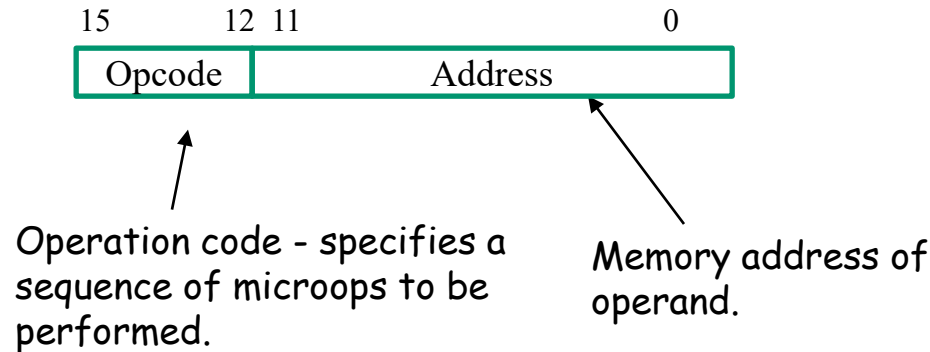
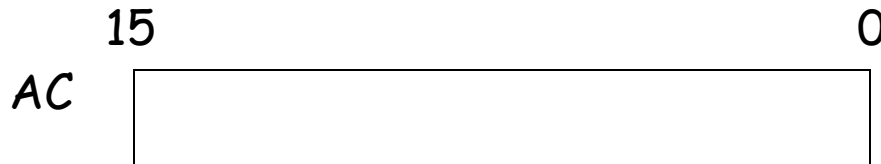


# Chapter 5: Basic Computer Organization and Design

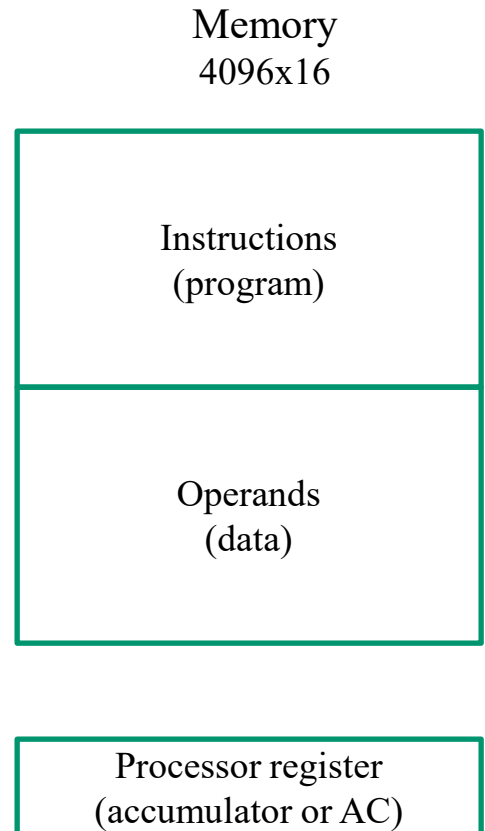
Very simple instruction code format.



Our computer has one processor register, called the accumulator, *AC*



Memory is used to hold the sequence of instructions (the program) and the operands (data).

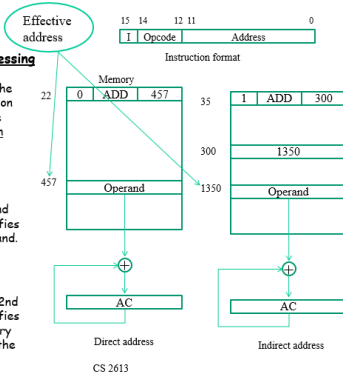


### Three types of addressing

1. **immediate operand:** the 2nd part of the instruction (bits 0-11) is actually the operand data (not used in this chapter)

2. **direct addressing:** 2nd part of instruction specifies the address of the operand.

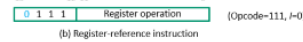
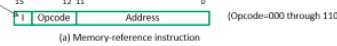
3. **indirect addressing:** 2nd part of instruction specifies the address in the memory in which the address of the operand is stored.



### Computer Instructions

Basic computer instruction formats.

I=0: Direct Address  
I=1: Indirect Address



CS 2613

### Control Unit:

- Two possible types of control:
  - hardwired (assumed in Chapter 5)
  - microprogrammed (covered in a later chapter)

From Figure 5-6, note that counter may not always count to 15, because it may be cleared.

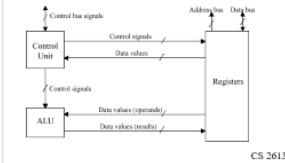


Figure 5-4 Basic computer system connected to a control bus.

### Instruction Cycle (four subcycles)

1. **Fetch** an instruction from memory
2. **Decode** the instruction
3. **Read** the effective **address** from memory (if indirect addressing is used)
4. **Execute** the instruction.

Click for CS 2613 Fetch/Decode/Execute Animation



### Register Instructions: Operations during T<sub>3</sub>

TABLE 5-3 Execution of Register-Reference Instructions  
 D, J, T<sub>3</sub> = r (common to all register-reference instructions)  
 R(r) = R (in R(R)-1) that specifies the operation)

CLA	rR <sub>0</sub>	AC ← 0	Clear AC
CLE	rR <sub>0</sub>	E ← 0	Clear E
CMA	rR <sub>0</sub>	AC ← ~AC	Complement AC
CME	rR <sub>0</sub>	E ← ~E	Complement E
CIR	rR <sub>0</sub>	AC ← shl AC, AC(15) ← E, E ← AC(15)	Circulate right
CIL	rR <sub>0</sub>	AC ← shr AC, AC(0) ← E, E ← AC(0)	Circulate left
INC	rR <sub>0</sub>	AC ← AC + 1	Increment AC
SPA	rR <sub>0</sub>	If (AC(15) = 0) then (PC ← PC + 1)	Skip if positive
SNA	rR <sub>0</sub>	If (AC(15) = 1) then (PC ← PC + 1)	Skip if negative
SZA	rR <sub>0</sub>	If (AC = 0) then (PC ← PC + 1)	Skip if AC zero
SZE	rR <sub>0</sub>	If (E = 0) then (PC ← PC + 1)	Skip if E zero
HLT	rR <sub>0</sub>	S ← 0 (S is a start-stop flip-flop)	Halt computer

(Sequence Counter)

Note that the entry r: SC ← 0 is not actually an instruction, but occurs with every register-reference instruction.

Q: How could the start-stop flip flop be used to stop the computer?

A: Could be ANDed with the control input to the sequence counter increment (INR) line.

CS 2613

### M-R Instructions: Operations during T<sub>4</sub> onwards

**Note:** Be careful when looking at Table 5-4 p. 145. Those are not technically RTL statements because they take more than one clock cycle.

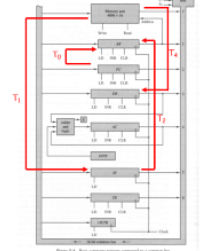
**AND**

$$D_0T_4: DR \leftarrow M[AR]$$

$$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$$

Note: There is an implied, "SC ← SC + 1" here.

T<sub>4</sub>: AR ← PC  
 T<sub>5</sub>: DR ← M[AR]  
 T<sub>6</sub>: SC ← 0  
 D<sub>1</sub>: T<sub>7</sub>: Nothing (NOP)  
 D<sub>2</sub>: T<sub>8</sub>: AR ← M[AR]



### I/O Instructions

I/O Device - a terminal with a keyboard (for input) and printer (for output).

- INPR and OUTR each hold eight bits
- Associated with INPR is a 1-bit flag, FGI.

When a key is struck on the keyboard, the 8-bit code is shifted into INPR, and FGI is set to 1 by the input device.

- **FGI=1 means data is in INPR** and is ready to be read.
- After transferring INPR into AC, computer should clear FGI.

When the **output device** is ready to receive an output character, it sets **FGO** to 1.

After loading data into OUTR, the computer clears FGO, and does not send more data until **output device sets FGO back to 1**.

CS 2613

### Interrupts

**Note:** In some applications, polling is necessary because there may not be other processing that can be done until input (or output) is performed.

However, in some cases interrupt control is more efficient (the I/O device interrupts the computer to let it know a transfer is ready).

When the **IEN** flag is set to 1, then the flags **FGI** and **FGO** interrupt the computer.

Gives the programmer control of whether to enable the interrupt facility.

### Design of Control Logic Gates (of Fig 5-16)

Inputs:  
 • 2 decoders  
 • 1 flip flop  
 • Bits 0-11 of IR  
 • Bits 0-15 of AC to check if AC=0 and sign (AC(15))  
 • Bits 0-15 of DR to check if DR=0  
 • 7 flip flops

Control Logic Gates

Outputs: Signals to control  
 • 9 registers  
 • R/W of memory  
 • Set, clear, complement FF's  
 • S<sub>0</sub>, S<sub>1</sub>, S<sub>2</sub> to select a register for bus  
 • AC adder and logic circuit



Effective  
address



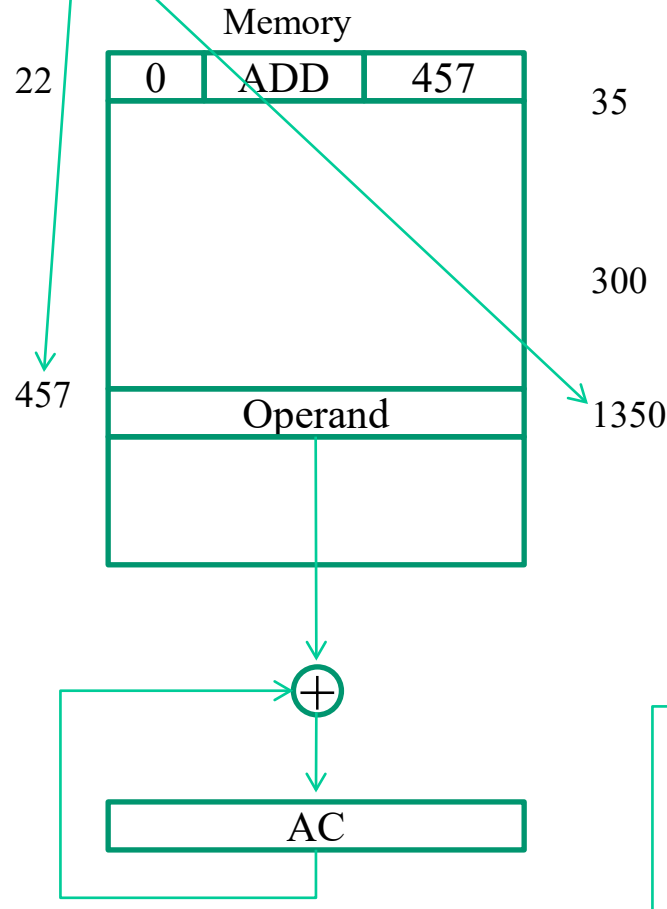
Instruction format

## Three types of addressing

1. *immediate operand*: the 2nd part of the instruction (bits 0-11) is actually the operand data (not used in this chapter)

2. *direct addressing*: 2nd part of instruction specifies the address of the operand.

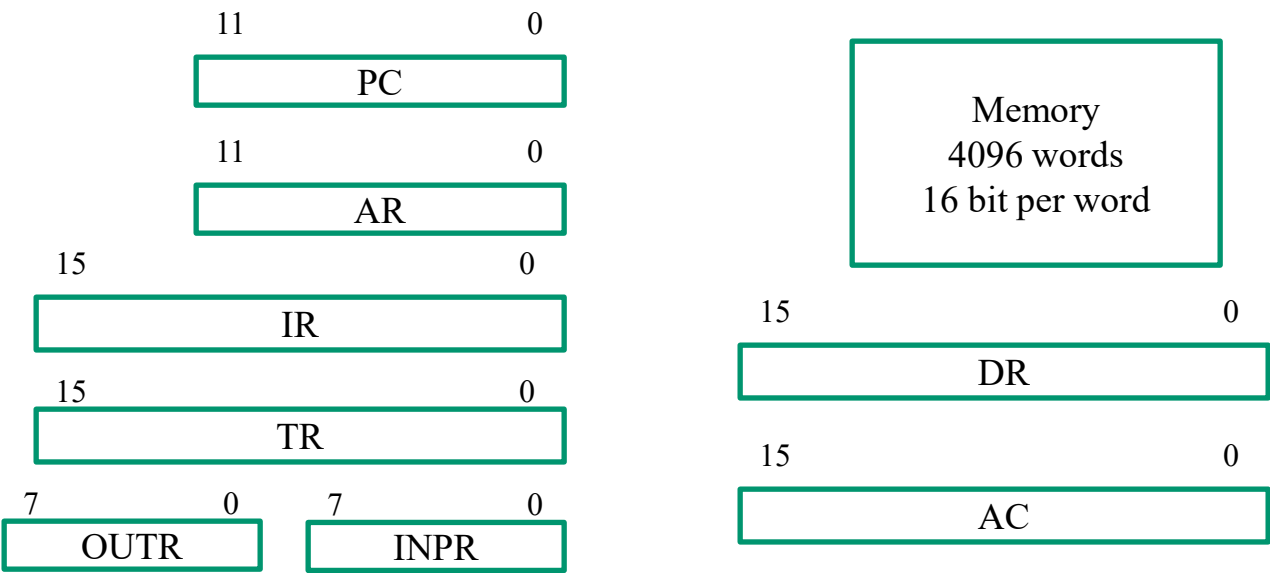
3. *indirect addressing*: 2nd part of instruction specifies the address in the memory in which the address of the operand is stored.



