:
  - ***Hardwired*** – control logic is implemented with gates, flip-flops, decoders and other digital circuits.
  - ***Microprogrammed*** – control information is stored in a control program, which is programmed to perform the necessary steps to implement instructions.

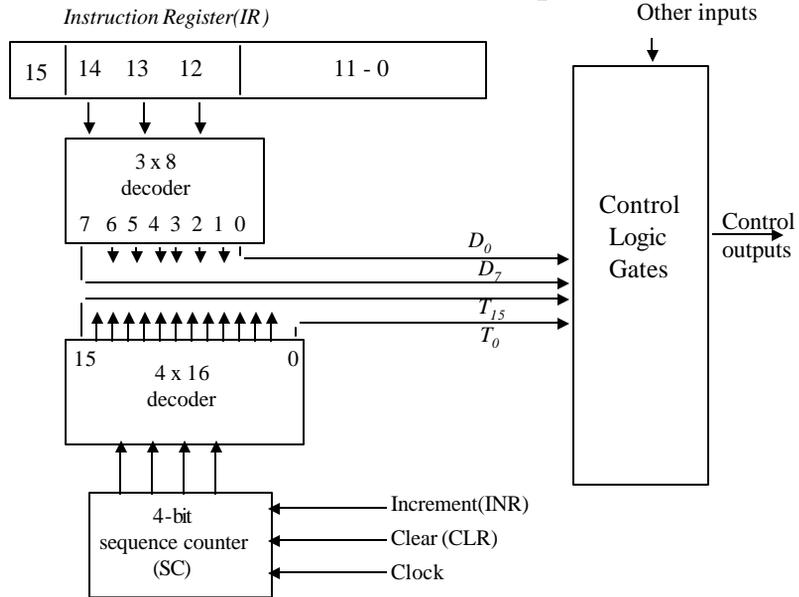
## Timing Signals

- Timing signals are generated by the sequence counter (SC), which receives as inputs the clock pulse, increment and clear.
- The SC's outputs are decoded into 16 timing signal  $T_0$  through  $T_{15}$ , which are used to control the sequence of operations.
- The RTL statement

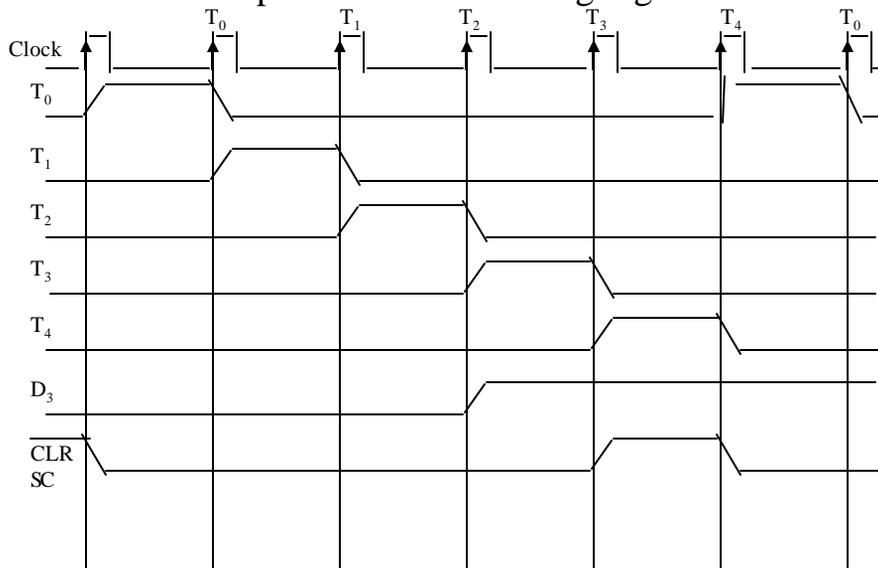
$$D_3T_4: SC \rightarrow 0$$

resets the sequence counter to zero; the next timing signal is  $T_0$

## Control Unit of Basic Computer



## Examples of Control Timing Signals



## Instruction Cycle

- The instructions of a program are carried out by a process called the *instruction cycle*.
- The instruction cycle consists of these phases:
  - Fetch an instruction from memory
  - Decode the instruction
  - Read the effective address from memory if the operand has an indirect address.
  - Execute the instruction.

