Section 5

Basic Computer Organization and Design

Slides with white background courtesy of Mano text for this class

1

Mano's Basic Computer

- Memory unit with 4096 16-bit words
- Registers: *AR*, *PC*, *DR*, *AC*, *IR*, *TR*, *OUTR*, *INPR*, *SC*
- Flip-flops: *I*, *S*, *E*, *R*, *IEN*, *FGI*, *FGO*
- 3 x 8 op decoder and 4 x 16 timing decoder
- 16-bit common bus
- Control logic gates
- Adder and logic circuit connected to input of *AC*

2

Instruction Code

- Computer instruction is binary code that specifies a sequence of microoperations
- Operation code + Address
  - Op code must have n bits for $\leq 2^n$ operations
  - Op code sometimes called a macrooperation
  - Address is register or memory location
    - Memory location is operand address
- Shorten "instruction code" to "instruction"
- Instructions and data in memory

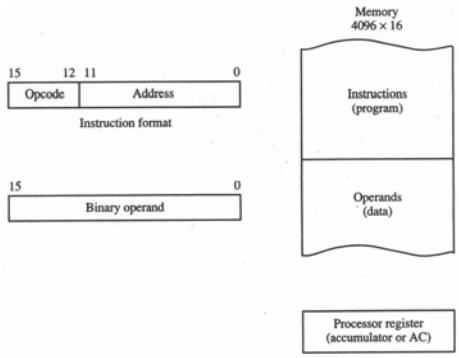
3

Stored Program Organization

- One processor register
  - *AC* – accumulator
- Instruction format
  - 4-bit op code
  - 12-bit address (for $2^{12} = 4096$ memory words)
- Instruction execution cycle
  - Read 16-bit instruction from memory
  - Use 12-bit address to fetch operand from memory
  - Execute 4-bit op code

4

Stored Program Organization



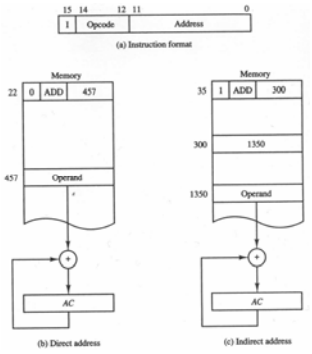
5

Address Types

- 12-bit instruction address
  - Immediate
    - Actual data value
  - Direct
    - Memory address where data (operand) resides
  - Indirect
    - Memory address where memory address of data (operand) resides
- Effective address is the address of the operand
- Lead bit of instruction used as indirect flag

6

Direct / Indirect Address



7

Basic 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

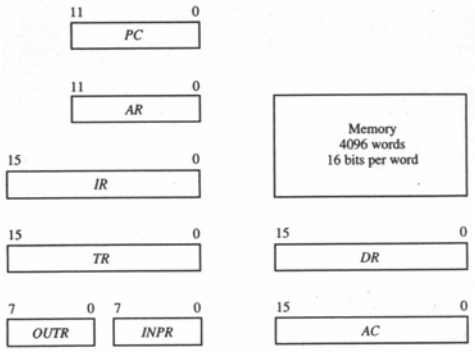
8

Program Counter (*PC*)

- Holds memory address of next instruction
- Next instruction is fetched after current instruction completes execution cycle
- *PC* is incremented right after instruction is fetched from memory
- *PC* value can be replaced by new address when executing a branch instruction

9

Registers + Memory Layout



10

Register Control Inputs

- Load (LD)
- Increment (INR)
- Clear (CLR)

11

Common Bus

- Connects registers and memory
- Specific output selected by $S_2S_1S_0$
  - When register has < 16 bits, high-order bus bits are set to 0
- Register with LD enabled reads data from bus
- Memory with Write enabled reads bus
- Memory with Read enabled puts data on bus
  - When $S_2S_1S_0 = 111$

12

Address Register (*AR*)

- Always used to specify address within memory unit
- Dedicated register eliminates need for separate address bus
- Content of any register output connected to the bus can be written to memory
- Any register input connected to bus can be target of memory read
  - As long as its LD is enabled

13

Accumulator (*AC*)