Direct address

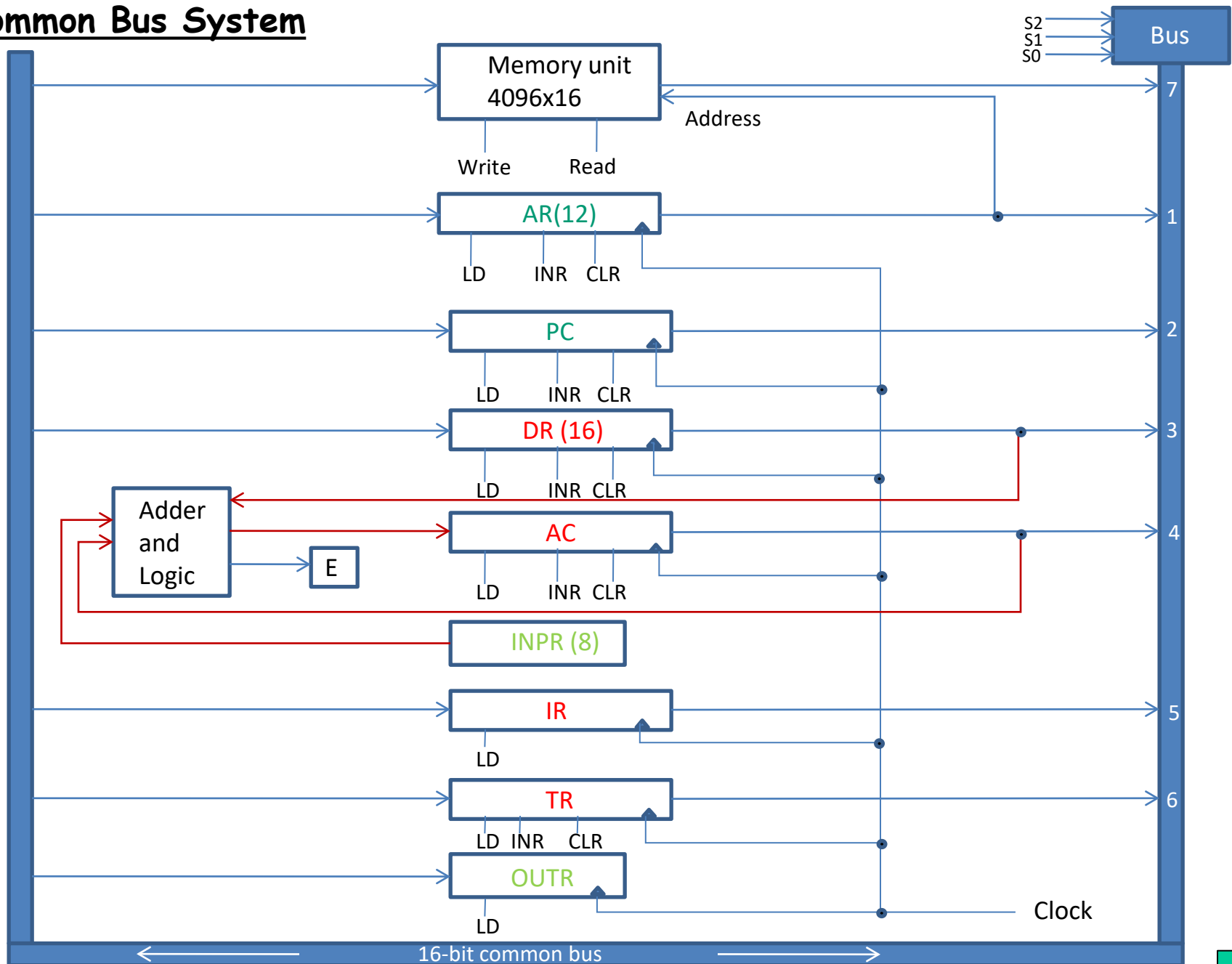
Indirect address

# Computer Registers



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



## Notes:

1. The bus could be implemented with MUXs or tri-state buffers.
2. The LD (load), INR, and CLR lines of the registers are independent (they are not tied together).
3. A "write" to the memory unit is analogous to loading a register.
4. DR and AC are used for arithmetic operations.
5. Any register can receive data from memory after a read, except AC and INPR.
6. The E bit is the "extended" AC bit (e.g., used for carry-out for addition).
7. AR and PC are only 12 bits. Why?

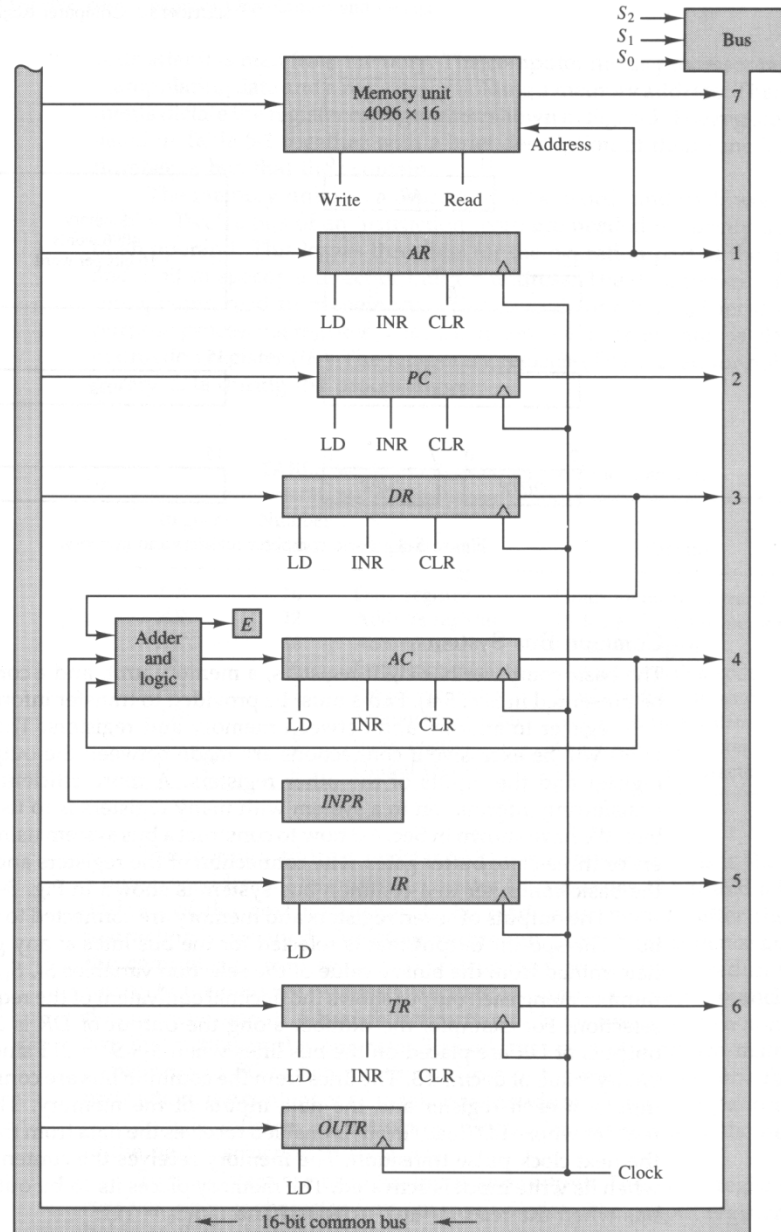


Figure 5-4 Basic computer registers connected to a common bus.



- 5-3. The following control inputs are active in the bus system shown in Fig. 5-4. For each case, specify the register transfer that will be executed during the next clock transition.

	$S_2$	$S_1$	$S_0$	LD of register	Memory	Adder
a.	1	1	1	<i>IR</i>	Read	—
b.	1	1	0	<i>PC</i>	—	—
c.	1	0	0	<i>DR</i>	Write	—
d.	0	0	0	<i>AC</i>	—	Add

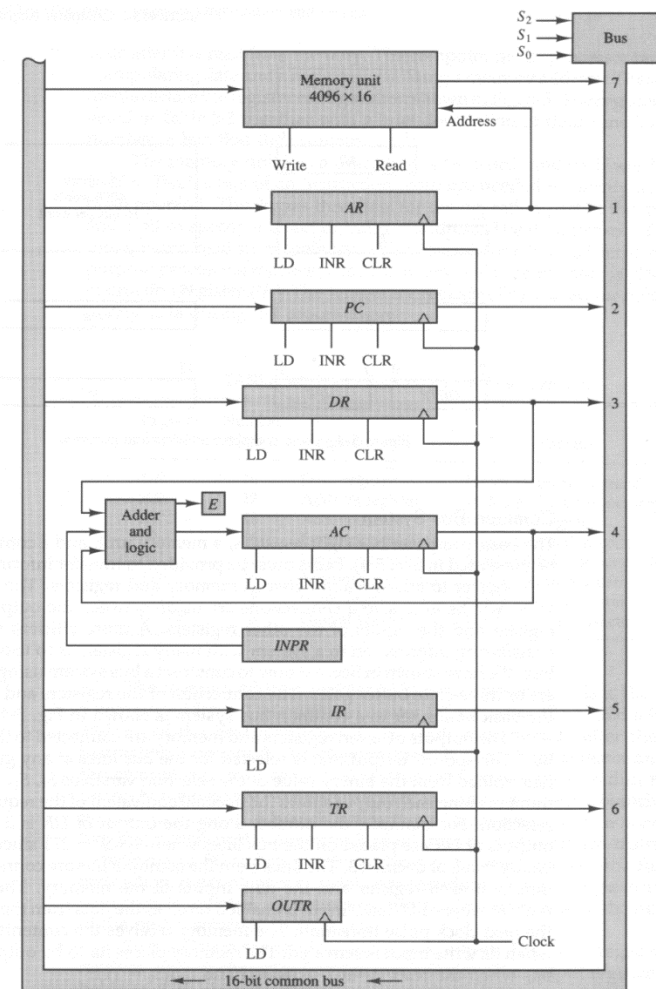
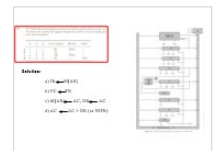


Figure 5-4 Basic computer registers connected to a common bus.



- 5-5. Explain why each of the following microoperations cannot be executed during a single clock pulse in the system shown in Fig. 5-4. Specify a sequence of microoperations that will perform the operation.
- $IR \leftarrow M[PC]$
  - $AC \leftarrow AC + TR$
  - $DR \leftarrow DR + AC$  (AC does not change)

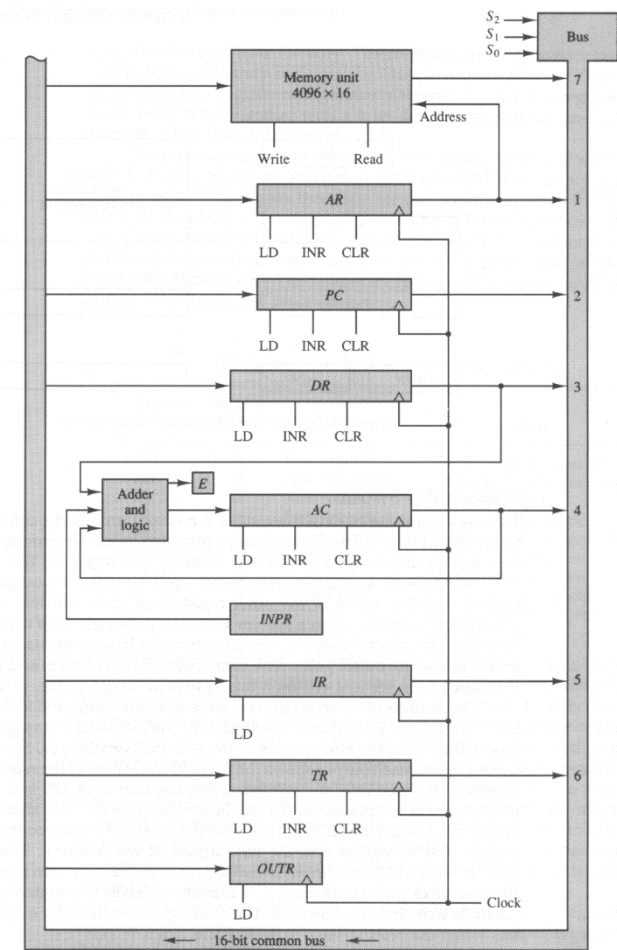


Figure 5-4 Basic computer registers connected to a common bus.





# Computer Instructions

Basic computer instruction formats.

I=0: Direct Address  
I=1: Indirect Address



(a) Memory-reference instruction



(b) Register-reference instruction



(c) Input-output instruction



15 14 ..... 12 11 ..... 0

I	Opcode	Address
0 1 1 1	Register operation	
1 1 1 1	I/O operation	

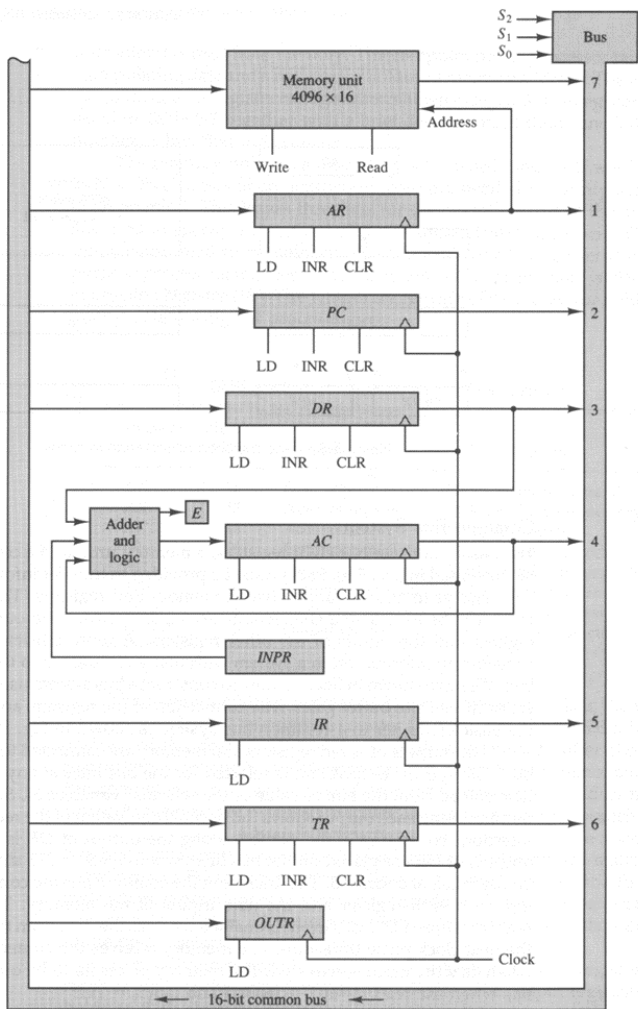


Figure 5-4 Basic computer registers connected to a common bus.

