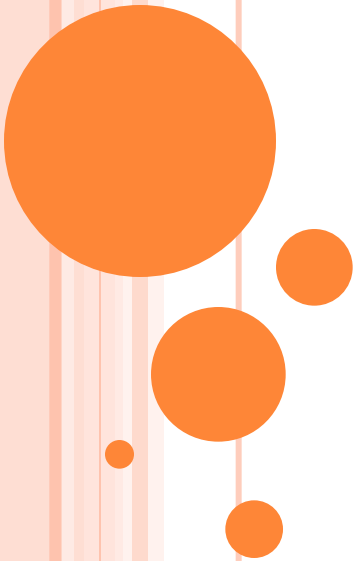
COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT 3: CONTROL UNIT



Instruction Format:

- In general, based on the number of operands or reference made in the instruction. Instructions can be classified into three categories:-

1. Three Address Instruction
2. Two address instruction
3. One address Instruction
4. Zero address instruction.

Three Address Instructions

- Example: X=(A+B)(C+D)
- Operands are in memory address symbolized by the letters A,B,C,D, result stored memory address of X

ADD T1, A, B	M[T1] ← M[A] + M[B]
ADD T2, C, D	M[T2] ← M[C] + M[D]
MUX X, T1, T2	M[X] ← M[T1] X M[T2]

OR

ADD R1, A, B	R1 ← M[A] + M[B]
ADD R2, C, D	R2 ← M[C] + M[D]
MUX X, R1, R2	M[X]← R1 X R2

- +: Short program, 3 instructions
- -: Binary coded instruction require more bits to specify three addresses

Two Address Instructions

- The first operand address also serves as the implied address for the result

MOVE T1, A	M[T1] ← M[A]
ADD T1, B	M[T1] ← M[T1] + M[B]
MOVE X, C	M[X] ← M[C]
ADD X, D	M[X] ← M[X] + M[D]
MUX X, T1	M[X] ← M[X] X M[T1]

- 5 instructions

One Address Instructions

- Implied address: a register called *an accumulator ACC* for one operand and the result, *single-accumulator architecture*

LD	A	ACC ← M[A]
ADD	B	ACC ← ACC + M[B]
ST	X	M[X] ← ACC
LD	C	ACC ← M[C]
ADD	D	ACC ← ACC + M[D]
MUX	X	ACC ← ACC X M[X]
ST	X	M[X] ← ACC

7 instructions

- All operations are between the ACC register and a memory operand

Zero Address Instructions

- Use stack (FILO):
  - ADD — TOS ← TOS + TOS_{-1}
  - PUSH X — TOS ← M[X]
  - POP X — M[X] ← TOS

PUSH A	TOS ← M[A]	8 instructions
PUSH B	TOS ← M[B]	
ADD	TOS ← TOS + TOS_{-1}	
PUSH C	TOS ← M[C]	
PUSH D	TOS ← M[D]	
ADD	TOS ← TOS + TOS_{-1}	
MUX	TOS ← TOS X TOS_{-1}	
POP X	M[X] ← TOS	

- Data manipulation operations: between the stack elements
- Transfer operations: between the stack and the memory

Mode/Opcode

Timing circuit:

- In order to perform an instruction, we need to perform a number of micro-operations, and these micro-operations must be timed.

$AC \leftarrow PC$

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

Micro-Operations:

- A computer executes a program
- Fetch/execute cycle
- Each cycle has a number of steps
  (see pipelining) Called micro-operations
- Each step does very little
- Atomic operation of CPU

Micro Operations:

- Can be performed during one clock period.

  Examples: shift, move, count, add, load etc.

- Can be classified into 4 categories:

1. Register transfer micro-operations
2. Arithmetic micro-operations
3. Logic micro-operations
4. Shift micro-operations

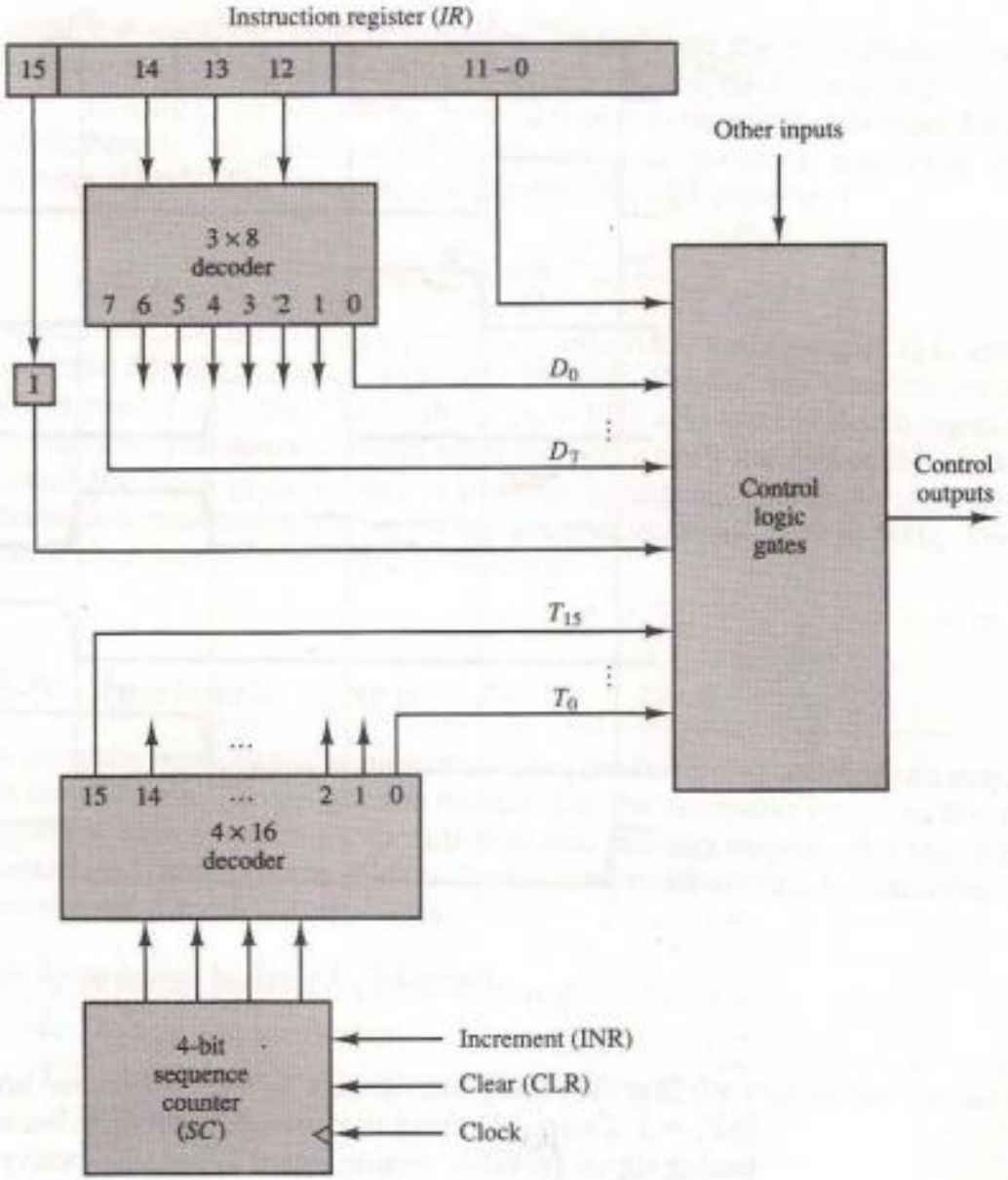
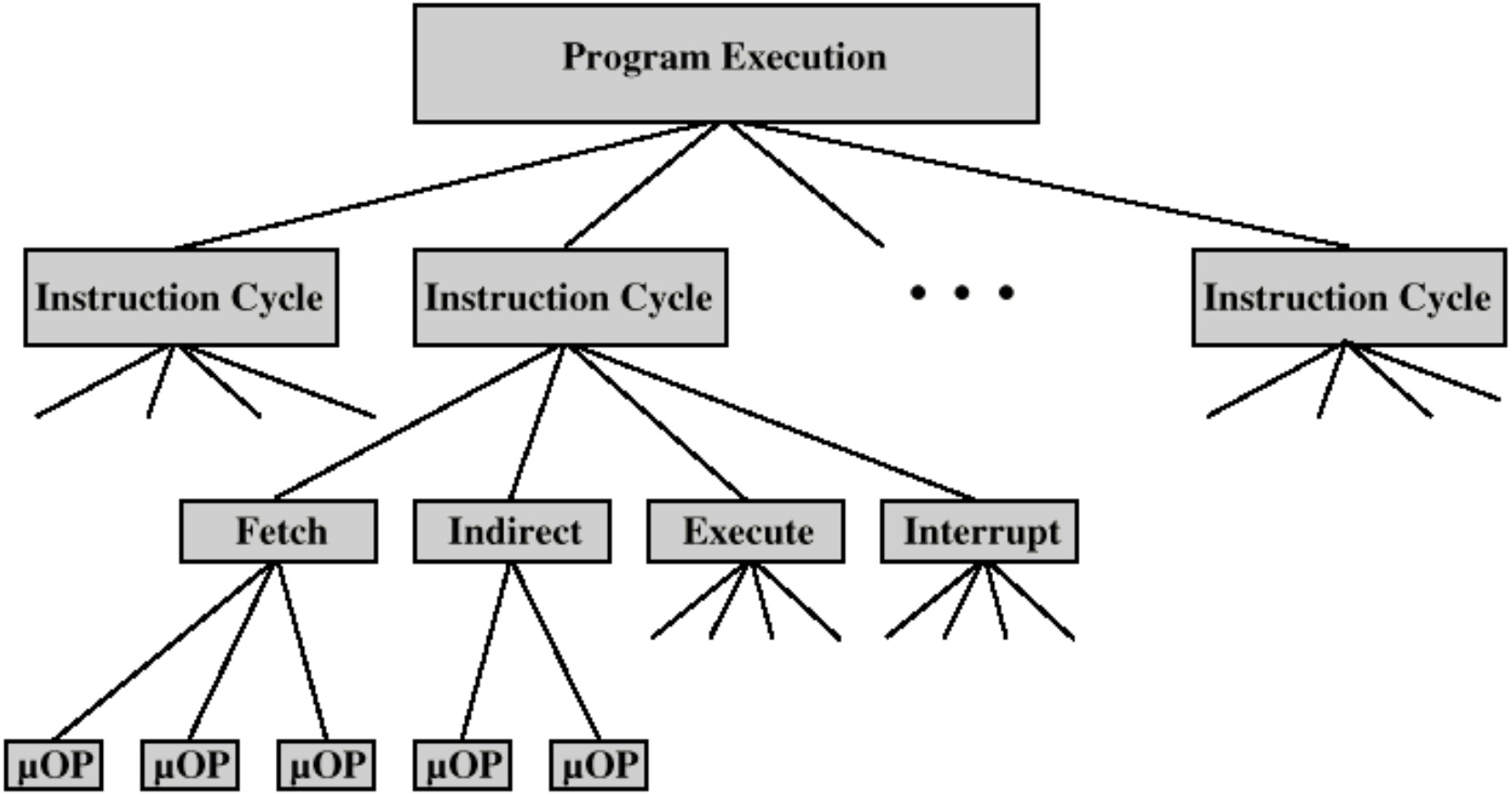


