Computer Organization and Architecture

Dr Binu P Chacko
Associate Professor
Department of Computer Science
Prajyoti Niketan College, Pudukad, THRISSUR

Complements

Types (for base r): r's complement, (r-1)'s complement

- 9's complement: Subtract each digit from 9
- 10's complement: 9's complement of the number + 1
- 1's complement: change 1 into 0, and 0 into 1
- 2's complement: 1's complement of the number + 1

Subtraction (using 10's complement, 2's complement)

738 – 503 = 235
738 + 497 = 1235

503 – 738 = -235
503 + 262 = 765 -> Negative 235

1101 – 1010 = 0011
1101 + 0110 = 10011

1010 - 1101
1010 + 0011 = 1101 -> Neg 0011

Fixed-point Representation

Sign bit (leftmost bit): 0 (positive), 1 (negative)

Integer Representation

Positive number: 0 followed by magnitude.

E.g. +14 -> 00001110

Negative number: -14

Signed-magnitude: 1 0001110

Signed-1's complement: 1 1110001

Signed-2's complement: 1 1110010

Addition

```
  +6     00000110          -6     11111010
+13      00001101        +13      00001101
-----    ----------      -----    ----------
+19      00010011         +7    100000111 (discard carry)
-------------------------------------------------------------
 +6      00000110                  -6       11111010
-13      11110011                  -13      11110011
-----    ----------                -----    ----------
-7       11111001 (2's complement of 7)      -19      111101101
```

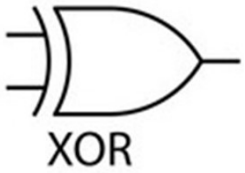
Cont…

Subtraction

(±A) – (+B) = (±A) + (-B)

(±A) – (-B) = (±A) + (+B)

Carries:	0 1	carries:	1 0
+ 70	0 1000110	- 70	1 0111010
+ 80	0 1010000	- 80	1 0110000
---	----------	---	----------
+150	1 0010110	-150	0 1101010



1 (overflow)

Decimal Fixed-point representation: + 375 + (-240) = + 135

(sign)

0 375	0000 0011 0111 0101	
+9 760	1001 0111 0110 0000	(10's complement)
-------	----------	
0 135	0000 0001 0011 0101	(discard carry)

Floating-point Representation

		Fraction	Exponent	(mantissa and exponent
+ 309.48	->	+0.30948	+03	are represented in the register)
		(normalized – MSB is non zero)		
+ 1001.11	->	01001110	000100	
		sign bit		

ANSI 32-bit format

Sign of mantissa

13 = 1101= 0.1101 x 2^4 = 00000100 11010000 00000000 00000000

-17 = -10001 = -0.10001 x 2^5 = 10000101 10001000 00000000 00000000

-0.125 = -0.001 = -0.1 x 2^{-2} = 11111110 10000000 00000000 00000000

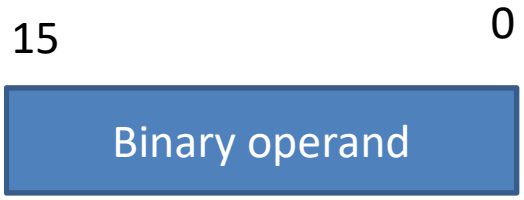
2's complement

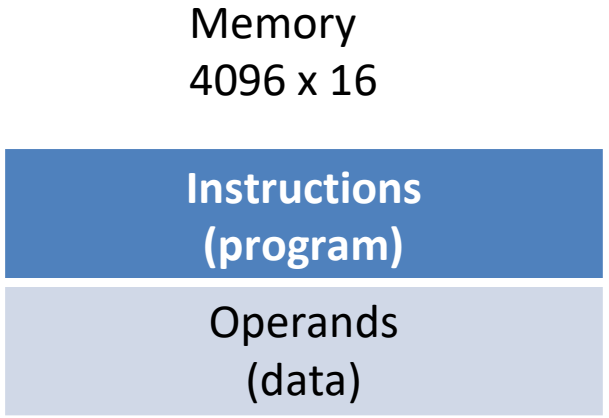
Instruction Codes

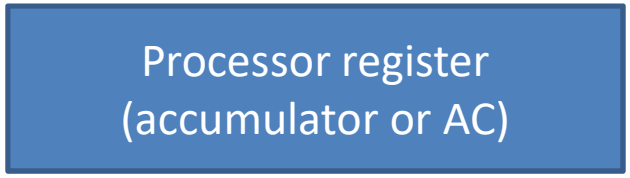
- Computer organization is defined by its internal registers, timing and control structure, and the set of instructions that it uses
- A computer is capable of executing various micro-operations, and can be instructed as to what specific sequence of operations it must perform
- A computer instruction is a binary code that specifies a sequence of micro-operations for the computer
- Stored program concept: ability to store and execute instructions
- Instruction code = opcode + operands
- The number of bits required for the operation code depends on the total number of operations available in the computer
- Computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations needed for the h/w implementation of the operation

Stored Program Organization

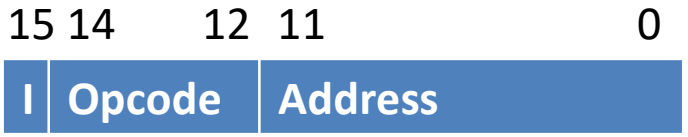
Instruction format



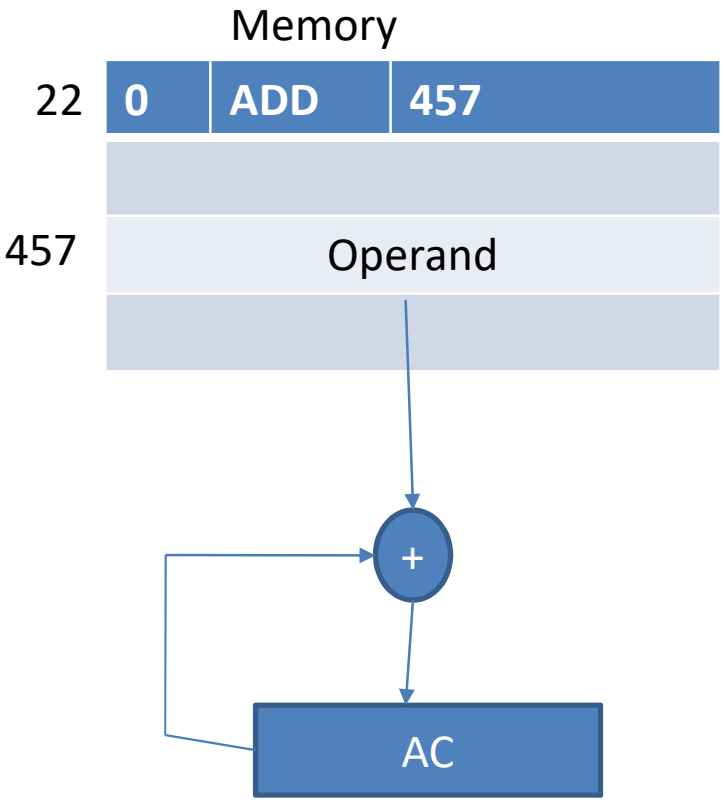




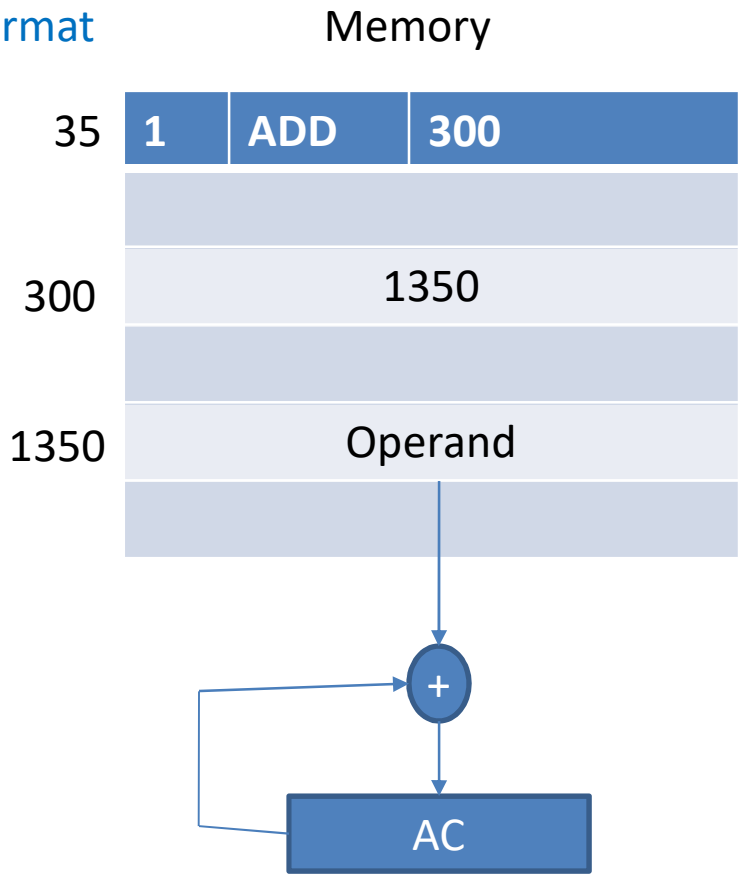
Indirect Address



Instruction format



Direct address

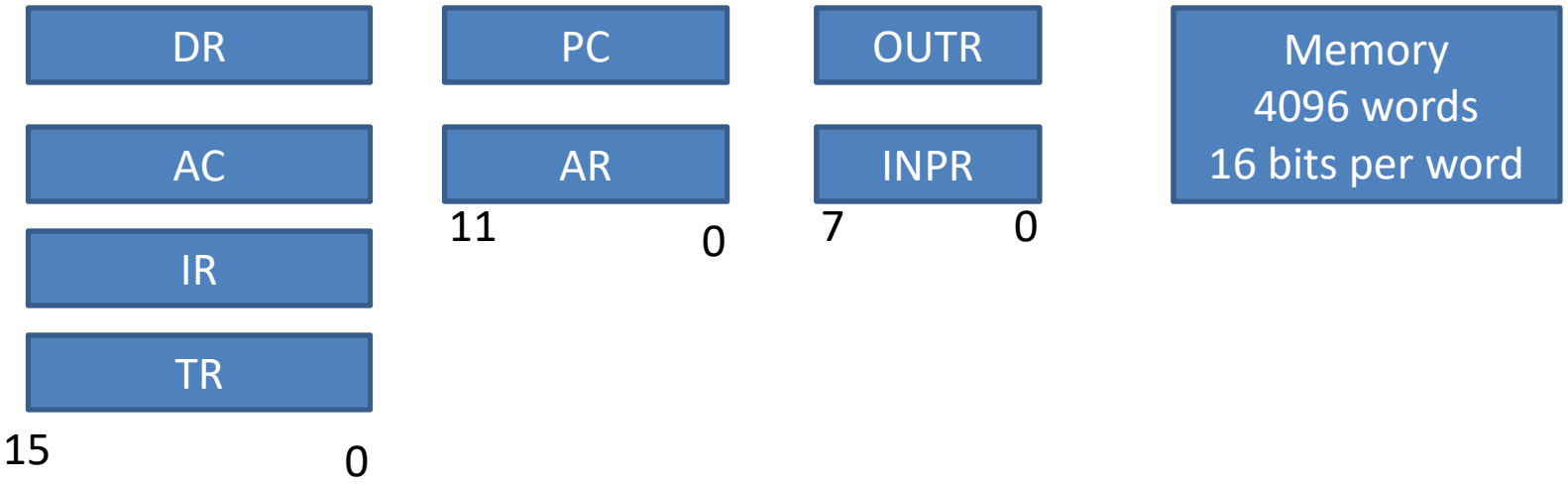


Indirect address

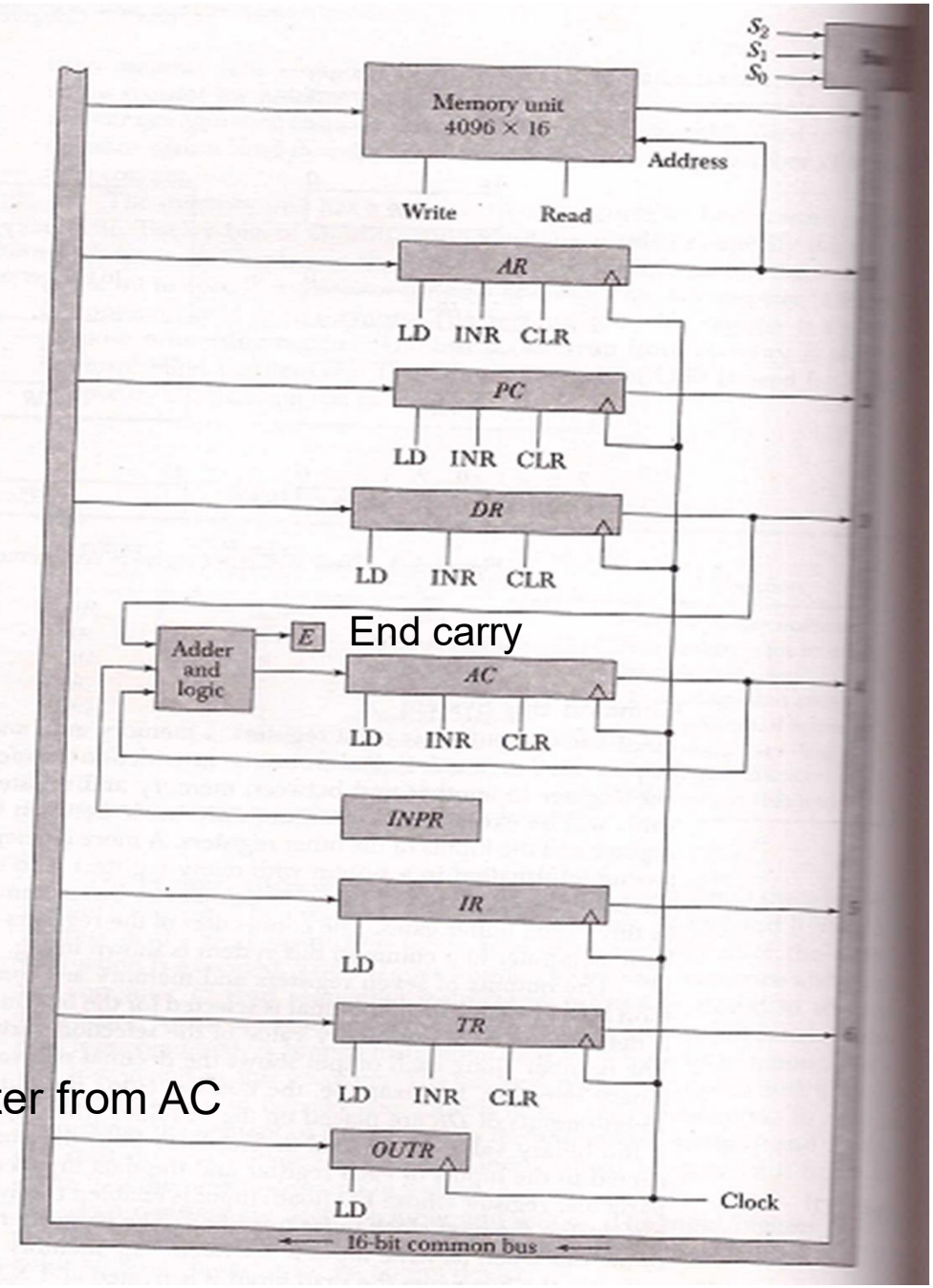
- Effective address 457, 1350

Computer Registers

Register symbol	No 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 during processing
INPR	8	Input register	Holds 8 bit character from an I/P device
OUTR	8	Output register	Holds 8 bit character for an O/P device