## Fetch and Decode

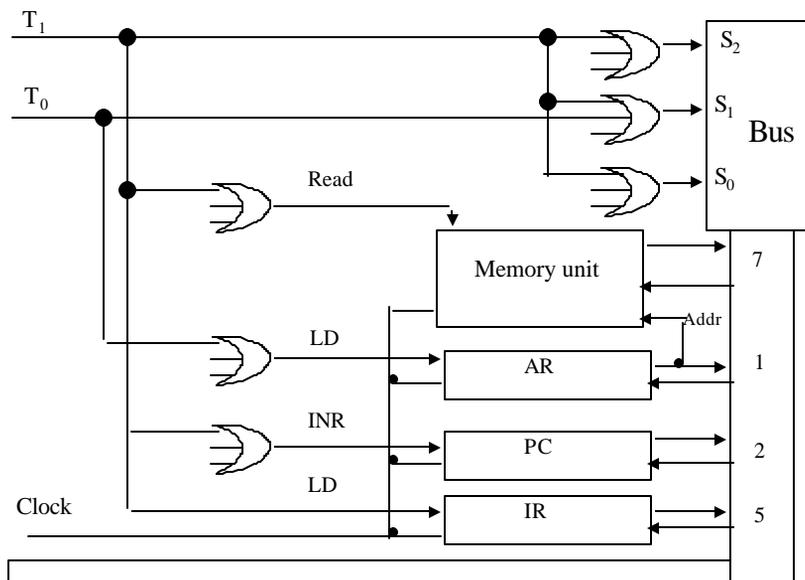
- Initially, the PC has stored the address of the instruction about to be executed and the SC is cleared to 0.
- With each clock pulses the SC is incremented and the timing signals go through the sequence  $T_0$ ,  $T_1$ ,  $T_2$ , etc.
- It is necessary to load the AR with the PC's address (it is connected to memory address inputs):

$$T_0: AR \leftarrow PC$$

## Fetch and Decode

- Subsequently, as we fetch the instruction to be executed, we must increment the program counter so that it points to the next instruction:
- $T_1$ :  $IR \leftarrow M[AR], PC \leftarrow PC + 1$
- In order to carry out the instruction, we must decode and prepare to fetch the operand. In the event it is an indirect operand, we need to have the indirect addressing bit as well:
- $T_2$ :  $D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14),$   
 $AR \leftarrow IR(0-11), I \leftarrow IR(15)$

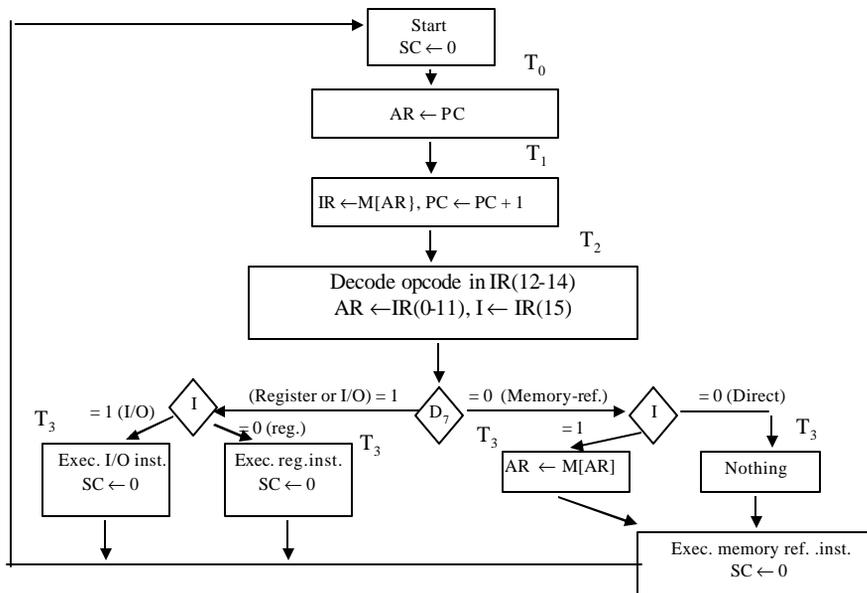
### Register Transfers For the Fetch Phase