Figure 5-6 Control unit of basic computer.


Program Execution
Instruction Cycle
Instruction Cycle
Instruction Cycle
Fetch
Indirect
Execute
Interrupt
µOP
µOP
µOP
µOP
µOP

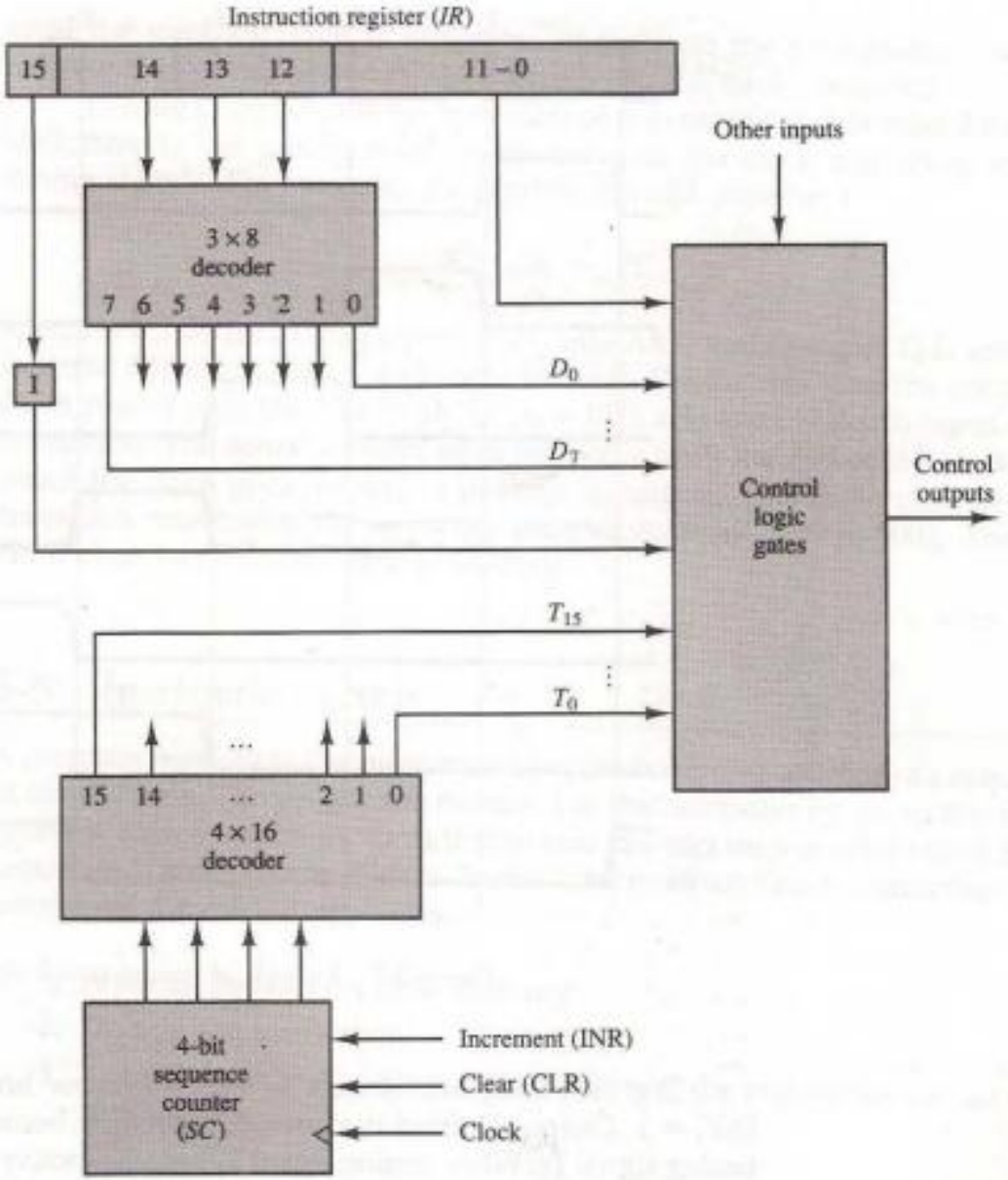


Figure 5-6 Control unit of basic computer.

Instruction Cycle:

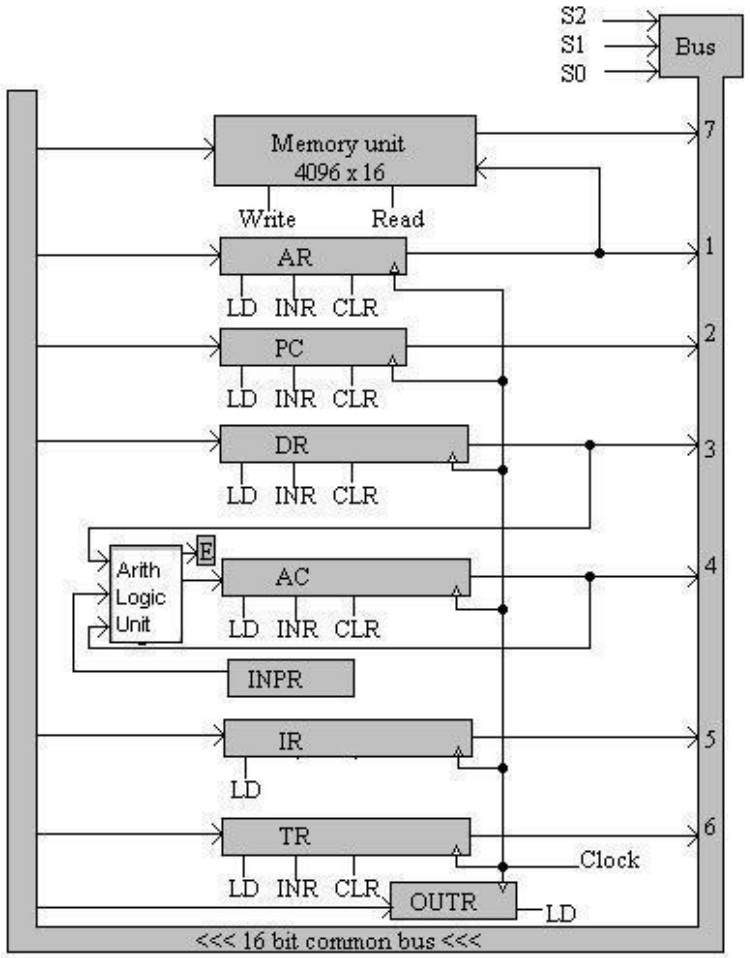
- A program is a sequence of instructions. It is executed by going through a cycle of each instruction. Each instruction cycle is subdivided into a sequence of sub cycles or phases. Each instruction cycle consists of these phases:

1. Fetch inst. from the memory
2. Decode
3. Read the effective address
4. Execute
5. Fetch and decode

Example: int i = 5 ;

Fetch - 4 Registers:

- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched



Fetch Sequence

- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches

Fetch Sequence (symbolic)

- t1: MAR <- (PC)
- t2: MBR <- (memory)
- PC <- (PC) +1
- t3: IR <- (MBR)
- (tx = time unit/clock cycle)

or

- t1: MAR <- (PC)
- t2: MBR <- (memory)
- PC <- (PC) +1
- t3: IR <- (MBR)

Rules for Clock Cycle Grouping

- Proper sequence must be followed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - Use ALU
  - May need additional micro-operations

EXAMPLE OF TIMING:

T_0: $AR \leftarrow PC$

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

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

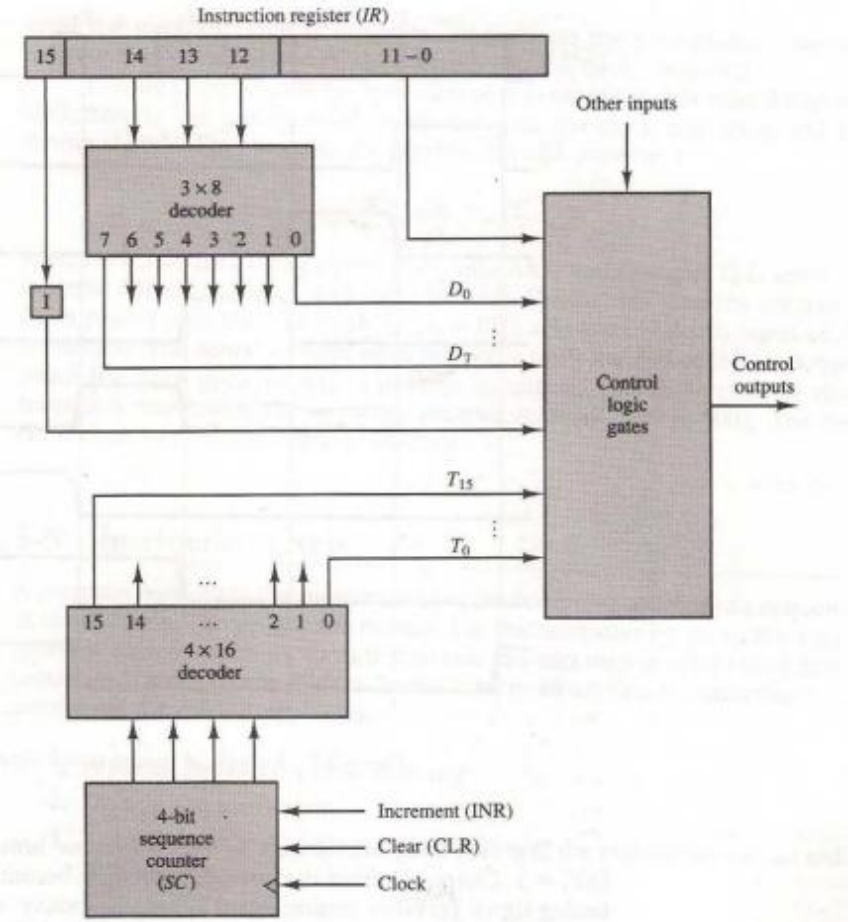


Figure 5-6 Control unit of basic computer.