Common Bus System

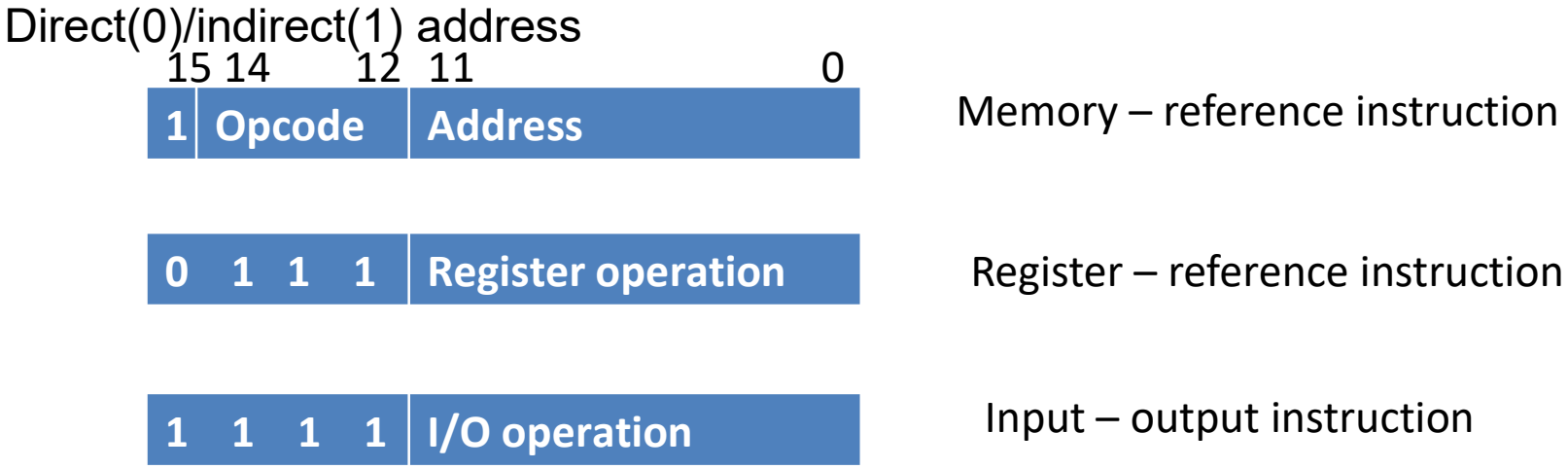


Character from AC

INR is performed using a counter

Computer Instructions

- Instruction code formats



Symbol	Description
INP	Input character to AC
OUT	Output character from AC
SKI	Skip on input flag
SKO	Skip on output flag
ION	Interrupt on
IOF	Interrupt off

Cont…

Symbol	Description
AND	AND memory word to AC
ADD	ADD memory word to AC
LDA	Load memory word to AC
STA	Store content of AC in memory
BUN	Branch unconditionally
BSA	Branch and save return address
ISZ	Increment and skip if zero
CLA	Clear AC
CLE	Clear E
CMA	Complement AC
CME	Complement E
CIR	Circulate right AC and E
CIL	Circulate left AC and E
INC	Increment AC
SPA	Skip next instruction if AC positive
SNA	Skip next instruction if AC negative
SZA	Skip next instruction if AC zero
SZE	Skip next instruction if E zero
HLT	Halt computer

Memory-reference instructions

Register-reference instructions

- 4 digit Hexcode/16-bit binary code for each instruction

Cont…

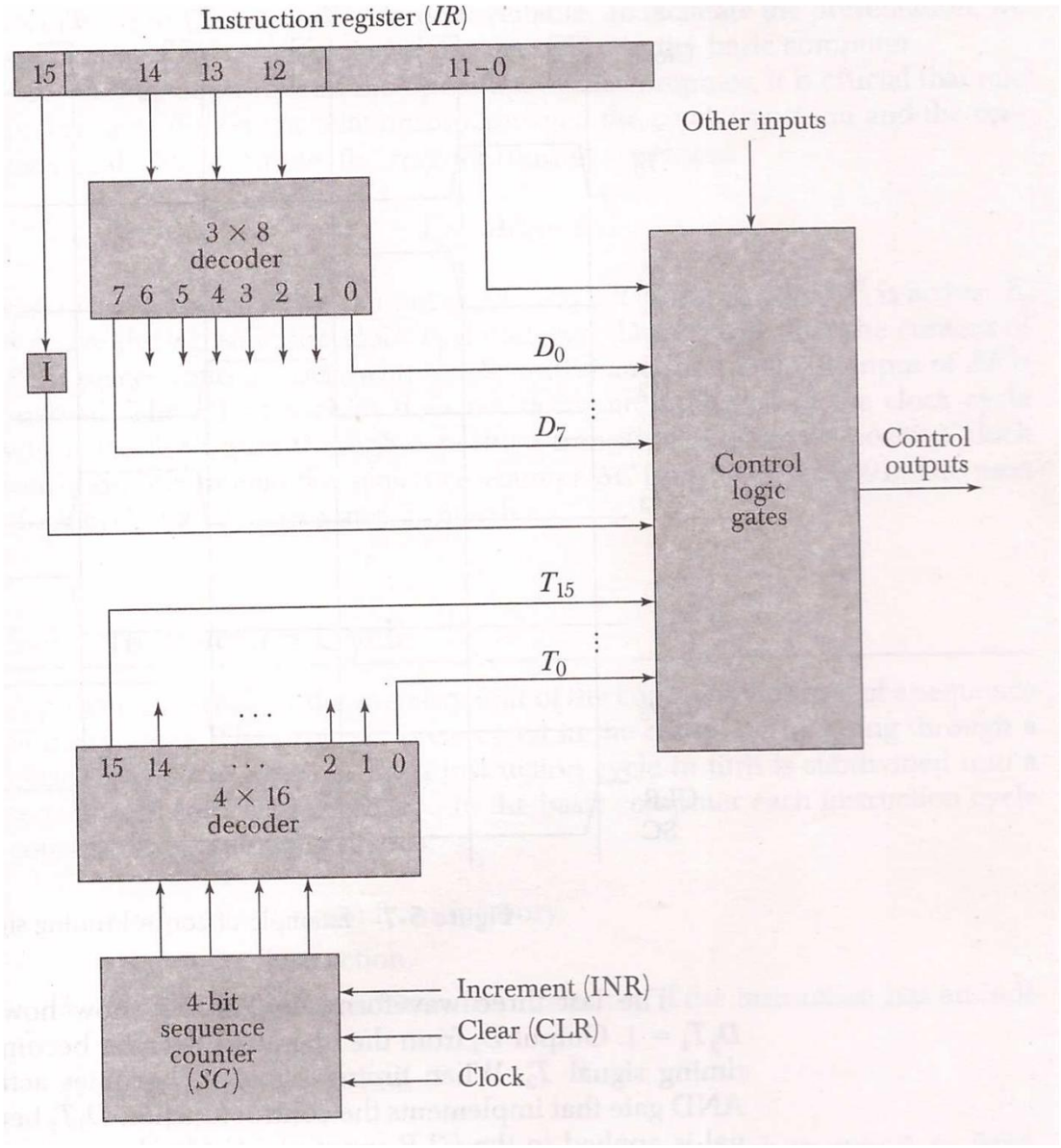
Instructions are categorized into

- Arithmetic, logical, and shift instructions
- Instructions for moving information to and from memory and processor registers – binary information is stored in memory and computations are done in registers
- Program control instructions (branch instructions) together with instructions that check status conditions
- Input and output instructions

Timing and Control

- Timing for all registers is controlled by a master clock generator
- Clock pulses change the state of a register when the register is enabled by a control signal
- Control signals are generated in the control unit
- They provide control inputs for multiplexers, registers, and microoperations for the accumulator
- Two types of control organization: hardwired control and microprogrammed control
- Hardwired organization: control logic is implemented with gates, flip-flops, decoders, and other digital circuits
- Modification requires changes in wiring among various components
- Microprogrammed organization: control information is stored in a control memory. This memory is programmed to initiate the required sequence of microoperations
- Modifications can be done by updating the microprogram

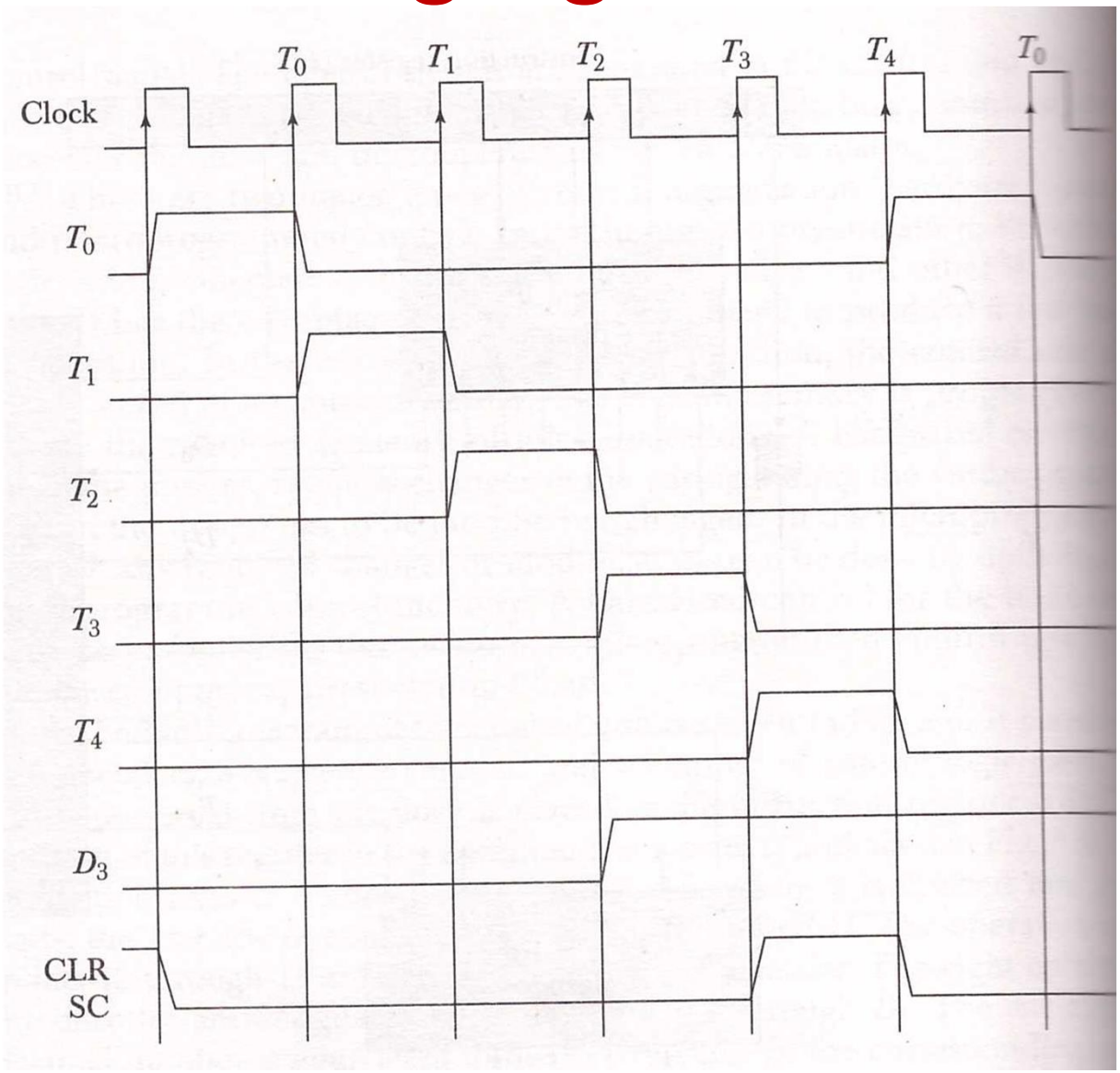
Control Unit



Control Timing Signal

- D_3T_4: SC <- 0

SC is cleared when D_3T_4=1



Instruction Cycle

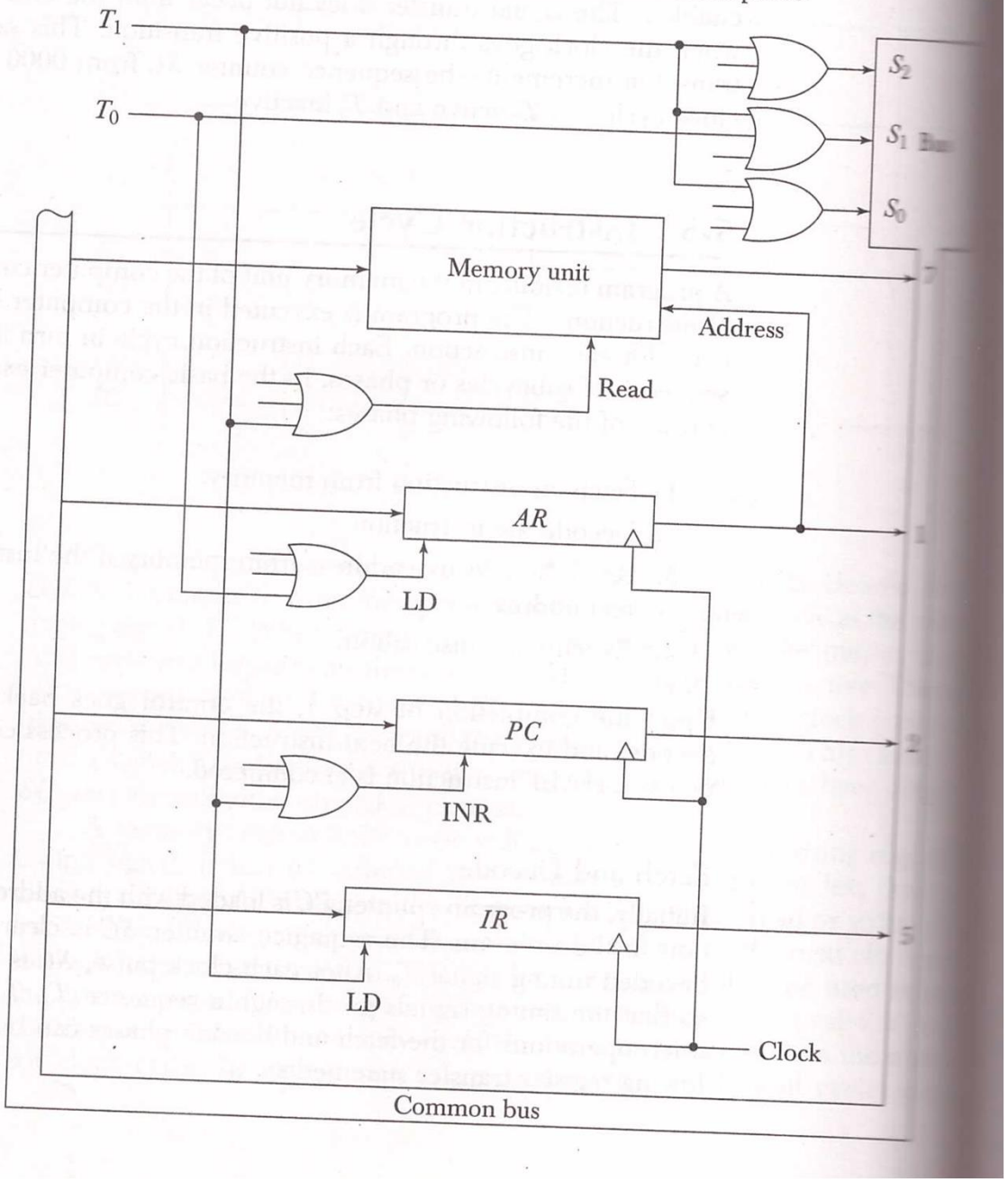
- Fetch the instruction from the memory
- Decode the instruction
- Read effective address from memory if the instruction has an indirect address
- Execute the instruction
- Fetch and decode: Initially PC is loaded with address of the first instruction. SC is cleared to 0. After each clock pulse, SC is incremented by 1 so that timing signals go through the sequence T_0, T_1, T_2, and so on.
- microoperations