Symbol	Hexadecimal 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 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
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag



**Q:** How to decode the register-reference instructions.

**Q:** How to decode the I/O instructions.

**A:** Look closely at Table 5-2.

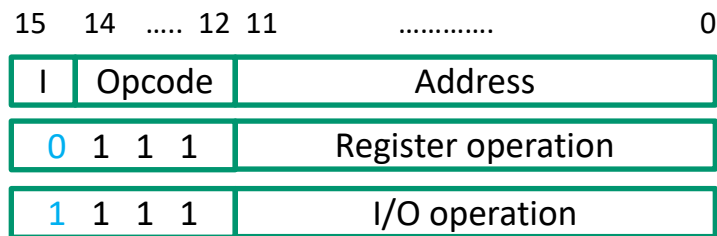
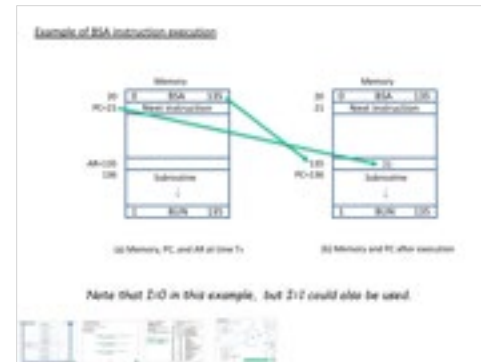


TABLE 5-2 Basic Computer Instructions

Symbol	Hexadecimal 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 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

# BUN and BSA



- 5-1.** A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.
- How many bits are there in the operation code, the register code part, and the address part?
  - Draw the instruction word format and indicate the number of bits in each part.
  - How many bits are there in the data and address inputs of the memory?



**5-6.** Consider the instruction formats of the basic computer shown in Fig. 5-5 and the list of instructions given in Table 5-2. For each of the following 16-bit instructions, give the equivalent four-digit hexadecimal code and explain in your own words what it is that the instruction is going to perform.

a. 0001 0000 0010 0100

b. 1011 0001 0010 0100

c. 0111 0000 0010 0000

TABLE 5-2 Basic Computer Instructions

Symbol	Hexadecimal 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 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



# I/O Instructions

I/O Device - a terminal with a keyboard (for input) and printer (for output).

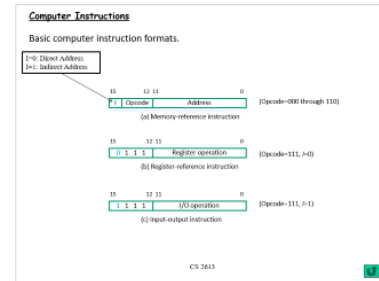
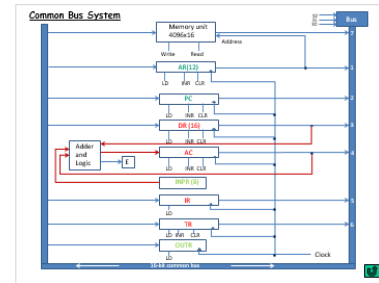
- INPR and OUTF each hold eight bits
- Associated with INPR is a 1-bit flag, FGI.

When a key is struck on the keyboard, the 8-bit code is shifted into INPR, and FGI is set to 1 by the input device.

- **FGI=1 means data is in INPR** and is ready to be read.
- After transferring INPR into AC, computer should clear FGI.

When the output device is ready to receive an output character, it sets FGO to 1.

After loading data into OUTF, the computer clears FGO, and does not send more data until **output device sets FGO back to 1**.

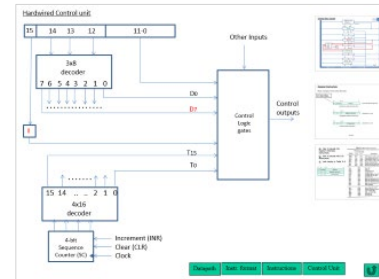


Q: How to decode the register-reference instructions.

Q: How to decode the I/O instructions.

A: Look closely at Table 5-2.

Hexadecimal code	Opcode	Description
0000	0000	Clear AC
0001	0001	Clear E
0010	0010	Complement AC
0011	0011	Complement E
0100	0100	Complement right AC and E
0101	0101	Increment AC
0110	0110	Skip next instruction if AC positive
0111	0111	Skip next instruction if AC negative
1000	1000	Input character to AC
1001	1001	Output character from AC
1010	1010	Skip on input flag
1011	1011	Skip on output flag
1100	1100	Interrupt on
1101	1101	Interrupt off



# Input Output Instructions

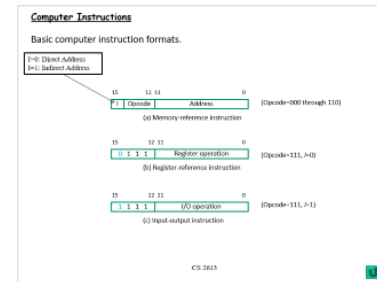
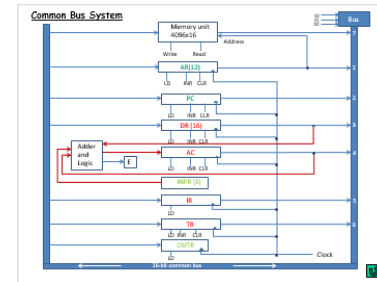
D7IT3 = p (common to all input output instructions)  
 IR(i) = Bi[bit in IR(6-11) that specifies the operation]

	p:	SC ← 0	Clear SC
INP	pB11	AC(0-7) ← INPR, FGI ← 0	Input character
OUT	pB10	OUTR ← AC(0-7), FGO ← 0	Output character
SKI	pB9	If(FGI=1) then (PC ← PC+1)	Skip on input flag
SKO	pB8	If(FGO=1) then (PC ← PC+1)	Skip on output flag
ION	pB7	IEN ← 1	Interrupt enable on
IOF	pB6	IEN ← 0	Interrupt enable off

## Typical programmed control transfer (polling)

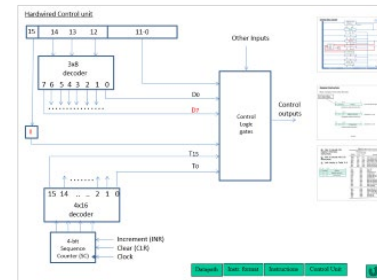
10	SKI
11	BUN 10
12	INP
...	
27	SKO
28	BUN 27
29	OUT

CS 2613



Q: How to decode the register-reference instructions.  
 Q: How to decode the I/O instructions.  
 A: Look closely at Table 5-2.

Symbol	Hexadecimal code	Description
AND	xxxx	AND memory word to AC
ASR	xxxx	ASR memory word to AC
LDN	xxxx	Load memory word to AC
STA	xxxx	Store content of AC to memory
BRN	xxxx	Branch nonconditionally
BRA	xxxx	Branch and save return address
DEC	xxxx	Decrement and skip if zero
CLA	7000	Clear AC
CLE	7000	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIN	7001	Conditionally right AC and E
CIL	7001	Conditionally left AC and E
INC	7001	Increment AC
SPA	7001	Skip next instruction if AC positive
SNA	7001	Skip next instruction if AC negative
SZA	7001	Skip next instruction if AC zero
SZC	7001	Skip next instruction if E is 0
HLT	7001	Halt program
DIP	F000	Input character to AC
OUT	F000	Output character from AC
SKI	F000	Skip on input flag
SKO	F000	Skip on output flag
ION	F000	Interrupt on
IOF	F000	Interrupt off





- 5-10. An instruction at address 021 in the basic computer has  $I = 0$ , an operation code of the AND instruction, and an address part equal to 083 (all numbers are in hexadecimal). The memory word at address 083 contains the operand B8F2 and the content of AC is A937. Go over the instruction cycle and determine the contents of the following registers at the end of the execute phase: PC, AR, DR, AC, and IR. Repeat the problem six more times starting with an operation code of another memory-reference instruction.



- 5-12.** The content of *PC* in the basic computer is 3AF (all numbers are in hexadecimal). The content of *AC* is 7EC3. The content of memory at address 3AF is 932E. The content of memory at address 32E is 09AC. The content of memory at address 9AC is 8B9F.
- What is the instruction that will be fetched and executed next?
  - Show the binary operation that will be performed in the *AC* when the instruction is executed.
  - Give the contents of registers *PC*, *AR*, *DR*, *AC*, and *IR* in hexadecimal and the values of *E*, *I*, and the sequence counter *SC* in binary at the end of the instruction cycle.



## Control Unit:

Two possible types of control:

- hardwired (assumed in Chapter 5)
- microprogrammed  
(covered in a later chapter)

From Figure 5-6, note that counter may not always count to 15, because it may be cleared.

## Control Unit

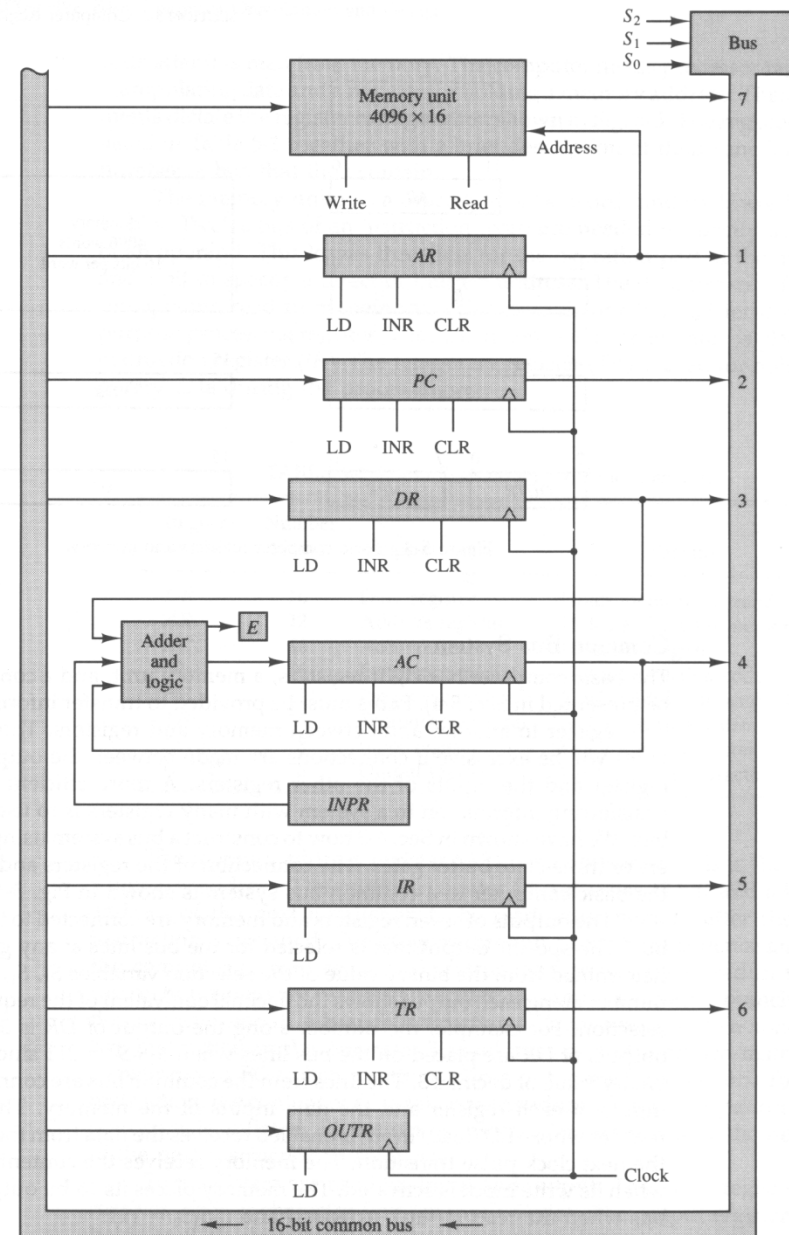
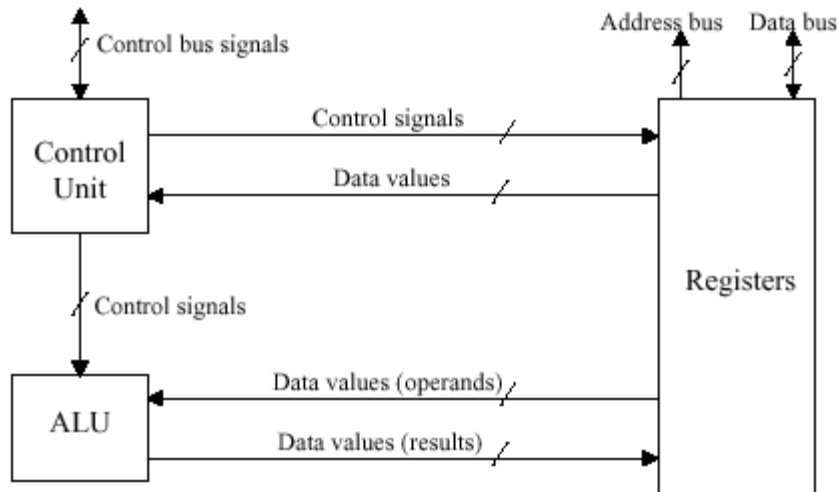
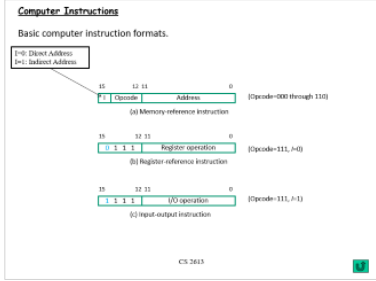
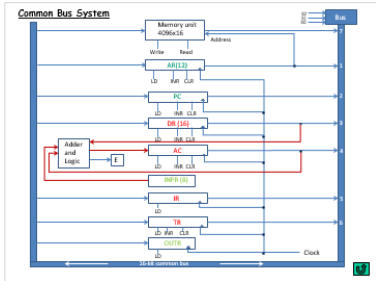
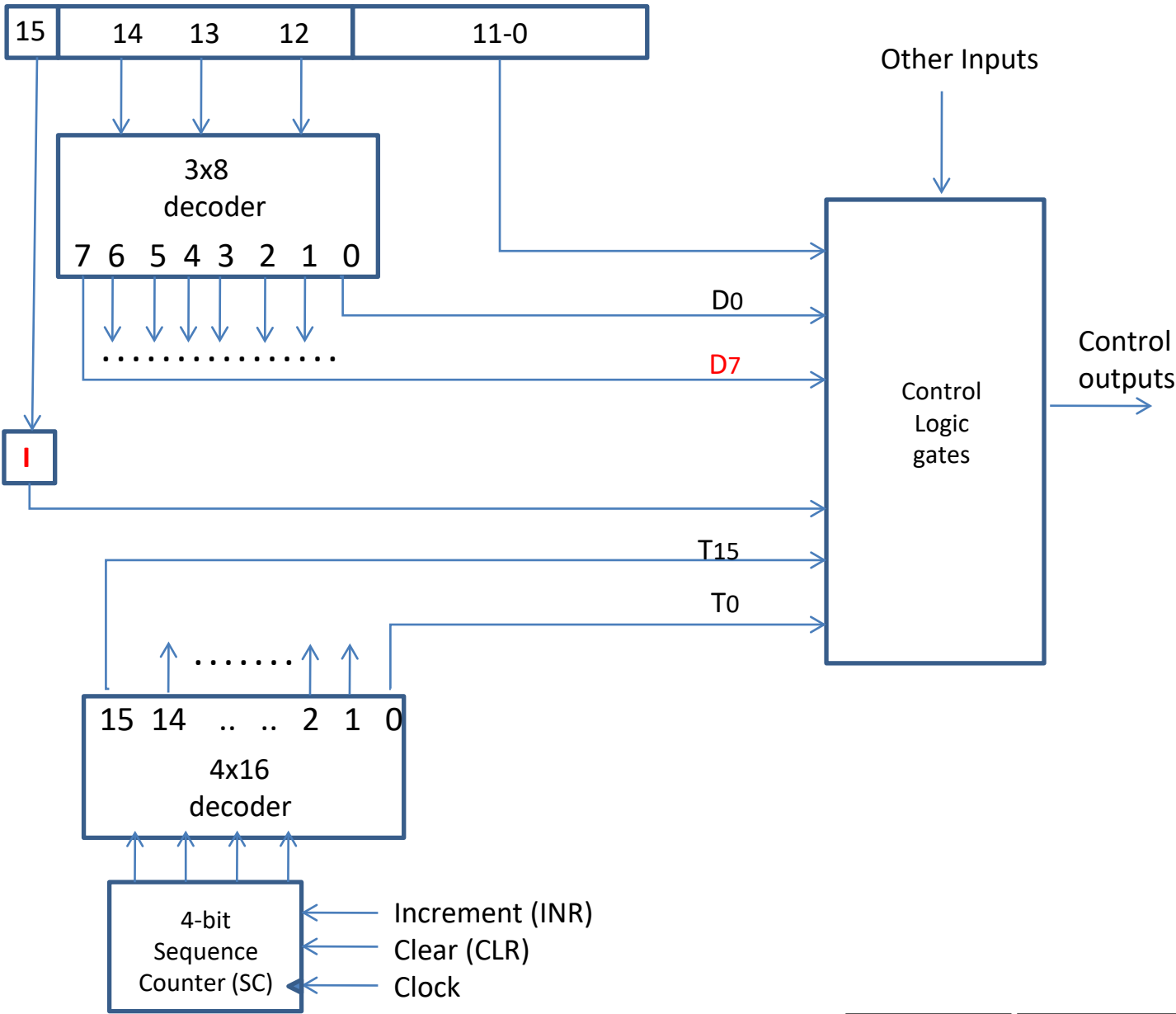