- Input comes from adder and logic circuit
- Adder and logic circuit
  - Input
    - 16-bit output of *AC*
    - 16-bit data register (*DR*)
    - 8-bit input register (*INPR*)
  - Output
    - 16-bit input of *AC*
    - *E* flip-flop (extended *AC* bit, aka overflow)
- *DR* and *AC* input used for arithmetic and logic microoperations

14

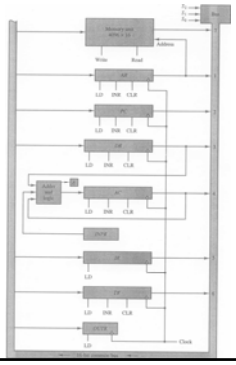
Timing Is Everything

- Content of any register output connected to the bus can be applied to the bus and content of any register input connected to the bus can be loaded from the bus during the same clock cycle
- These 2 microoperations can be executed at the same time

  $DR \leftarrow AC$ and $AC \leftarrow DR$

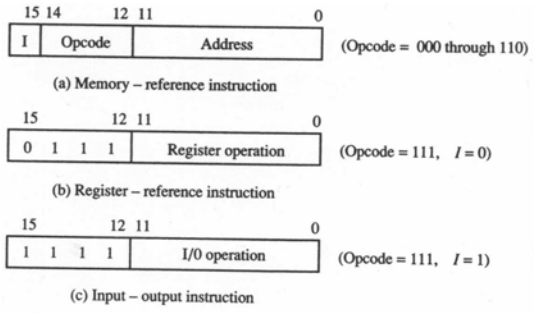
15

Bus Connections



16

Basic Instruction Formats



17

Instruction Format

- Only 3 bits used for op code
- Looks like only 8 different op codes are possible
- Wrong!
- For op code 111, one of the low-order 12 bits is turned on to extend the op code definition

18

Basic 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

19

Instruction Set Completeness

- Arithmetic, logical, and shift
- Move data from and to memory and registers
- Program control and status check
- Input and output
  - (I/O, I/O, it's off to the bus we go…)

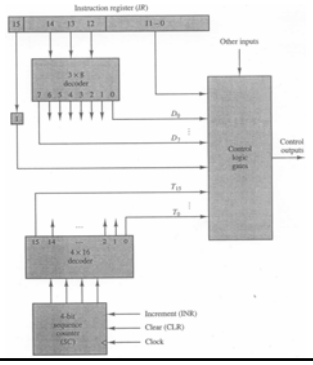
20

Control Unit

- Instruction read from memory and put in *IR*
- Leftmost bit put in *I* flip-flop
- 3-bit op code decoded with 3 x 8 decoder into D_0 to D_7
- 4-bit sequence counter (*SC*) decoded with 4 x 16 decoder into T_0 to T_{15} (timing signals)
- *I*, D_0 to D_7, T_0 to T_{15}, rightmost 12 bits of *IR*, and other inputs are fed into control and logic gates

21

Control Unit



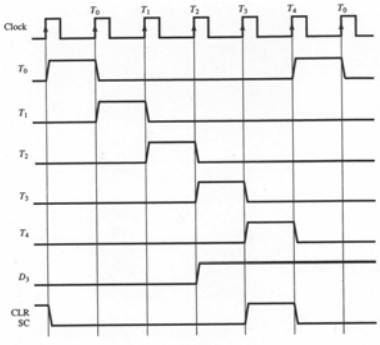
22

Sequence Counter (*SC*)

- Inputs are increment (INR) and clear (CLR)
- Example
  - *SC* incremented to provide T_0, T_1, T_2, T_3, and T_4
  - At time T_4, *SC* is cleared to 0 if D_3 is active
  - Written as: D_3T_4: $SC \leftarrow 0$

23

Timing Diagram



24

Instruction Cycle

- Fetch instruction from memory
- Decode the instruction
- Read effective address from memory if indirect address
- Execute the instruction

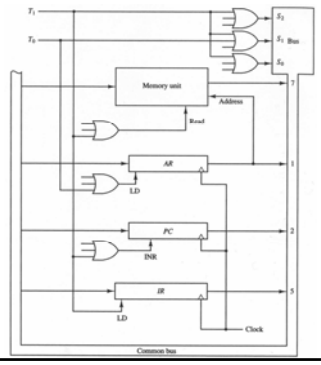
25

Fetch And Decode

- *SC* cleared to 0, generating timing signal T_0
- After each clock pulse, *SC* is incremented
- Fetch and decode microoperations
  - T_0: $AR \leftarrow PC$
  - T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
  - T_2: $D_0,\ldots D_7 \leftarrow$ decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11)$, $I \leftarrow IR(15)$