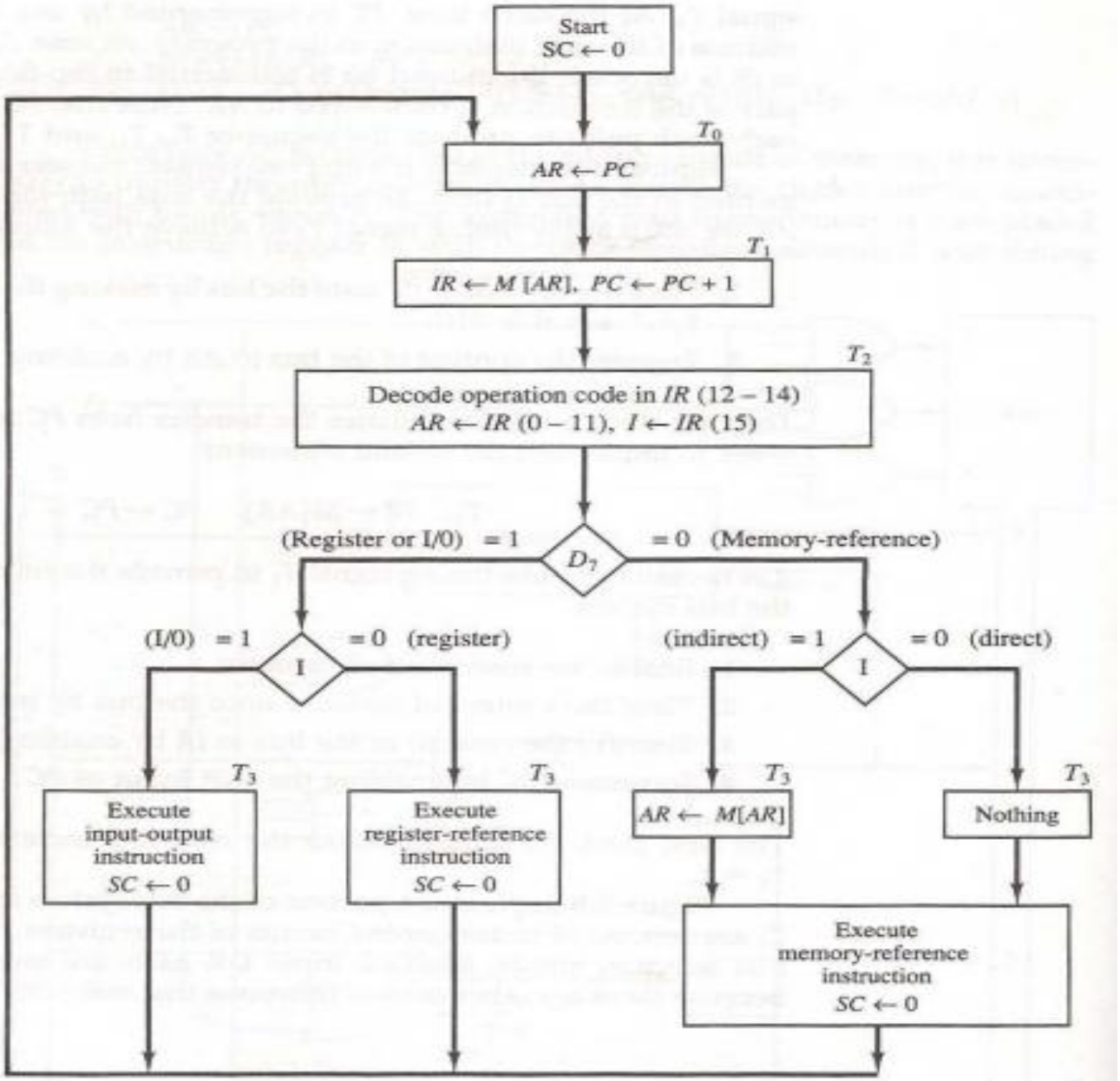


Figure 5-9 Flowchart for instruction cycle (initial configuration).

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 $PC \leftarrow PC + 1$

Memory reference and register reference instructions

$D_7 I' T_3 = r$ (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

TABLE 5-5 Input-Output Instructions

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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$	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

Symbol	Hexadecimal code $I = 0$	$I = 1$	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

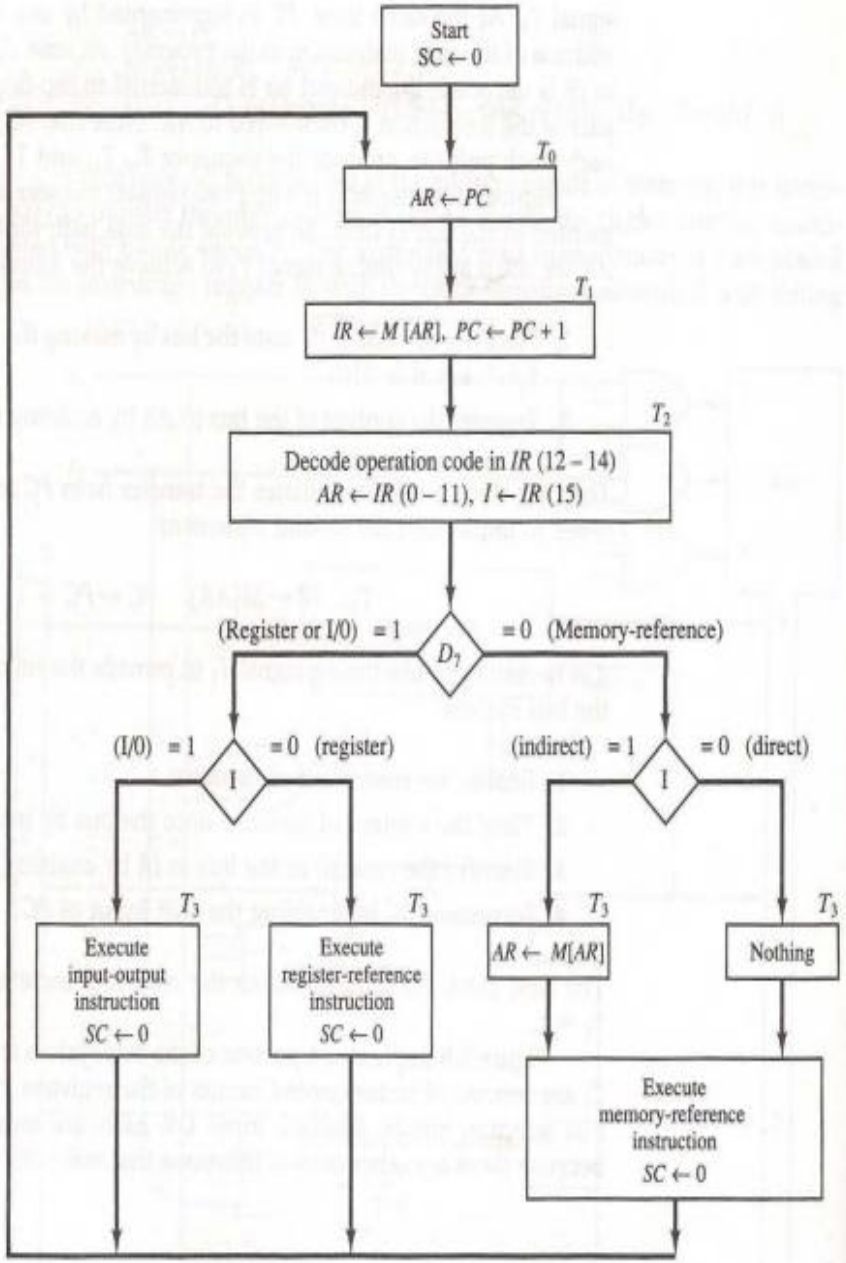


Figure 5-9 Flowchart for instruction cycle (initial configuration).