T_0 : AR <- PC

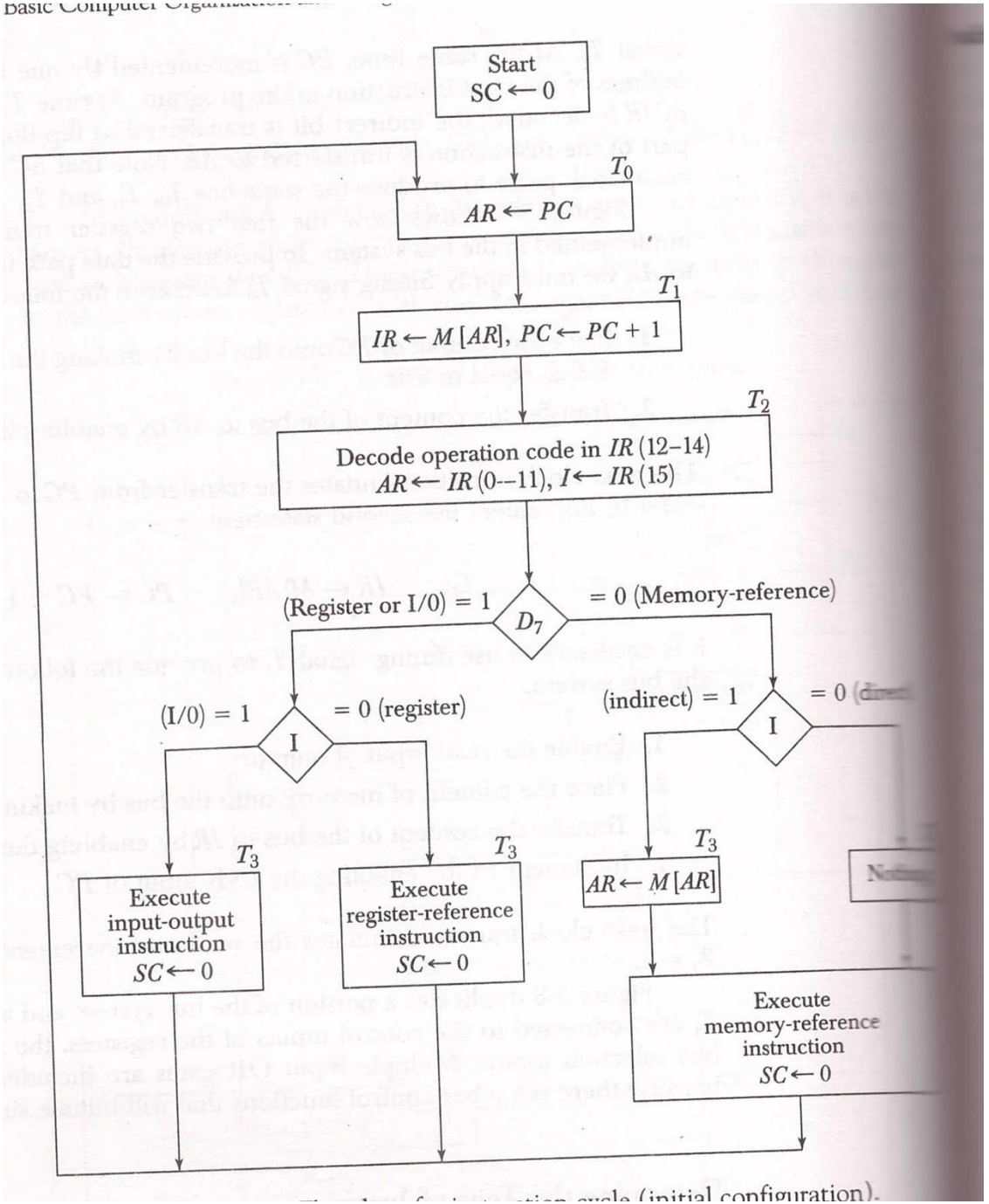
T_1 : IR <- M[AR], PC <- PC + 1

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

Register transfers for the fetch phase.



- Determine the type of instruction in T_3



- Register-reference instructions use bits 0 to 11 to specify one of the 12 instructions
- These instructions are executed in T_3

TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC^* \rightarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Memory Reference Instructions

- Execution starts at T_4

TABLE 5-4 Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

Cont…

- AND to AC

D_0T_4 : DR <- M[AR]

D_0T_5 : AC <- AC Λ DR, SC <- 0

- ADD to AC

D_1T_4 : DR <- M[AR]

D_1T_5 : AC <- AC + DR, E <- C_{out}, SC <- 0

- Output carry C_{out} is transferred to E (extended accumulator) flip flop
- LDA

D_2T_4 : DR <- M[AR]

D_2T_5 : AC <- DR, SC <- 0

- STA

D_3T_4 : M[AR] <- AC, SC <- 0

Cont…

- BUN

D_4T_4 : PC <- AR, SC <- 0

- BSA

D_5T_4 : M[AR] <- PC, AR <- AR + 1

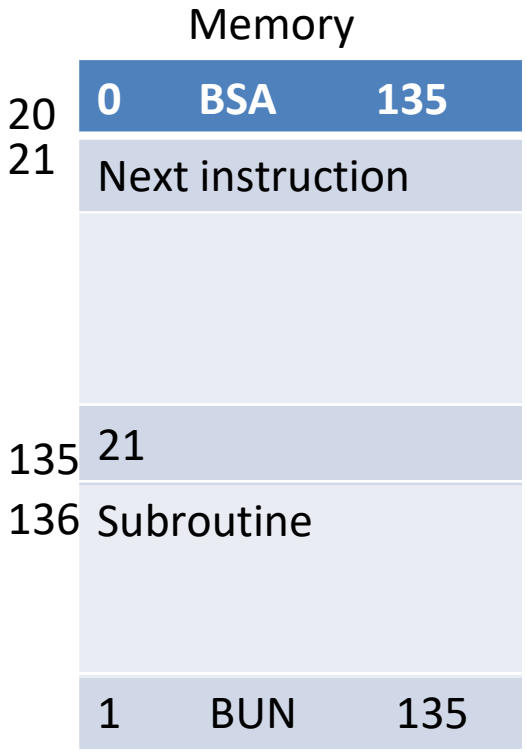
D_5T_5 : PC <- AR, SC <- 0

- ISZ

D_6T_4 : DR <- M[AR]

D_6T_5 : DR <- DR + 1

D_6T_6 : M[AR] <- DR, if (DR = 0) then (PC <- PC + 1), SC <- 0



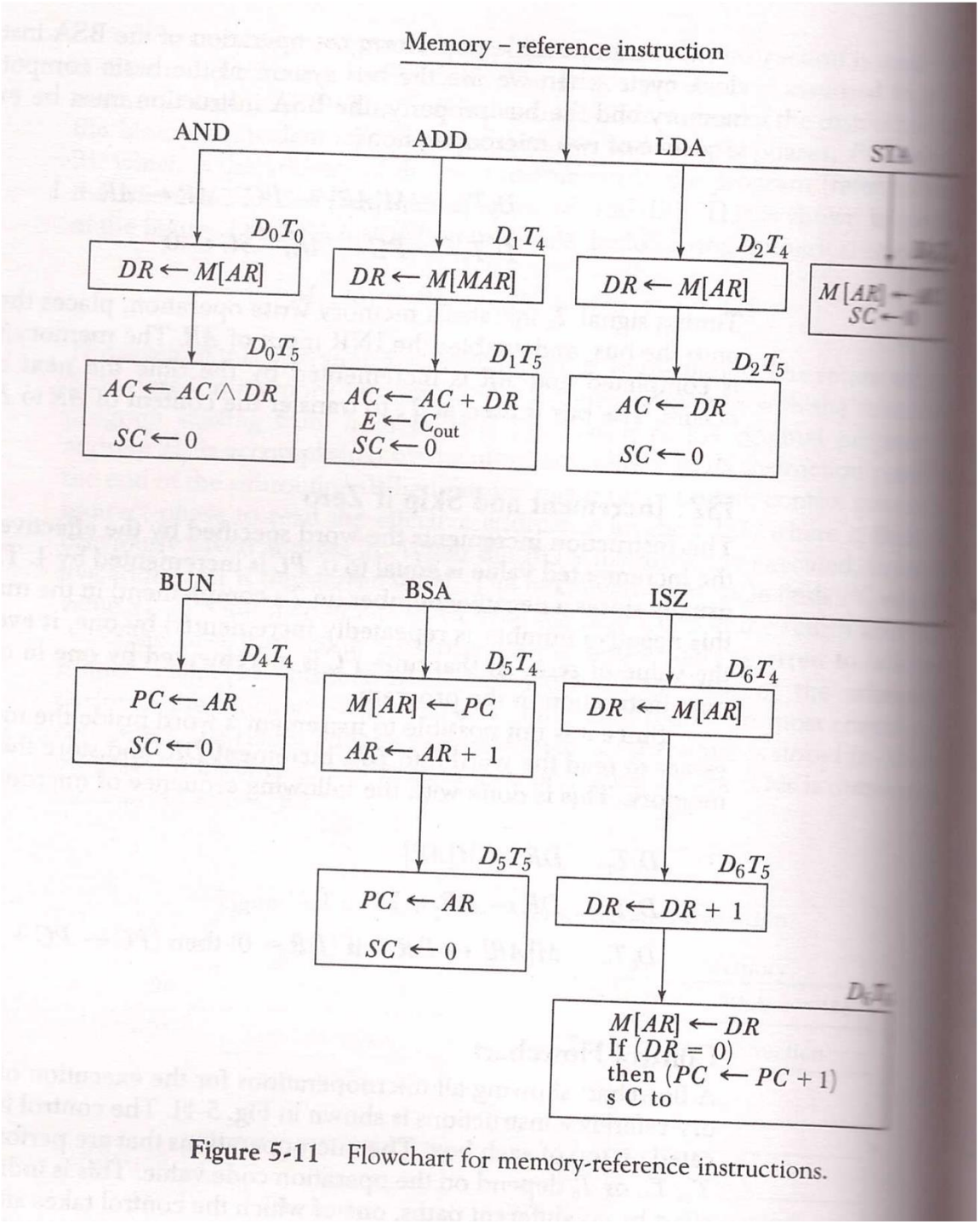
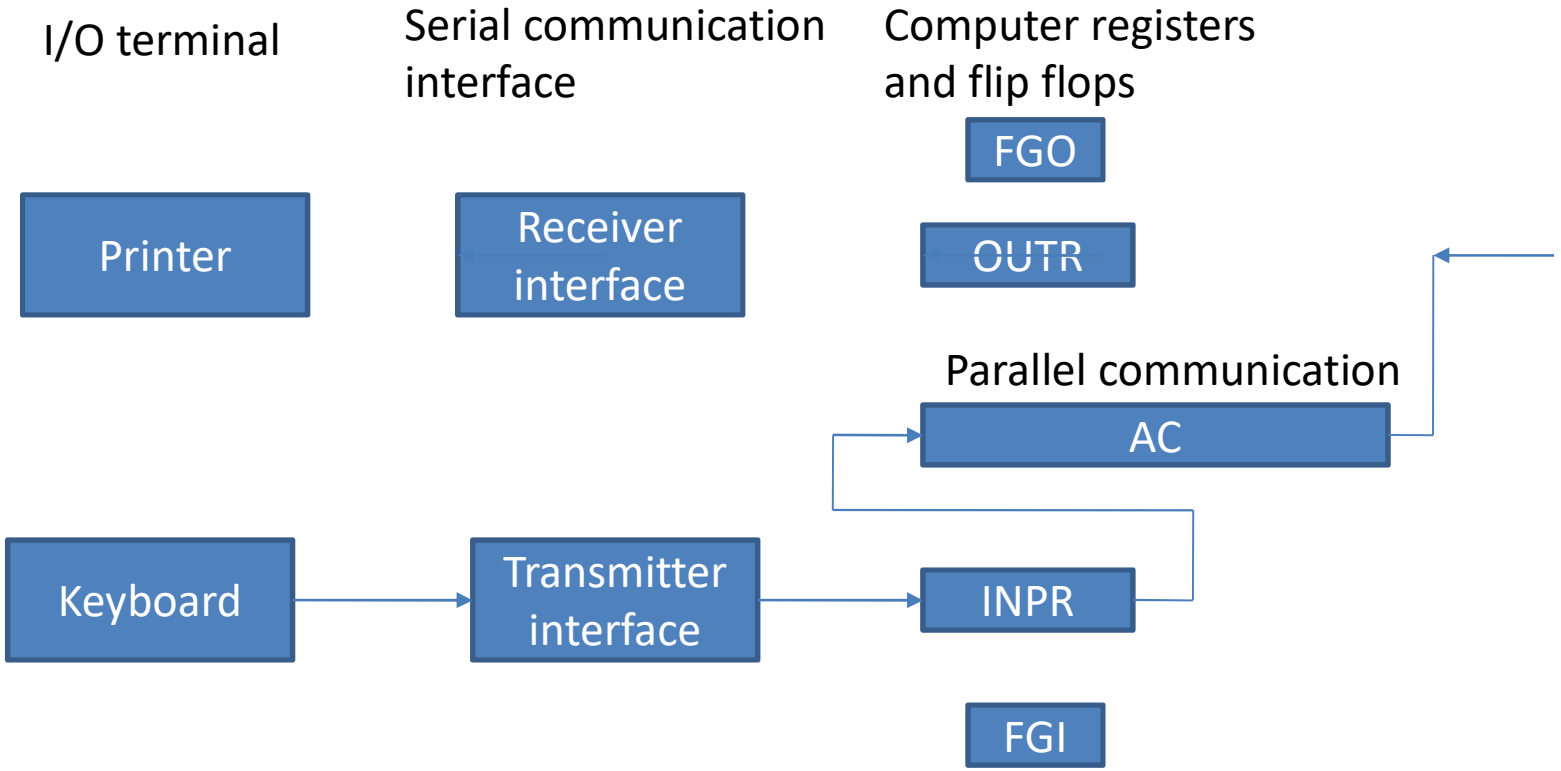


Figure 5-11 Flowchart for memory-reference instructions.

