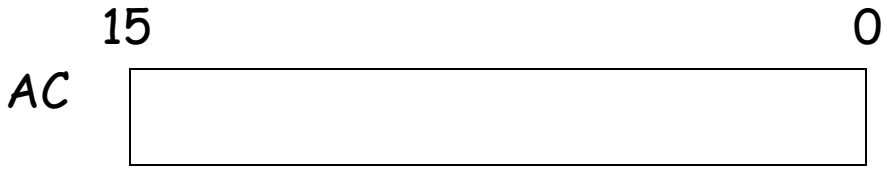
Chapter 5: Basic Computer Organization and Design

Very simple instruction code format.

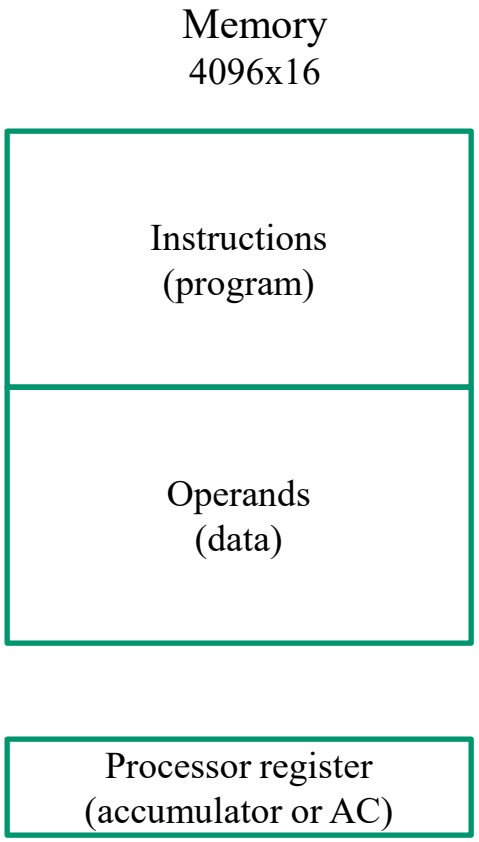
Operation code - specifies a sequence of microops to be performed.

Memory address of operand.

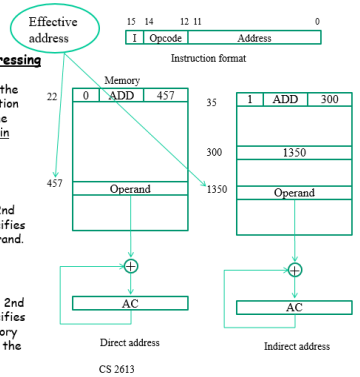
Our computer has one processor register, called the accumulator, *AC*



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



Three types of addressing



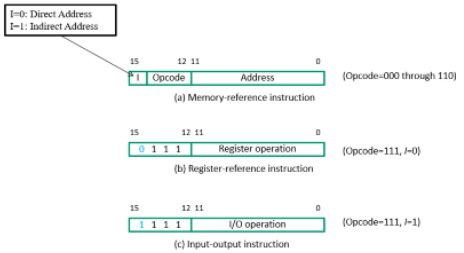
1. *immediate operand:* the 2nd part of the instruction (bits 0-11) is actually the operand data (not used in this chapter)

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.



Control Unit:

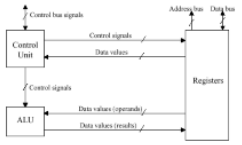
Two possible types of control:

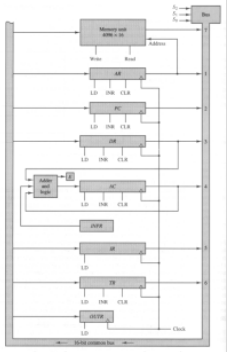
- hardwired (assumed in Chapter 5)
- microprogrammed

(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





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

Register Instructions: Operations during T_3

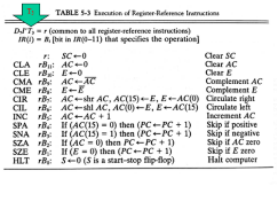


TABLE 5-3 Execution of Register-Reference Instructions

$D_7T_3 = r$ (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

(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.

M-R Instructions: Operations during T_4 onwards

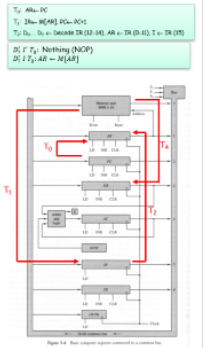
Note: Be careful when looking at Table 5-4 p. 145. Those are not technically RTL statements because they take more than one clock cycle.

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.



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.

Interrupts

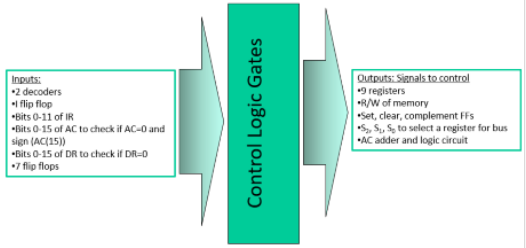
Note: In some applications, polling is necessary because there may not be other processing that can be done until input (or output) is performed.

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)

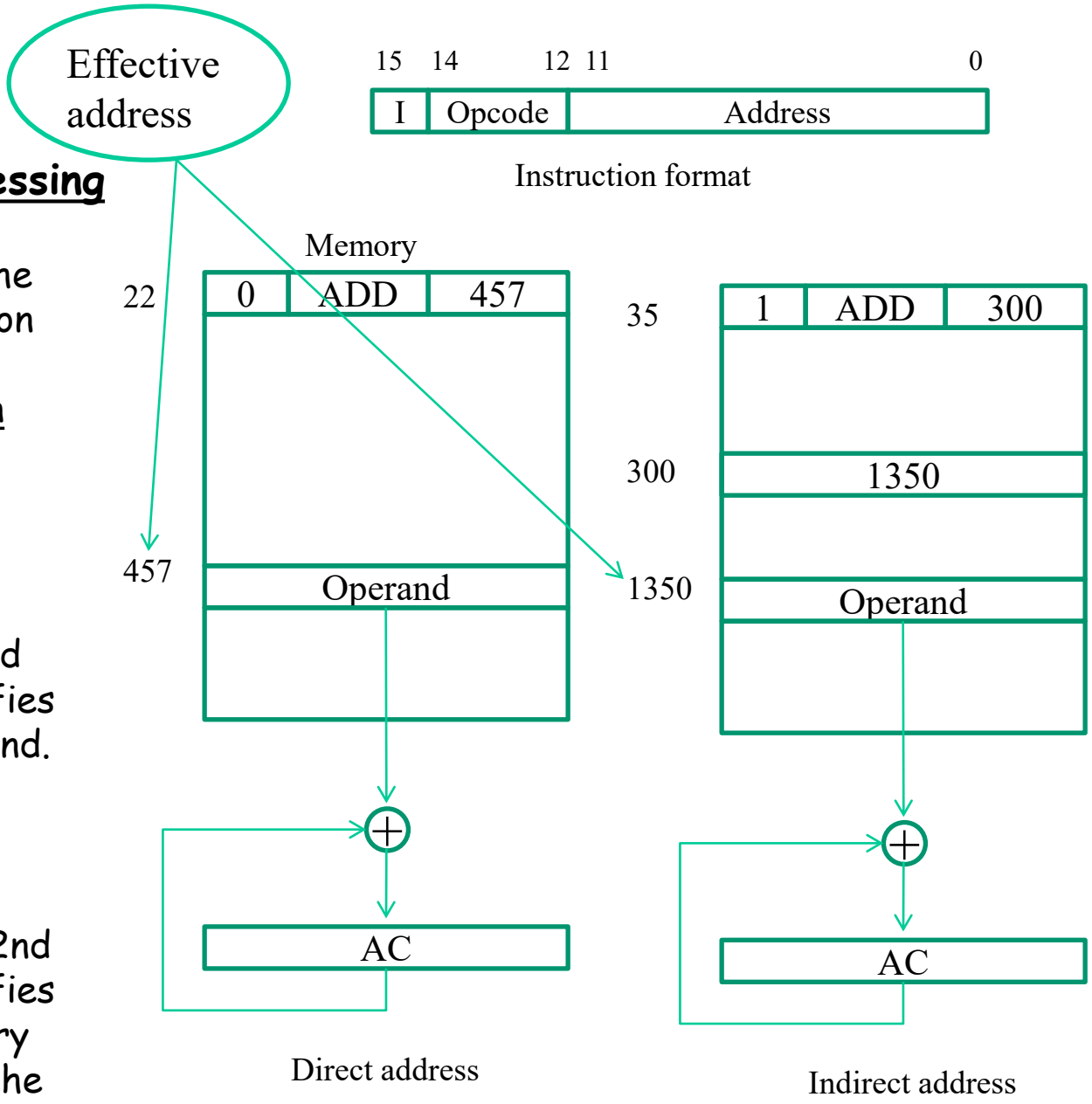


Three types of addressing

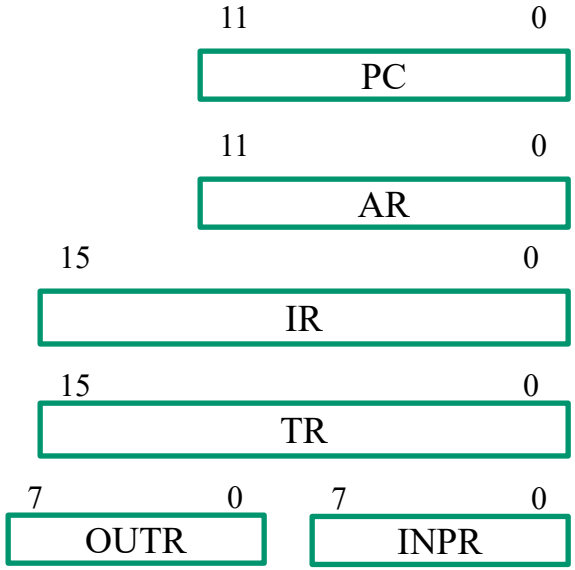
1. *immediate operand:* the 2nd part of the instruction (bits 0-11) is actually the operand data (not used in this chapter)

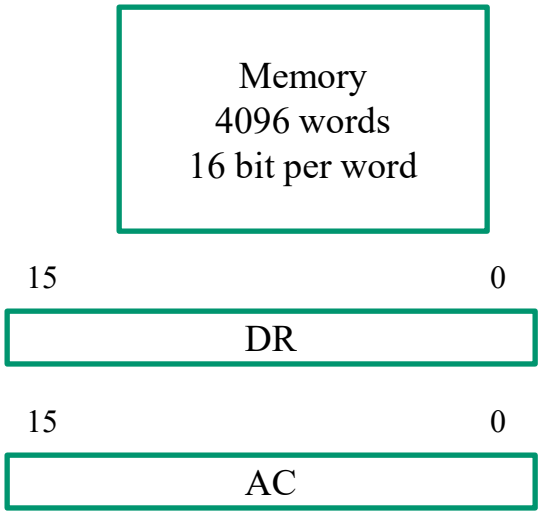
2. *direct addressing:* 2nd part of instruction specifies the address of the operand.

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



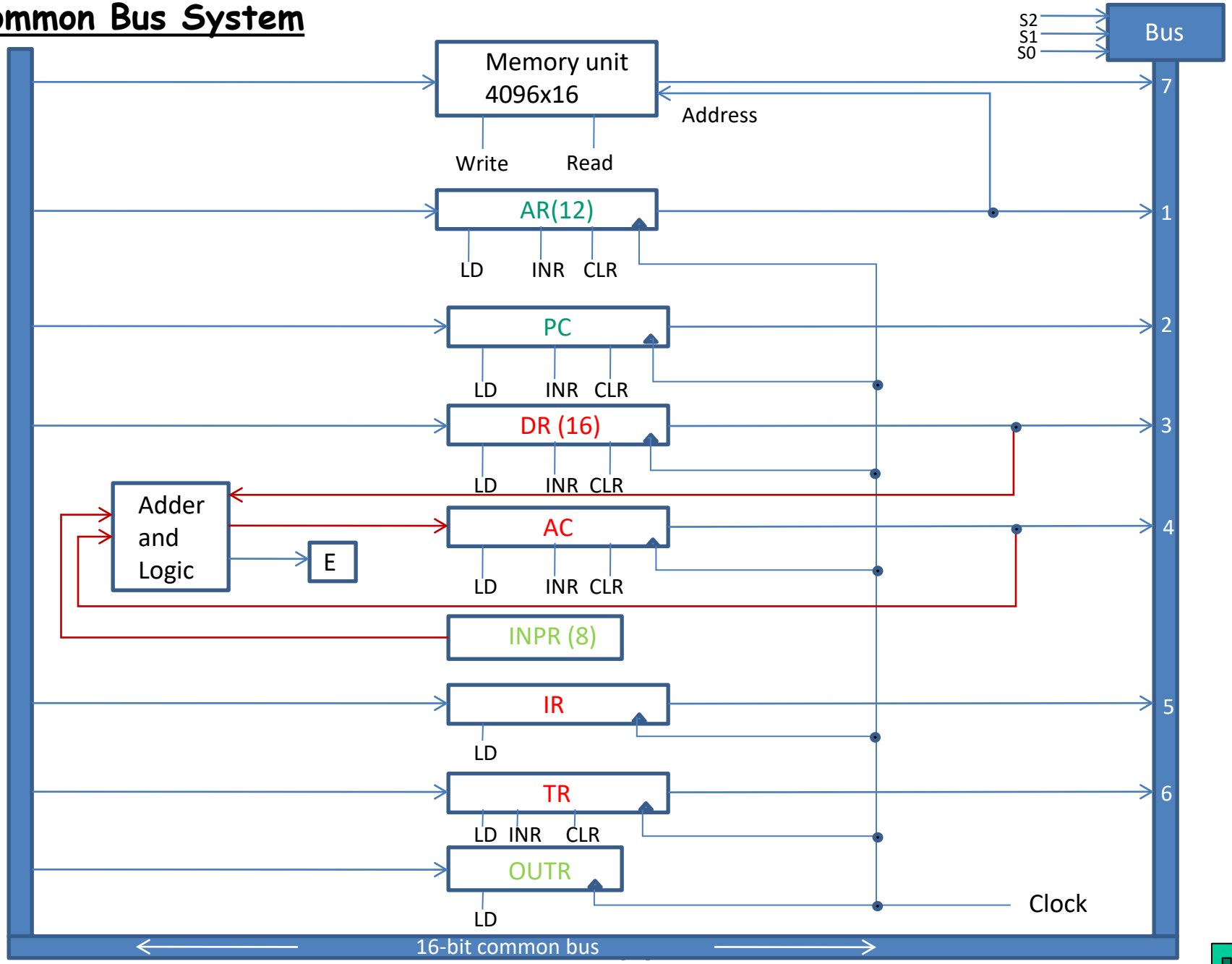
Computer Registers





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?