## Type of Instruction and Addressing

- During time  $T_3$ , the control unit determines if this is a memory-reference, register-reference or input/output instruction.
  - The latter two are distinguished by the I (indirect) bit.
  - If it is a memory-reference instruction, the I bit will determine direct or indirect addressing.
- The four separate paths are:
  - $D_7'IT_3$ :  $AR \rightarrow M[AR]$
  - $D_7'IT_3$ : Nothing
  - $D_7IT_3$ : Execute a register-reference instruction
  - $D_7IT_3$ : Execute an input-output instruction

### Flowchart For Instruction Cycle



## Execution of Register-Reference Instructions

$D_7I'T_3 = r$  (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	$rB_9$	$AC \leftarrow AC'$	Complement AC
CME	$rB_8$	$E \leftarrow E'$	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

## Execution of Register-Reference Instructions

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	Op. Decoder	Symb. Desc.
AND	D <sub>0</sub>	$AC \leftarrow AC \wedge M[AR]$
ADD	D <sub>1</sub>	$AC \leftarrow AC + M[AR],$ $E \leftarrow C_{out}$
LDA	D <sub>2</sub>	$AC \leftarrow M[AR]$
STA	D <sub>3</sub>	$M[AR] \leftarrow AC$
BUN	D <sub>4</sub>	$PC \leftarrow AR$
BSA	D <sub>5</sub>	$M[AR] \leftarrow PC$ $PC \leftarrow AR + 1$
ISZ	D <sub>6</sub>	$M[AR] \leftarrow M[AR] + 1$ If $M[AR] + 1 = 0$ Then $PC \leftarrow PC + 1$

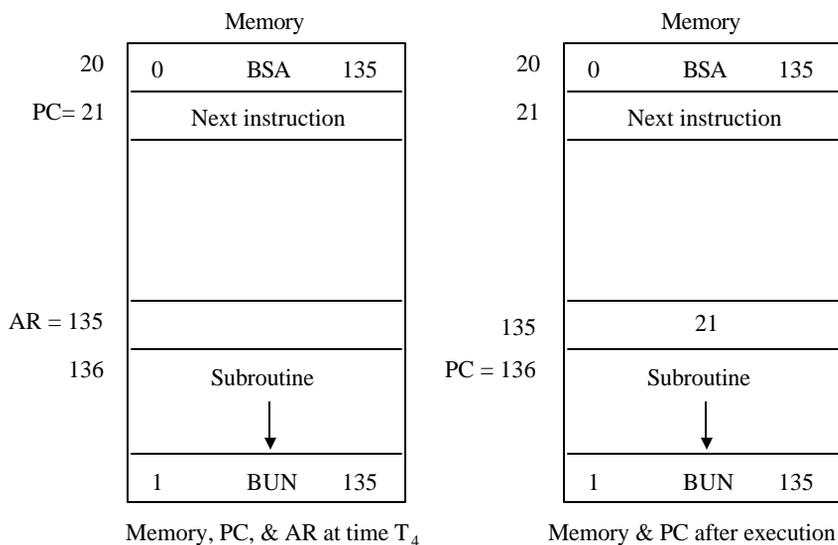
## Memory-Reference Instructions

- All memory-reference instructions have to wait until T<sub>4</sub> so that the timing is the same whether the operand is direct or indirect.
- **AND**, **ADD** and **LDA** must all be performed in two steps because AC can only be access via DR:
- **AND**:  
D<sub>0</sub>T<sub>4</sub>:  $DR \leftarrow M[AR]$   
D<sub>0</sub>T<sub>5</sub>:  $AC \leftarrow AC \wedge DR, SC \leftarrow 0$
- **ADD**:  
D<sub>1</sub>T<sub>4</sub>:  $DR \leftarrow M[AR]$   
D<sub>1</sub>T<sub>5</sub>:  $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
- **LDA**:  
D<sub>2</sub>T<sub>4</sub>:  $DR \leftarrow M[AR]$   
D<sub>0</sub>T<sub>5</sub>:  $AC \leftarrow DR, SC \leftarrow 0$

## Memory-Reference Instructions (continued)

- **STA** stores the contents of the AC, which can be applied directly to the bus:  
 $D_3T_4: \quad M[AR] \leftarrow AC, SC \leftarrow 0$
- **BUN** transfers control unconditionally to the effective address indicated by the effective address:  
 $D_4T_4: \quad PC \leftarrow AR, SC \leftarrow 0$
- **BSA** is used to branch to a subprogram. This requires saving the return address, which is saved at the operand's effective address with the subprogram beginning one word later in memory:  
 $D_5T_4: \quad M[AR] \leftarrow PC, AR \leftarrow AR + 1$   
 $D_5T_5: \quad PC \leftarrow AR, SC \leftarrow 0$