I/O Configuration



- Initially FGI is cleared to 0. When a key is struck in the keyboard, 8-bit alphanumeric code is shifted into INPR serially and FGI is set to 1. Then, the information from INPR is transferred in parallel into AC and FGI is cleared to 0
- Initially FGO is set to 1. The information from AC is transferred in parallel to OUTR and FGO is cleared to 0. Printer accepts the information and prints it, and then sets FGO to 1

I/O Instructions

$D_7IT_3 = p$ (common to all I/O instructions) $IR(i) = B_i$ [bit in IR(6 – 11) that specifies the instruction}			
	p	SC <- 0	Clear SC
INP	pB_{11}	AC(0 – 7) <- INPR, FGI <- 0	Input character
OUT	pB_{10}	OUTR <- AC(0 – 7), FGO <- 0	Output character
SKI	pB_9	If (FGI = 1) then (PC <- PC + 1)	Skip on input flag
SKO	pB_8	If (FGO = 1) then (PC <- PC + 1)	Skip on output flag
ION	pB_7	IEN <- 1	Interrupt enable on
IOF	pB_6	IEN <- 0	Interrupt enable off

I/O instructions are used to transfer data to and from AC, to check flag bits and for controlling the interrupt facility

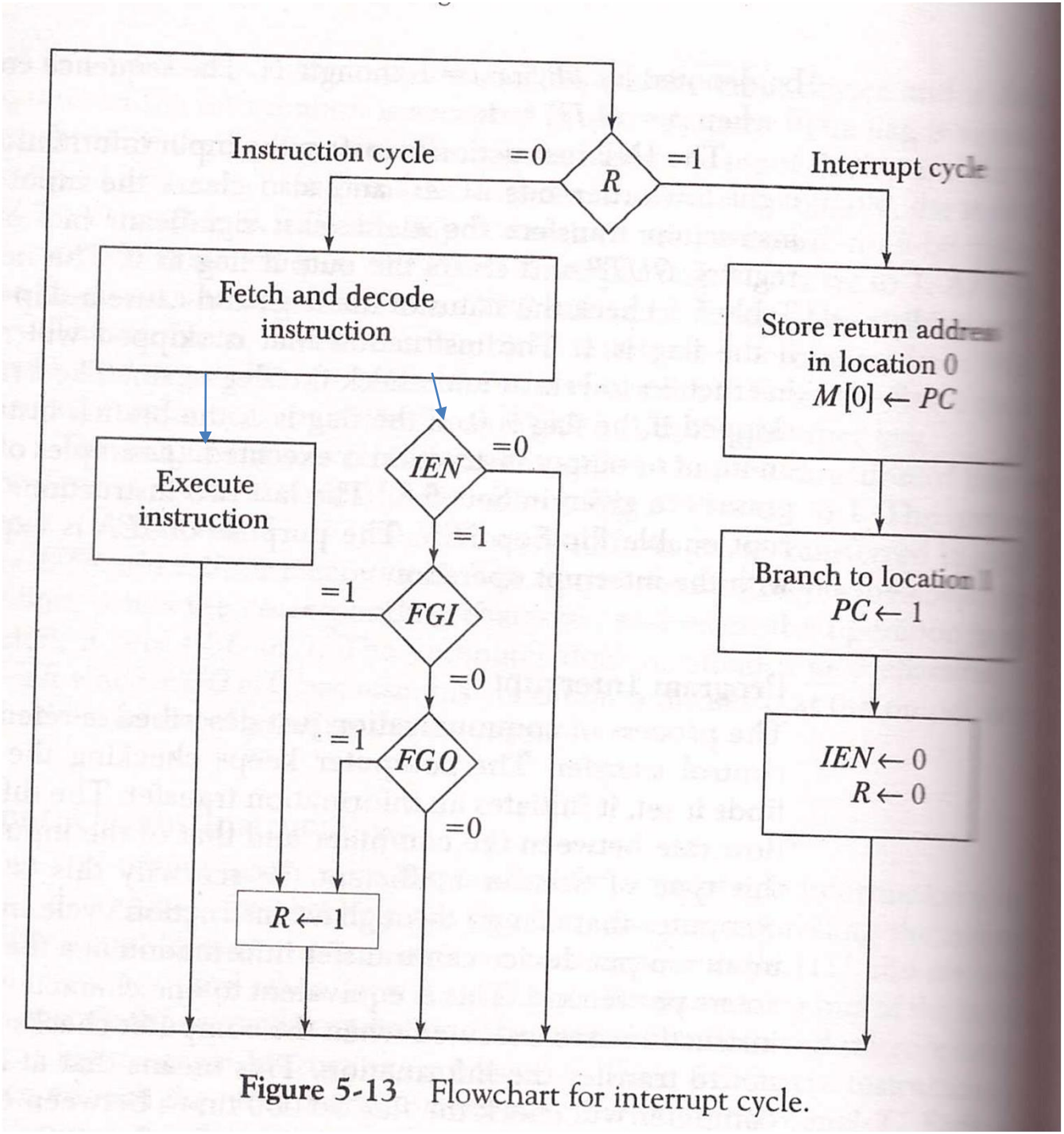
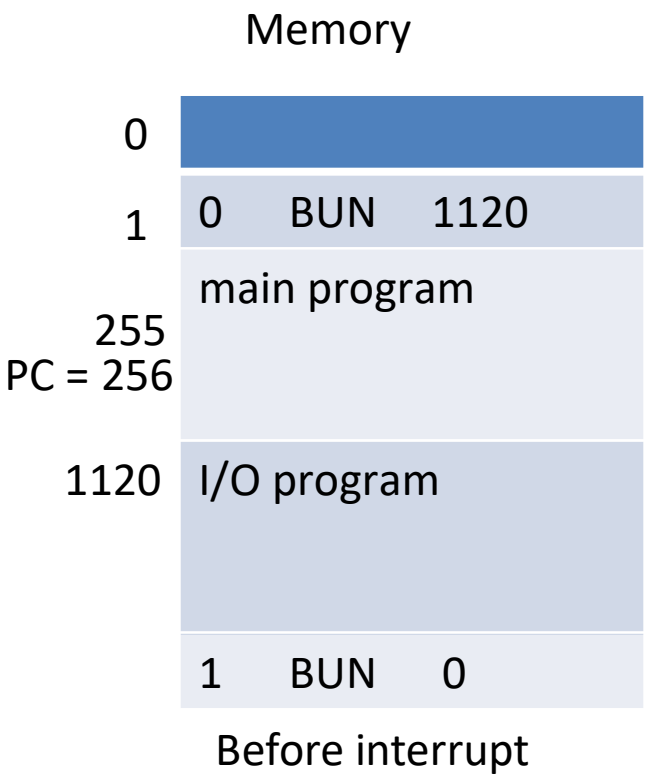
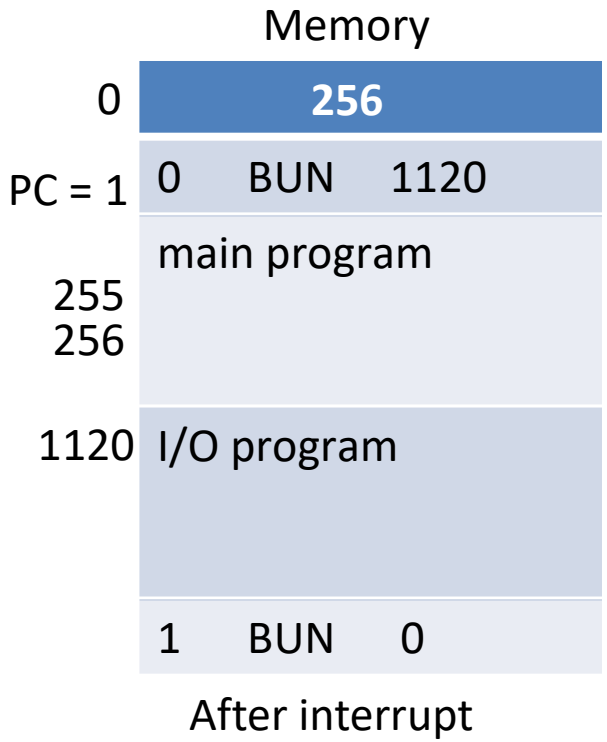


Figure 5-13 Flowchart for interrupt cycle.

Interrupt Cycle



Cont…



Register transfer statements

Condition for setting flip flop R to 1

$T_0'T_1'T_2'(IEN)(FGI + FGO)$: R <- 1

RT_0: AR <- 0, TR <- PC

RT_1: M[AR] <- TR, PC <- 0

RT_2: PC <- PC + 1, IEN <- 0, R <- 0, SC <- 0

Computer

Components

- Memory unit
- Registers: AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
- Flip-flops: I, S, E, R, IEN, FGI, FGO
- Decoders: 3 x 8 operation decoder, 4 x 16 timing decoder
- 16 bit common bus
- Control logic gates
- Adder and logic circuit

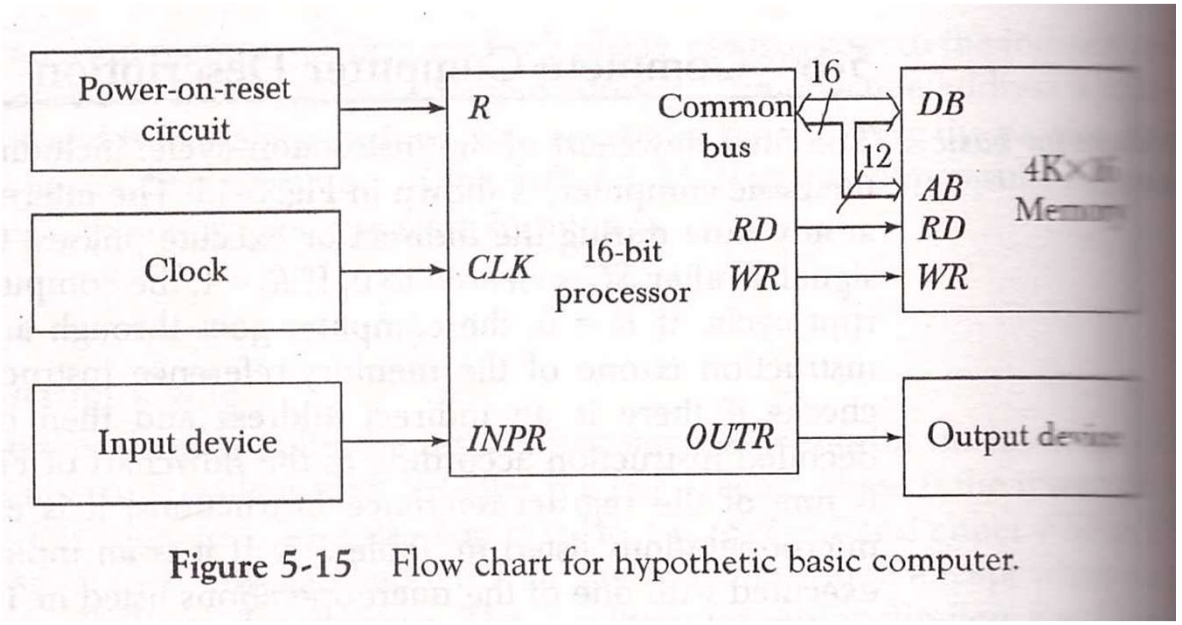


Figure 5-15 Flow chart for hypothetic basic computer.

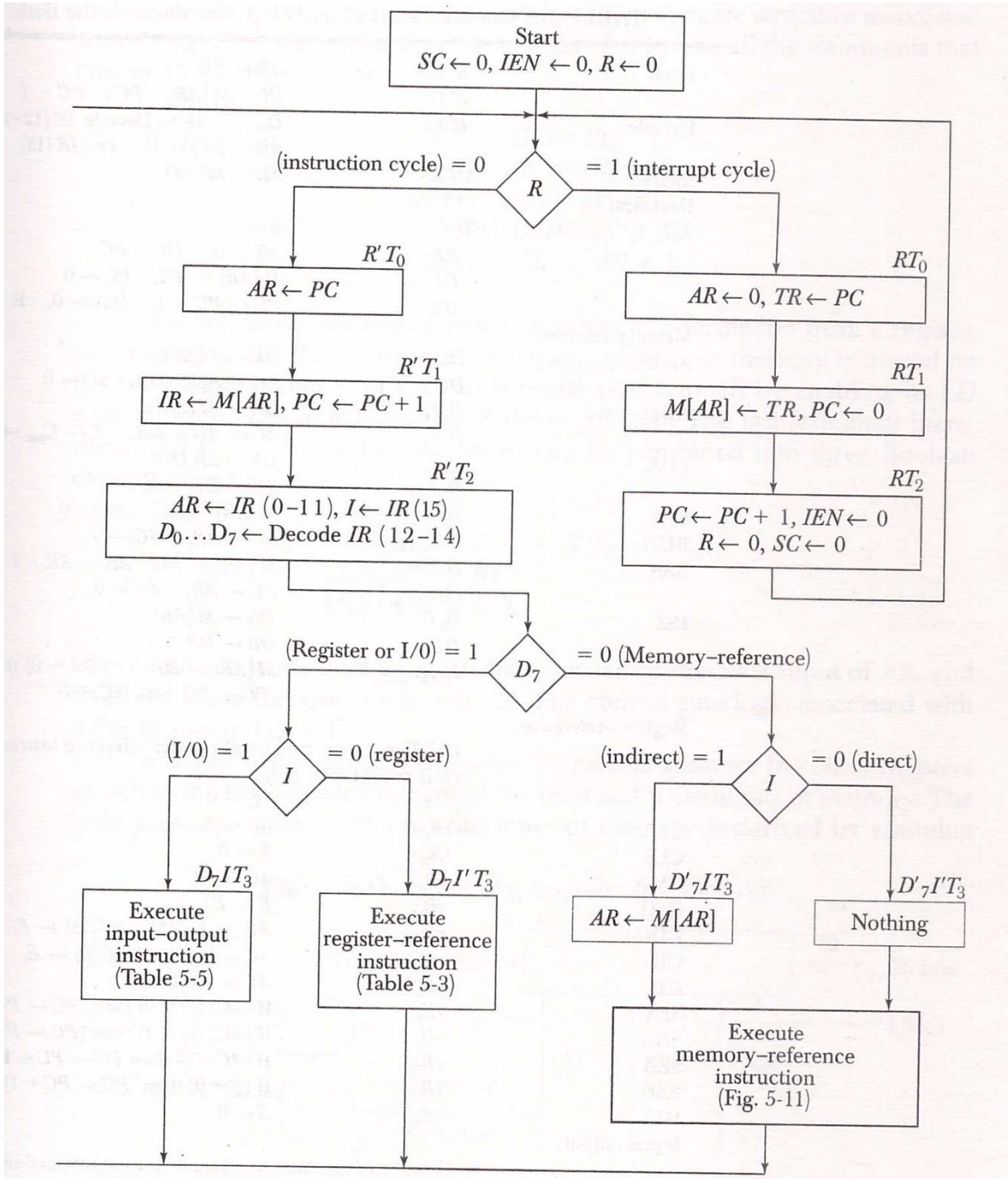


Figure 5-16 Flowchart for computer operation.