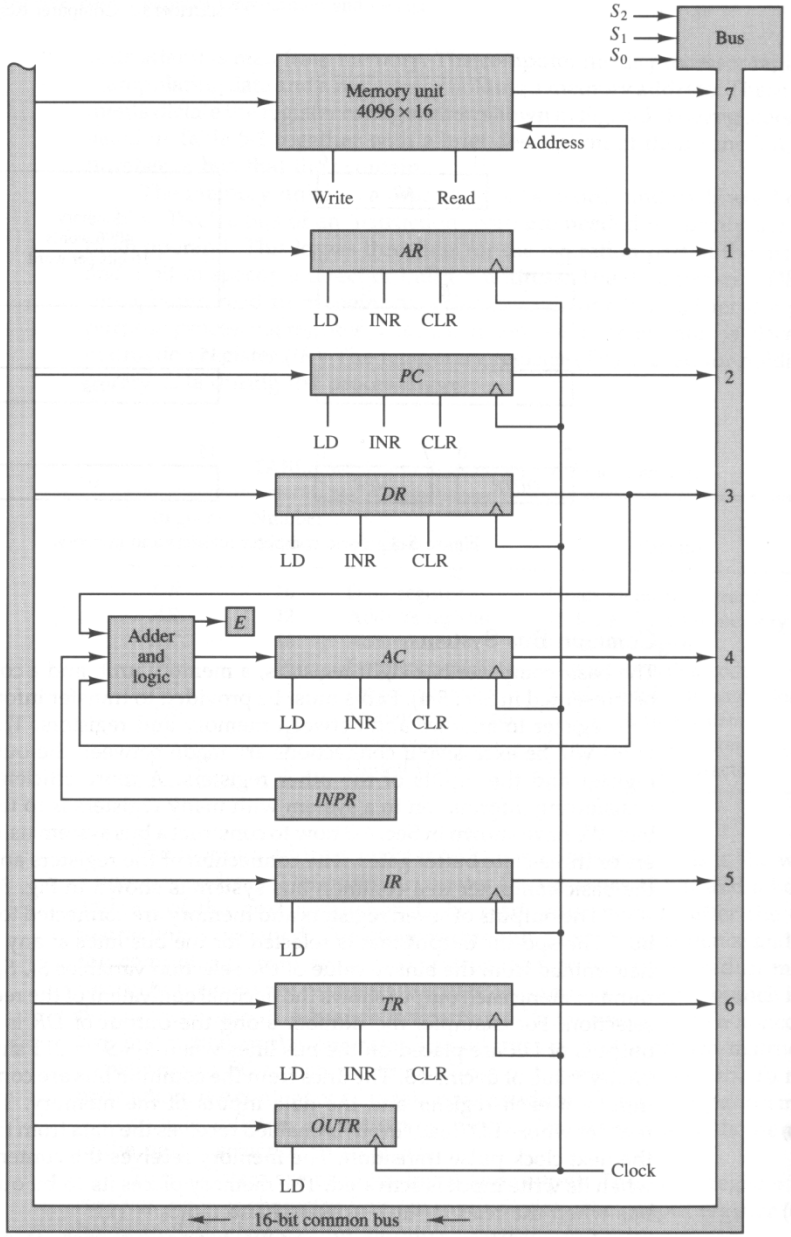


Figure 5-4 Basic computer registers connected to a common bus.

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	*IR*	Read	—
b.	1	1	0	*PC*	—	—
c.	1	0	0	*DR*	Write	—
d.	0	0	0	*AC*	—	Add

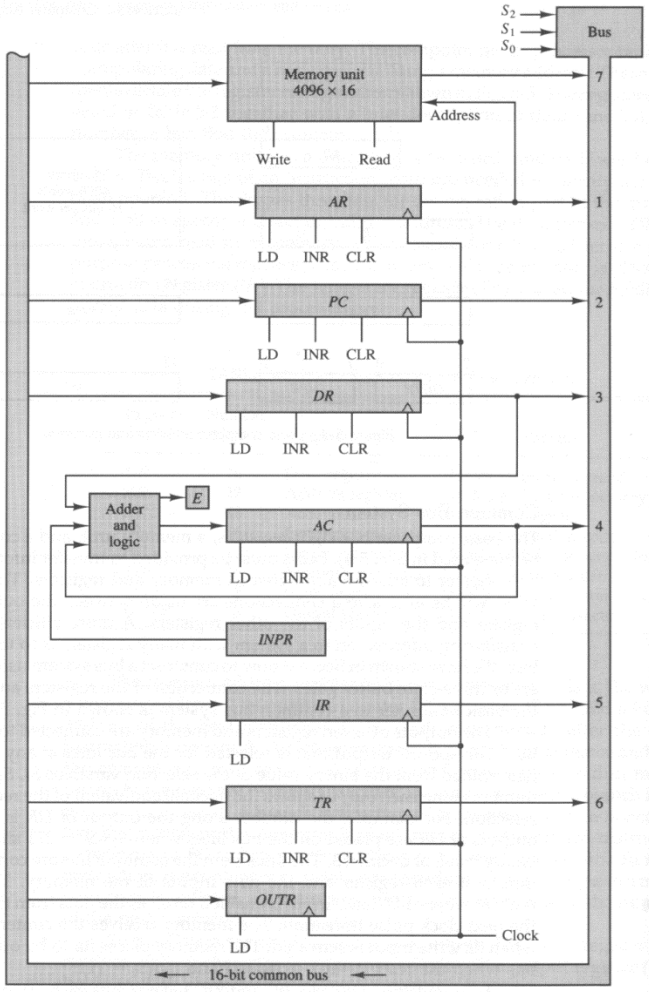


Figure 5-4 Basic computer registers connected to a common bus.

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.

a. $IR \leftarrow M[PC]$

b. $AC \leftarrow AC + TR$

c. $DR \leftarrow DR + AC$ (AC does not change)

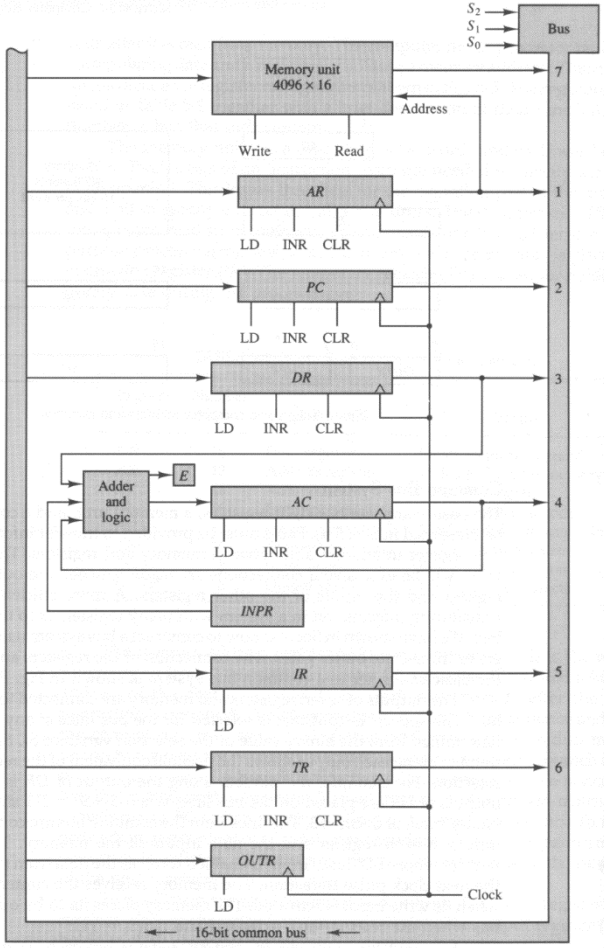
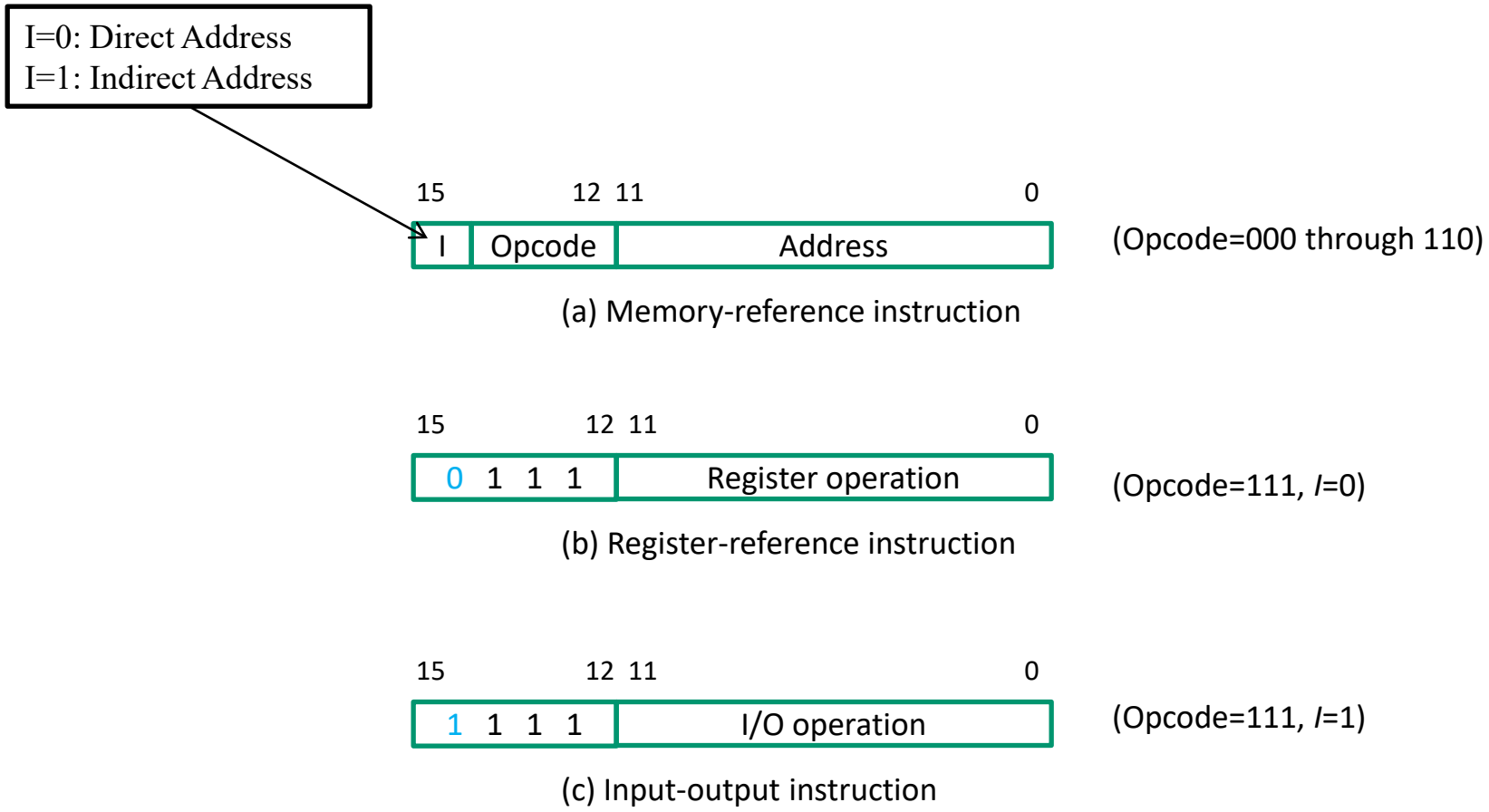


Figure 5-4 Basic computer registers connected to a common bus.

Computer Instructions

Basic computer instruction formats.