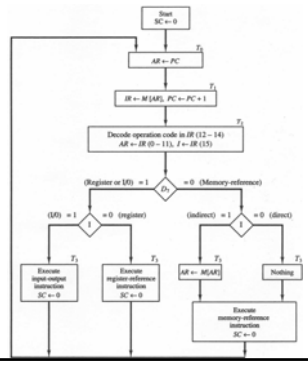
26

Fetch Phase



27

Instruction Cycle Flowchart



28

Instruction Paths

- D'_7IT_3: $AR \leftarrow M[AR]$
- $D'_7I'T_3$: Do nothing
- $D_7I'T_3$: Execute a register-reference instruction
- D_7IT_3: Execute an I/O instruction

29

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:	$AC \leftarrow$ 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

30

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 $PC \leftarrow PC + 1$

31

AND to *AC*

- D_0T_4: $DR \leftarrow M[AR]$
- D_0T_5: $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

32

ADD to *AC*

- D_1T_4: $DR \leftarrow M[AR]$
- D_1T_5: $AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$

33

LDA: Load *AC*

- D_2T_4: $DR \leftarrow M[AR]$
- D_2T_5: $AC \leftarrow DR$, $SC \leftarrow 0$

34

STA: Store *AC*

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

35

BUN: Branch Unconditionally

- D_4T_4: $PC \leftarrow AR$, $SC \leftarrow 0$

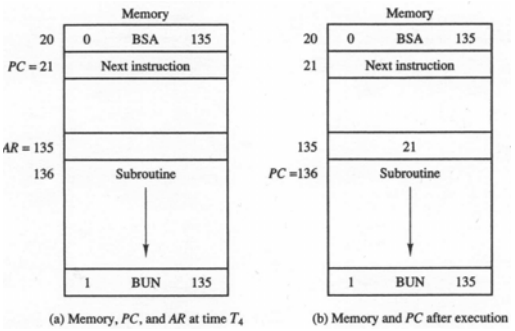
36

BSA: Branch & Save Return Address

- D_5T_4: $M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$
- D_5T_5: $PC \leftarrow AR,\ SC \leftarrow 0$

37

BSA Example



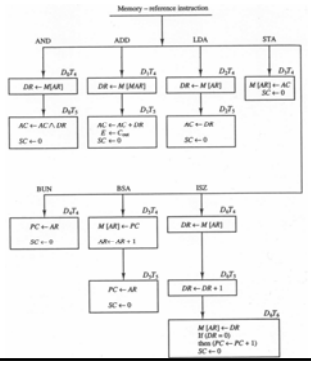
38

ISZ: Increment & Skip if Zero

- Increment word specified by effective address
  - If value = 0, increment *PC*
- D_6T_4: $DR \leftarrow M[AR]$
- D_6T_5: $DR \leftarrow DR + 1$
- D_6T_6: $M[AR] \leftarrow DR,\ SC \leftarrow 0,$
  if $(DR = 0)$ then $(PC \leftarrow PC + 1)$

39

Memory-Reference Instructions



40

Input Register *INPR*

- 1-bit input flip-flop *FGI*
  - Initially cleared to 0
- When key hit on keyboard
  - 8-bit alphanumeric code is shifted into *INPR*
  - Input flag *FGI* set to 1
  - No more input can be accepted from keyboard
- Computer checks *FGI*, when set to 1
  - Parallel transfer from *INPR* to *AC*
  - *FGI* cleared to 0
  - More input can now be accepted from keyboard

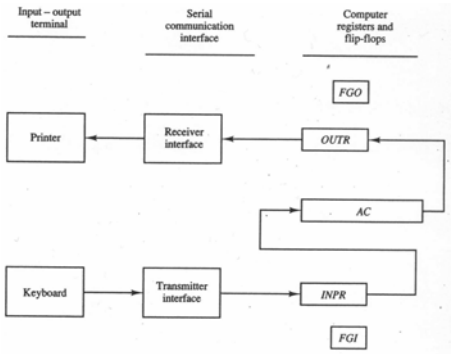
41

Output Register *OUTR*

- 1-bit output flip-flop *FGO*
  - Initially set to 1
- Computer checks *FGO*, when set to 1
  - Parallel transfer from *AC* to *OUTR*
  - *FGO* cleared to 0
  - No more output can be sent from computer
- Output device accepts 8-bit character
  - *FGO* set to 1
  - More output can now be sent from computer