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	$D_7'IT_3$:	$AR \leftarrow M[AR]$
Interrupt:		
$T_0T_1'T_2'(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	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:	$AC \leftarrow AC \wedge DR$, $SC \rightarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out} \rightarrow SC \leftarrow 0$
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 instruct ions	
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	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

Statements that change the contents of AR

$R'T_0$: AR <- PC

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

$D_7'IT_3$: AR <- M[AR]

RT_0: AR <- 0

D_5T_4: AR <- AR + 1

Control I/P of registers

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

$CLR(AR) = RT_0$

$INR(AR) = D_5T_4$

Read I/P from Memory (<-M[AR])

$Read = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$

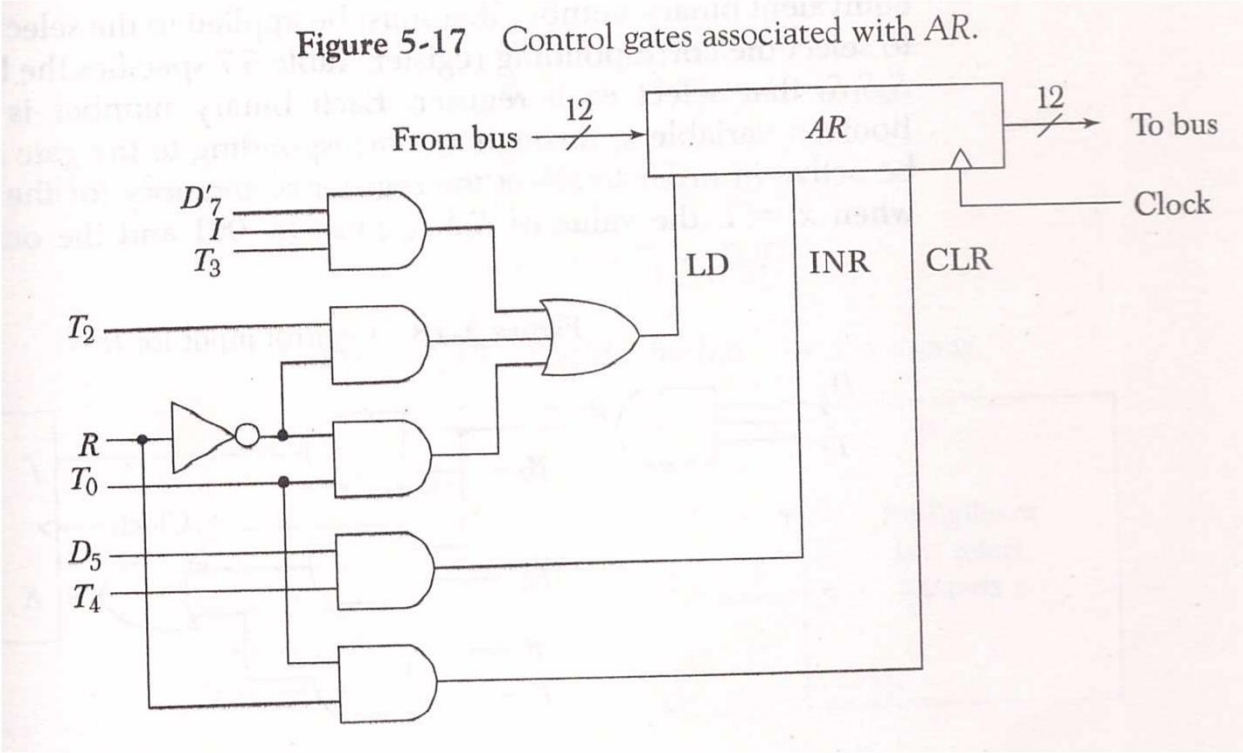


Figure 5-17 Control gates associated with AR.

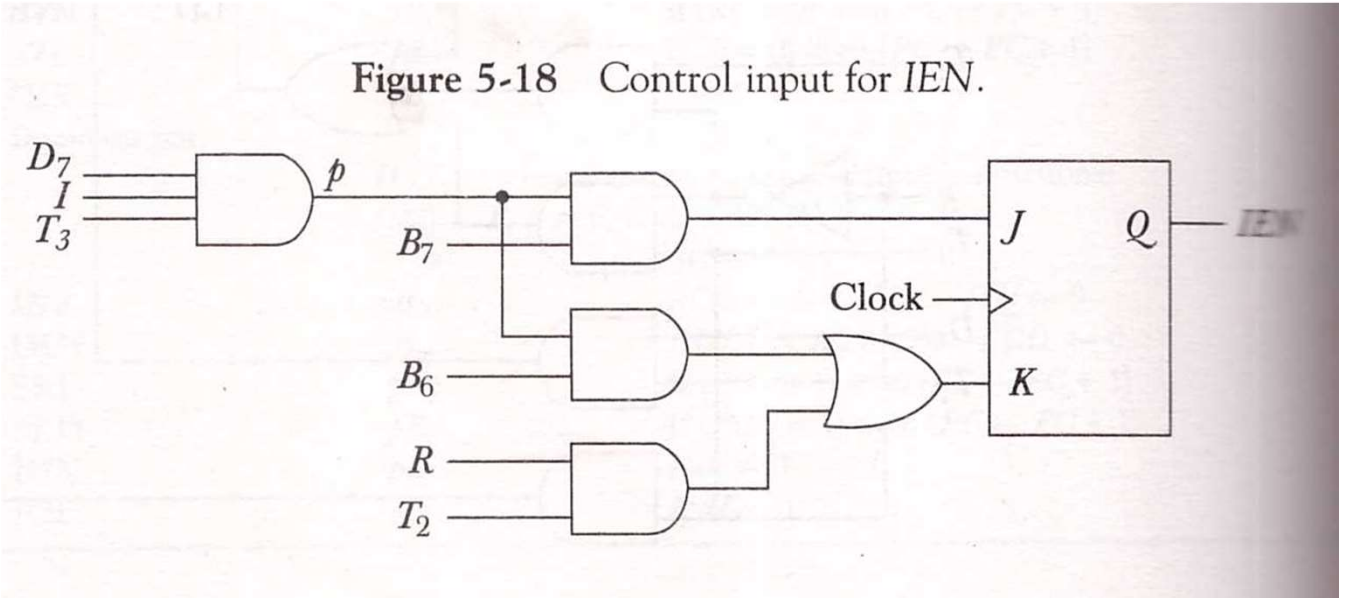
Control of Flip-flops

pB_7: IEN <- 1 ION

pB_6: IEN <- 0 IOF

RT_2: IEN <- 0 at the end of interrupt cycle

where p = D_7IT_3 and B_6 and B_7 are bits 6 and 7 of IR



Control of Common Bus

Encoder for bus selection circuit										
Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

Cont…

Boolean functions for the encoder

$S_0 = x_1 + x_3 + x_5 + x_7$

$S_1 = x_2 + x_3 + x_6 + x_7$

$S_2 = x_4 + x_5 + x_6 + x_7$

Logic that makes $x_1 = 1$ (statements that have AR as source)

D_4T_4 : PC <- AR

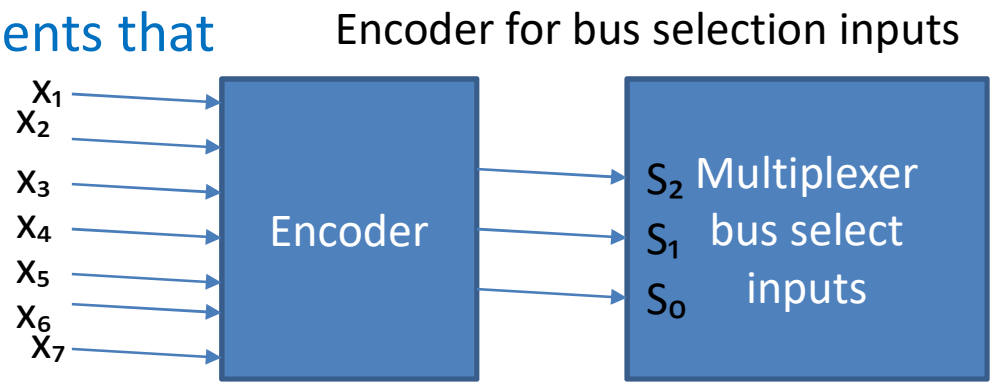
D_5T_5 : PC <- AR

Boolean function for x_1

$x_1 = D_4T_4 + D_5T_5$

Boolean function for x_7 (same as memory read)

$x_7 = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$



Design of Accumulator Logic

Register transfer statements

D_0T_5 : AC <- AC Λ DR

D_1T_5 : AC <- AC + DR

D_2T_5 : AC <- DR

pB_{11} : AC(0-7) <- INPR

rB_{11} : AC <- 0

rB_9 : AC <- $\overline{AC}$

rB_7 : AC <- shr AC, AC(15) <- E

rB_6 : AC <- shl AC, AC(0) <- E

rB_5 : AC <- AC + 1

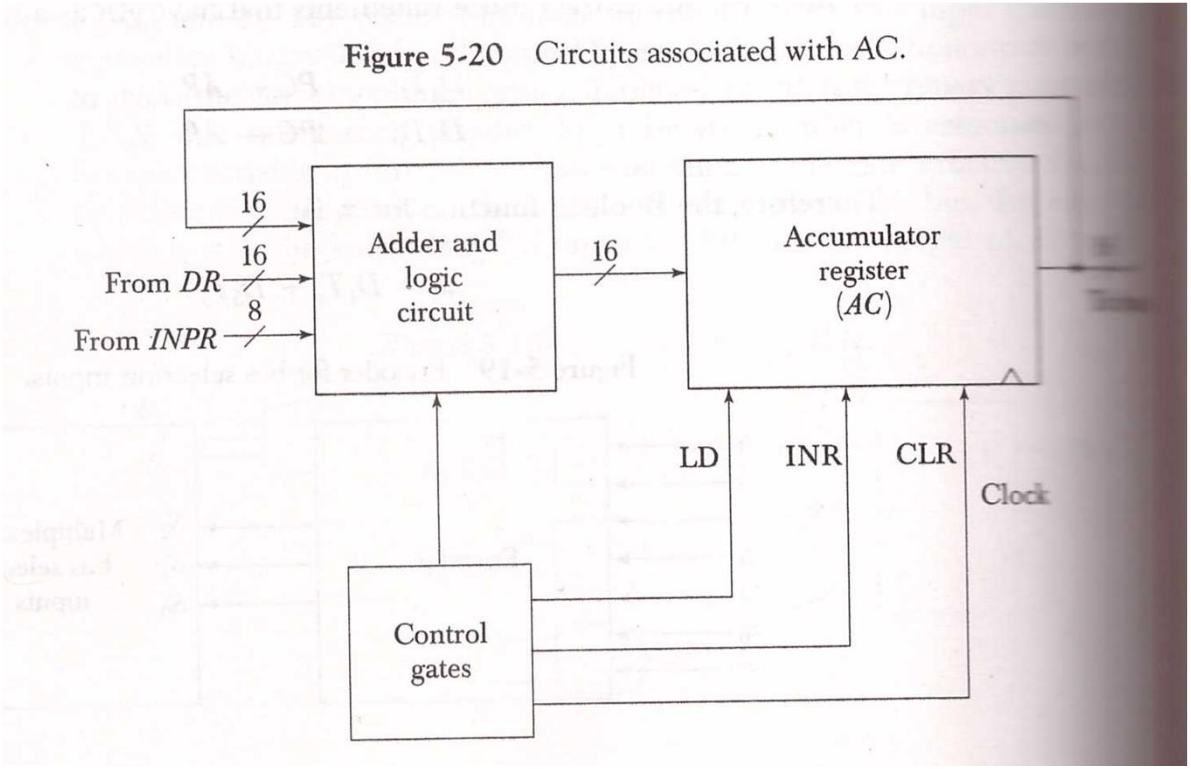
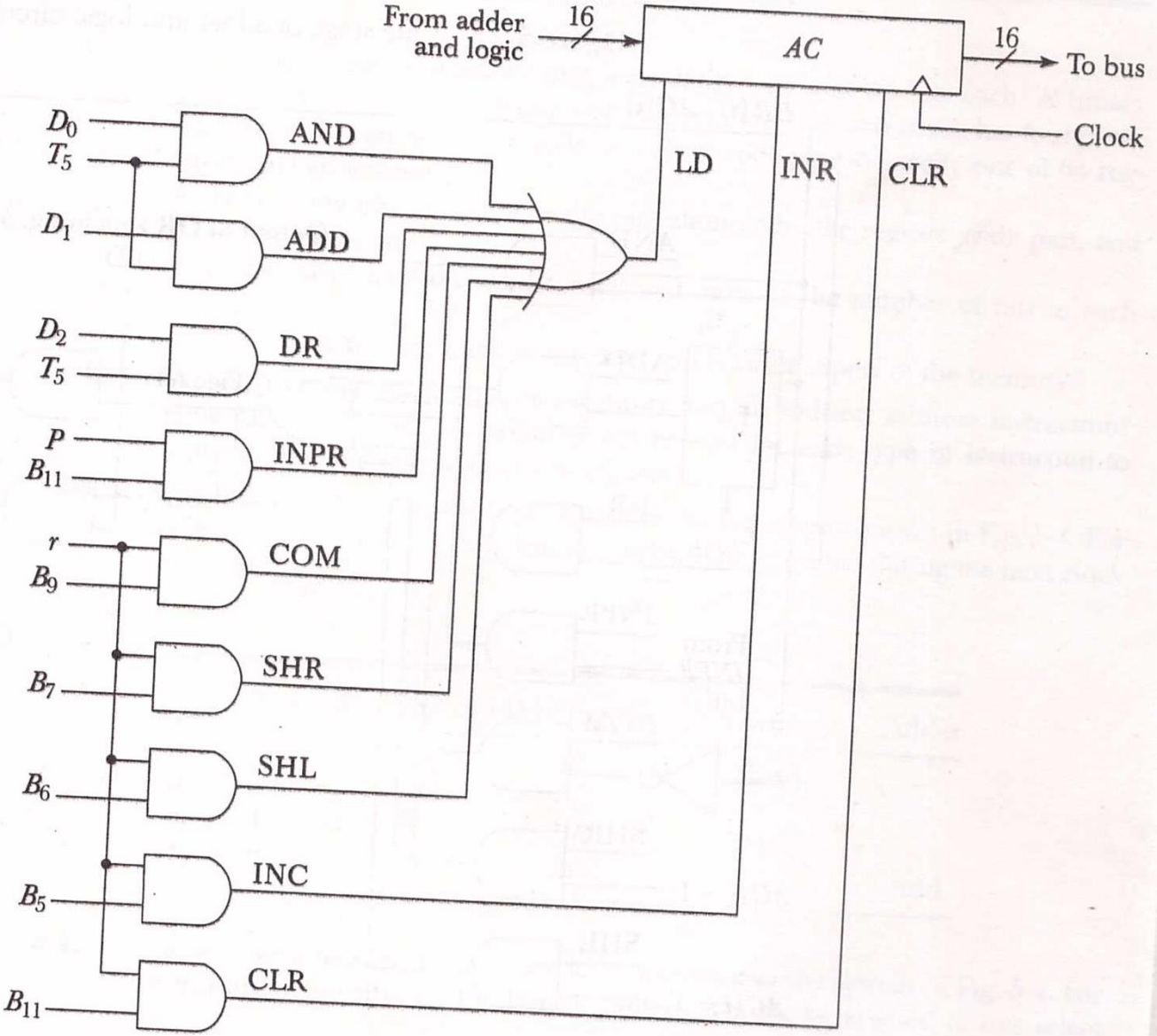


Figure 5-20 Circuits associated with AC.