15	14	.....	12	11	.....	0
I	Opcode				Address	
0	1	1	1		Register operation	
1	1	1	1		I/O operation	

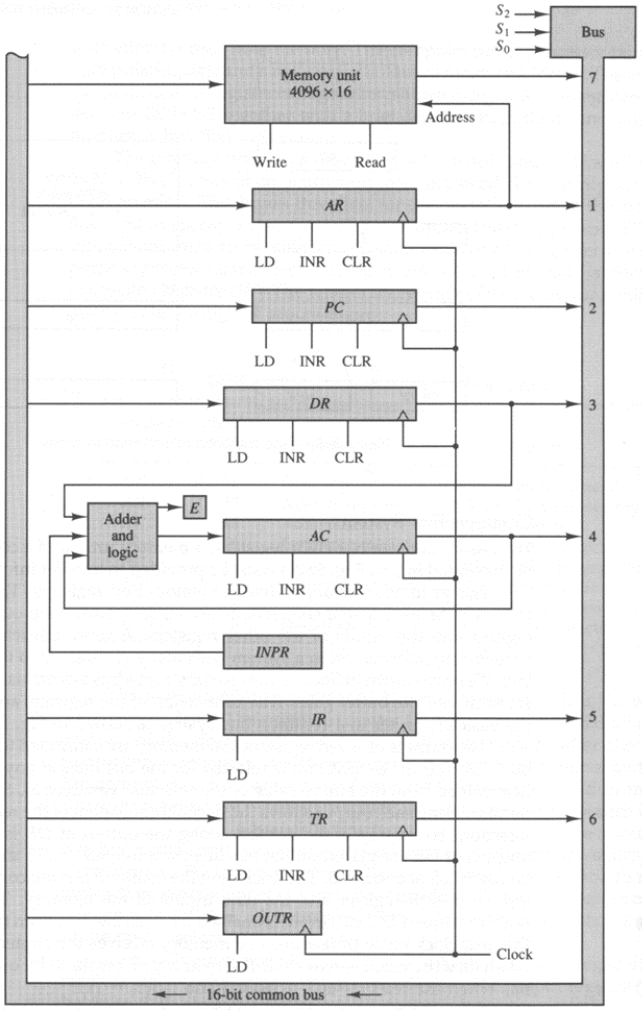


Figure 5-4 Basic computer registers connected to a common bus.

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.

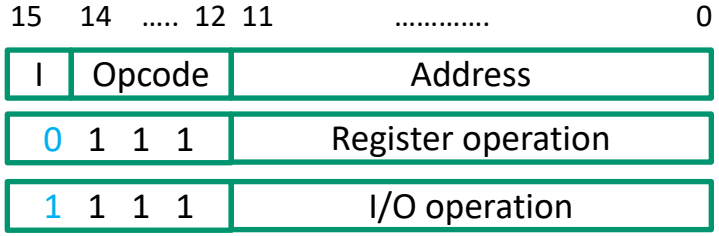
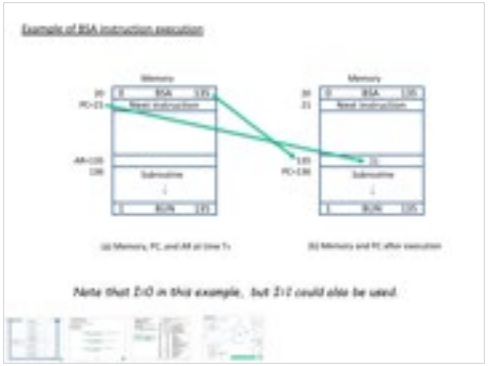


TABLE 5-2 Basic Computer Instructions

Symbol	Hexadecimal code $I = 0$	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

BUN and BSA


BUN: (Branch unconditionally)
Recall that indirect addressing could've been used.
D_3T_4: $PC \leftarrow AR$, $SC \leftarrow 0$
(Recall PC was already incremented at T_1.)
BSA: (Branch and Save Return Address)
D_3T_4: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
D_3T_5: $PC \leftarrow AR$, $SC \leftarrow 0$
Similar to Function Call


Example of BSA instruction execution
Memory
Memory
Subroutine
Subroutine
Note that I=0 in this example, but I=1 could also be used.

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.

a. How many bits are there in the operation code, the register code part, and the address part?

b. Draw the instruction word format and indicate the number of bits in each part.

c. 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

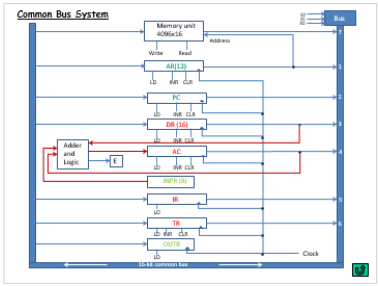
b. 1011 0001 0010 0100

c. 0111 0000 0010 0000

TABLE 5-2 Basic Computer Instructions

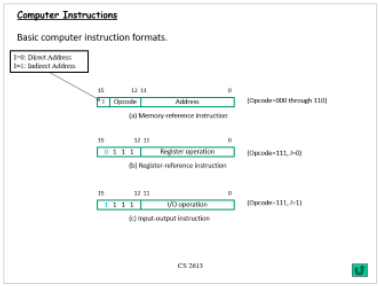
Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return 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

I/O Instructions



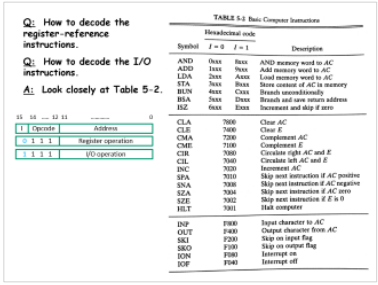
I/O Device - a terminal with a keyboard (for input) and printer (for output).

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



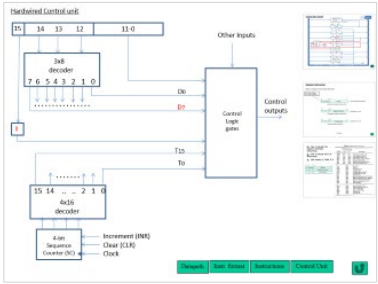
When a key is struck on the keyboard, the 8-bit code is shifted into INPR, and FGI is set to 1 by the input device.

- 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.



Input Output Instructions

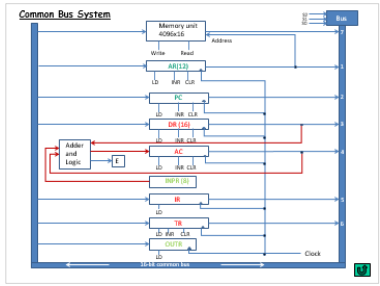
D7IT3 = p (common to all input output instructions)
IR(i) = Bi[bit in IR(6-11) that specifies the operation]

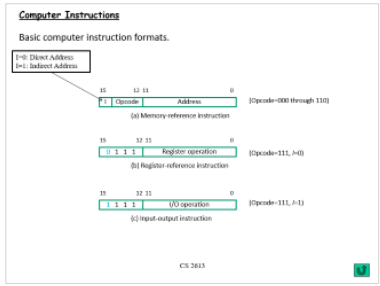
	p:	SC←0	Clear SC
INP	pB11	AC(0-7)←INPR, FGI←0	Input character
OUT	pB10	OUTR←AC(0-7), FGO←0	Output character
SKI	pB9	If(FGI=1)then (PC←PC+1)	Skip on input flag
SKO	pB8	If(FGO=1)then (PC←PC+1)	Skip on output flag
ION	pB7	IEN←1	Interrupt enable on
IOF	pB6	IEN←0	Interrupt enable off

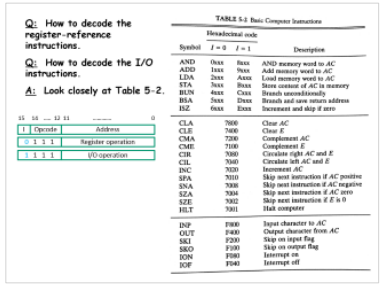
Typical programmed control transfer (polling)

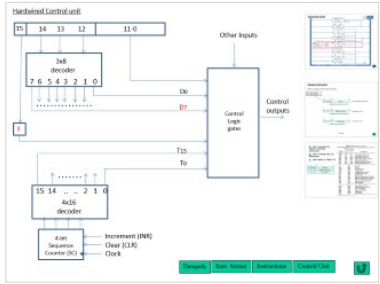
10	SKI
11	BUN 10
12	INP
. . . . .	
27	SKO
28	BUN 27
29	OUT

CS 2613









Datapath | Instr. format | Instructions | Control Unit

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.

a. What is the instruction that will be fetched and executed next?

b. Show the binary operation that will be performed in the *AC* when the instruction is executed.

c. 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:

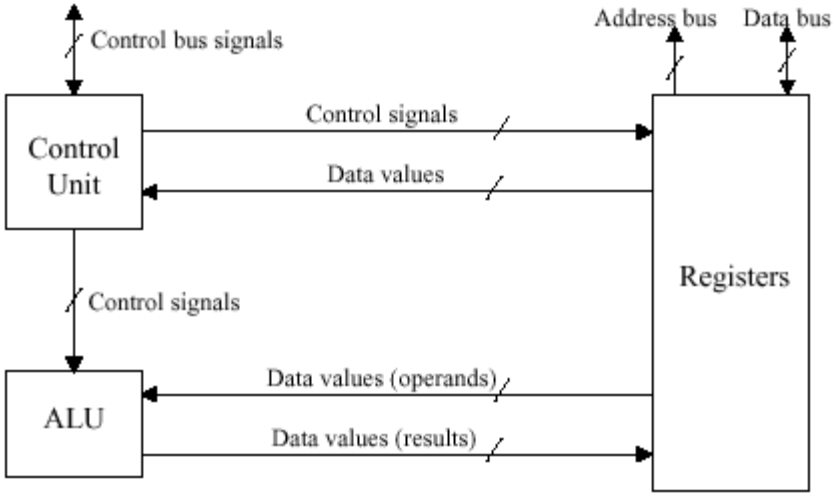
Two possible types of control:

- 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



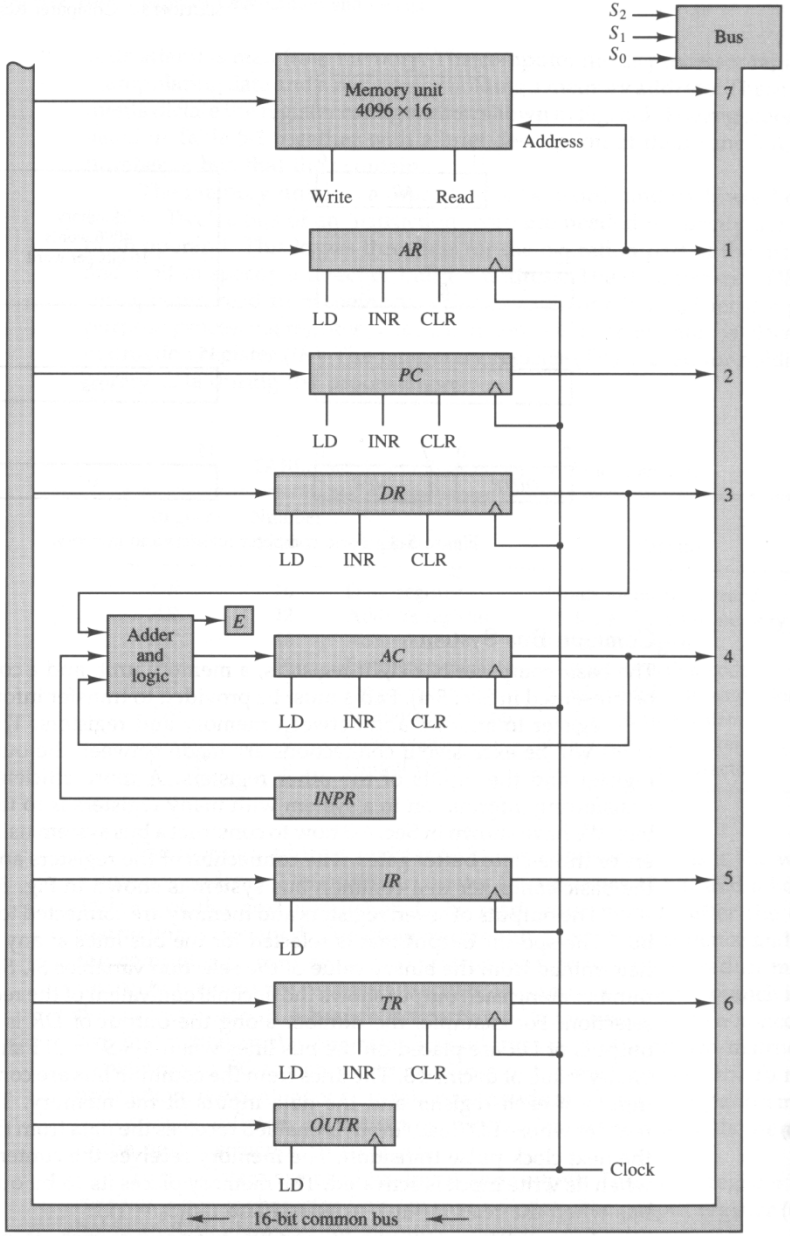
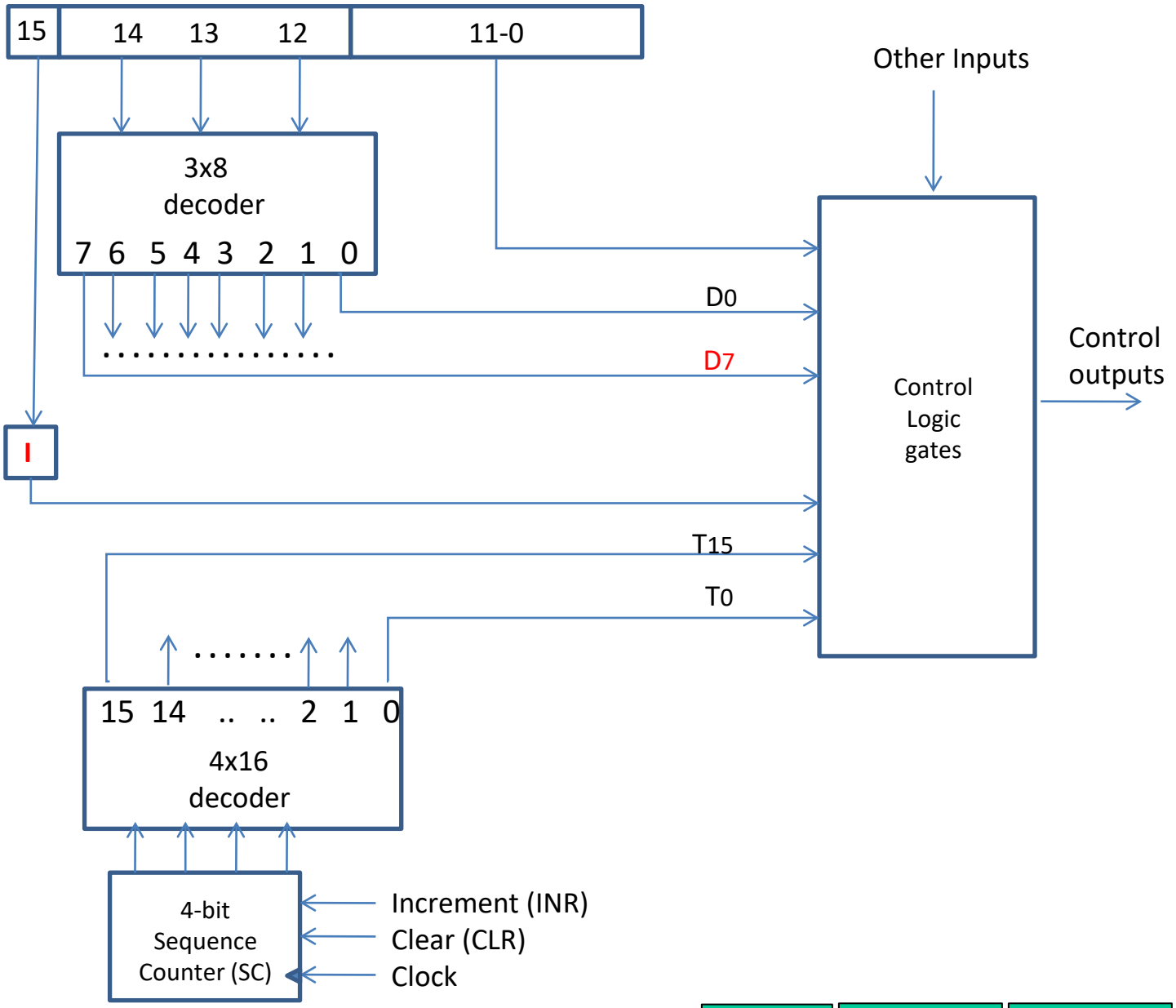
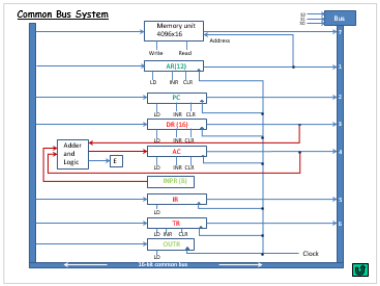
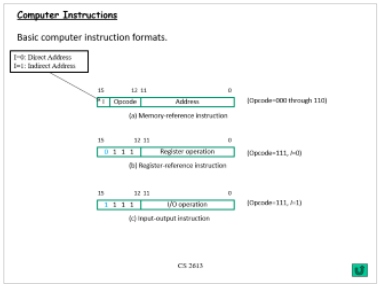