42

Input-Output Configuration



43

Input-Output Instructions

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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If ($FGI = 1$) then ($PC \leftarrow PC + 1$)	Skip on input flag
SKO	pB_8:	If ($FGO = 1$) then ($PC \leftarrow PC + 1$)	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

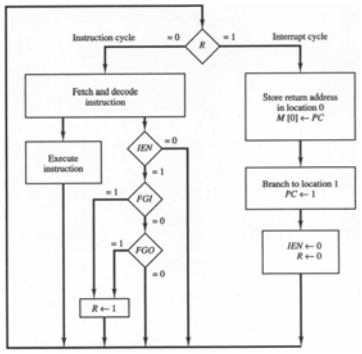
44

Interrupt Enable *IEN*

- Having computer constantly check *FGI* and *FGO* via an executable instruction is a waste of time
- Instead, *IEN* is programmatically set, effectively saying "let me know if you need me"
  - Meanwhile, it keeps executing instructions
- During each execution cycle, if computer detects *FGI* or *FGO* is set, then *R* is set to 1
- The interrupt happens when the computer is ready to fetch the next instruction
  - *R* = 0 means go through instruction cycle
  - *R* = 1 means go through interrupt cycle

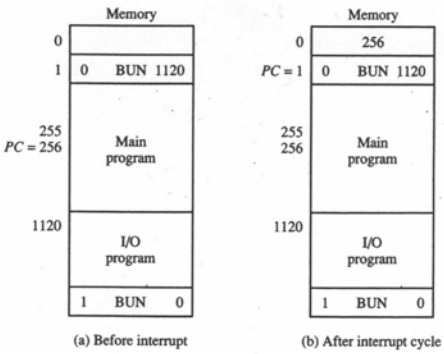
45

Interrupt Flowchart



46

Interrupt Cycle Example



47

Interrupt Cycle

- Condition for setting *R* to 1
  $T'_0T'_1T'_2(IEN)(FGI + FGO)$: $R \leftarrow 1$
- Fetch phase modified to service interrupt
  RT_0: $AR \leftarrow 0$, $TR \leftarrow PC$
  RT_1: $M[AR] \leftarrow TR$, $PC \leftarrow 0$
  RT_2: $PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$

48

Computer Operation Flowchart

49

Inputs To Control Logic Gates

- Two decoders
  - 8-bit instruction and 16-bit sequence
- Seven flip-flops: *I*, *S*, *E*, *R*, *IEN*, *FGI*, *FGO*
- *IR* bits 0 through 11
- *AC* bits 0 through 15
  - Check if *AC* = 0 and check sign bit
- *DR* bits 0 through 15
  - Check if *DR* = 0

50

Outputs Of Control Logic Gates

- Control inputs of nine registers
- Control read & write inputs of memory
- Set, clear, or complement flip-flops
- S_2, S_1, and S_0 to select a register for the bus
- Control AC adder and logic circuit

51

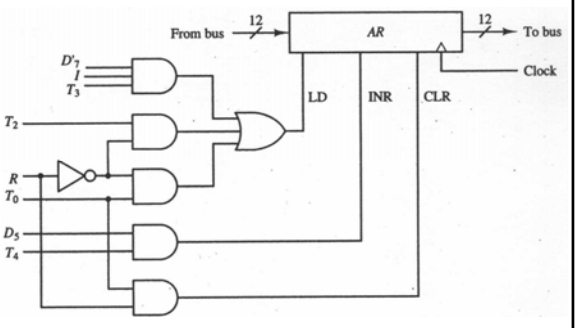
AR Control Gates

- Register control inputs: LD, INR, and CLR
- Find all statements that alter *AR* contents

$R'T_0$:	$AR \leftarrow PC$	LD
$R'T_2$:	$AR \leftarrow IR(0\text{-}11)$	LD
D'_7IT_3:	$AR \leftarrow M[AR]$	LD
RT_0:	$AR \leftarrow 0$	CLR
D_5T_4:	$AR \leftarrow AR + 1$	INR