Figure 5-21 Gate structure for controlling the LD, INR, and CLR of AC.



Adder & Logic Circuit

- Adder & logic circuit is divided into 16 stages, with each state corr: to one bit of AC
- When LD is enabled, 16 inputs I_i (i = 0, 1, …, 15) are transferred to AC(0-15)
- Inputs of the gates with symbolic names come from the outputs of gates marked with the same symbolic name in Fig 5.21

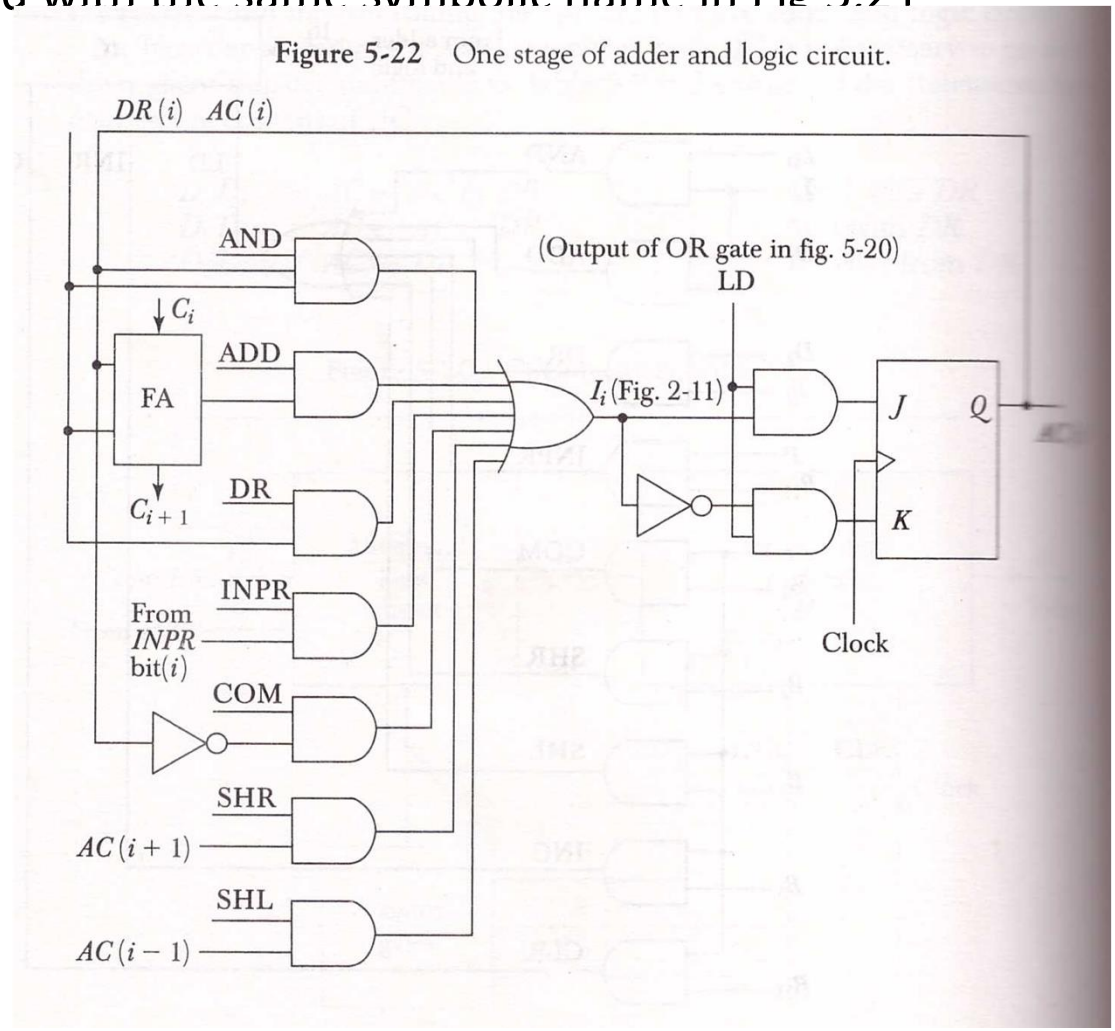


Figure 5-22 One stage of adder and logic circuit.

Microprogrammed control

- Function of a CU is to initiate a sequence of microoperations

Methods of implementing a CU

- Hardwired control: control signals are generated by hardware
- Use of fixed instructions, fixed logic blocks of and/or arrays, encoder, decoder
- High speed operation, expensive, relatively complex, no flexibility
- E.g. Intel 8085, Motorola 6802, Zilog 80, RISC CPUs
- Microprogrammed control: A CU whose binary control variables are stored in memory. An elegant and systematic method for controlling the microoperation sequences
- The control function that specifies a microoperation is a binary variable. Control variables are represented by control words. Each word in control memory (part of CU) contains a *microinstruction* (specifies one or more microoperations). A sequence of microinstructions constitutes a microprogram
- E.g. Intel 8080, Motorola 68000, CISC CPUs

Cont…

- Dynamic microprogramming permits a microprogram to be loaded from a magnetic disk
- Each machine instruction initiates a series of microinstructions
- Microinstructions generate microoperations to fetch the instruction from *main memory*; to evaluate effective address, to execute the operation specified by the instruction, and to return control to the fetch phase to repeat the cycle for the next instruction
- Microinstruction contains bits for initiating microoperations and bits that determine the address sequence for the *control memory*
- Functions of microprogram sequencer : increment CAR by 1, load CAR with an address from control memory, transfer an external address, or load an initial address to start the control operations
- CDR (also called pipeline register) allows the execution of microoperations simultaneously with the generation of the next microinstruction
- Adv : The H/W configuration need not be changed for different operations; instead specify a different set of microinstructions for control memory

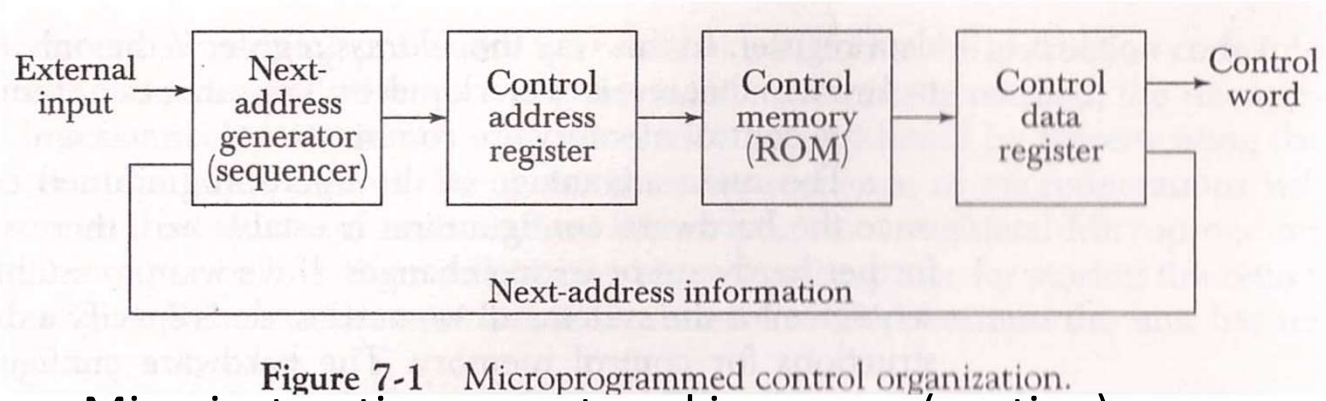


Figure 7-1 Microprogrammed control organization.

Address sequencing

- Microinstructions are stored in groups (routine)
- Each computer instruction has its own microprogram routine to generate microoperations that execute the instruction
- An initial address is loaded into CAR when power is on. This address is the address o the first microinstruction that activates the **instruction fetch routine**. The fetch routine may be sequenced by incrementing CAR. At the end of the fetch routine, instruction is in IR
- **Determine effective address of the operand** – addressing modes. This routine can be reached through a branch microinstruction, which is conditioned on the status of mode bits of the instruction. When this routine is completed, address of the operand is in MAR
- **Generate microoperations to execute the instruction**. Mapping – a rule that transforms the instruction code into a control memory address where routine is located. Microprograms that employ subroutines will require an external register for storing the return address ● At the end of the execution of the instruction, **return to fetch routine** by executing an unconditional branch instruction

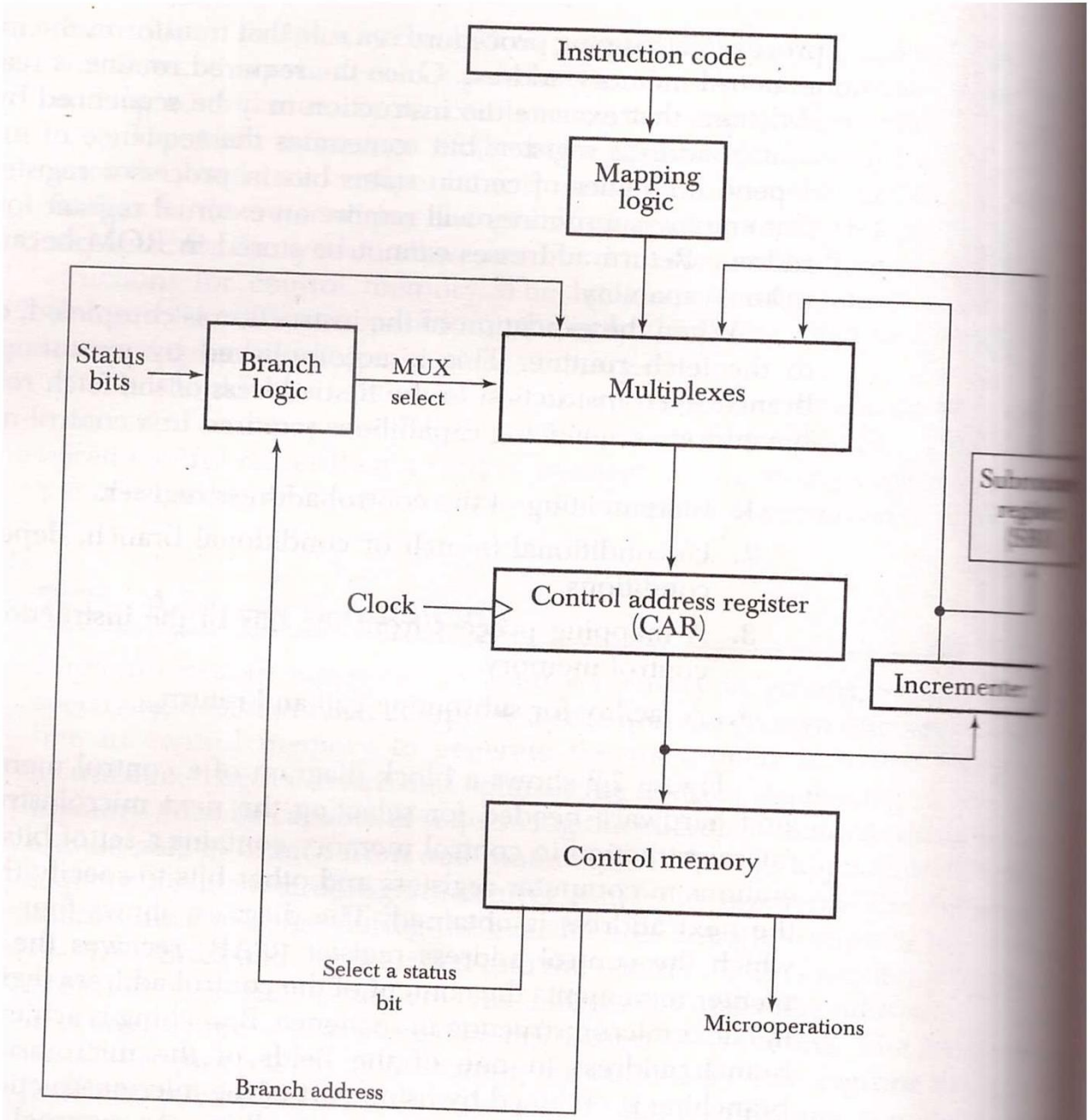


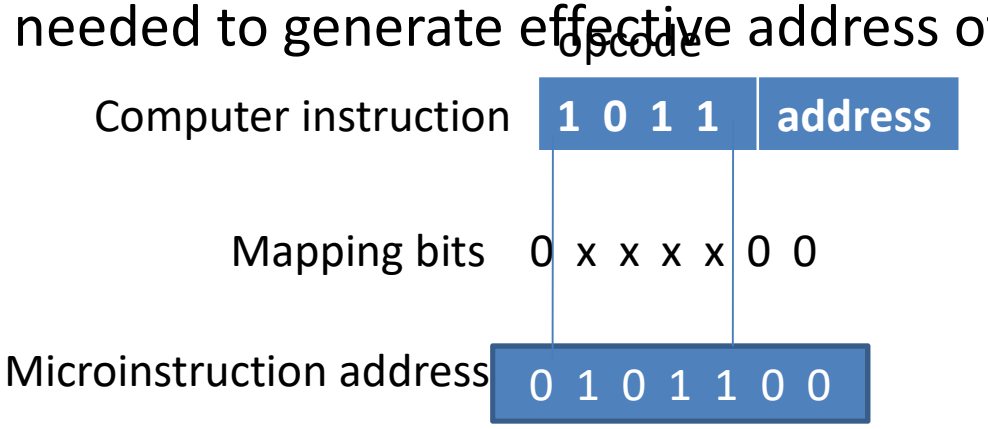
Figure 7-2 Selection of address for control memory

Cont…

- Branch logic : Test the specified condition and branch to the indicated address if the condition is met; otherwise, CAR is incremented
- Status bits together with branch address in the microinstruction control the conditional branch decisions generated in the branch logic
- Suppose 3 bits in the microinstruction are used to specify any one of 8 status bit conditions. These bits provide selection variables for the multiplexer. If the selected status bit is in 1 state, o/p of the multiplexer is 1; otherwise 0. o/p 1 generates a control signal to transfer the branch address from the microinstruction into CAR. o/p 0 causes CAR to be incremented
- Status bits provide information such as carry-out of an adder, sign bit of a number, mode bits of an instruction, and input/output status conditions

Mapping of Instruction

- 4 bit operation code should be converted into a 7 bit address for the control memory containing 128 words
- For each computer instruction, a microprogram routine may contain four microinstructions. If the routine needs more than 4 microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines
- Subroutine: Microinstructions can be saved by employing subroutines that use common sections of microcode. Return address is stored in SBR. E.g. sequence of microoperations needed to generate effective address of the operand