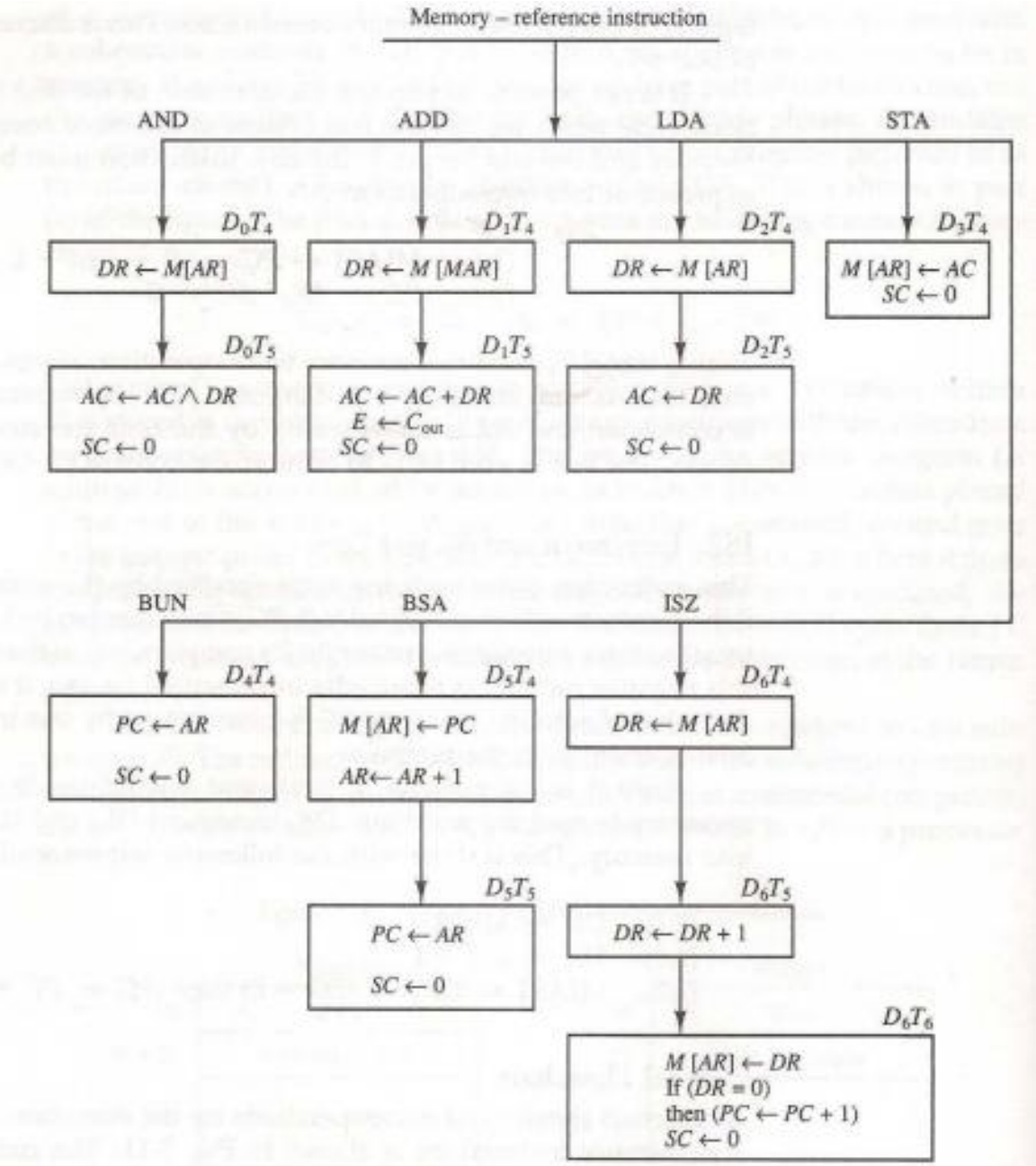
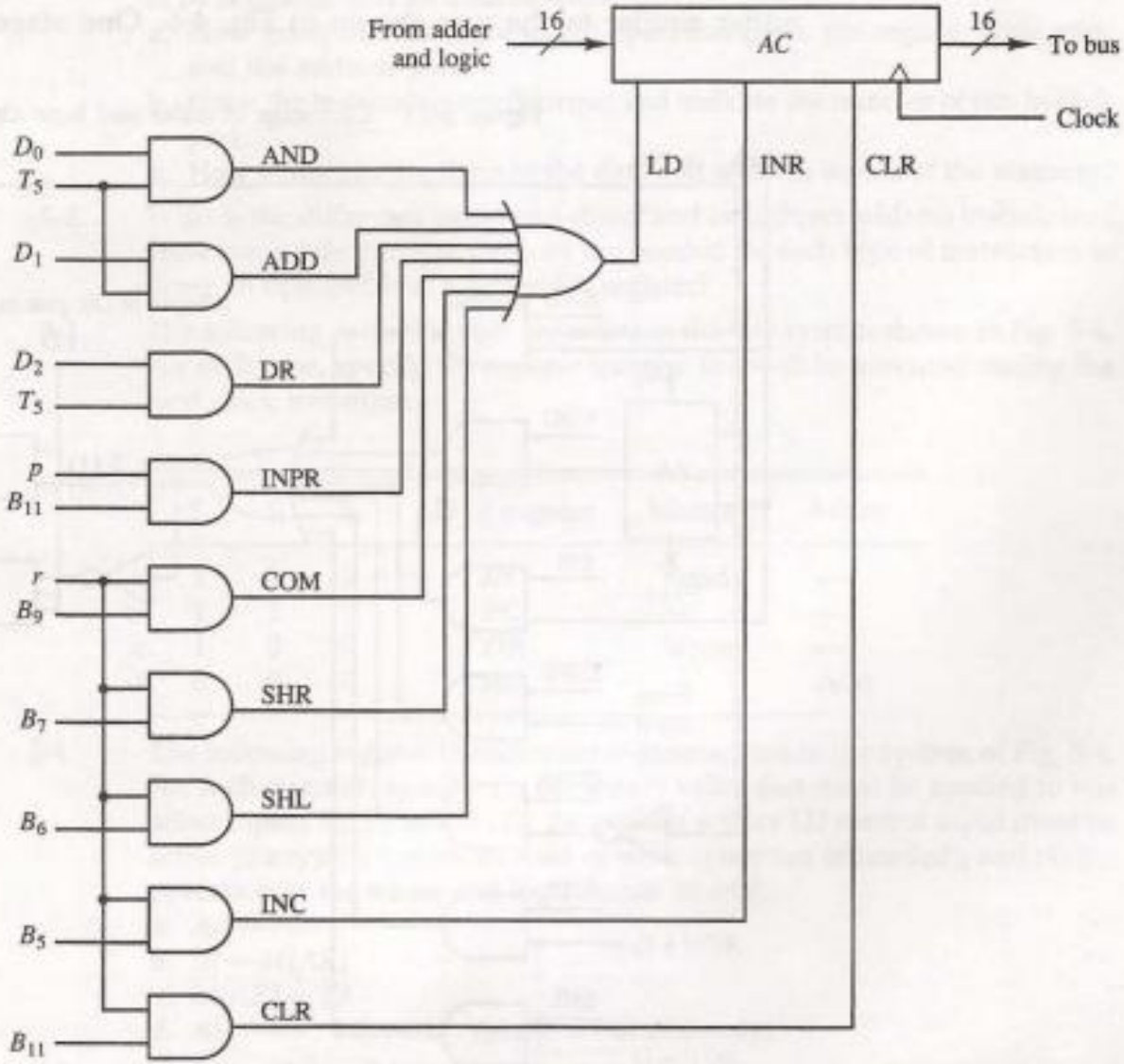