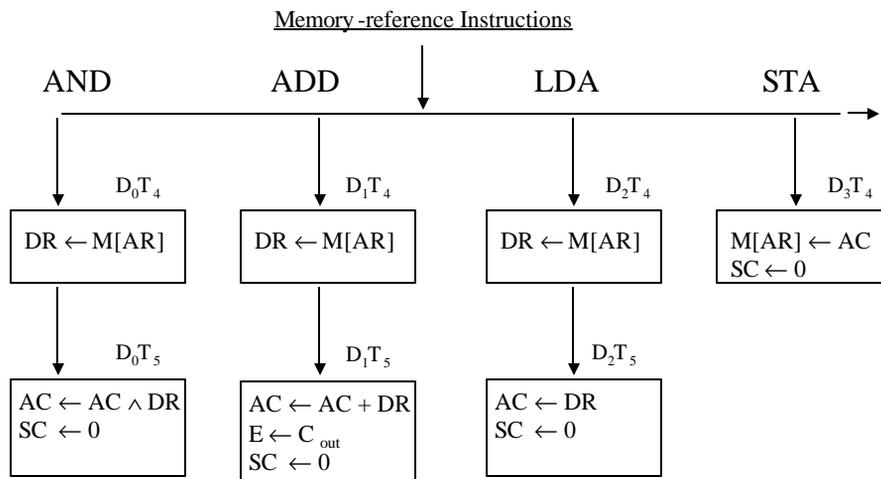
### Example of BSA Instruction Execution



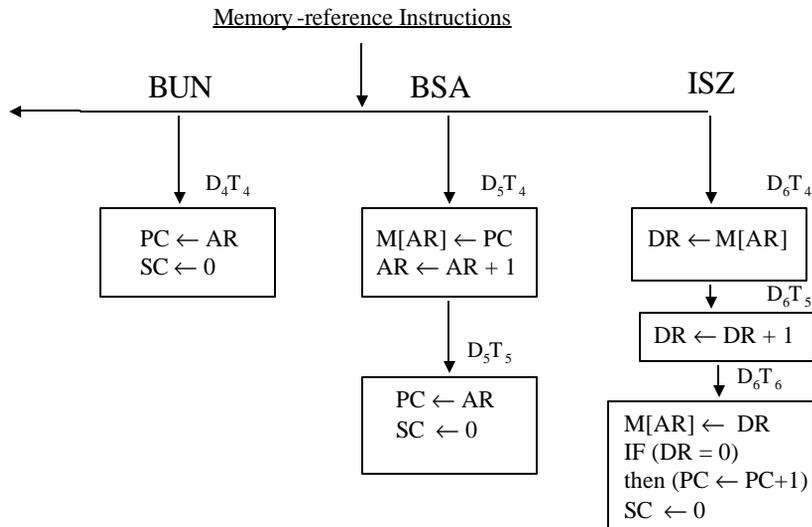
## Memory-Reference Instructions (continued)

- **ISZ** skips the next instruction if the operand stored at the effective address is 0. This requires that the PC incremented, which cannot be done directly:
- $D_6T_4$ :  $DR \leftarrow M[AR]$   
 $D_6T_5$ :  $DR \leftarrow DR + 1$   
 $D_6T_6$ :  $M[AR] \leftarrow DR$ ,  
 if  $(DR = 0)$  then  $(PC \leftarrow PC + 1)$ ,  
 $SC \leftarrow 0$