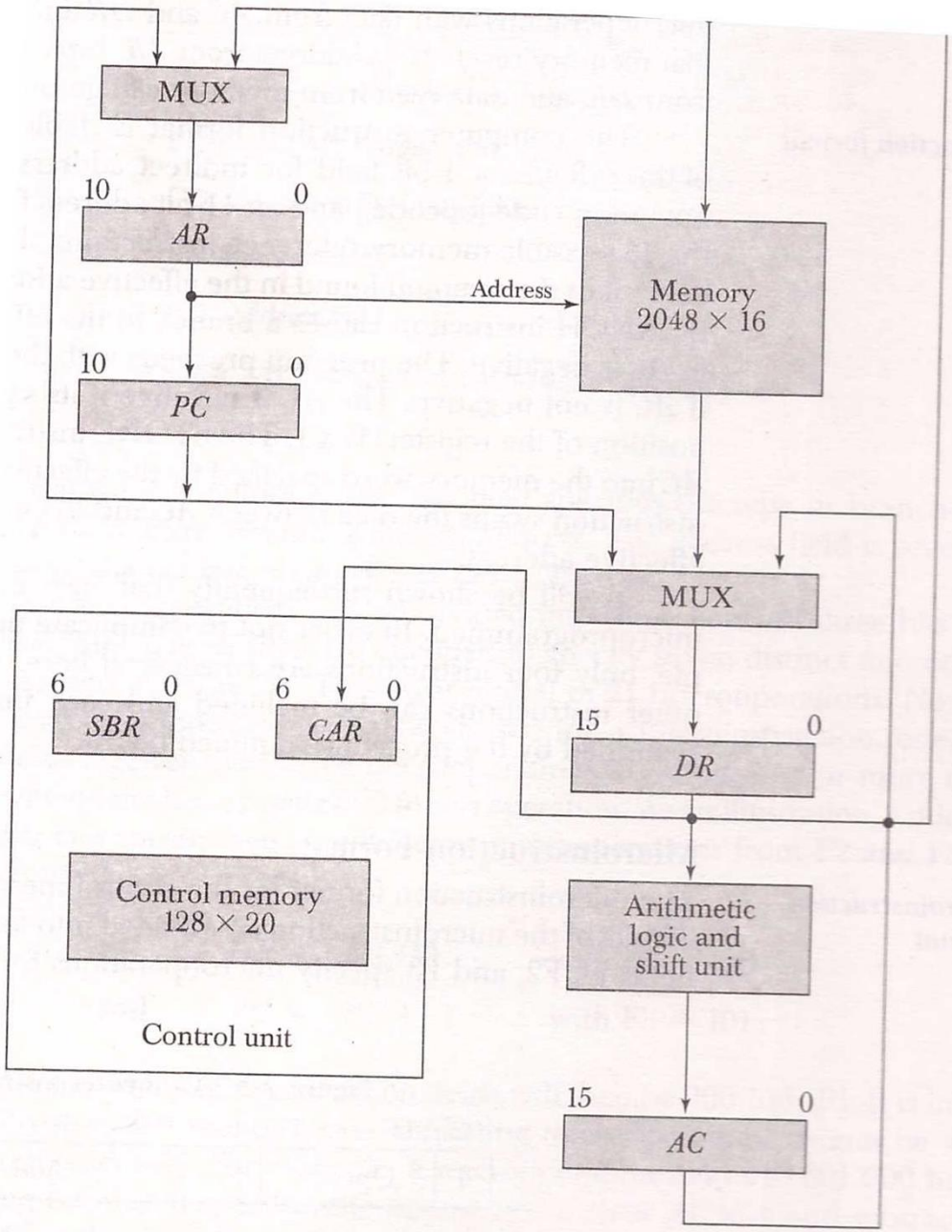


Figure 7-4 Computer hardware fonfiguration.

Computer Configuration

- Transfer of information among registers in the processor is done through MUX
- DR <- AC, PC or memory
- AR <- PC or DR
- PC <- AR
- AC <- ALU <- AC, DR
- Memory receives its address from AR
- DR Memory

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR (15)	I	Indirect address bit
10	AC (15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Cont…

- CD is used in conjunction with BR

BR	Symbol	Function
00	JMP	CAR <- AD if condition = 1 CAR <- CAR + 1 if condition = 0
01	CALL	CAR <- AD, SBR <- CAR + 1 if condition = 1 CAR <- CAR + 1 if condition = 0
10	RET	CAR <- SBR (Return from subroutine)
11	MAP	CAR (2 – 5) <- DR (11 -14), CAR (0, 1, 6) <- 0

- BR is used in conjunction with AD
- JMP and CALL instructions depend on CD
- RET and MAP are independent of CD and AD
- A symbolic (assembly language) microprogram is created using Symbols in the above tables
- A symbolic microinstruction is divided into five fields: label, microoperations, CD, BR and AD
- Microoperations field consists of one, two or three symbols separated by commas. There may be no more than one symbol from each F field
- AD specifies the address in 3 ways-1. With symbolic address 2. NEXT 3. left empty for RET or M

- Microinstructions for fetch routine

Cont…

AR <- PC

DR <- M[AR], PC <- PC + 1

AR <- DR(0 – 10), CAR(2 – 5) <- DR(11 – 14), CAR(0, 1, 6) <- 0

- Symbolic microprogram

```
        ORG  64
FETCH: PCTAR          U     JMP   NEXT
       READ, INCPC    U     JMP   NEXT
       DRTAR          U     MAP
```

- Binary microprogram

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Symbolic Microprogram (Partial)				
Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH

Cont…

Cont…

	ORG 64			
FETCH:	**PCTAR**	**U**	**JMP**	**NEXT**
	READ, INCPC	**U**	**JMP**	**NEXT**
	DRTAR	**U**	**MAP**	
INDRCT:	**READ**	**U**	**JMP**	**NEXT**
	DRTAR	**U**	**RET**	

- The various fields in microinstruction format provide control bits to initiate microoperations in the system. Each field requires a decoder to produce the corresponding control signals
- First 64 words are occupied by routines for 16 instructions, remaining 64 words may be used for any purpose
- Microprogram assembler is used to translate a symbolic microprogram into its binary equivalent

Design of Control Unit

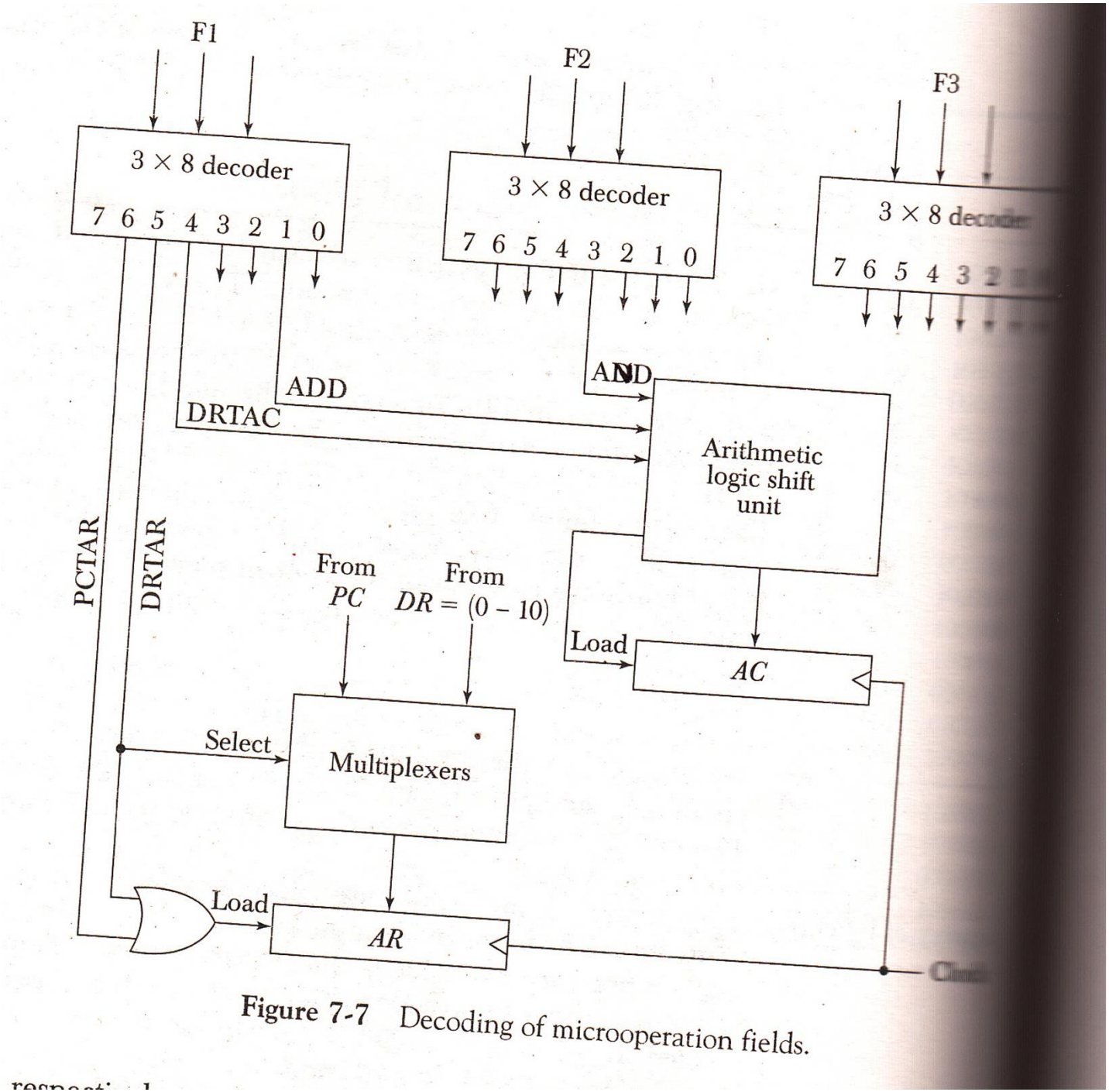


Figure 7-7 Decoding of microoperation fields.

Microprogram Sequencer

- Basic components of µp CU: control memory, circuits that select next address
- The address selection part is called microprogram sequencer
- Purpose: To present an address to the CM so that a microinstruction may be read and executed
- The next address logic of the sequencer determines the specific address source to be loaded into CAR. The choice of address source is guided by the next address information bits that the sequencer receives from the present microinstruction
- CD field of microinstruction selects one of the status bits in MUX2. If this bit is 1, T (test) variable is 1; otherwise 0
- T value together with 2 bits of BR field will go to input logic circuit. The input logic will determine the type of operations (increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations) that are available in the unit
- A sequencer will have a register stack of about 4 to 8 levels deep. So a number of subroutines can be active at the same time

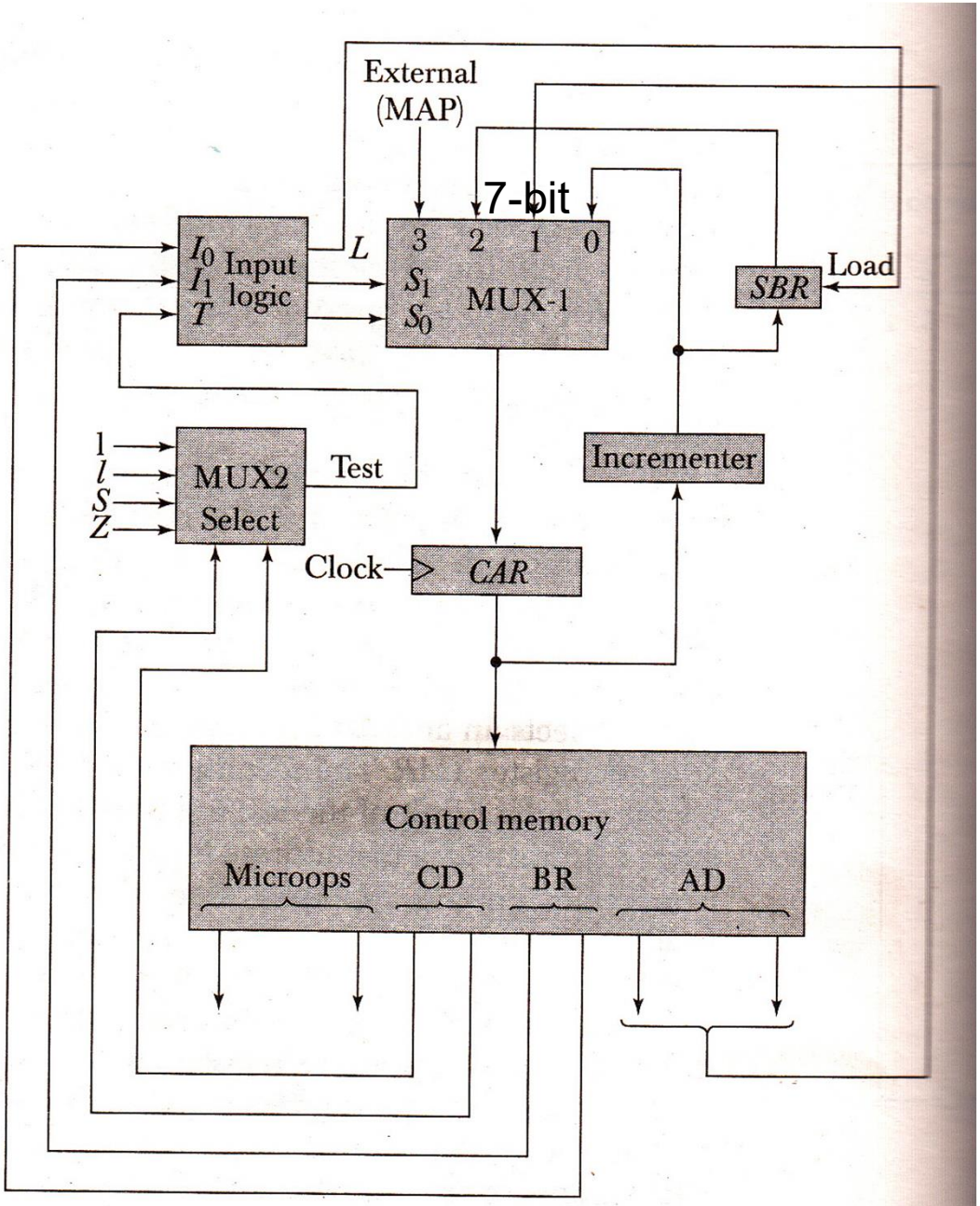
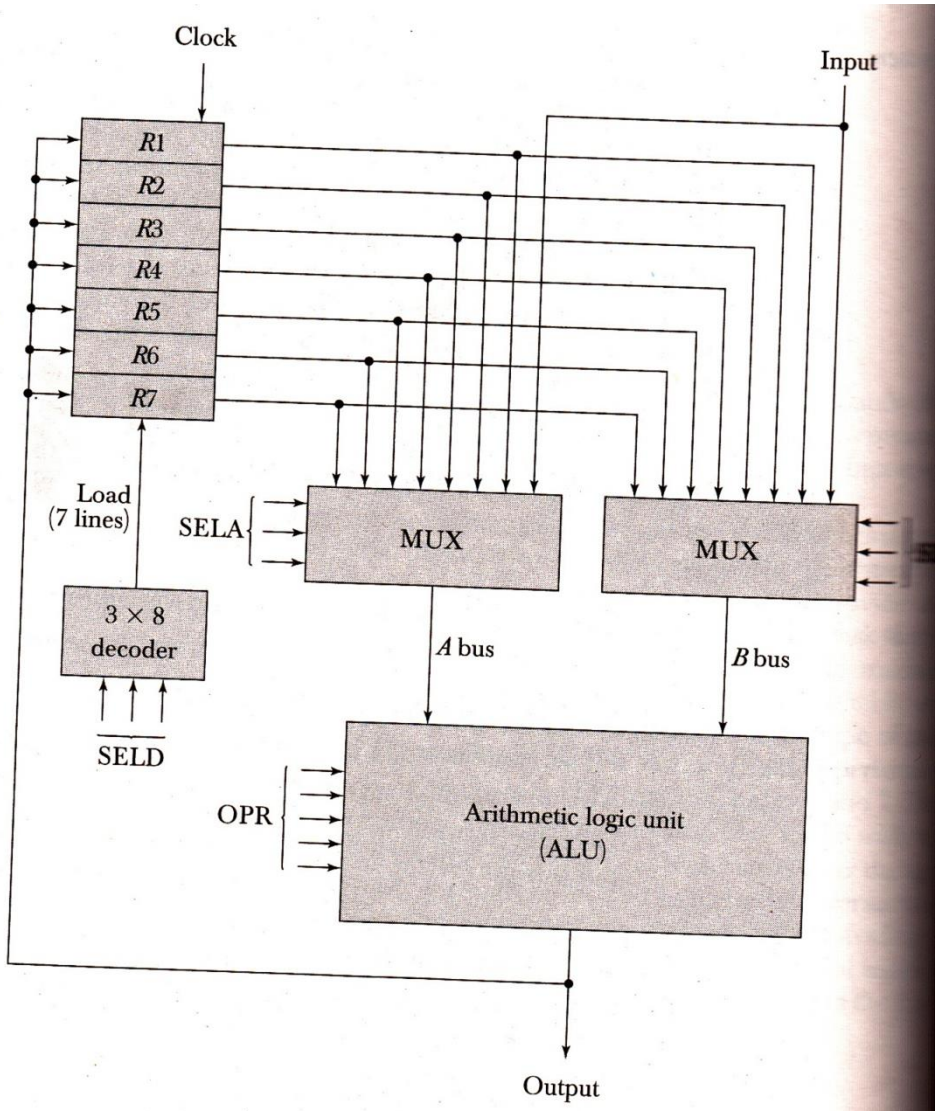
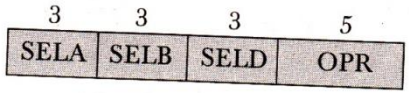