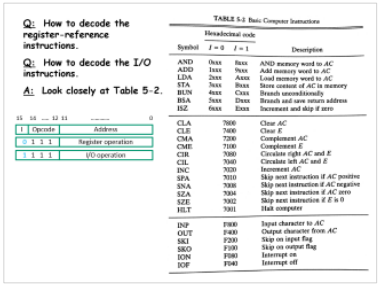
Figure 5-4 Basic computer registers connected to a common bus.

Hardwired Control unit



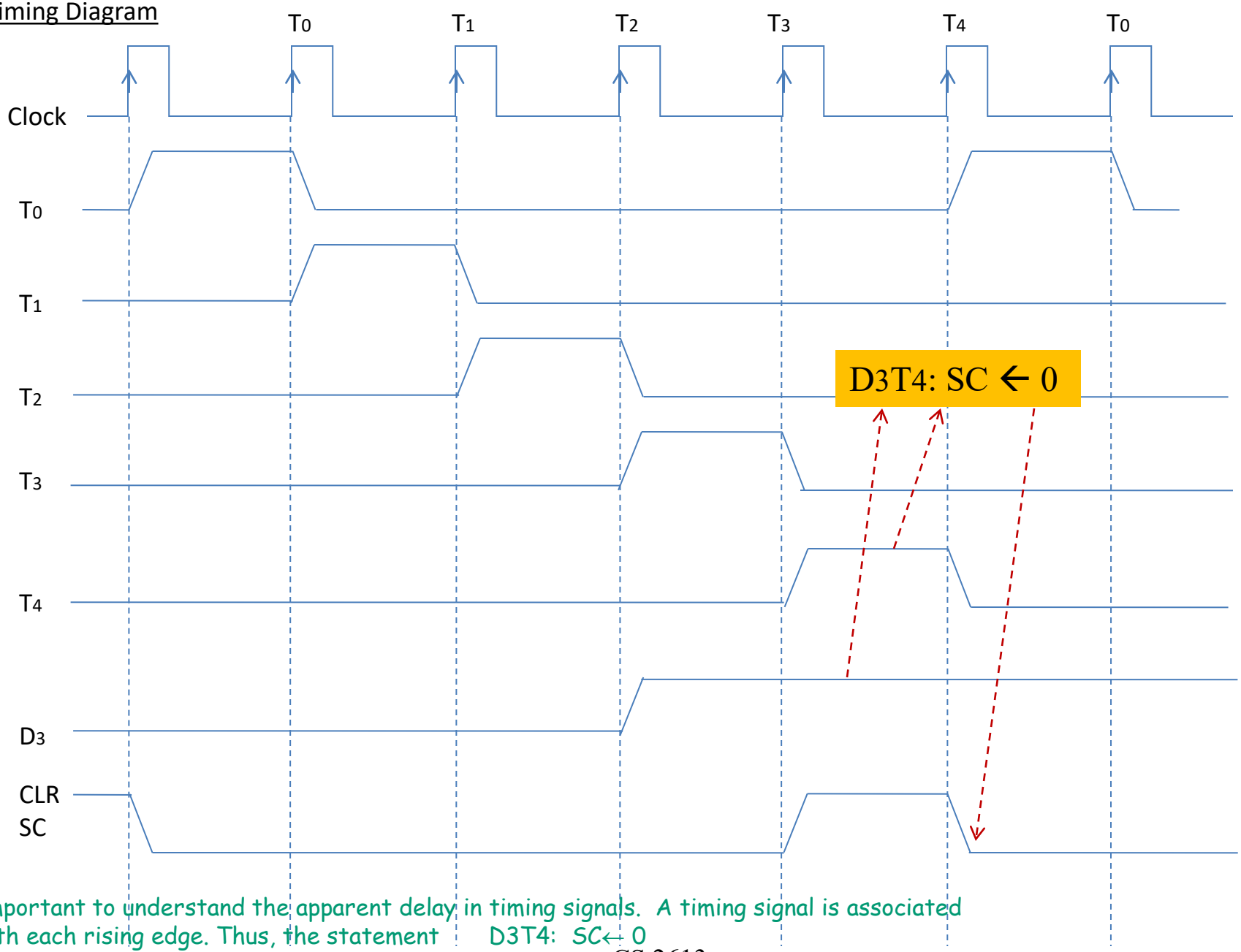






Datapath | Instr. format | Instructions | Control Unit

Timing Diagram

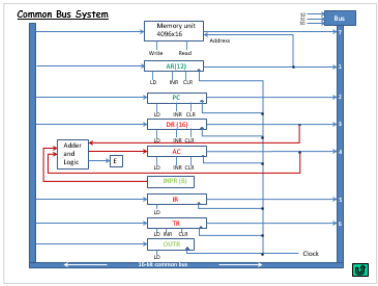


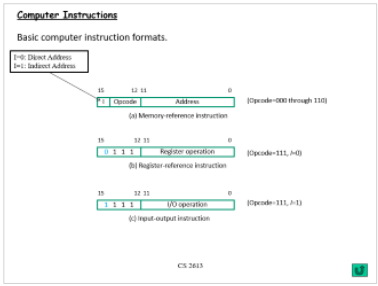
Important to understand the apparent delay in timing signals. A timing signal is associated with each rising edge. Thus, the statement D3T4: SC← 0

is not performed at the beginning of the clock cycle where D3 and T4 are activated, but at the next rising edge

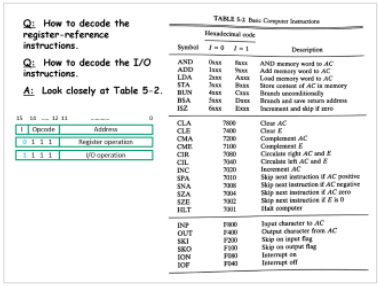
Instruction Cycle (four subcycles)

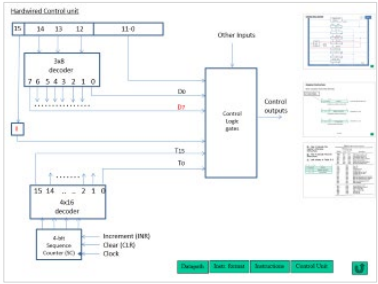
1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory (if indirect addressing is used)
4. Execute the instruction.

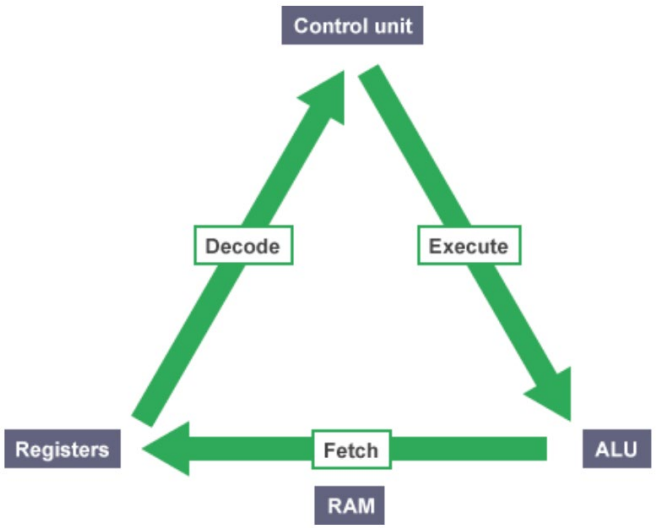




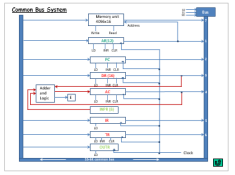
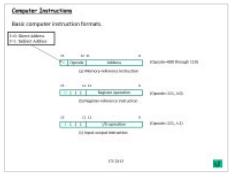
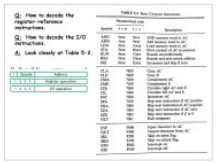
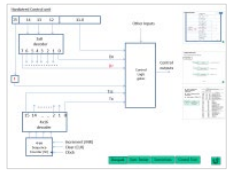
Click for
CPU Fetch-Decode-Execute Animation







RTL specification for "fetch and load" phase:

This sequence is done for all instructions.

T_0: $AR \leftarrow PC$

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

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

$SC \leftarrow SC + 1$ 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.

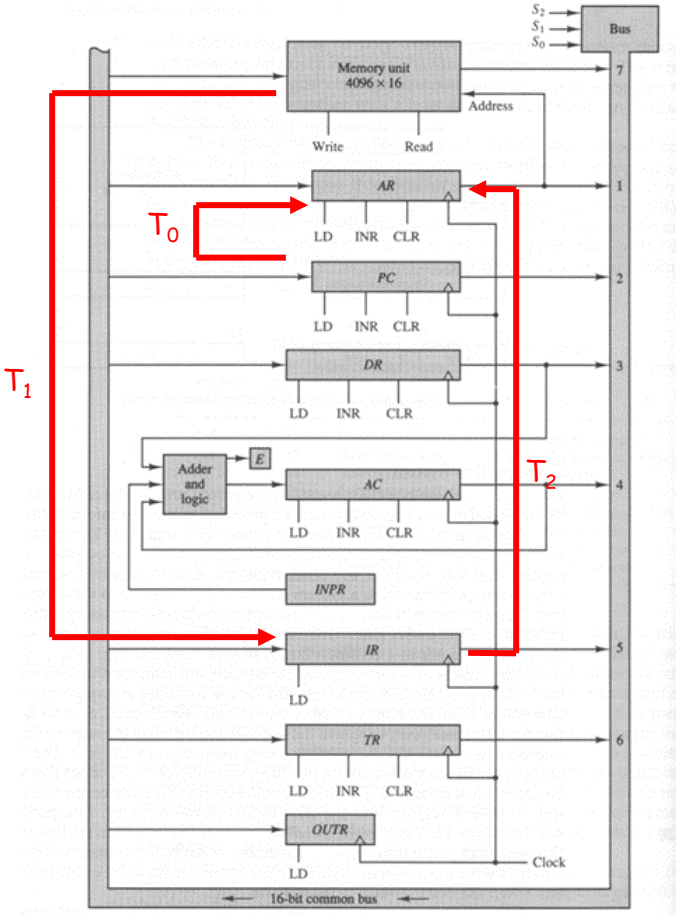
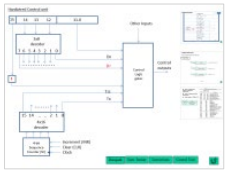