Figure 5-4 Basic computer registers connected to a common bus.

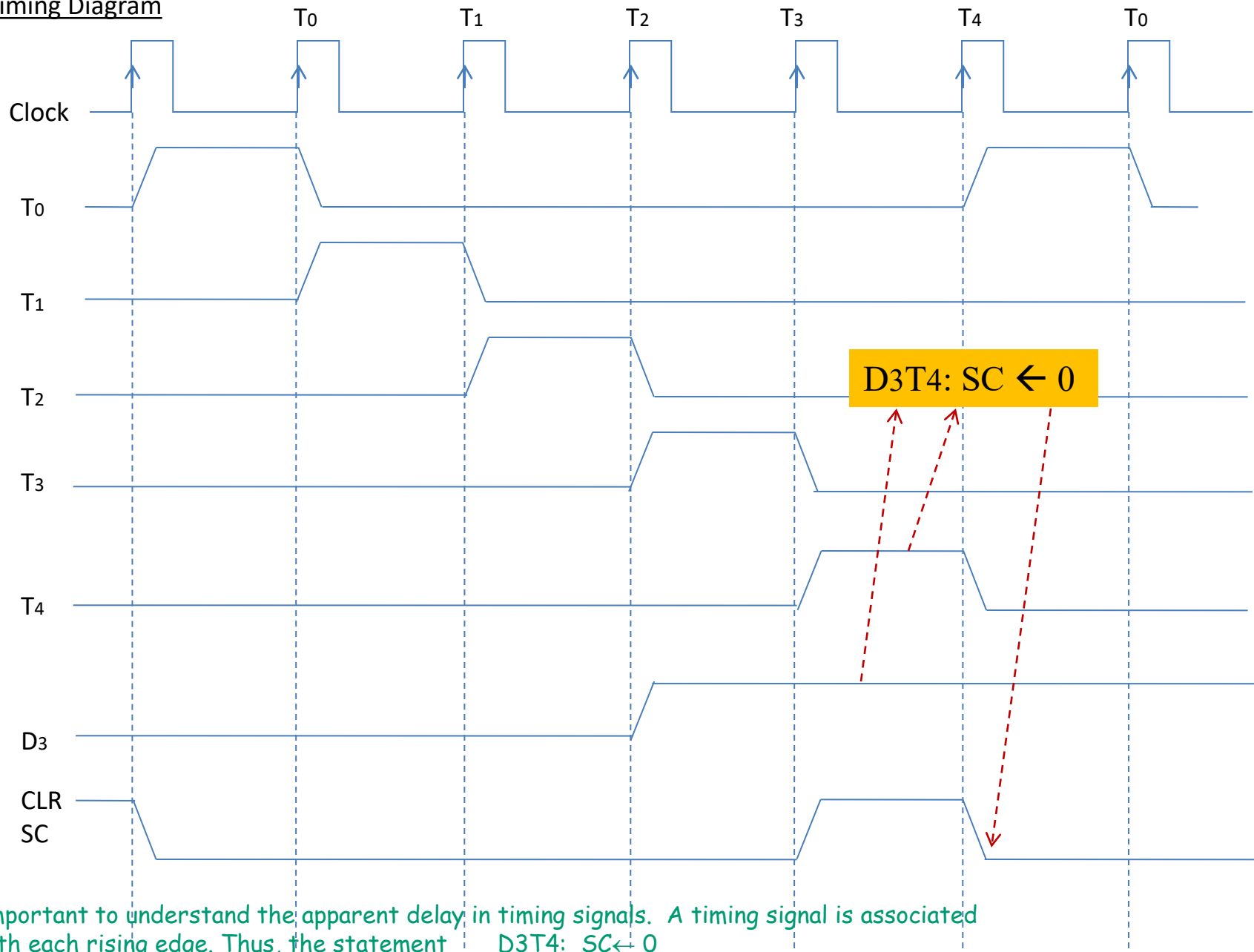
Hardwired Control unit



Q: How to decode the register-reference instructions.  
Q: How to decode the I/O instructions.  
A: Look closely at Table 5-2.

Symbol	$I = 0$	$I = 1$	Description
AND	xxxx	xxxx	AND memory word to AC
ASD	xxxx	xxxx	Add memory word to AC
LDA	xxxx	xxxx	Load memory word to AC
STA	xxxx	xxxx	Store content of AC to memory
BRN	xxxx	xxxx	Branch nonconditionally
BRA	xxxx	xxxx	Branch and save return address
DEC	xxxx	xxxx	Decrement and skip if zero
CLA	xxxx	xxxx	Clear AC
CLE	xxxx	xxxx	Clear E
CMA	xxxx	xxxx	Complement AC
CME	xxxx	xxxx	Complement E
CIR	xxxx	xxxx	Circle right AC and E
CIL	xxxx	xxxx	Circle left AC and E
INC	xxxx	xxxx	Increment AC
SPA	xxxx	xxxx	Skip next instruction if AC positive
SNA	xxxx	xxxx	Skip next instruction if AC negative
SZA	xxxx	xxxx	Skip next instruction if Z is 0
SZE	xxxx	xxxx	Skip next instruction if E is 0
HLT	xxxx	xxxx	Halt computer
DIP	xxxx	xxxx	Input character to AC
OUT	xxxx	xxxx	Output character from AC
SIF	xxxx	xxxx	Skip on input flag
SOF	xxxx	xxxx	Skip on output flag
IOF	xxxx	xxxx	Interrupt on
IOF	xxxx	xxxx	Interrupt off

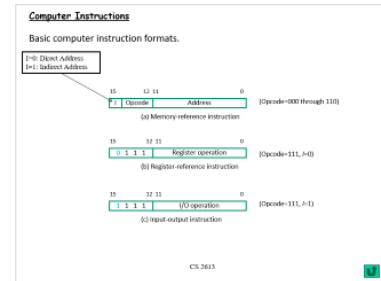
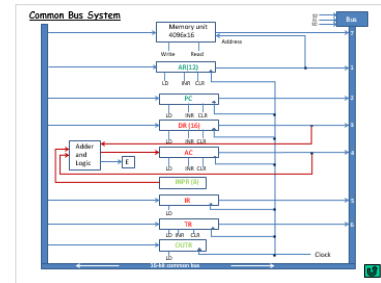
## Timing Diagram



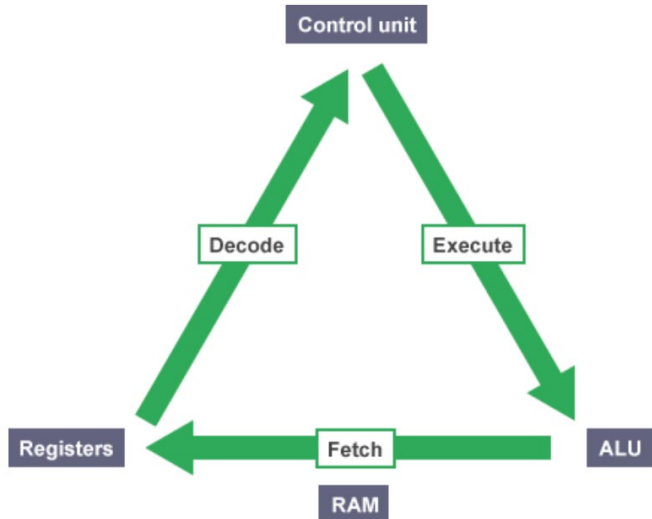
Important to understand the apparent delay in timing signals. A timing signal is associated with each rising edge. Thus, the statement **D3T4: SC  $\leftarrow$  0** is not performed at the beginning of the clock cycle where D3 and T4 are activated, but at the next rising edge

# Instruction Cycle (four subcycles)

1. **Fetch** an instruction from memory
2. **Decode** the instruction
3. Read the effective **address** from memory (if indirect addressing is used)
4. Execute the instruction.



[Click for CPU Fetch-Decode-Execute Animation](#)

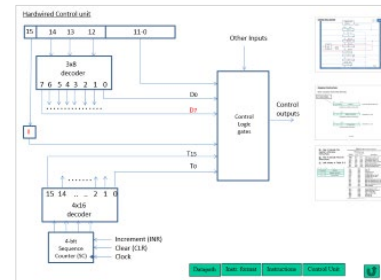


**Q: How to decode the register-reference instructions.**

**Q: How to decode the I/O instructions.**

**A: Look closely at Table 5-2.**

Symbol	I = 0	I = 1	Description
AND	xxxx	xxxx	AND memory word to AC
ASR	xxxx	xxxx	Arithmetic shift right
LDN	xxxx	xxxx	Load memory word to AC
STA	xxxx	xxxx	Store contents of AC to memory
BRN	xxxx	xxxx	Branch nonconditionally
BRA	xxxx	xxxx	Branch and save return address
DEC	xxxx	xxxx	Decrement and skip if zero
CLA	xxxx	xxxx	Clear AC
CLE	xxxx	xxxx	Clear E
CMA	xxxx	xxxx	Complement AC
CME	xxxx	xxxx	Complement E
CIR	xxxx	xxxx	Circle right AC and E
CIL	xxxx	xxxx	Circle left AC and E
INC	xxxx	xxxx	Increment AC
INP	xxxx	xxxx	Input instruction if AC positive
SPA	xxxx	xxxx	Skip next instruction if AC positive
SNA	xxxx	xxxx	Skip next instruction if AC negative
SZA	xxxx	xxxx	Skip next instruction if Z is 0
SZE	xxxx	xxxx	Skip next instruction if E is 0
HLT	xxxx	xxxx	HALT computer
DIP	xxxx	xxxx	Input character to AC
OUT	xxxx	xxxx	Output character from AC
SKI	xxxx	xxxx	Skip on input flag
SNO	xxxx	xxxx	Skip on output flag
SON	xxxx	xxxx	Interrupt on
IOF	xxxx	xxxx	Interrupt off



# RTL specification for "fetch and load" phase:

This sequence is done for all instructions.

$$\left\{ \begin{array}{l} T_0: AR \leftarrow PC \\ T_1: IR \leftarrow M[AR], PC \leftarrow PC+1 \\ T_2: D_0 \dots D_7 \leftarrow \text{Decode IR (12-14)}, AR \leftarrow IR (0-11), I \leftarrow IR (15) \end{array} \right.$$

$SC \leftarrow SC + 1$  assumed for every microop  
 $SC \leftarrow 0$  shown explicitly.

**Q:** What happens on activation of timing signal  $T_3$ ?

**A:** Depends on the status of  $D_7$  and  $I$ .

If  $D_7 = 0$ , then a memory reference instruction is to be executed.

If  $D_7 = 1$ , then a register-reference or I/O instruction is to be executed.