52

AR Control Gates



53

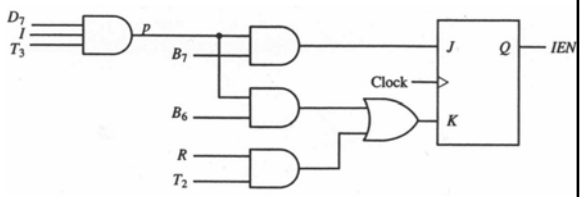
IEN Control Gates

- Find all statements that change *IEN*

$D_7IT_3B_7$:	$IEN \leftarrow 1$
$D_7IT_3B_6$:	$IEN \leftarrow 0$
RT_2:	$IEN \leftarrow 1$

54

IEN Control Gates



55

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

56

Boolean Functions For Encoder

- $S_0 = x_1 + x_3 + x_5 + x_7$
- $S_1 = x_2 + x_3 + x_6 + x_7$
- $S_2 = x_4 + x_5 + x_6 + x_7$

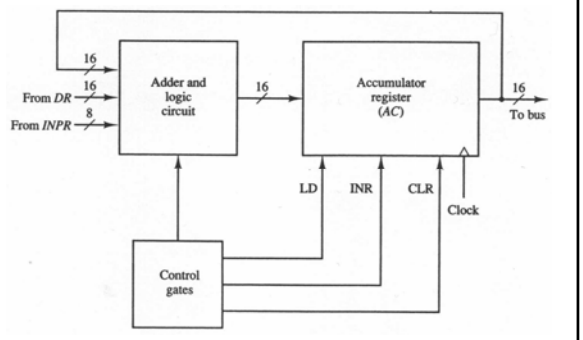
57

Boolean Function For x_1

- Find logic that makes $x_1 = 1$ by finding instructions that have *AR* as a source
  D_4T_4: $PC \leftarrow AR$
  D_5T_5: $PC \leftarrow AR$
- Therefore $x_1 = D_4T_4 + D_5T_5$
- Repeat for other six inputs

58

Circuits Associated With AC



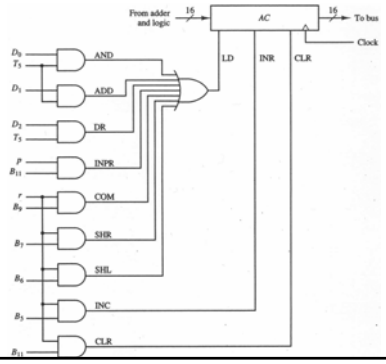
59

AC Control Gates

- Find all statements that alter *AC* contents
  D_0T_5: $AC \leftarrow AC \wedge DR$
  D_1T_5: $AC \leftarrow AC + DR$
  D_2T_5: $AC \leftarrow DR$
  $D_7IT_3B_{11}$: $AC(0\text{-}7) \leftarrow INPR$
  $D_7I'T_3B_9$: $AC \leftarrow \overline{AC}$
  $D_7I'T_3B_7$: $AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E$
  $D_7I'T_3B_6$: $AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E$
  $D_7I'T_3B_{11}$: $AC \leftarrow 0$
  $D_7I'T_3B_5$: $AC \leftarrow AC + 1$

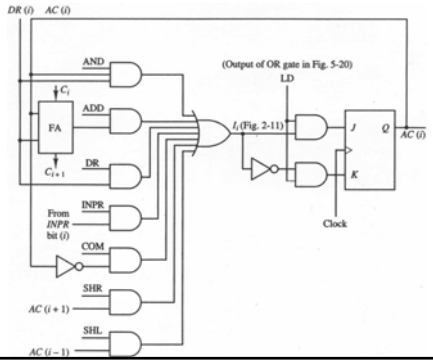
60

AC Control Gates



61

Adder And Logic Circuit



62

Mano's Basic Computer

- Memory unit with 4096 16-bit words
- Registers: *AR*, *PC*, *DR*, *AC*, *IR*, *TR*, *OUTR*, *INPR*, *SC*
- Flip-flops: *I*, *S*, *E*, *R*, *IEN*, *FGI*, *FGO*
- 3 x 8 op decoder and 4 x 16 timing decoder
- 16-bit common bus
- Control logic gates
- Adder and logic circuit connected to input of *AC*

63