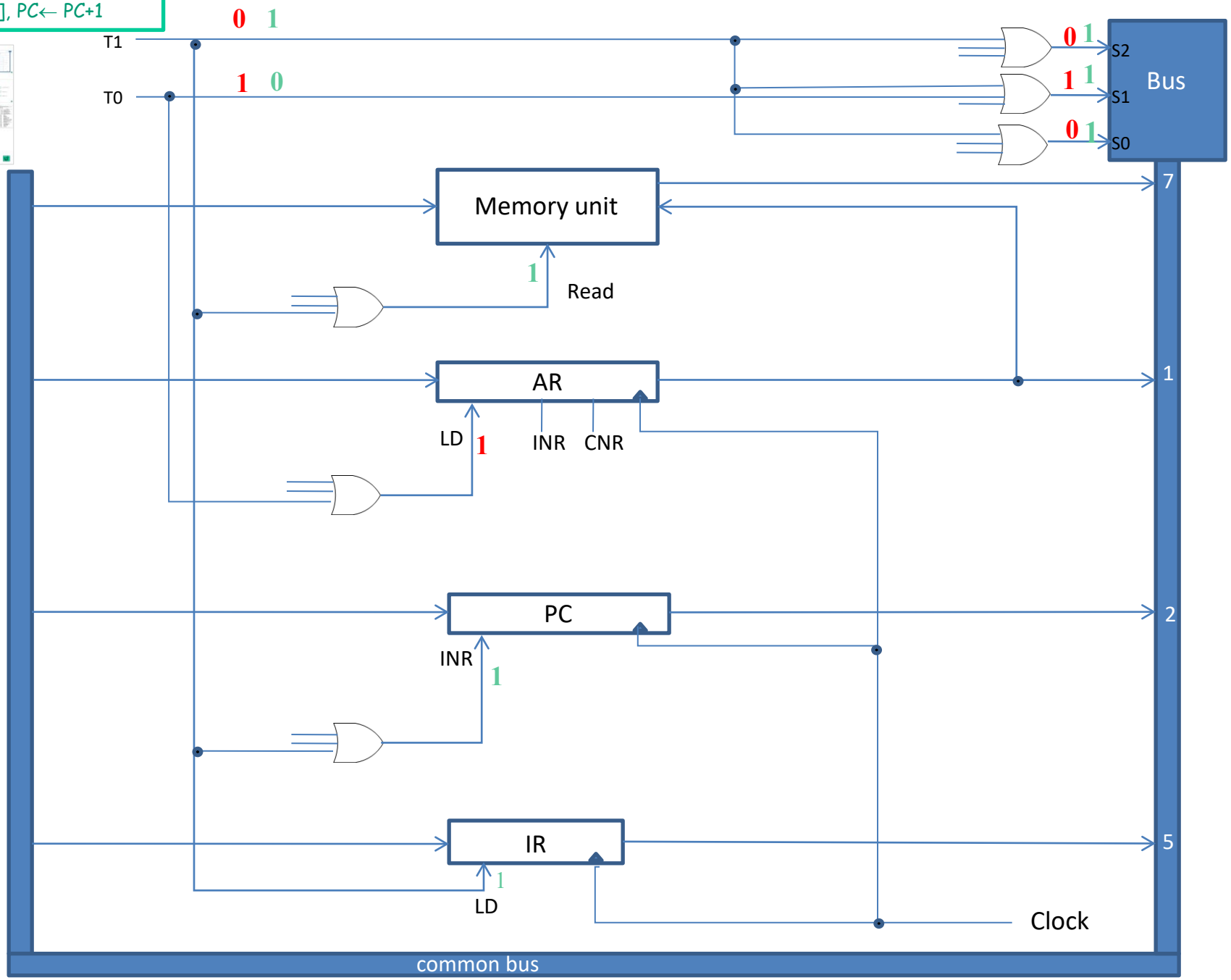
Figure 5-4 Basic computer registers connected to a common bus.

Implementation of:

T_0: AR← PC

T_1: IR← M[AR], PC← PC+1





Four specific cases

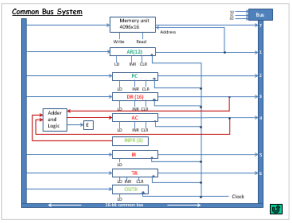
Nearly the same, but indirect addressing needs one extra cycle to get the effective address.

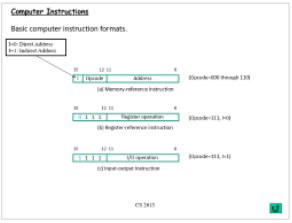
$D_7' I' \Rightarrow$ Memory-reference with direct addressing

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

$D_7 I' \Rightarrow$ Register reference instruction

$D_7 I \Rightarrow$ I/O instruction

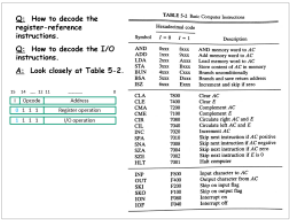




$D_7' I' T_3$: Nothing (NOP)

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

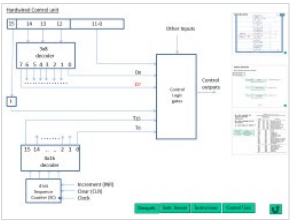
(T_4 will be used to start executing memory reference instructions).



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

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

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



Register Instructions: Operations during T_3

T_3

TABLE 5-3 Execution of Register-Reference Instructions

$D_7 I' T_3 = r$ (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

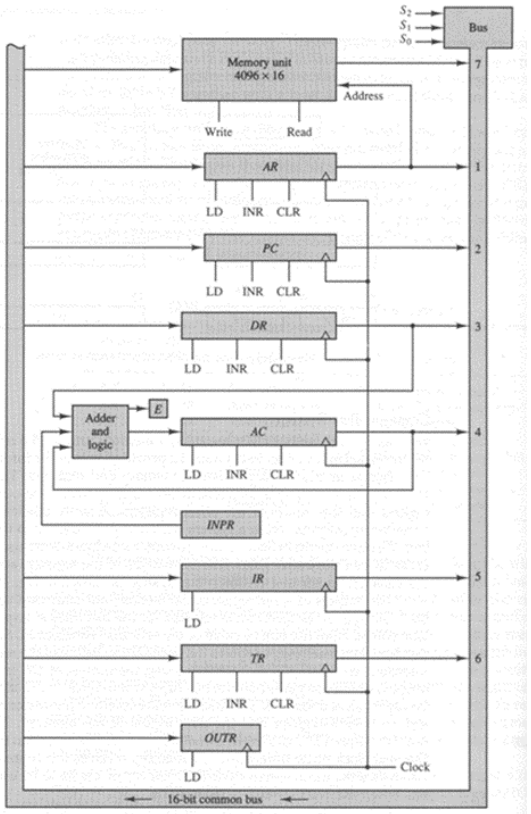
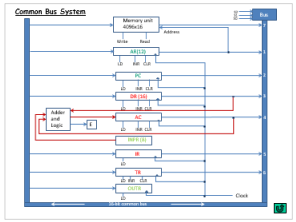
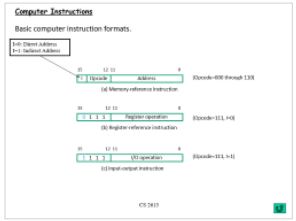
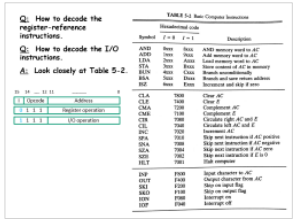
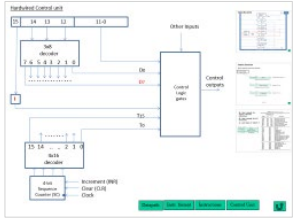


Figure 5-4 Basic computer registers connected to a common bus.









(Sequence Counter)

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

<u>Q:</u> How could the start-stop flip flop be used to stop the computer?

<u>A:</u> 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 $rB_4 AC(15)'$: $PC \leftarrow PC + 1$

SNA $rB_3 AC(15)$: $PC \leftarrow PC + 1$

SZA $rB_2 Z$: $PC \leftarrow PC + 1$

SZE $rB_1 E$: $PC \leftarrow PC + 1$

M-R Instructions: Operations during T_4 onwards

T_0: AR← PC

T_1: IR← M[AR], PC← PC+1

T_2: $D_0 \ldots D_7$ ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15)

$D_7' I' T_3$: Nothing (NOP)

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

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

$D_0 T_4: DR \leftarrow M[AR]$

$D_0 T_5: AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$

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

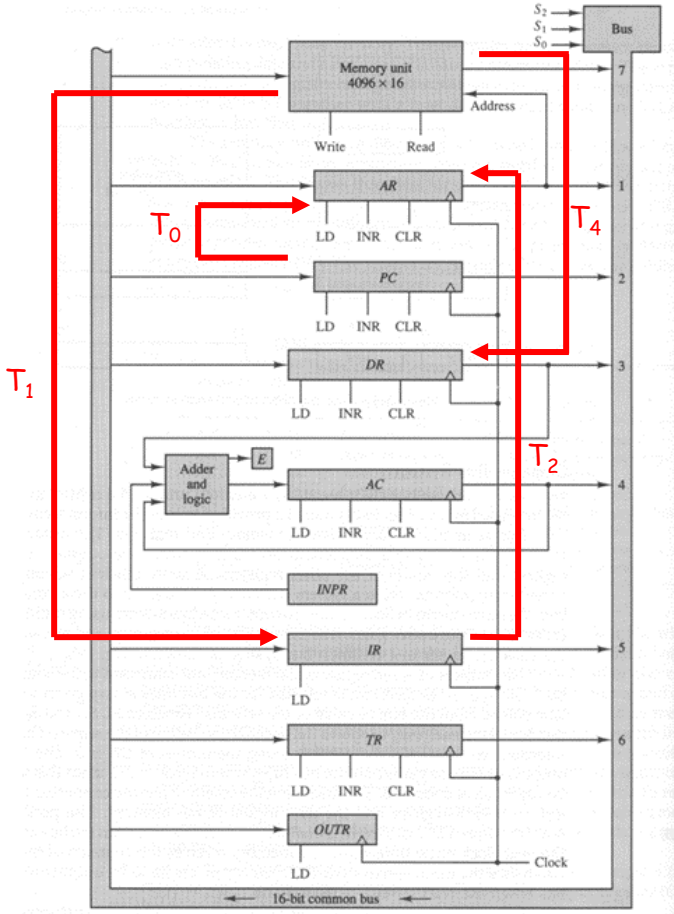
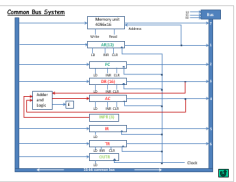
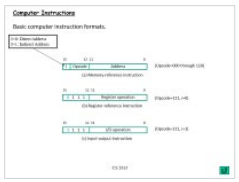
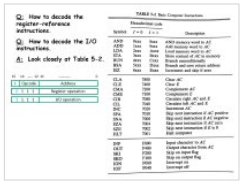
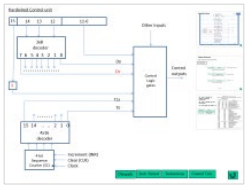


Figure 5-4 Basic computer registers connected to a common bus.

T_0: AR← PC

T_1: IR← M[AR], PC← PC+1

T_2: $D_0 \ldots D_7$ ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15)

$D_7' I' T_3$: Nothing (NOP)

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

ADD

$D_1 T_4$: $DR \leftarrow M[AR]$

$D_1 T_5$: $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$

LDA (load to AC)

$D_2 T_4: DR \leftarrow M[AR]$

$D_2 T_5: AC \leftarrow DR, SC \leftarrow 0$

STA (store AC)

$D_3 T_4$: $M[AR] \leftarrow AC, SC \leftarrow 0$

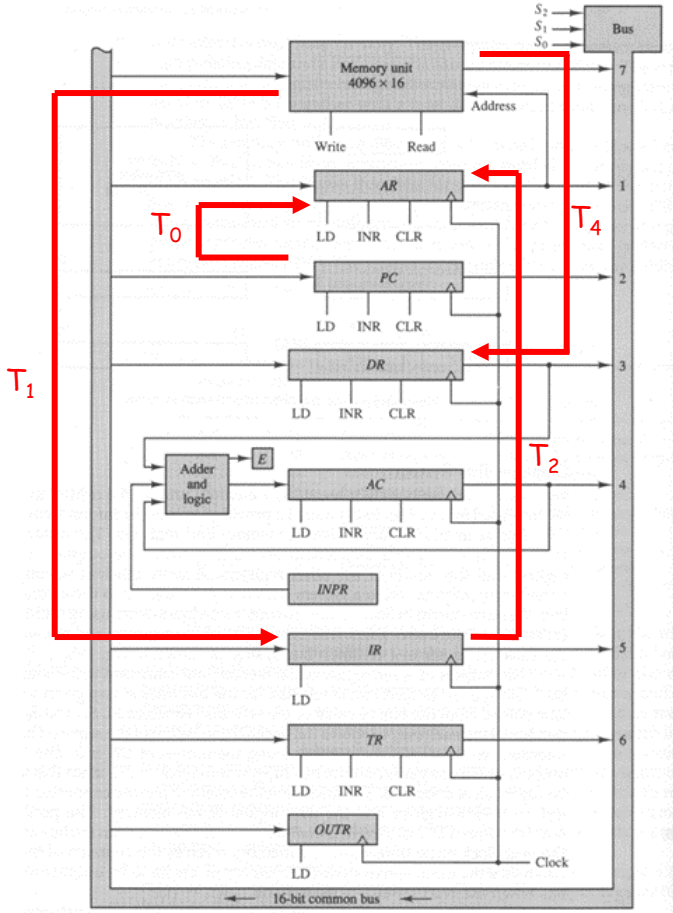
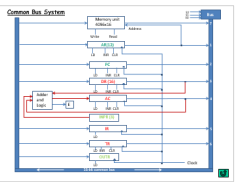
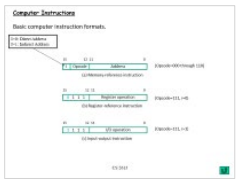
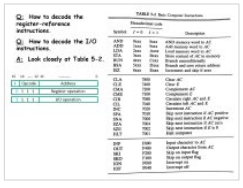
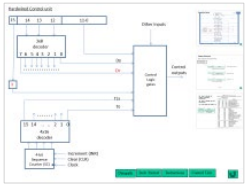


Figure 5-4 Basic computer registers connected to a common bus.