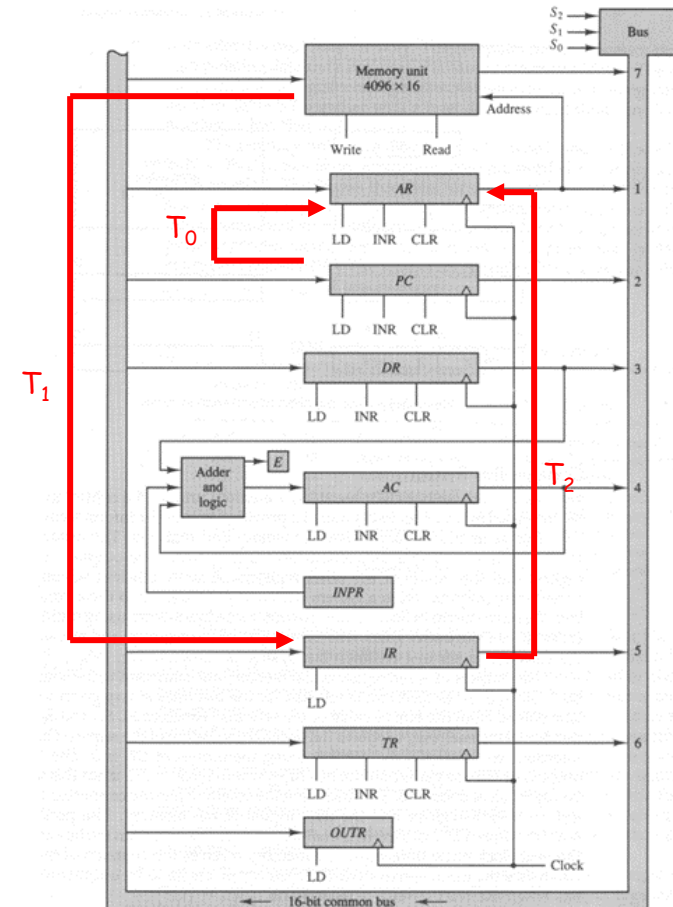
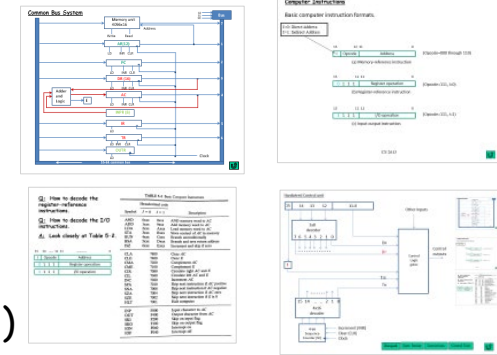


Figure 5-4 Basic computer registers connected to a common bus.

Datapath

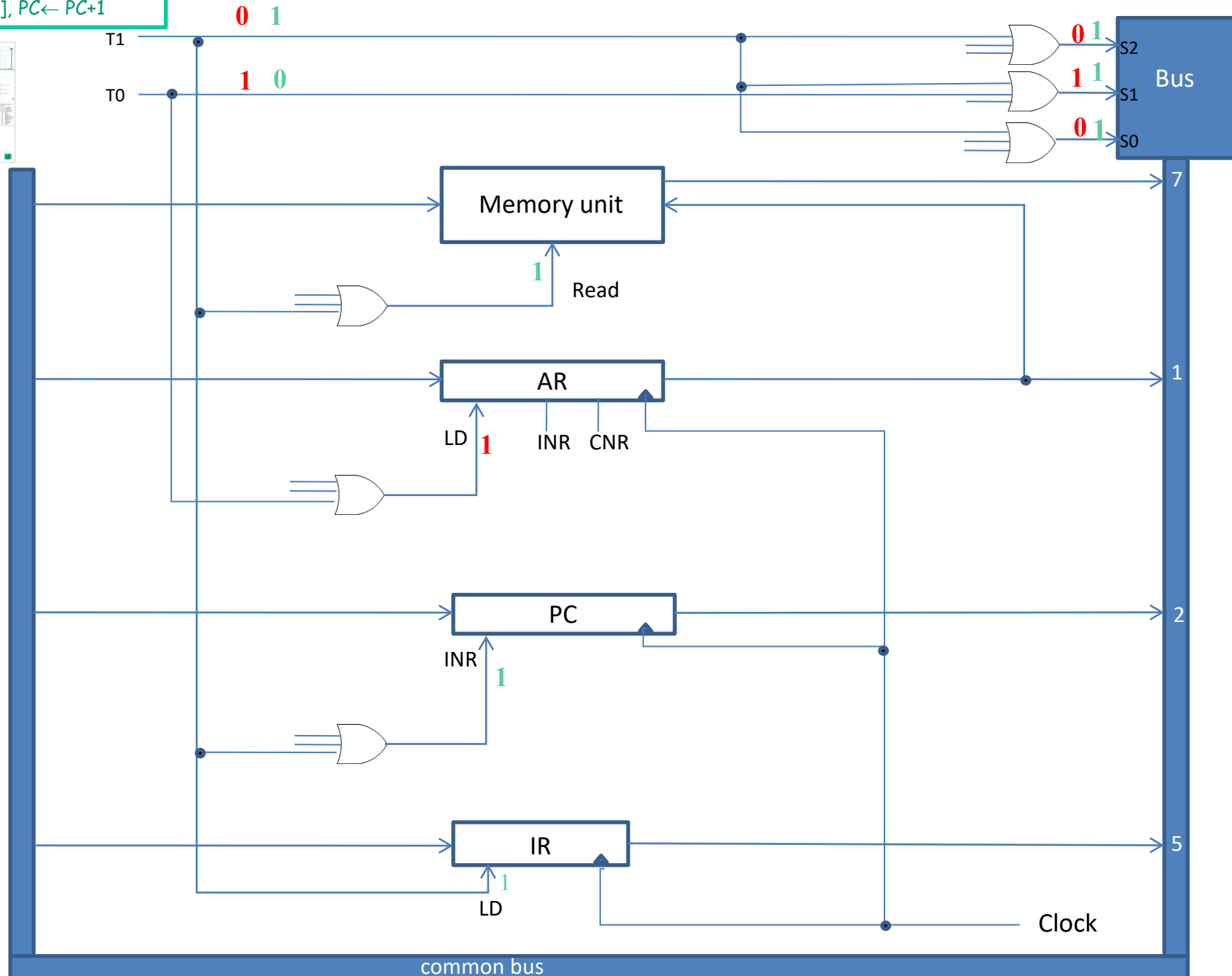
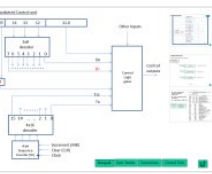
Instr. format

Instructions

Control Unit



Implementation of:  
 $T_0: AR \leftarrow PC$   
 $T_1: IR \leftarrow M[AR], PC \leftarrow PC+1$





### Four specific cases

Nearly the same, but indirect addressing needs one extra cycle to get the effective address.

$D_7' I' \Rightarrow$  Memory-reference with direct addressing

$D_7'I \Rightarrow$  Memory-reference with indirect addressing

$D_7I' \Rightarrow$  Register reference instruction

$$D_7I \Rightarrow \text{I/O instruction}$$

$D'_7 I' T_3$ : Nothing (NOP)

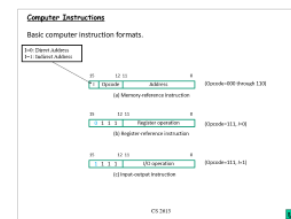
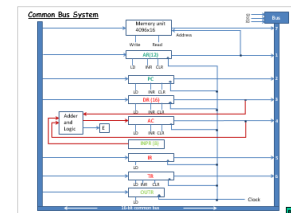
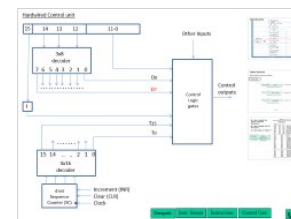
$$D'_7 \mid T_3: AR \leftarrow M[AR]$$

( $T_4$  will be used to start executing memory reference instructions).

**For other two cases, we can execute the appropriate instruction on  $T_3$**

$D_7 I' T_3$ : Start executing register-ref. instr.

$D_7 \mid T_3$ : Start executing I/O instr.

[illegible]

# Register Instructions: Operations during $T_3$

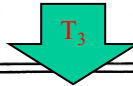


TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$  (common to all register-reference instructions)  
 $IR(i) = B_i$  [bit in  $IR(0-11)$  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

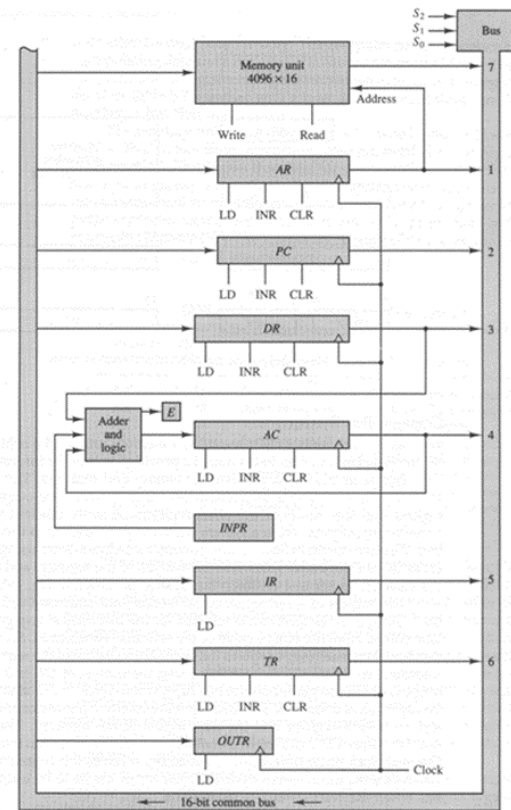
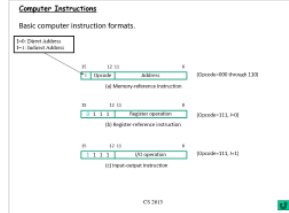
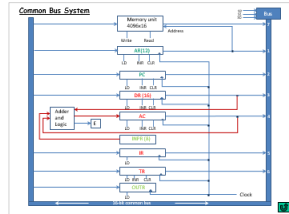


Figure 5-4 Basic computer registers connected to a common bus.

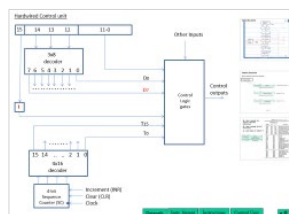


Q: How to decode the register-reference instructions.

Q: How to decode the I/O instructions.

A: Look closely at Table 5-2.

Register	Control	Description
CLP	Clear $PC$	
CLR	Clear $E$	
CMA	Complement $AC$	
CME	Complement $E$	
CIR	Circulate $AC$ and $E$	
CIL	Circulate $AC$ and $E$	
INC	Increment $AC$	
SPA	Skip next instruction if $AC(15)$ is 0	
SNA	Skip next instruction if $AC(15)$ is 1	
SZA	Skip next instruction if $AC$ is 0	
SZE	Skip next instruction if $E$ is 0	
HLT	Halt computer	



(Sequence Counter)

Note that the entry  $r: SC \leftarrow 0$  is not actually an instruction, but occurs with every register-reference instruction.

Q: How could the start-stop flip flop be used to stop the computer?

A: Could be ANDed with the control input to the sequence counter increment (INR) line.

## Alternate ways to specify "skip" instructions.

Assume AC register has a special "Z" bit to denote zero.

SPA  $rB_4AC(15)'$ :  $PC \leftarrow PC + 1$   
SNA  $rB_3AC(15)$ :  $PC \leftarrow PC + 1$   
SZA  $rB_2Z$ :  $PC \leftarrow PC + 1$   
SZE  $rB_1E$ :  $PC \leftarrow PC + 1$

# M-R Instructions: Operations during T<sub>4</sub> onwards

**Note:** Be careful when looking at Table 5-4 p. 145. Those are not technically RTL statements because they take more than one clock cycle.

**AND**

$D_0T_4: DR \leftarrow M[AR]$

$D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$

Note: There is an implied, "SC  $\leftarrow$  SC + 1" here.

T<sub>0</sub>: AR  $\leftarrow$  PC

T<sub>1</sub>: IR  $\leftarrow$  M[AR], PC  $\leftarrow$  PC + 1

T<sub>2</sub>: D<sub>0</sub>... D<sub>7</sub>  $\leftarrow$  Decode IR (12-14), AR  $\leftarrow$  IR (0-11), I  $\leftarrow$  IR (15)

D<sub>7</sub>' I' T<sub>3</sub>: Nothing (NOP)

D<sub>7</sub>' I T<sub>3</sub>: AR  $\leftarrow$  M[AR]

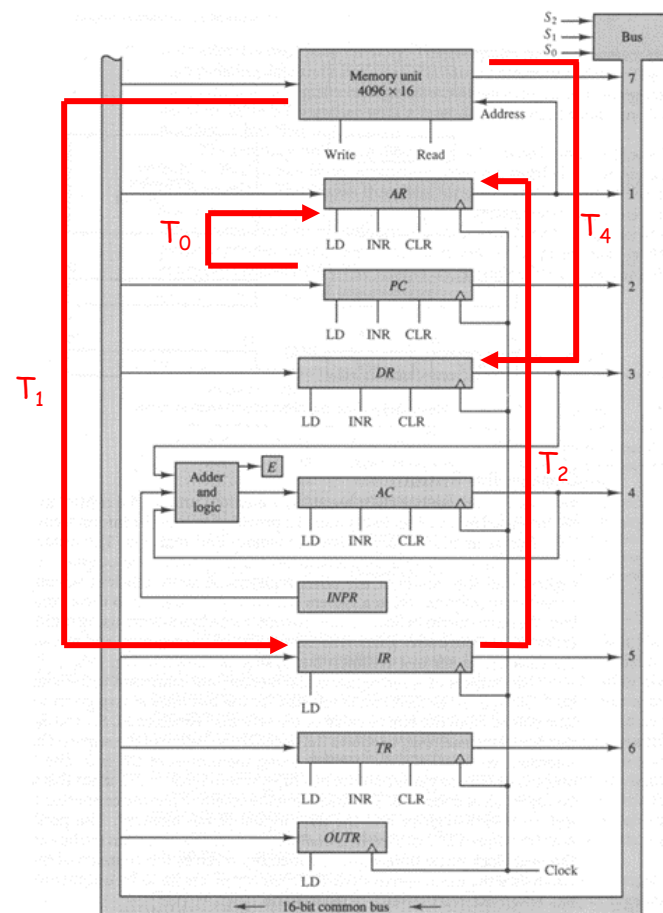


Figure 5-4 Basic computer registers connected to a common bus.