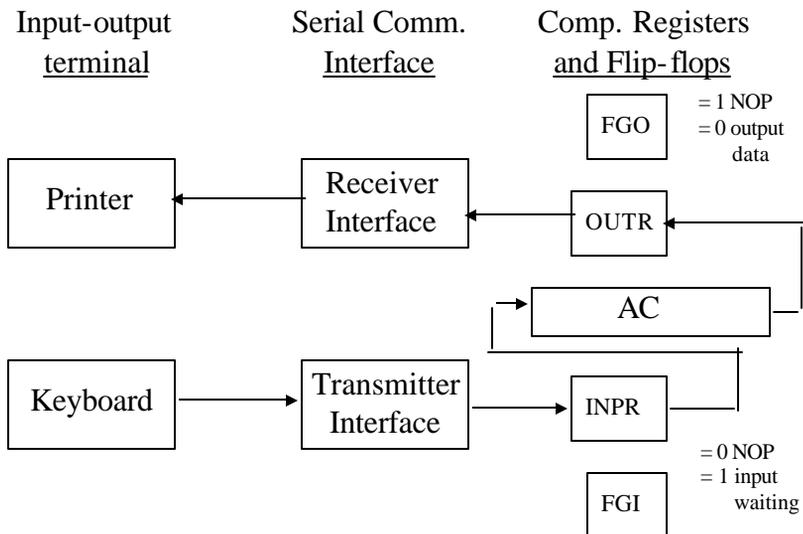
### Flowchart For Memory-Reference Instructions



## Flowchart For Memory-Reference Instructions (continued)



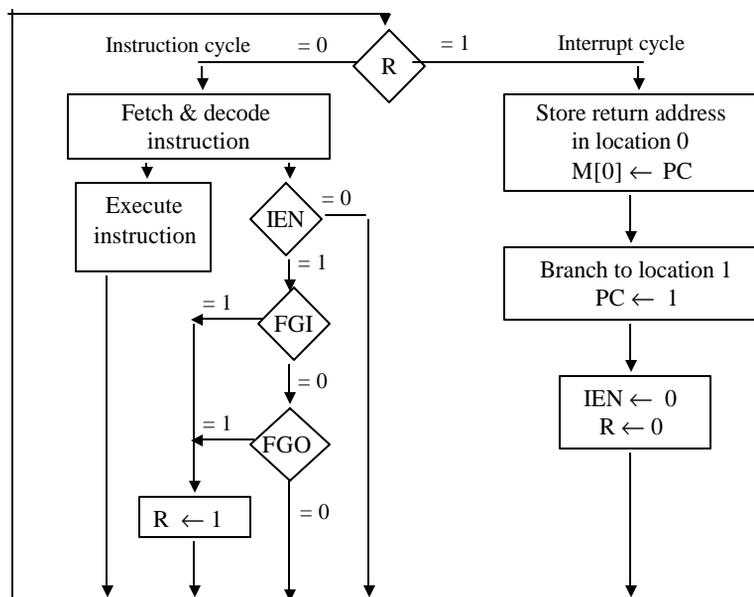
## Input-Output Configuration



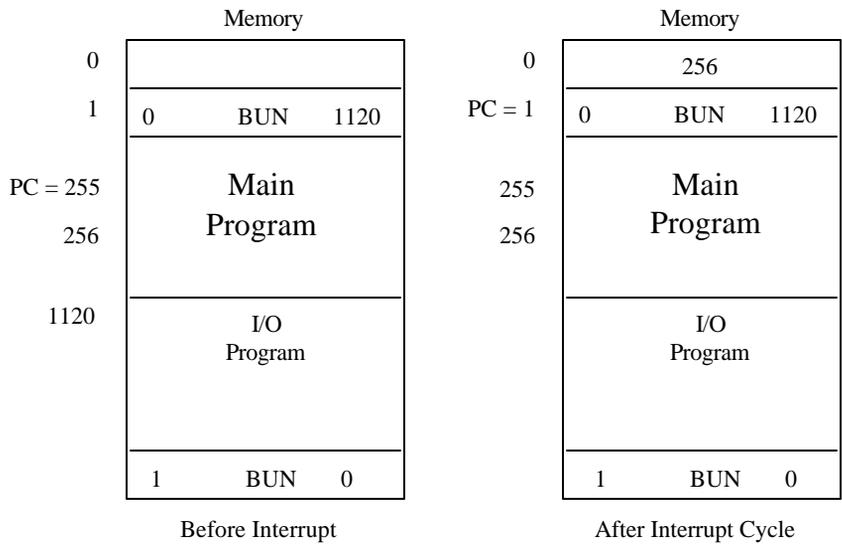
# Input-Output Instructions

	p	$SC \leftarrow 0$	Clear SC
INP	pB <sup>11</sup>	$AC(0-7) \leftarrow INPR,$ $FGI \leftarrow 0$	Input character
OUT	pB <sup>10</sup>	$OUTR \leftarrow AC(0-7),$ $FGO \leftarrow 0$	Output character
SKI	pB <sup>9</sup>	If (FGI = 1) Then $PC \leftarrow PC + 1$	Skip on input flag
SKO	pB <sup>8</sup>	If (FGO = 1) Then $PC \leftarrow PC + 1$	Skip on output flag
ION	pB <sup>7</sup>	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB <sup>6</sup>	$IEN \leftarrow 0$	Interrupt enable off