T_0: AR← PC

T_1: IR← M[AR], PC← PC+1

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

$D_7' I' T_3$: Nothing (NOP)

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

BUN: (branch unconditionally)

Recall that indirect addressing could've been used.

$D_4T_4: \; PC \leftarrow AR, \qquad SC \leftarrow 0$

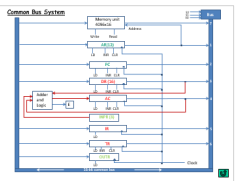
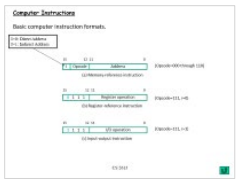
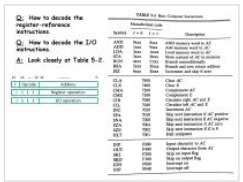
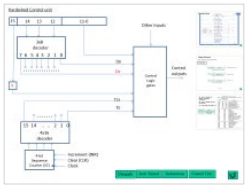
(Recall PC was already incremented at T_1.)

BSA: (Branch and Save Return Address)

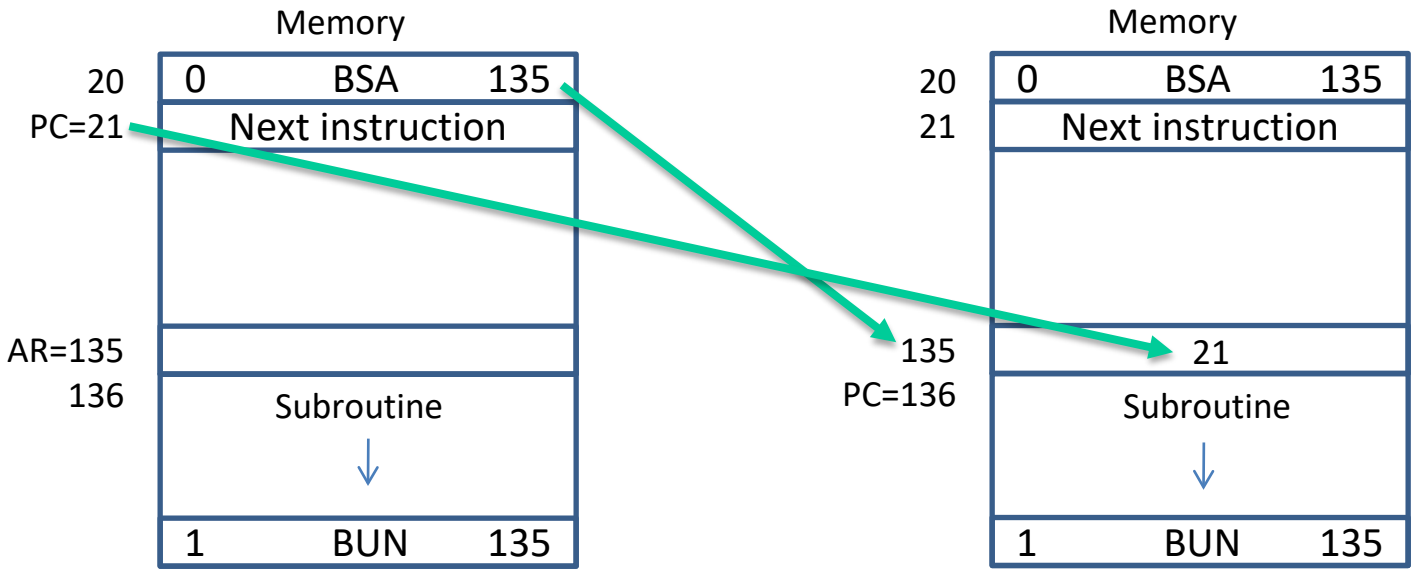
$D_5T_4: \; M[AR] \leftarrow PC, \qquad AR \leftarrow AR + 1$

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

Similar to Function Call

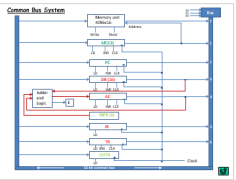
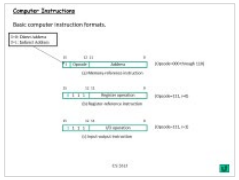
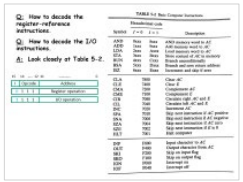
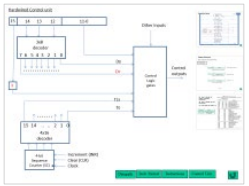
Example of BSA instruction execution



(a) Memory, PC, and AR at time T_4

(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← PC

T_1: IR← M[AR], PC← PC+1

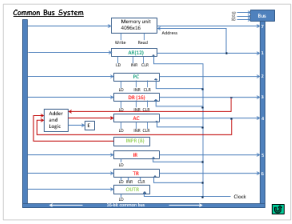
T_2: D_0… D_7 ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15)

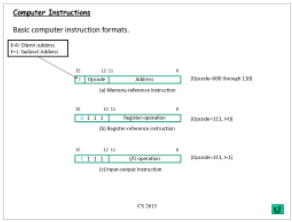
$$D_6T_4: \quad DR \leftarrow M[AR]$$
$$D_6T_5: \quad DR \leftarrow DR + 1$$
$$D_6T_6: \quad M[\mathrm{AR}] \leftarrow DR, \text{ if } (DR = 0) \text{ then } (\mathrm{PC} \leftarrow \mathrm{PC} + 1), \mathrm{SC} \leftarrow 0$$

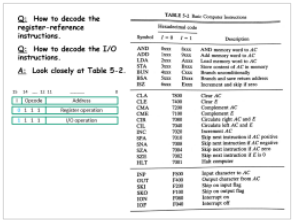
Can be used to implement a "for" loop.

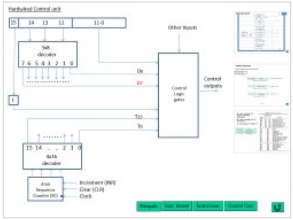
Use a negative number (stored in memory) as the loop index. As the index is incremented, it will eventually equal to zero.

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

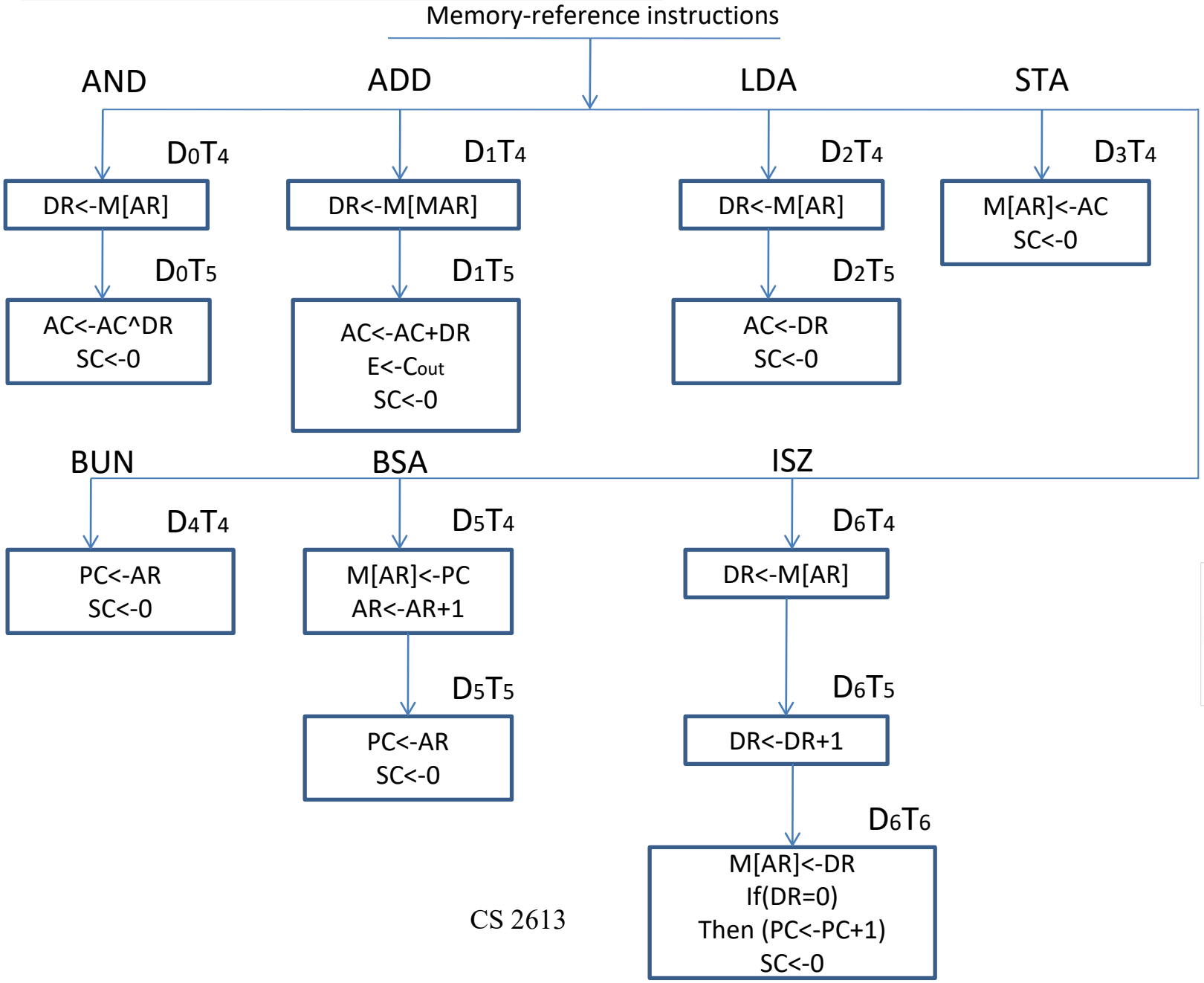


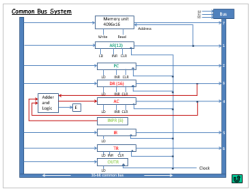


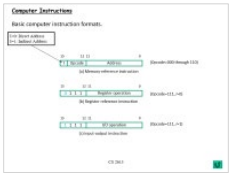


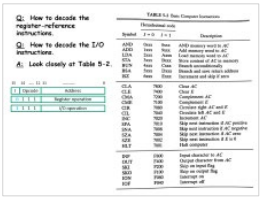


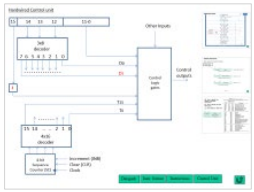
Flowchart for memory-reference instruction











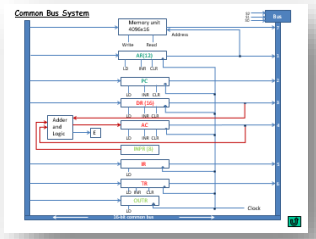
Interrupts

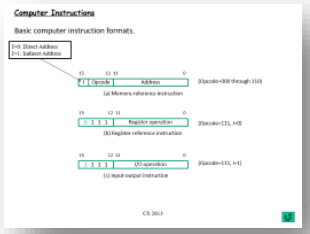
Note: In some applications, polling is necessary because there may not be other processing that can be done until input (or output) is performed.

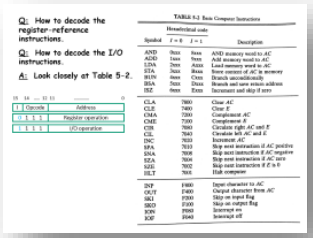
However, in some cases interrupt control is more efficient (the I/O device interrupts the computer to let it know a transfer is ready).

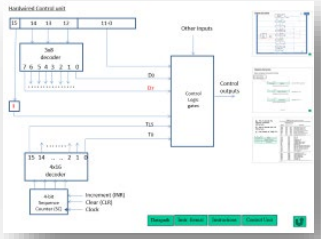
When the IEN flag is set to 1, then the flags FGI and FGO interrupt the computer.

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









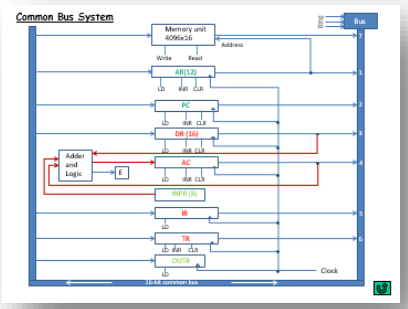
The interrupt flag R is set to 1 when either FGI or FGO is set AND the IEN = 1, so R = (IEN)(FGI + FGO)…

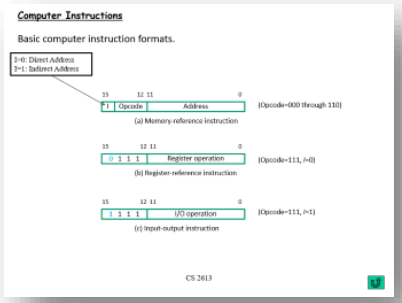
…However, the setting of R is performed only during the "Execution Phase" of each instruction:

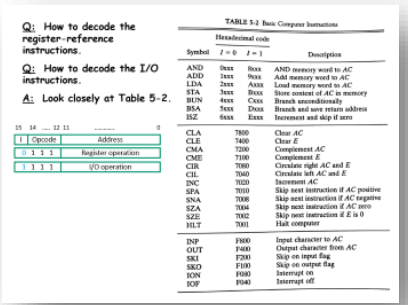
$$T_0' T_1' T_2' (IEN)(FGI + FGO): R \leftarrow 1$$

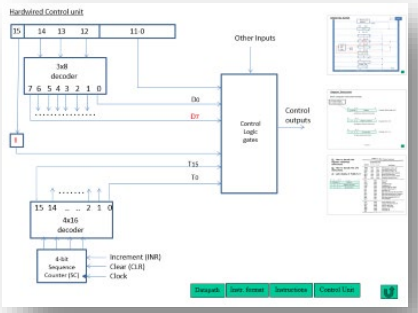
Q: Why not just put $T3$ here?