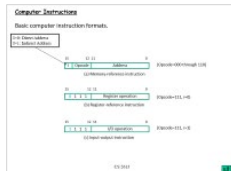
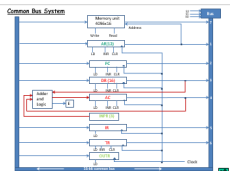
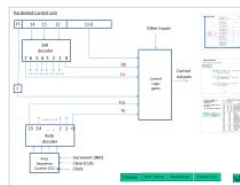


TABLE 5-4 Basic computer instructions

Instruction	Operation	Comments
LD	Load	Load register with data from memory
ST	Store	Store register data to memory
INR	Increment	Increment register by 1
CLR	Clear	Clear register to 0
ADD	Add	Add register to accumulator
SUB	Subtract	Subtract register from accumulator
MUL	Multiply	Multiply register by accumulator
DIV	Divide	Divide register by accumulator
AND	AND	AND register with accumulator
OR	OR	OR register with accumulator
XOR	XOR	XOR register with accumulator
NOT	NOT	NOT register
JMP	Jump	Jump to address in register
JZ	Jump if zero	Jump if accumulator is zero
JNZ	Jump if not zero	Jump if accumulator is not zero
JPOS	Jump if positive	Jump if accumulator is positive
JNPOS	Jump if not positive	Jump if accumulator is not positive
JPL	Jump if less than or equal	Jump if register is less than or equal to accumulator
JNPL	Jump if not less than or equal	Jump if register is not less than or equal to accumulator
JGT	Jump if greater than	Jump if register is greater than accumulator
JNGT	Jump if not greater than	Jump if register is not greater than accumulator
JGE	Jump if greater than or equal	Jump if register is greater than or equal to accumulator
JNGE	Jump if not greater than or equal	Jump if register is not greater than or equal to accumulator
JLE	Jump if less than	Jump if register is less than accumulator
JNLE	Jump if not less than	Jump if register is not less than accumulator
JG	Jump if greater	Jump if register is greater than accumulator
JNG	Jump if not greater	Jump if register is not greater than accumulator
JGE	Jump if greater than or equal	Jump if register is greater than or equal to accumulator
JNGE	Jump if not greater than or equal	Jump if register is not greater than or equal to accumulator
JLE	Jump if less than	Jump if register is less than accumulator
JNLE	Jump if not less than	Jump if register is not less than accumulator
JG	Jump if greater	Jump if register is greater than accumulator
JNG	Jump if not greater	Jump if register is not greater than accumulator



## ADD

$D_1T_4: DR \leftarrow M[AR]$

$D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$

## LDA (load to AC)

$D_2T_4: DR \leftarrow M[AR]$

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

## STA (store AC)

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

$T_0: AR \leftarrow PC$

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

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

$D_7I'T_3: \text{Nothing (NOP)}$

$D_7IT_3: AR \leftarrow M[AR]$

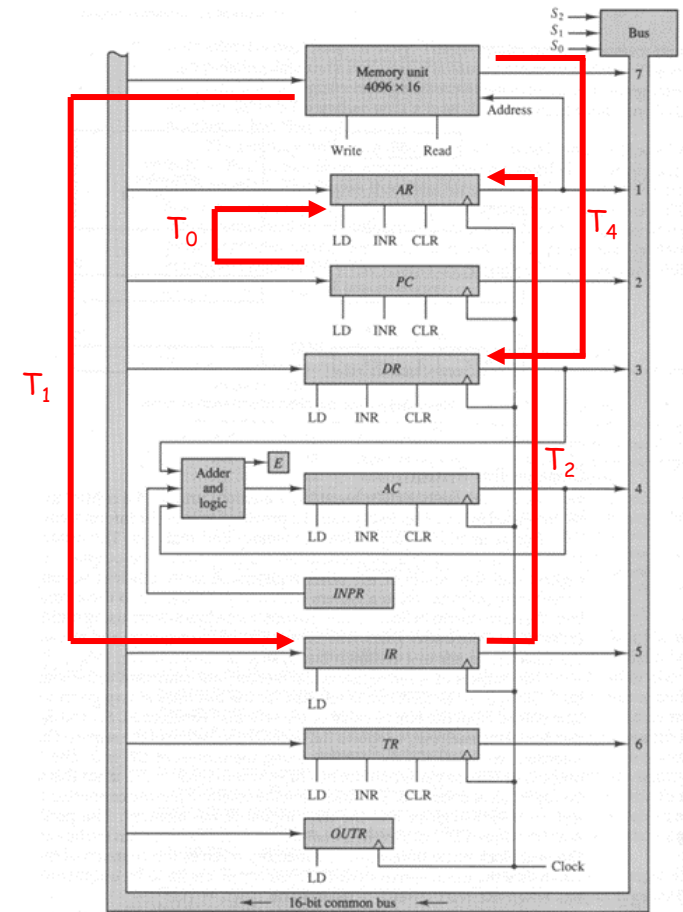


Figure 5-4 Basic computer registers connected to a common bus.



$T_0: AR \leftarrow PC$

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

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

$D_7' I' T_3: \text{Nothing (NOP)}$

$D_7' I T_3: AR \leftarrow M[AR]$

## BUN: (branch unconditionally)

Recall that indirect addressing could've been used.

$D_4 T_4: PC \leftarrow AR, \quad SC \leftarrow 0$

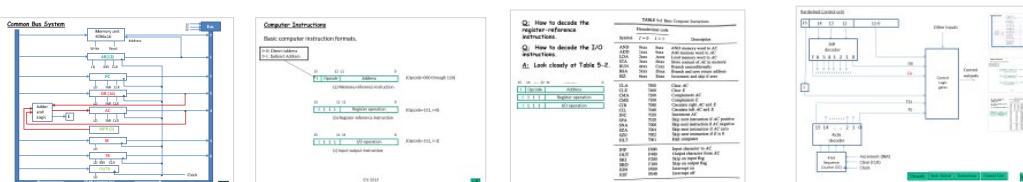
(Recall PC was already incremented at  $T_1$ .)

## BSA: (Branch and Save Return Address)

$D_5 T_4: M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$

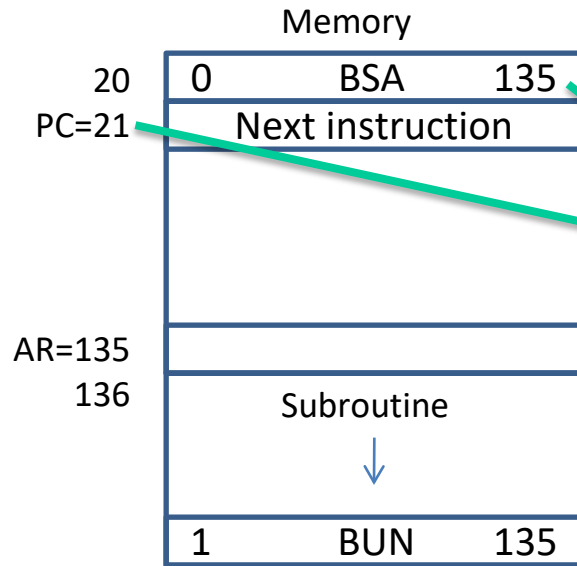
$D_5 T_5: PC \leftarrow AR, \quad SC \leftarrow 0$

Similar to Function Call

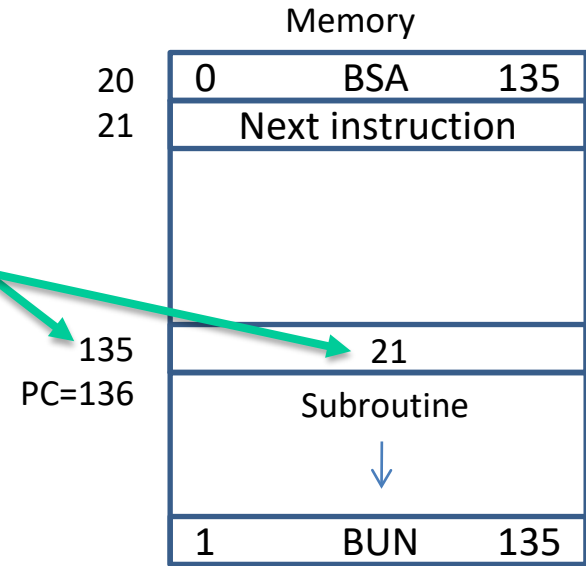




## Example of BSA instruction execution

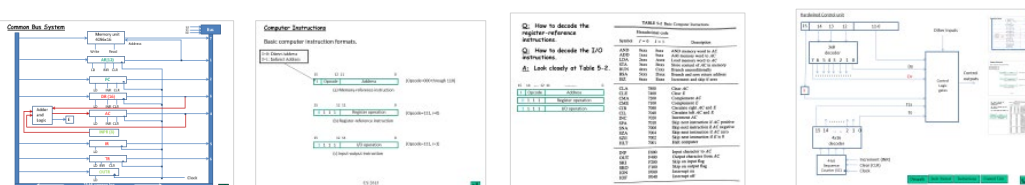


(a) Memory, PC, and AR at time T<sub>4</sub>



(b) Memory and PC after execution

*Note that  $I=0$  in this example, but  $I=1$  could also be used.*





### ISZ: (Increment and Skip if Zero)

$T_0: AR \leftarrow PC$

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

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

$$D_6T_4: \quad DR \leftarrow M[AR]$$
$$D_6T_5: \quad DR \leftarrow DR + 1$$
$$D_6T_6: \quad M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$$

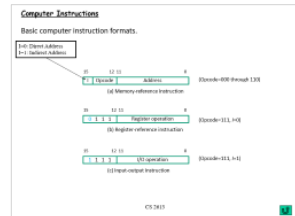
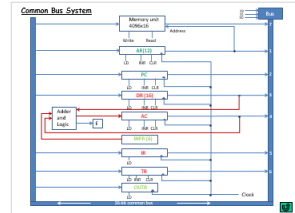
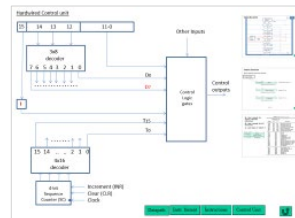
*Can be used to implement a "for" loop.*

Use a negative number (stored in memory) as the loop index. As the index is incremented, it will eventually equal to zero.

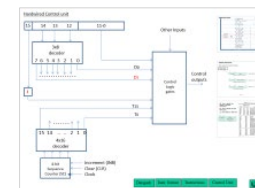
```

10      first instruction of loop body
      .....
20      ISZ 23
21      0 BUN 10
22      Next block of instructions
23      FFF0          ; Negative counter value

```

[illegible]

## Memory-reference instructions



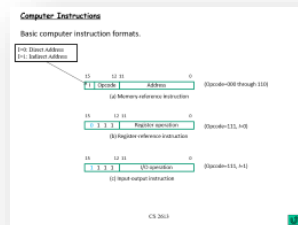
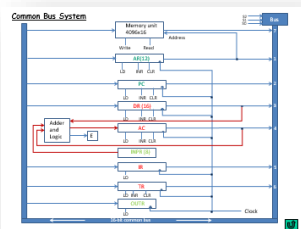
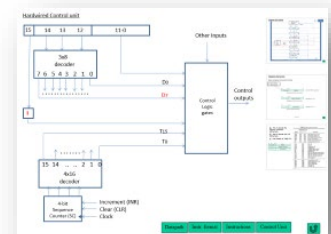
# Interrupts

**Note:** In some applications, polling is necessary because there may not be other processing that can be done until input (or output) is performed.

However, in some cases interrupt control is more efficient (the I/O device interrupts the computer to let it know a transfer is ready).

When the **IEN** flag is set to 1, then the flags **FGI** and **FGO** interrupt the computer.

Gives the programmer control of whether to enable the interrupt facility.

[illegible]

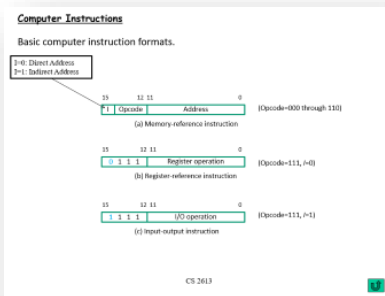
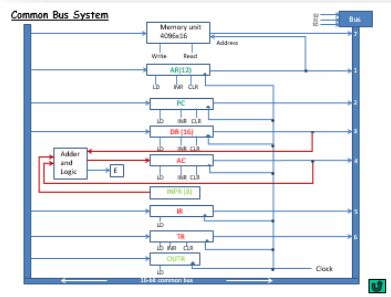
The **interrupt flag R** is set to 1 when either FGI or FGO is set AND the  $IEN = 1$ , so  $R = (IEN)(FGI + FGO) \dots$

...However, the setting of R is performed only during the “Execution Phase” of each instruction:

$$T'_0 T'_1 T'_2 (IEN)(FGI + FGO): R \leftarrow 1$$

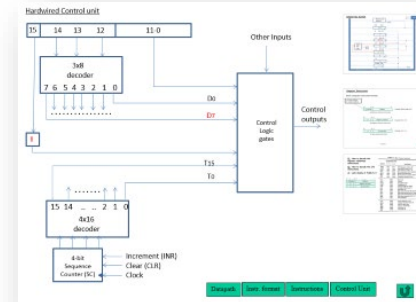
**Q:** Why not just put  $T_3$  here?

**A:** Recall that FGI and FGO are set asynchronously by the I/O device. Thus, above allows R to be set anytime during the execute cycle of an instruction.