Figure 7-8 Microprogram sequencer for a control memory.



(a) Block diagram



(b) Control word

Figure 8-2 Register set with common ALU.

CPU

- R1 <- R2 – R3

Cont…

Field	: SELA	SELB	SELD	OPR
Symbol	: R2	R3	R1	SUB
Control word	: 010	011	001	00101

ALU operations		
OPR	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Cont…

TABLE 8-3 Examples of Microoperations for the CPU

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100
$R6 \leftarrow R6 + 1$	R6	–	R6	INCA	110 000 110
$R7 \leftarrow R1$	R1	–	R7	TSFA	001 000 111
Output $\leftarrow R2$	R2	–	None	TSFA	010 000 000
Output $\leftarrow$ Input	Input	–	None	TSFA	000 000 000
$R4 \leftarrow$ sh1 $R4$	R4	–	R4	SHLA	100 000 100
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101

Stack Organisation

- Stack works on LIFO basis
- SP always points at the top item in the stack
- SP contains 6 bits for a 64-word stack
- FULL is set when stack is full, and EMTY is set when stack is empty
- DR holds the data to be written into or read out of the stack
- Push: SP <- SP + 1

  M[SP] <- DR

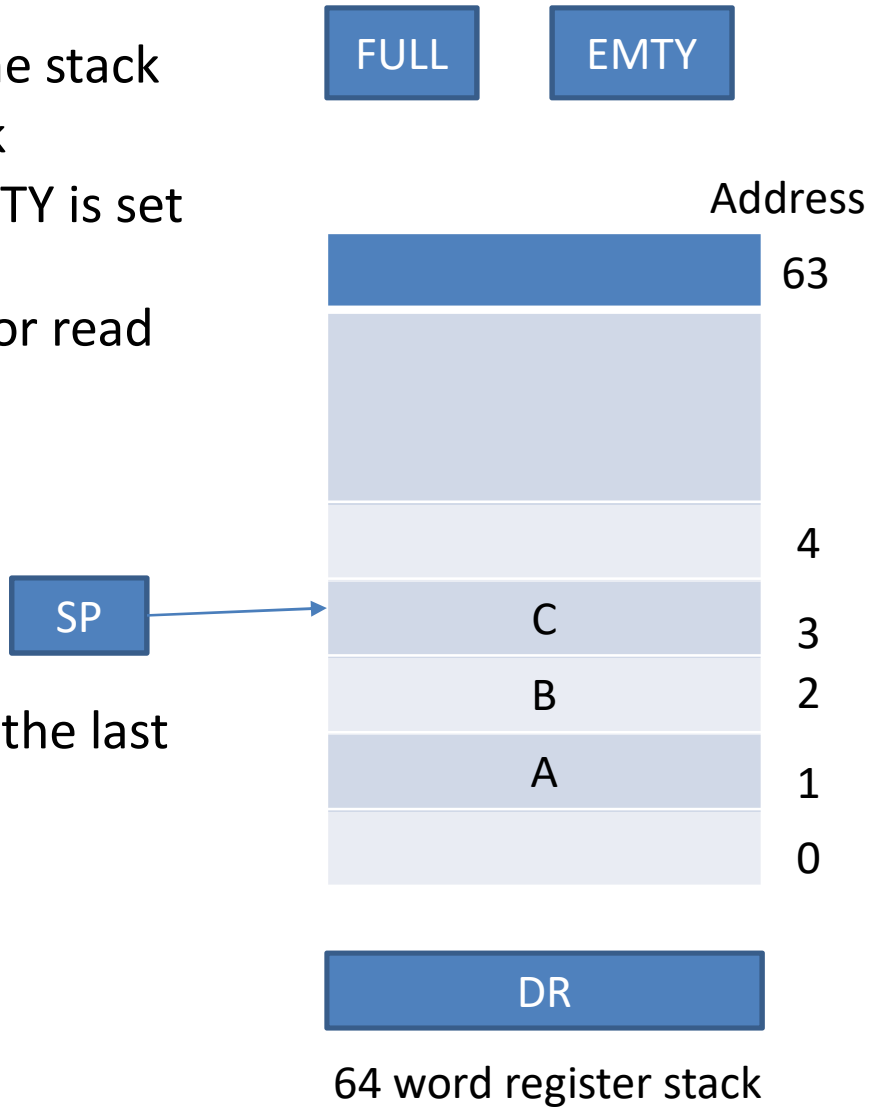
  If (SP = 0) then FULL <- 1

  EMTY <- 0
- First item is stored at location 1, and the last item at 0
- Pop: DR <- M[SP]

  SP <- SP – 1

  If (SP = 0) then EMTY <- 1

  FULL <- 0

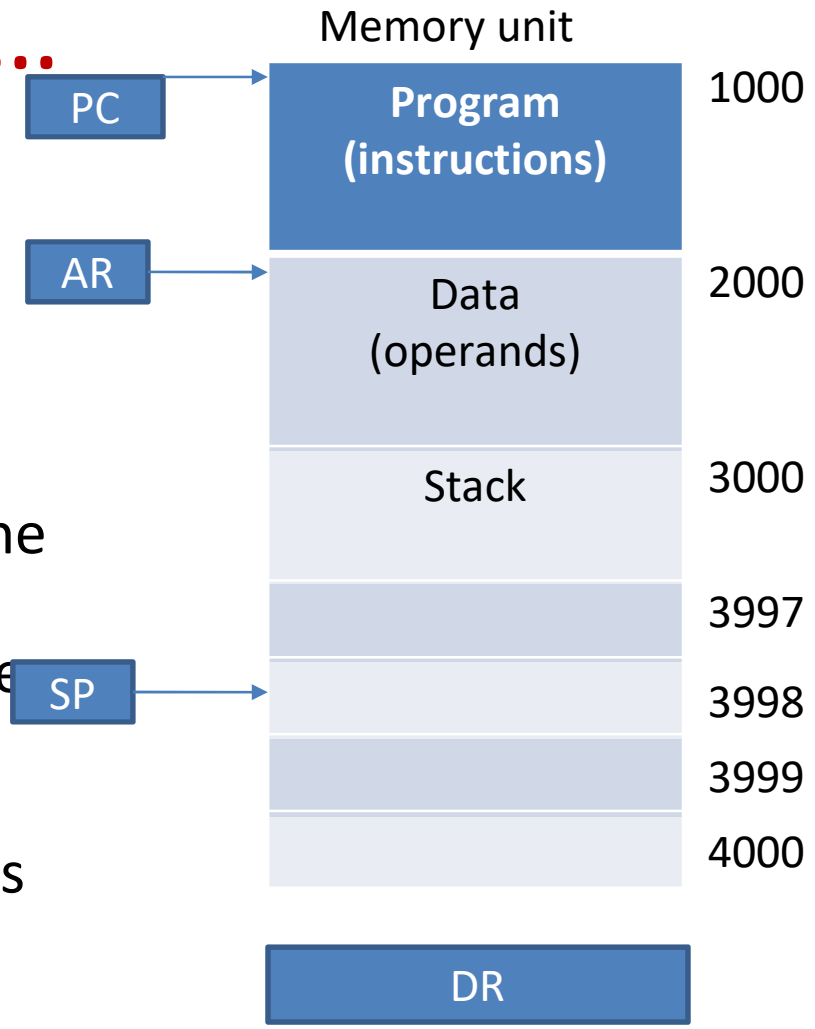


Cont…

- Push: SP <- SP – 1
  M[SP] <- DR
- Pop: DR <- M[SP]
  SP <- SP + 1
- Two registers are used to check the limits of the stack
- Stack organisation is very effective for evaluating arithmetic expressions
- Reverse polish notation (postfix) is suitable for stack manipulation

Infix: A * B + C * D

Postfix: AB * CD *+



Instruction Formats

- Consists of opcode field, address field and mode field

CPU organisations

- Single accumulator organization
- ADD X AC <- AC + M[X]
- General register organisation
- ADD R1, R2, R3 R1 <- R2 + R3
- ADD R1, R2 R1 <- R1 + R2
- ADD R1, X R1 <- R1 + M[X]
- Stack orgaisation
- PUSH X
- ADD

Cont…

- X = (A + B) * (C + D)
- Three address instructions
- ADD R1, A, B R1 <- M[A] + M[B]
- ADD R2, C, D R2 <- M[C] + M[D]
- MUL X, R1, R2 M[X] <- R1 * R2
- Computer : Cyber 170
- Two address instruction
- MOV R1, A R1 <- M[A]
- ADD R1, B R1 <- R1 + M[B]
- MOV R2, C R2 <- M[C]
- ADD R2, D R1 <- R2 + M[D]
- MUL R1, R2 R1 <- R1 * R2
- MOV X, R1 M[X] <- R1

Cont…

- One address instruction
- LOAD A — AC <- M[A]
- ADD B — AC <- AC + M[B]
- STORE T — M[T] <- AC
- LOAD C — AC <- M[C]
- ADD D — AC <- AC + M[D]
- MUL T — AC <- AC * M[T]
- STORE X — M[X] <- AC
- Zero address instruction
- PUSH A — TOS <- A
- PUSH B — TOS <- B
- ADD — TOS <- (A + B)
- PUSH C — TOS <- C
- PUSH D — TOS <- D
- ADD — TOS <- (C + D)
- MUL — TOS <- (C + D) * (A + B)
- POP X — M[X] <- TOS

RISC instruction

LOAD R1, A	R1 <- M[A]
LOAD R2, B	R2 <- M[B]
LOAD R3, C	R3 <- M[C]
LOAD R4, D	R4 <- M[D]
ADD R1, R1, R2	R1 <- R1 + R2
ADD R3, R3, R4	R3 <- R3 + R4
MUL R1, R1, R3	R1 <- R1 * R3
STORE X, R1	M[X] <- R1

Addressing Modes

- Instruction cycle
- Fetch the instruction from the memory: PC points towards the address of the next instruction to be executed
- Decode the instruction: Determine the operation to be performed, addressing mode of the instruction, and the location of the operands
- Execute the instruction
- Implied mode: operands are implicit in the instructions
- E.g. complement accumulator, zero address instructions
- Immediate mode: operand is specified in the instruction itself
- Register mode: operands are in registers
- Register indirect mode: address of the operand is in register
- Autoincrement or autodecrement mode: register is incremented or decremented after (or before) its value is used to access the memory

Cont…

- Direct address mode: address is specified in the instruction
- Indirect address mode: address of the effective address is given in the instruction
- Relative address mode: effective address = address part of the instruction + content of PC
- Indexed addressing mode: effective address = address part of the instruction + content of index register
- Base register addressing mode: effective address = address part of the instruction + content of base register (an address)
- E.g Load AC with a value

Computer Instructions

- Data transfer Instructions

Name	Mnemonics
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instructions

- ADDI : add two binary integers
- ADDF : add two floating point numbers
- ADDD : add two decimal numbers in BCD

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Cont…

- Logical and bit manipulation instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Cont…

- Shift instructions

OP REG TYPE RL COUNT

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control

- Instructions may alter the flow of control
- Conditional and unconditional branch and jump instructions
- SKP (zero address) instruction skips the next instruction
- Certain status bits are set as a result of compare and test (performs logical AND) operations

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Cont…

- Status bits : Carry, Sign, Zero, oVerflow

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
Unsigned compare conditions (A – B)		
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
Signed compare conditions (A – B)		
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Cont…

Conditional Branch instructions

Cont…

- Microoperations in a subroutine call

SP <- SP – 1

M[SP] <- PC

PC <- effective address

- Microoperations in a return statement

PC <- M[SP]

SP <- SP + 1

Interrupt

- Program interrupt: refers to the transfer of program control from a currently running program to another service program as a result of an external or internal request
- Interrupt procedure: interrupt is usually initiated by an external or internal signal
- Address of the interrupt service program is determined by the h/w
- Interrupt procedure stores all the information necessary to define the state of the CPU
- Content of program counter, registers and certain status conditions
- PSW (stored in a register) : collection of all status bit conditions. It includes status bits from the last ALU operation, interrupts that are allowed to occur, whether the CPU is in supervisor or user mode

Types of Interrupts

- External interrupts : come from I/O devices, from a timing device, from a circuit monitoring the power supply, or from any other external source
- Internal interrupts (traps) : arise from illegal or erroneous use of an instruction or data. E.g. register overflow, stack overflow, divide by zero, invalid opcode, protection violation
- Software interrupt : programmer initiates the interrupt by executing a special call instruction. It is used to switch from user mode to supervisor mode