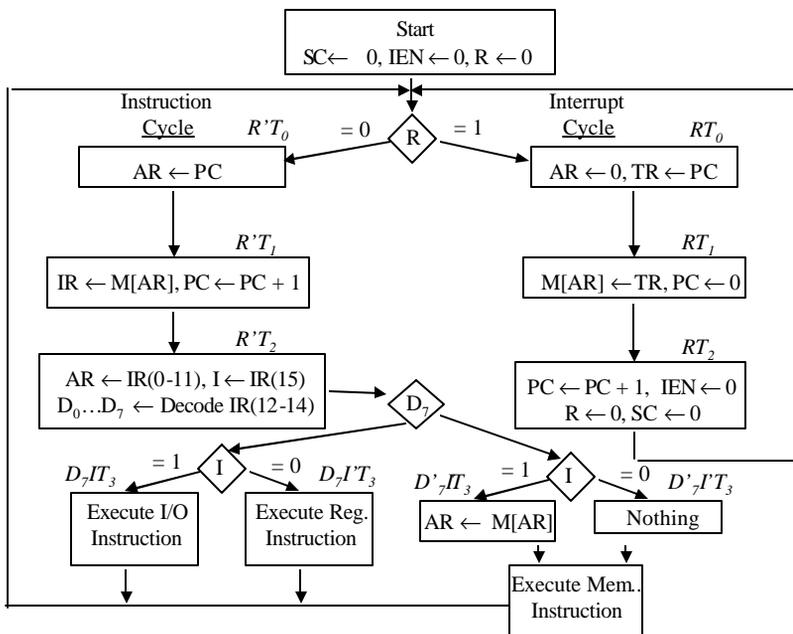
Flowchart For Interrupt Cycle



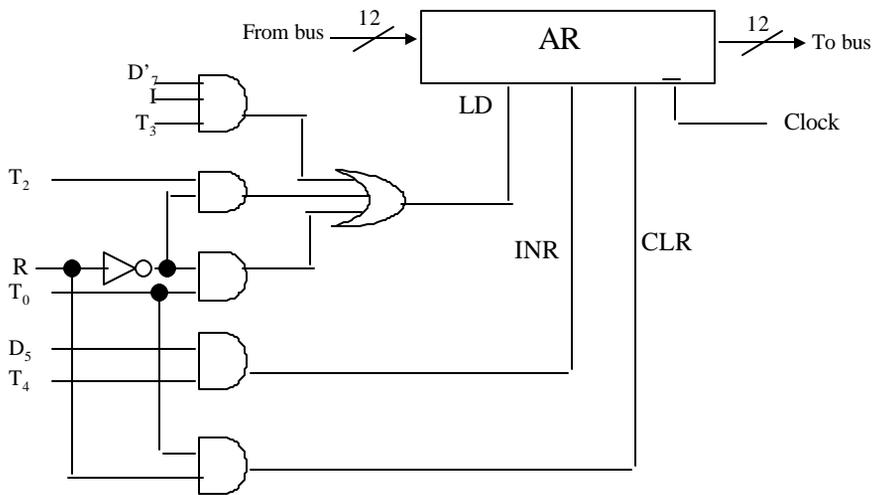
## Demonstration of the Interrupt Cycle



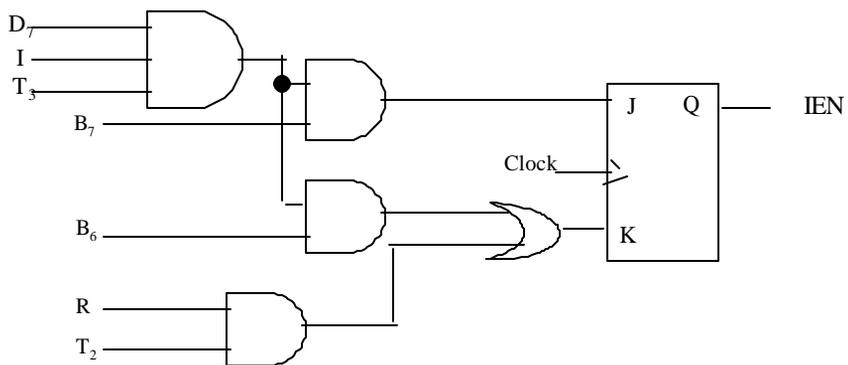
## Flowchart For Computer Operation



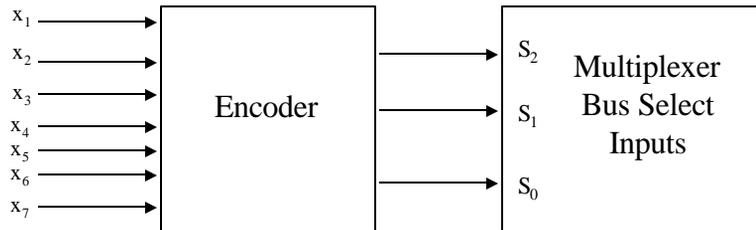
### Control Gates Associated With AR



### Control Inputs for IEN



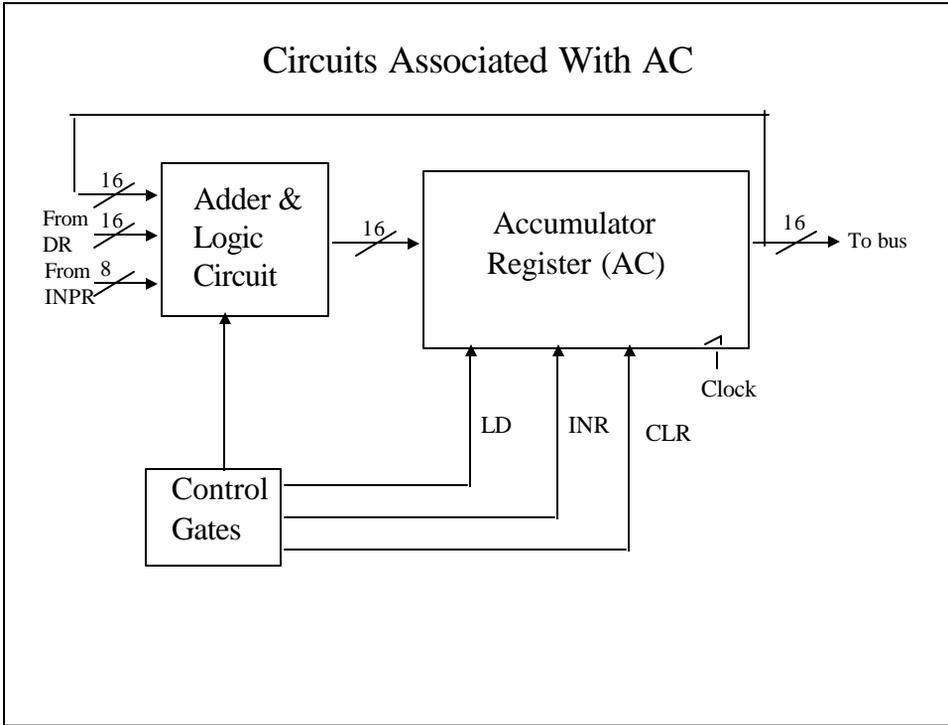
## Encoder for Bus Selection Inputs



## Encoder for Bus Selection Circuit

		<u>Inputs</u>					<u>Outputs</u>				
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$S_2$	$S_1$	$S_0$	Register Selected for Bus	
0	0	0	0	0	0	0	0	0	0	None	
1	0	0	0	0	0	0	0	0	1	AR	
0	1	0	0	0	0	0	0	1	0	PC	
0	0	1	0	0	0	0	0	1	1	DR	
0	0	0	1	0	0	0	1	0	0	AC	
0	0	0	0	1	0	0	1	0	1	IR	
0	0	0	0	0	1	0	1	1	0	TR	
0	0	0	0	0	0	1	1	1	1	Mem.	

## Circuits Associated With AC



## Gate structure for Controlling the LD, INR, CLR of AC