**Q: How to decode the register-reference instructions.**  
**Q: How to decode the I/O instructions.**  
**A: Look closely at Table 5-2.**

Hexadecimal code	Symbol	Description
0000	CLA	Clear AC
0001	CLR	Clear E
0002	CMA	Complement AC
0003	CME	Complement E
0004	CIR	Circular right AC and E
0005	CIL	Circular left AC and E
0006	INC	Increment AC
0007	ISL	Skip next instruction if AC positive
0008	SNA	Skip next instruction if AC negative
0009	SZA	Skip next instruction if AC zero
0010	SZE	Skip next instruction if E is 0
0011	HLT	Halt computer
0012	INP	Input character to AC
0013	OUT	Output character from AC
0014	SIF	Stop on input flag
0015	SOF	Stop on output flag
0016	IOF	Interrupt on
0017	IOF	Interrupt off



## Interrupt Cycle

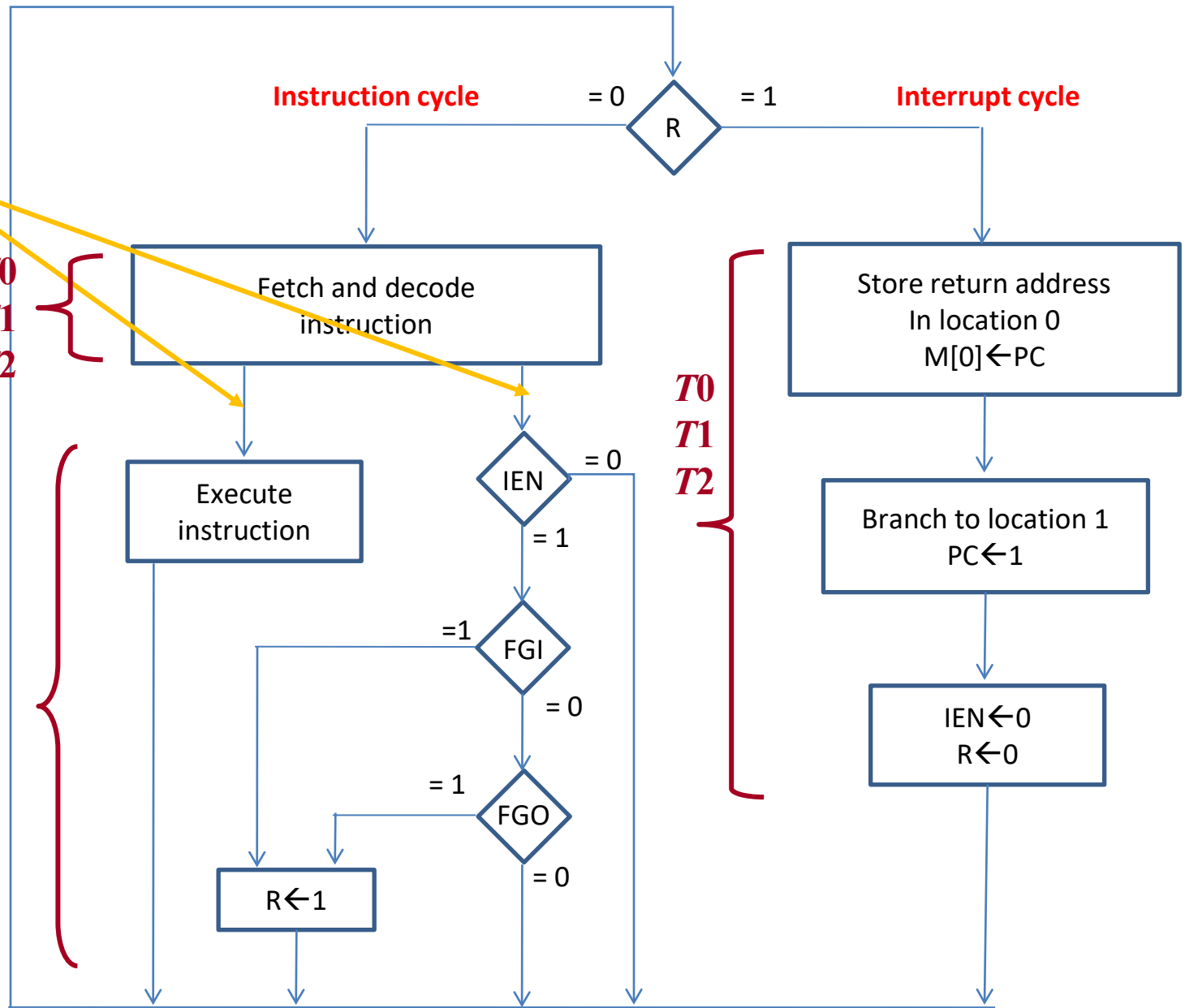
Both of these paths are activated at the same time (in parallel)

*T0*  
*T1*  
*T2*

*T3*  
*T4*  
...

Instruction cycle

Interrupt cycle



$T_0: AR \leftarrow PC$

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

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

$D'_7 I' T_3: \text{Nothing (NOP)}$

$D'_7 I T_3: AR \leftarrow M[AR]$

Control for fetch and decode microops must be modified...

*If  $R = 0$   
normal  
instruction  
cycle is  
performed.*

Must  
be added.

$R'T_0: AR \leftarrow PC$

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

$R'T_2: D_0 \dots D_7 \leftarrow \text{Decode } IR(12-14), AR \leftarrow IR(0-11), I \leftarrow IR(15)$

*If  $R = 1$ , interrupt cycle is  
performed, includes a  
modified fetch phase*

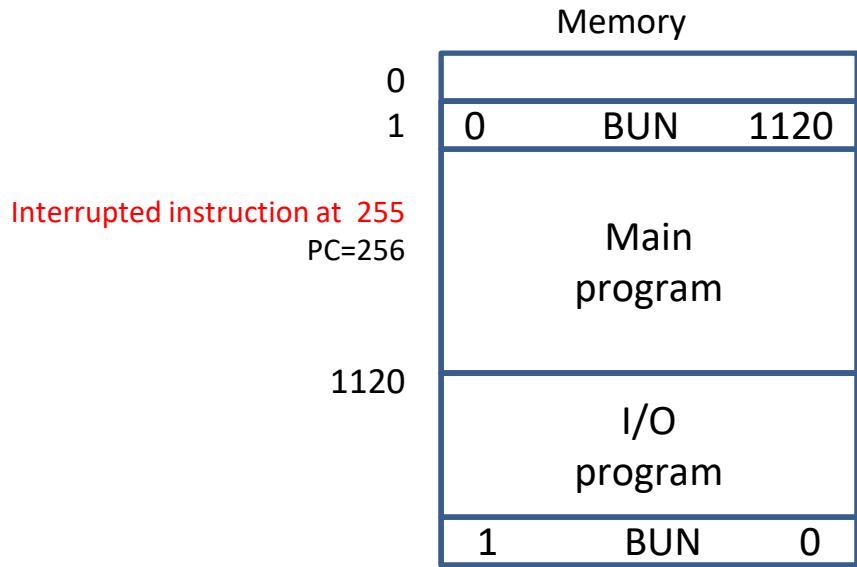
$RT_0: AR \leftarrow 0, TR \leftarrow PC$

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

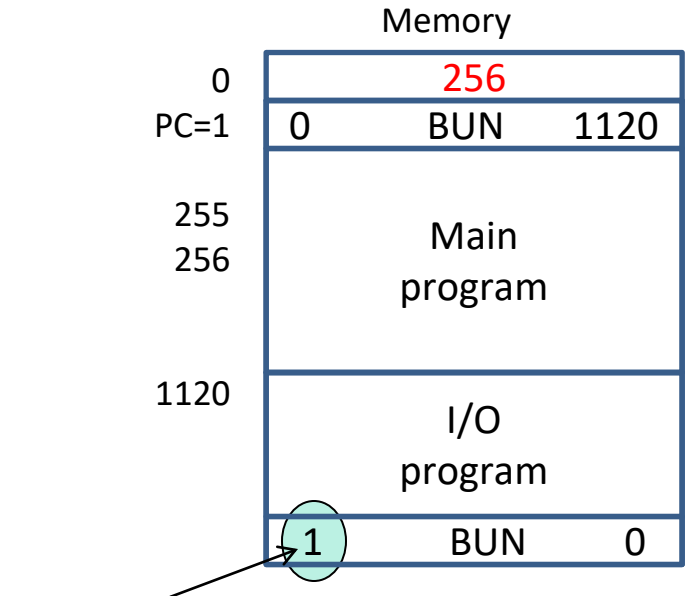
$RT_2: PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$

- Store PC in memory location 0
- Branch to location 1
- Clear IEN, R, and SC

# Demonstration of the interrupt cycle



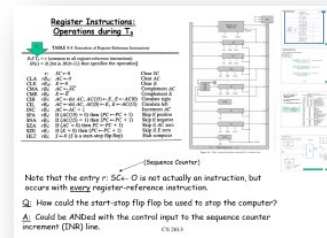
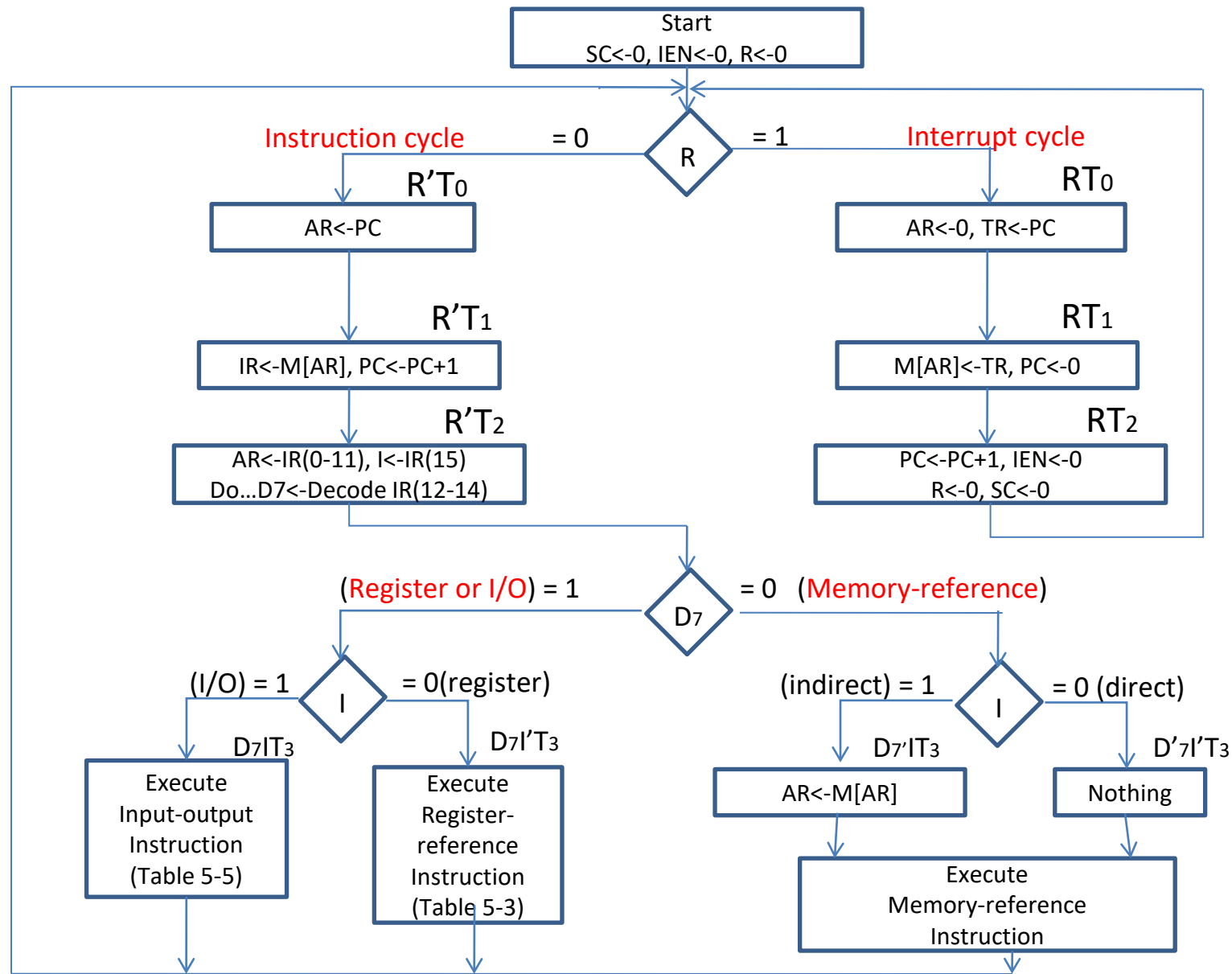
(a) Before interrupt



(b) after interrupt cycle



# Flowchart for complete computer operation





# Design of Control Logic Gates (of Fig 5-16)

## Inputs:

- 2 decoders
- 1 flip flop
- Bits 0-11 of IR
- Bits 0-15 of AC to check if AC=0 and sign (AC(15))
- Bits 0-15 of DR to check if DR=0
- 7 flip flops

## Control Logic Gates

## Outputs: Signals to control

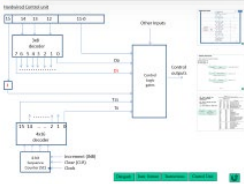
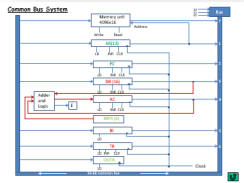
- 9 registers
- R/W of memory
- Set, clear, complement FFs
- S<sub>2</sub>, S<sub>1</sub>, S<sub>0</sub> to select a register for bus
- AC adder and logic circuit