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.

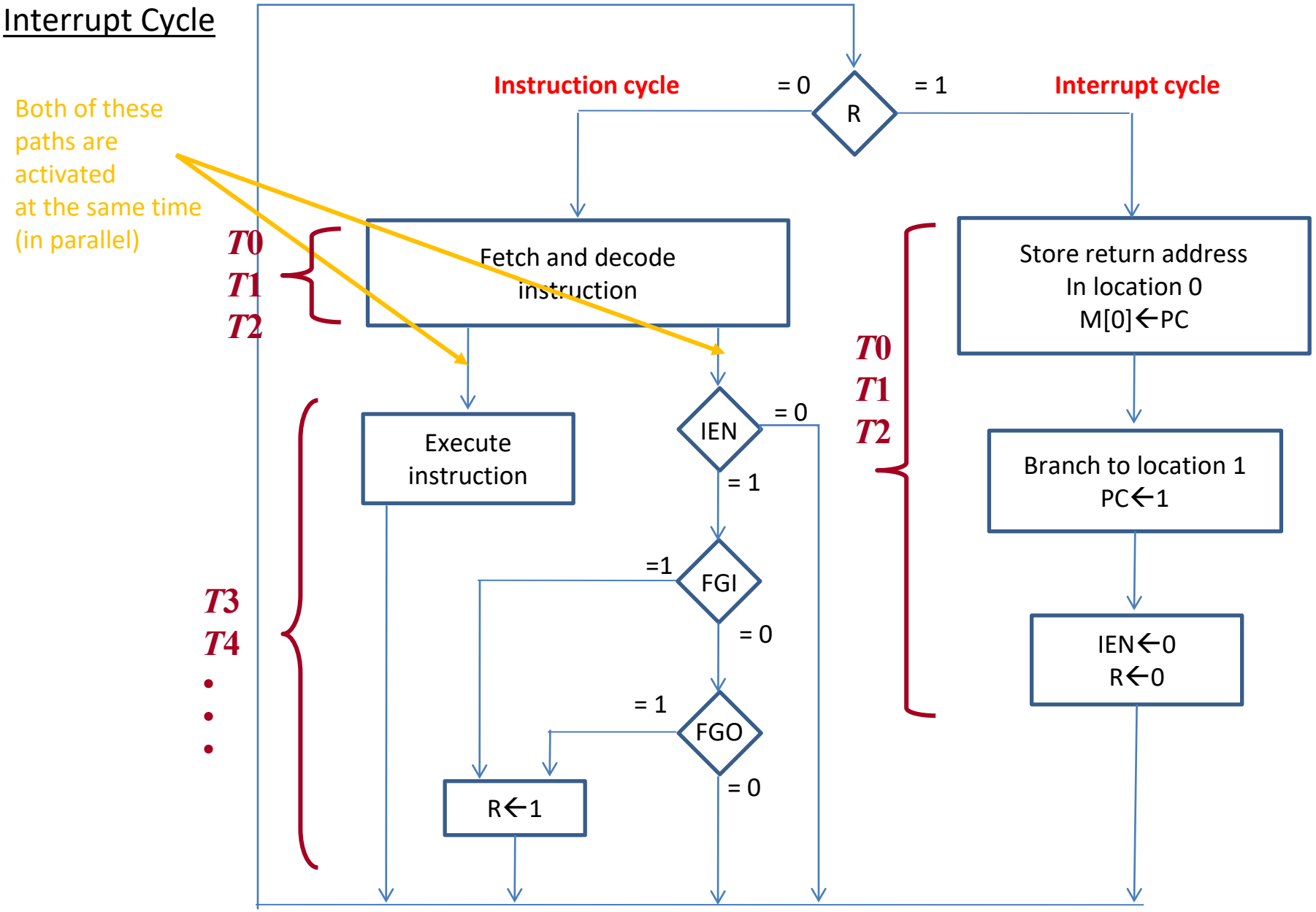








Interrupt Cycle



T_0: AR← PC

T_1: IR← M[AR], PC← PC+1

T_2: $D_0 \ldots D_7$ ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15)

$D_7' I' T_3$: 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. →

$R'T_0: AR \leftarrow PC$

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

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

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

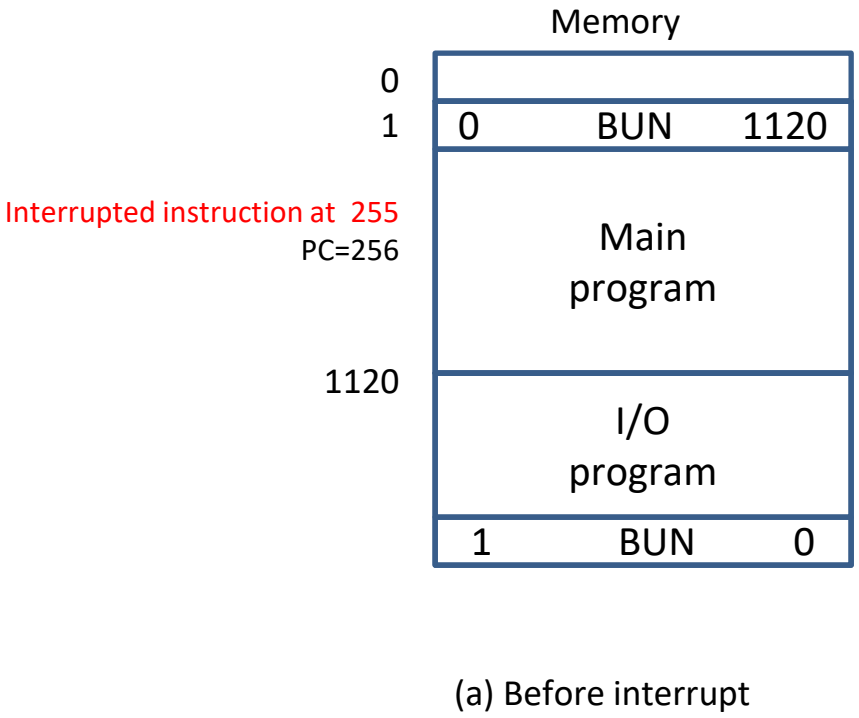
- Store PC in memory location 0
- Branch to location 1
- Clear IEN, R, and SC

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

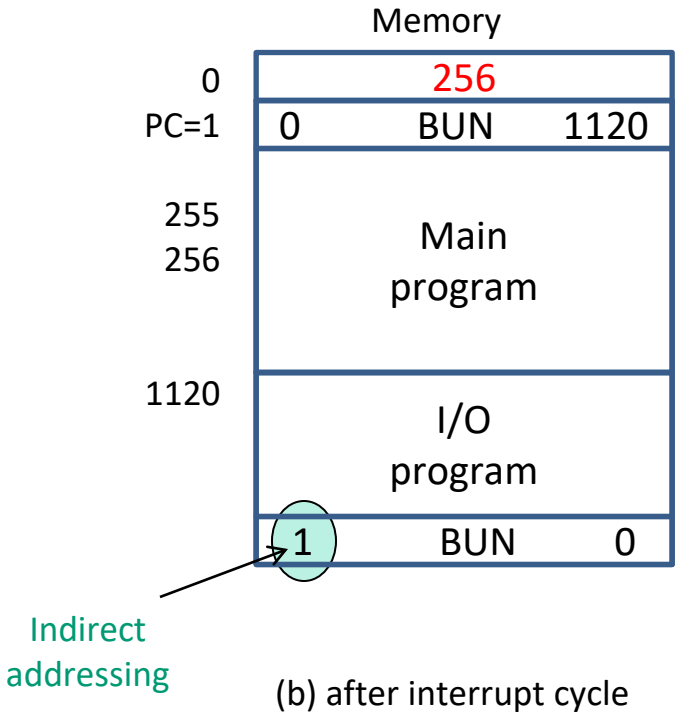
$RT_1: M[AR] \leftarrow TR, PC \leftarrow 0$

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

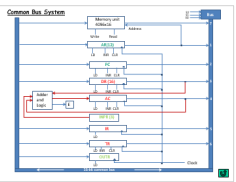
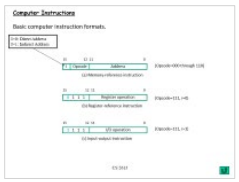
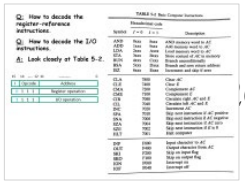
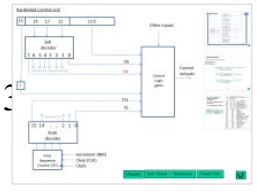
Demonstration of the interrupt cycle



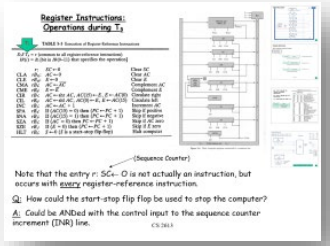
(a) Before interrupt



(b) after interrupt cycle

Flowchart for complete computer operation



Start
SC<-0, IEN<-0, R<-0

R

= 0 — Instruction cycle

R'T0
AR<-PC

R'T1
IR<-M[AR], PC<-PC+1

R'T2
AR<-IR(0-11), I<-IR(15)
Do…D7<-Decode IR(12-14)

= 1 — Interrupt cycle

RT0
AR<-0, TR<-PC

RT1
M[AR]<-TR, PC<-0

RT2
PC<-PC+1, IEN<-0
R<-0, SC<-0

D7

(Register or I/O) = 1

I

(I/O) = 1

D7IT3
Execute
Input-output
Instruction
(Table 5-5)

= 0(register)

D7I'T3
Execute
Register-
reference
Instruction
(Table 5-3)

= 0 (Memory-reference)

I

(indirect) = 1

D7'IT3
AR<-M[AR]

= 0 (direct)

D'7I'T3
Nothing

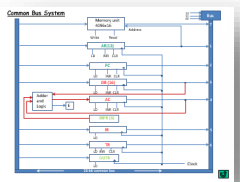
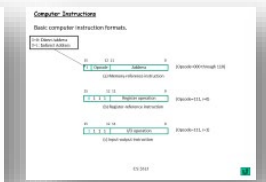
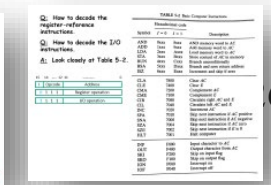
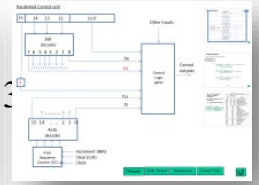
Execute
Memory-reference
Instruction

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

Inputs:
- 2 decoders
- I 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_2, S_1, S_0 to select a register for bus
- AC adder and logic circuit

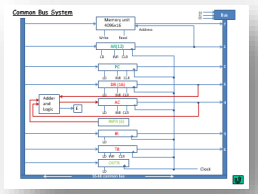





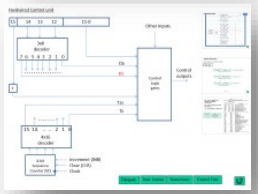
Control functions and microoperations for the basic computer

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	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	D'_7IT_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:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
		$D_7I'T_3 = r$ (common to all register-reference instructions)
		$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	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:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$





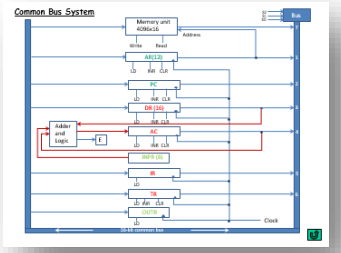
Control of Registers and Memory

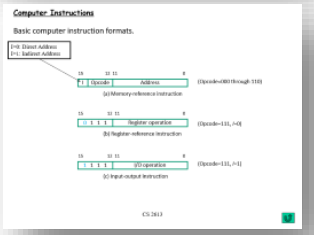
Ex. Consider all microops for which the content of AR is changed.