Control functions and microoperations for the basic computer

Microps where AR is changed

TABLE 5-6 Control Functions and Microoperations for the Basic Computer

Fetch	$R'T_0: AR \leftarrow PC$ $R'T_1: IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2: D_0, \dots, D_7 \leftarrow \text{Decode } IR(12-14),$ $AR \leftarrow IR(0-11), I \leftarrow IR(15)$
Indirect	$D_4IT_3: AR \leftarrow M[AR]$
Interrupt:	$T_0T_1T_2(IEN)(FGI + FGO): R \leftarrow 1$ $RT_0: AR \leftarrow 0, TR \leftarrow PC$ $RT_1: M[AR] \leftarrow TR, PC \leftarrow 0$ $RT_2: PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-reference:	
AND	$D_0T_4: DR \leftarrow M[AR]$ $D_0T_5: AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	$D_1T_4: DR \leftarrow M[AR]$ $D_1T_5: AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	$D_2T_4: DR \leftarrow M[AR]$ $D_2T_5: AC \leftarrow DR, SC \leftarrow 0$
STA	$D_3T_4: M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	$D_4T_4: PC \leftarrow AR, SC \leftarrow 0$
BSA	$D_5T_4: M[AR] \leftarrow PC, AR \leftarrow AR + 1$ $D_5T_5: PC \leftarrow AR, SC \leftarrow 0$
ISZ	$D_6T_4: DR \leftarrow M[AR]$ $D_6T_5: DR \leftarrow DR + 1$ $D_6T_6: M[AR] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), SC \leftarrow 0$
Register-reference:	
	$D_7IT_3 = r$ (common to all register-reference instructions) $IR(i) = B_i$ ( $i = 0, 1, 2, \dots, 11$ ) $r: SC \leftarrow 0$ $rB_{11}: AC \leftarrow 0$ $rB_{10}: E \leftarrow 0$ $rB_9: AC \leftarrow \overline{AC}$ $rB_8: E \leftarrow \overline{E}$ $rB_7: AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$ $rB_6: AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$ $rB_5: AC \leftarrow AC + 1$ $rB_4: \text{ If } (AC(15) = 0) \text{ then } (PC \leftarrow PC + 1)$ $rB_3: \text{ If } (AC(15) = 1) \text{ then } (PC \leftarrow PC + 1)$ $rB_2: \text{ If } (AC = 0) \text{ then } PC \leftarrow PC + 1$ $rB_1: \text{ If } (E = 0) \text{ then } (PC \leftarrow PC + 1)$ $rB_0: S \leftarrow 0$
Input-output:	
	$D_7IT_3 = p$ (common to all input-output instructions) $IR(i) = B_i$ ( $i = 6, 7, 8, 9, 10, 11$ ) $p: SC \leftarrow 0$ $pB_{11}: AC(0-7) \leftarrow \text{INPR}, FGI \leftarrow 0$ $pB_{10}: \text{OUTR} \leftarrow AC(0-7), FGO \leftarrow 0$ $pB_9: \text{ If } (FGI = 1) \text{ then } (PC \leftarrow PC + 1)$ $pB_8: \text{ If } (FGO = 1) \text{ then } (PC \leftarrow PC + 1)$ $pB_7: IEN \leftarrow 1$ $pB_6: IEN \leftarrow 0$



# Control of Registers and Memory

Ex. Consider all microops for which the **content of AR** is changed.

$$R'T_0: AR \leftarrow PC$$

$$R'T_2: AR \leftarrow IR(0 - 11)$$

$$D_7'IT_3: AR \leftarrow M[AR]$$

$$RT_0: AR \leftarrow 0$$

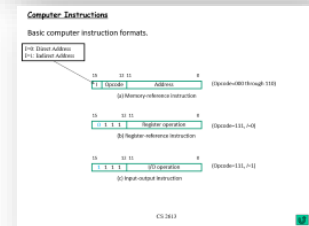
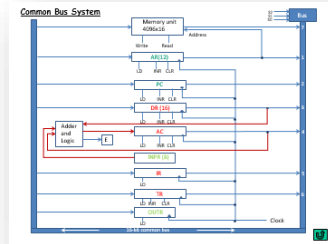
$$D_5T_4: AR \leftarrow AR + 1$$

So, the signals LD, CLR and INR (of AR) are controlled as follows:

$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$

$$CLR(AR) = RT_0$$

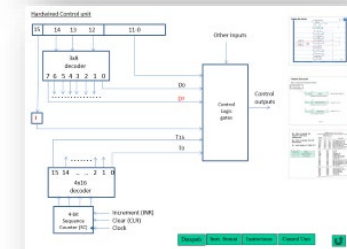
$$INR(AR) = D_5T_4$$



Q: How to decode the register-reference instructions.  
Q: How to decode the I/O instructions.  
A: Look closely at Table 5-2.

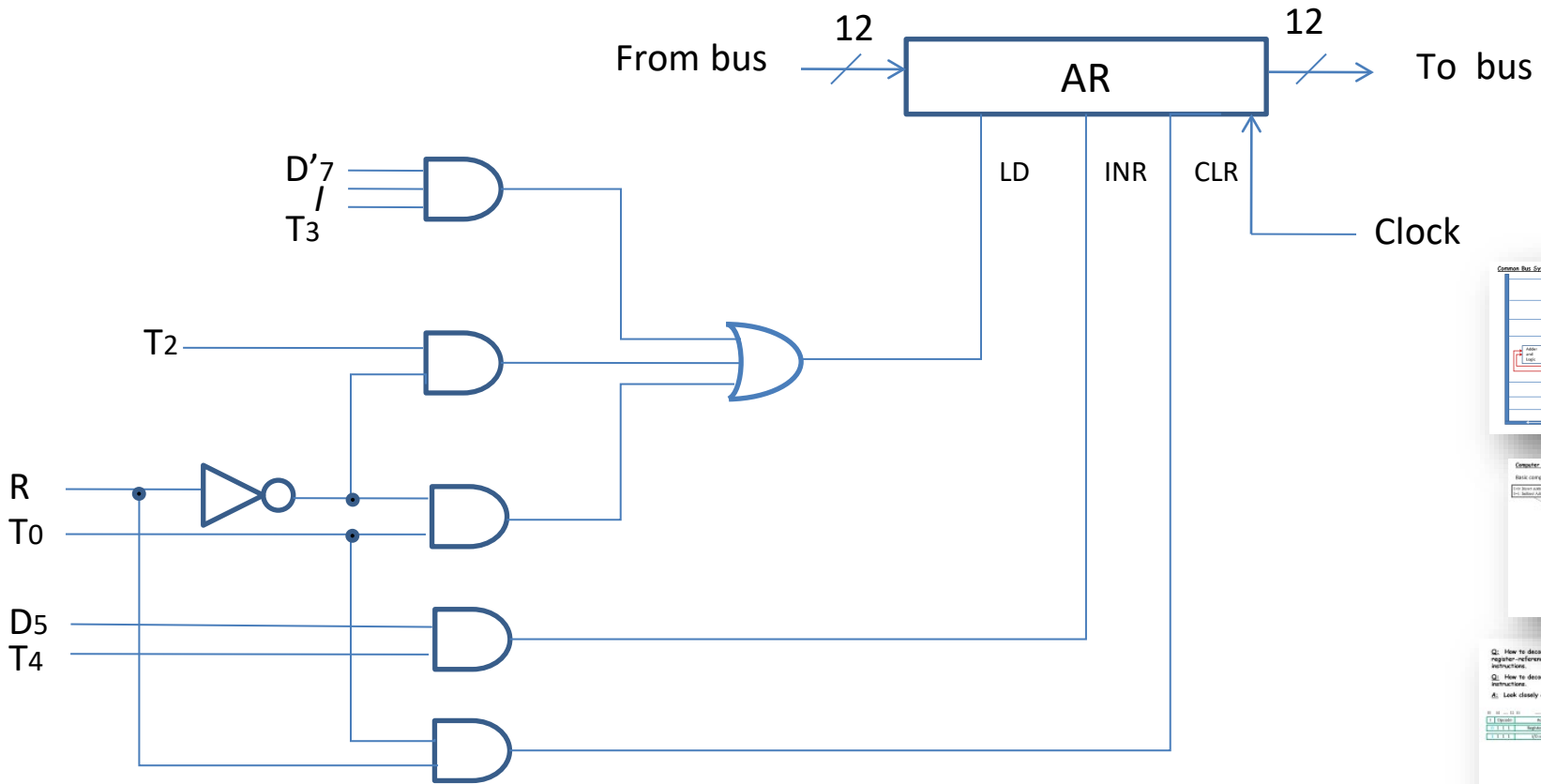
TABLE 5-2 Basic Computer Instructions

Symbol	Op	Op	Description
AND	0000	0000	AND memory word to AC
OR	0001	0000	OR memory word to AC
LDA	0010	0000	Load memory word to AC
STA	0011	0000	Store memory word to AC
BRA	0100	0000	Branch unconditionally
BRS	0101	0000	Branch and save return address
BEZ	0110	0000	Branch and save return address
CLR	0111	0000	Clear AC
CHS	1000	0000	Complement AC
CMR	1001	0000	Complement D
CLR	1010	0000	Clear right AC and D
CLL	1011	0000	Clear left AC and D
INC	1100	0000	Increment AC
DEC	1101	0000	Decrement AC
SFA	1110	0000	Stop next instruction if AC positive
SNA	1111	0000	Stop next instruction if AC negative
SZA	1000	0001	Stop next instruction if Z is 0
SNA	1001	0001	Stop next instruction if N is 1
SFV	1010	0001	Stop next instruction if V is 1
SFV	1011	0001	Stop next instruction if V is 1
INP	1100	0000	Input character to AC
OUT	1101	0000	Output character from AC
SKI	1110	0000	Skip on zero flag
SNO	1111	0000	Skip on negative flag
SKP	1000	0000	Skip on zero flag
SKN	1001	0000	Skip on negative flag
SKP	1010	0000	Skip on zero flag
SKN	1011	0000	Skip on negative flag



Similar logic design is performed for other registers and memory.

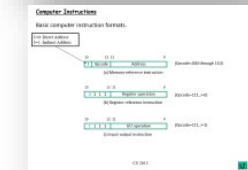
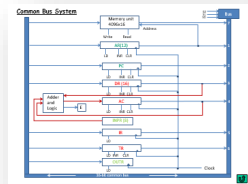
## Control gates associated with AR



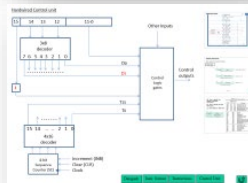
$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$

$$CLR(AR) = RT_0$$

$$INR(AR) = D_5 T_4$$



Hexadecimal value			Description
Symbol	Hex	I/O	
A0D	00	00	AD0 memory bus at 0
A0E	00	00	AD1 memory bus at 0
A0F	00	00	AD2 memory bus at 0
D0A	00	00	Local memory bus at 0
D0B	00	00	Local memory bus at 1
D0C	00	00	Local memory bus at 2
D0D	00	00	Local memory bus at 3
D0E	00	00	Local memory bus at 4
D0F	00	00	Local memory bus at 5
E0A	00	00	External memory bus at 0
E0B	00	00	External memory bus at 1
E0C	00	00	External memory bus at 2
E0D	00	00	External memory bus at 3
E0E	00	00	External memory bus at 4
E0F	00	00	External memory bus at 5
F0A	00	00	AD0 memory bus at 1
F0B	00	00	AD1 memory bus at 1
F0C	00	00	AD2 memory bus at 1
F0D	00	00	Local memory bus at 1
F0E	00	00	Local memory bus at 2
F0F	00	00	Local memory bus at 3
F10	00	00	Local memory bus at 4
F11	00	00	Local memory bus at 5
F12	00	00	Local memory bus at 6
F13	00	00	Local memory bus at 7
F14	00	00	Local memory bus at 8
F15	00	00	Local memory bus at 9
F16	00	00	Local memory bus at 10
F17	00	00	Local memory bus at 11
F18	00	00	Local memory bus at 12
F19	00	00	Local memory bus at 13
F1A	00	00	Local memory bus at 14
F1B	00	00	Local memory bus at 15
F1C	00	00	Local memory bus at 16
F1D	00	00	Local memory bus at 17
F1E	00	00	Local memory bus at 18
F1F	00	00	Local memory bus at 19



Control functions and misconfigurations for the base computer

**Table 1 1st Gen System with 16GB RAM in 8-Bay Config**

Item	Configuration
Processor	Intel® Xeon® E-2278 (28M Cache, up to 3.8 GHz, 10 cores)
Memory	16GB (4 x 4GB) DDR4-2400
Storage	256GB SATA 6Gbps (1 x 256GB SATA 6Gbps)
Network	2 x Intel® Ethernet® GbE ports (1 x RJ-45, 1 x SFP+)
Expansion	2 x PCIe 3.0 x16 slots (1 x full, 1 x half), 2 x PCIe 3.0 x4 slots (1 x full, 1 x half), 2 x PCIe 3.0 x1 slots (1 x full, 1 x half)
OS	Windows Server 2019 Standard Edition
Power	1500W Platinum Efficiency Power Supply
Form Factor	1U Rack Mountable

**Memory options**

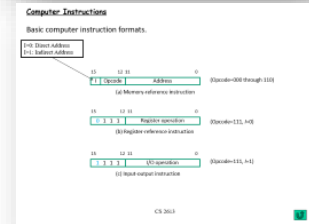
Table 1-1 shows a range of memory configurations that can be used with the system. The table lists the number of memory modules, the capacity of each module, and the total system memory. The table also lists the memory speed and the memory type.

**Disk options**

Table 1-2 shows a range of disk configurations that can be used with the system. The table lists the number of disks, the capacity of each disk, and the total system storage. The table also lists the disk type and the disk interface.

CS 2603

The diagram illustrates a Common Bus System. At the top, a 'Memory unit' is shown with an 'Address' input and a 'Data' output. Below it, an 'Address decoder' is connected to the 'Address' input of the memory unit. The decoder has a 'Select' input and an output labeled 'A[15:0]'. To the right, a 'Multiplexer' is connected to the 'Data' output of the memory unit. The multiplexer has a 'Select' input and an output labeled 'M[15:0]'. A 'Common Bus System' is represented by a horizontal line at the top. The 'Data' output of the memory unit and the 'M[15:0]' output of the multiplexer are connected to this bus. The 'A[15:0]' output of the address decoder is also connected to the bus. A 'Clock' signal is shown at the bottom right, connected to the clock inputs of both the address decoder and the multiplexer.

[illegible]

$$D_5T_5: PC \leftarrow AR$$

[illegible]

**5-15.** The memory unit of the basic computer shown in Fig. 5-3 is to be changed to a  $65,536 \times 16$  memory, requiring an address of 16 bits. The instruction format of a memory-reference instruction shown in Fig. 5-5(a) remains the same for  $I = 1$  (indirect address) with the address part of the instruction residing in positions 0 through 11. But when  $I = 0$  (direct address), the address of the instruction is given by the 16 bits in the next word following the instruction. Modify the microoperations during time  $T_2$ ,  $T_3$ , (and  $T_4$  if necessary) to conform with this configuration.

$65,536 \times 16$  memory = 16 bits