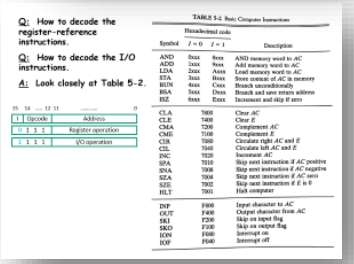
$$R'T_0: AR \leftarrow PC$$
$$R'T_2: AR \leftarrow IR(0-11)$$
$$D_7'IT_3: AR \leftarrow M[AR]$$
$$RT_0: AR \leftarrow 0$$
$$D_5T_4: AR \leftarrow AR + 1$$

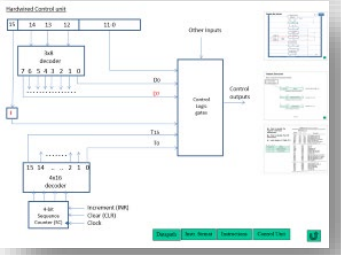
So, the signals LD, CLR and INR (of AR) are controlled as follows:

$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$



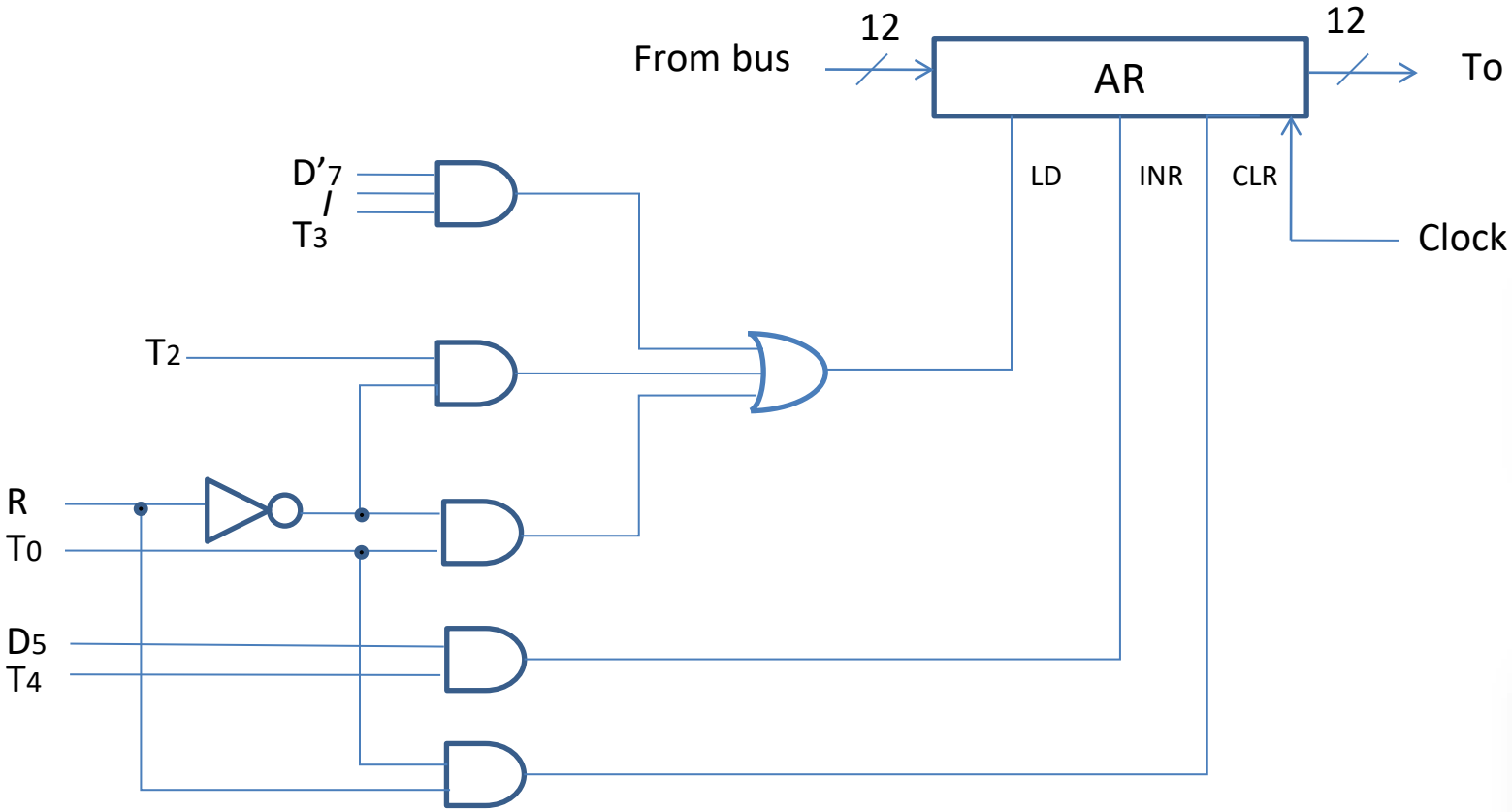


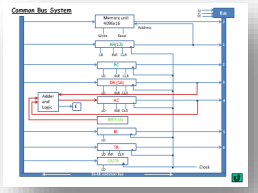


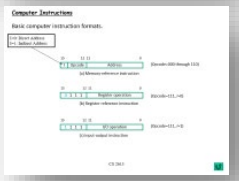


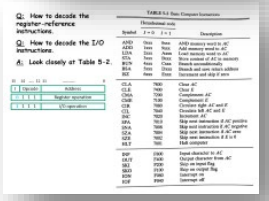
Similar logic design is performed for other registers and memory.

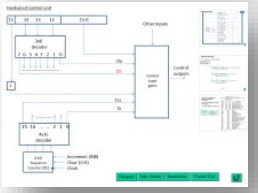
Control gates associated with AR







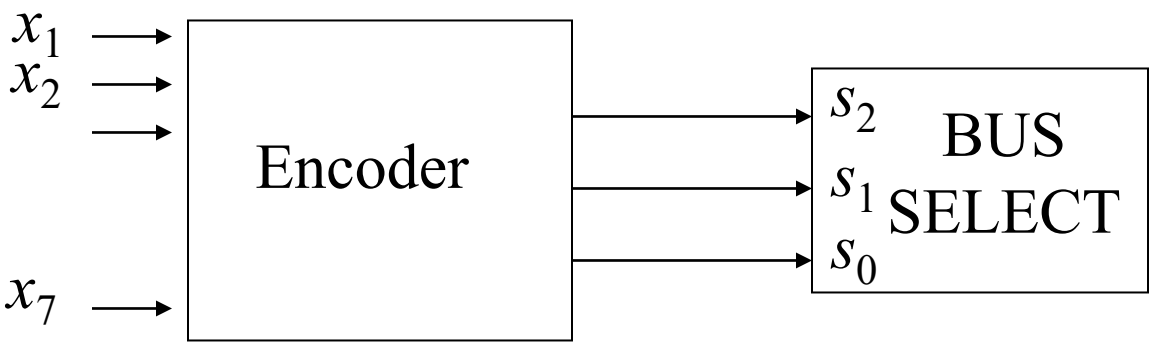




$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$

Control of the Bus

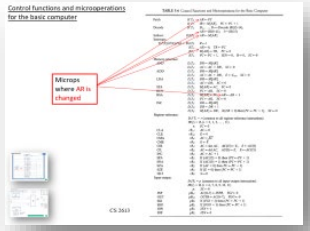
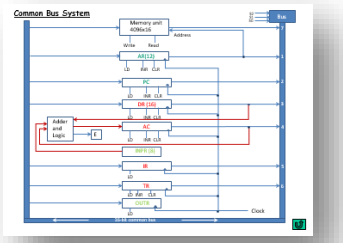
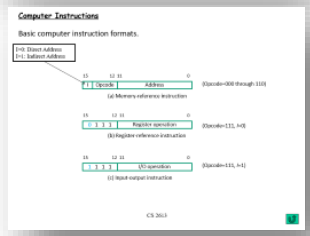
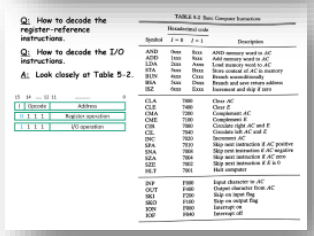
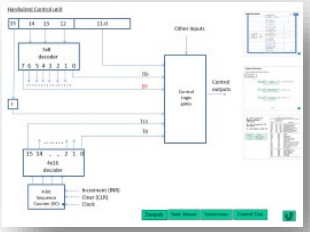
Let x_1 to x_7 denote signals for selection of registers and memory onto the bus, i.e., AR = 1, PC = 2, ... Mem unit = 7 (refer to Figure 5-4, next slide)



So, whenever AR is the source of microop, x_1=1.

$$D_4T_4: PC \leftarrow AR$$
$$D_5T_5: PC \leftarrow AR$$

Thus, $x_1 = D_4T_4 + D_5T_5$

5-15. The memory unit of the basic computer shown in Fig. 5-3 is to be changed to a 65,536 × 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 x 16 memory = 16 bits

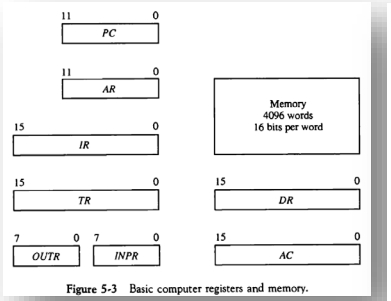
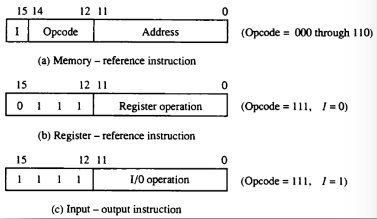


Figure 5-3 Basic computer registers and memory.

Figure 5-5 Basic computer instruction formats.



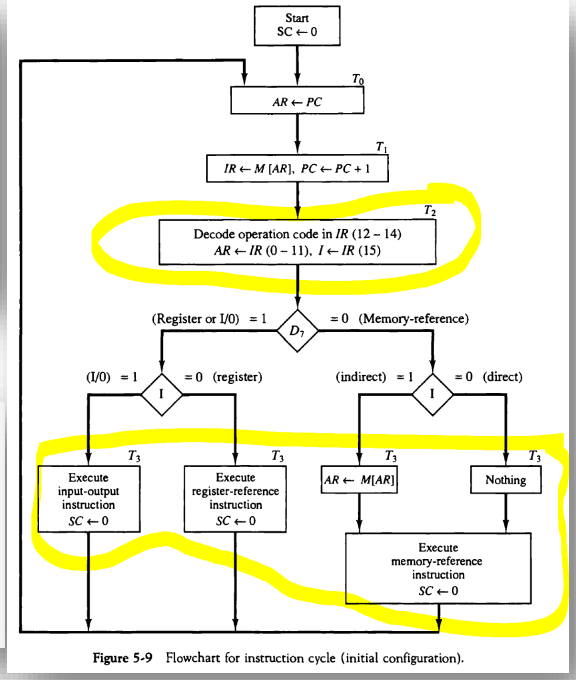
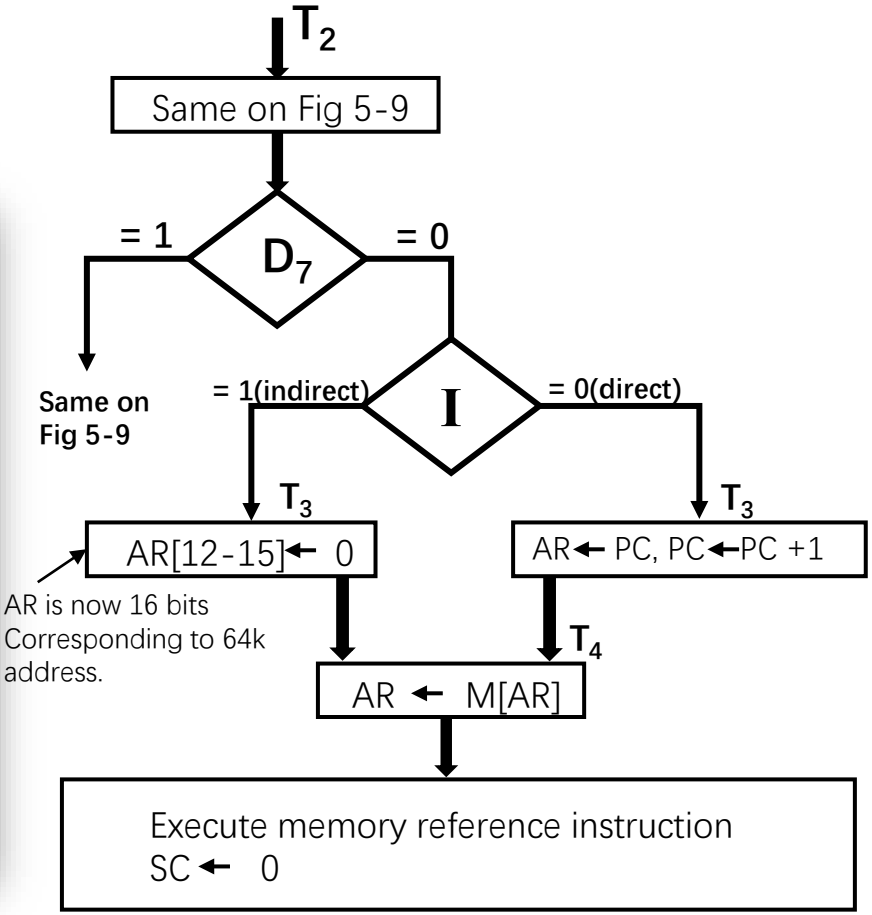
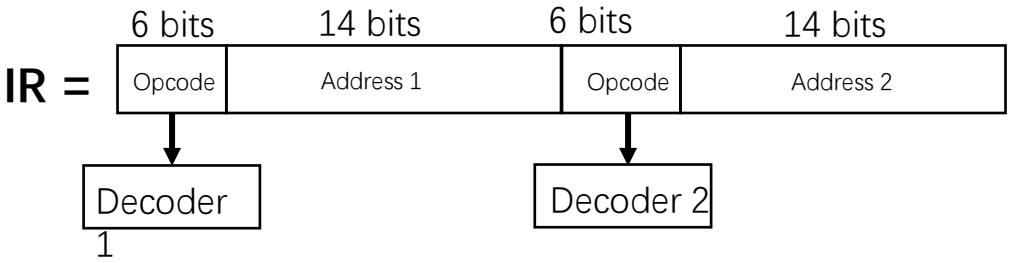


Figure 5-9 Flowchart for instruction cycle (initial configuration).



5-17. A digital computer has a memory unit with a capacity of 16,384 words, 40 bits per word. The instruction code format consists of six bits for the operation part and 14 bits for the address part (no indirect mode bit). Two instructions are packed in one memory word, and a 40-bit instruction register *IR* is available in the control unit. Formulate a procedure for fetching and executing instructions for this computer.



1. Read 40 bits instruction from memory to IR and increment PC
2. Decode Opcode 1
3. Execute instruction 1 using address 1
4. Decode Opcode 2
5. Execute instruction 2 using address 2
6. Go back to step 1