Figure 5-11 Flowchart for memory-reference instructions.

Figure 5-20 Gate structure for controlling the LD, INR, and CLR of *AC*.



RISC (REDUCED INSTRUCTION SET COMPUTER):

- RISC processors typically have a limited set of instructions, often performing basic operations like arithmetic, logic, and data movement.
- RISC processors aim to execute most instructions in a single clock cycle, resulting in faster instruction throughput.
- Few instructions and addressing modes.
- Hardwired rather than microprogrammed control.
- RISC processors typically have a larger number of registers, This reduces the need for memory accesses, improving performance.
- RISC computer has a small set of simple and general instructions, rather than large set of a complex and specialized ones.

Example: SUMMIT (2020), 200 petaFLOPS operations per second

CISC (Complex instruction set computer):

- CISC processors support a wide range of complex instructions that can perform multiple operations in a single instruction.
- CISC architectures often include memory-to-memory operations, where data can be transferred directly between memory locations without involving the CPU registers.
- Instructions in CISC architectures can vary in length, depending on the complexity of the operation being performed.
- CISC processors have specialized hardware to execute complex instructions efficiently. This hardware includes microcode, which translates complex instructions into a series of simpler micro-operations that can be executed by the CPU.
- Compared to RISC architectures, CISC architectures typically have a smaller number of general-purpose registers.
- # of instructions 100-250
- # of Addressing modes 5-20
- Variable length instruction format.

Designing of control unit:

- The control signal can be generated either by hardware (hardware CU) or by micro-program control unit (m/y + hardware).
- In Hardware control unit design, each control signal expressed as SOP expression and realized by digital hardware.
- Fixed logic circuits that correspond directly to the Boolean expression are used to generate control signal.

EXAMPLE OF TIMING:

T_0: $AR \leftarrow PC$

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

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

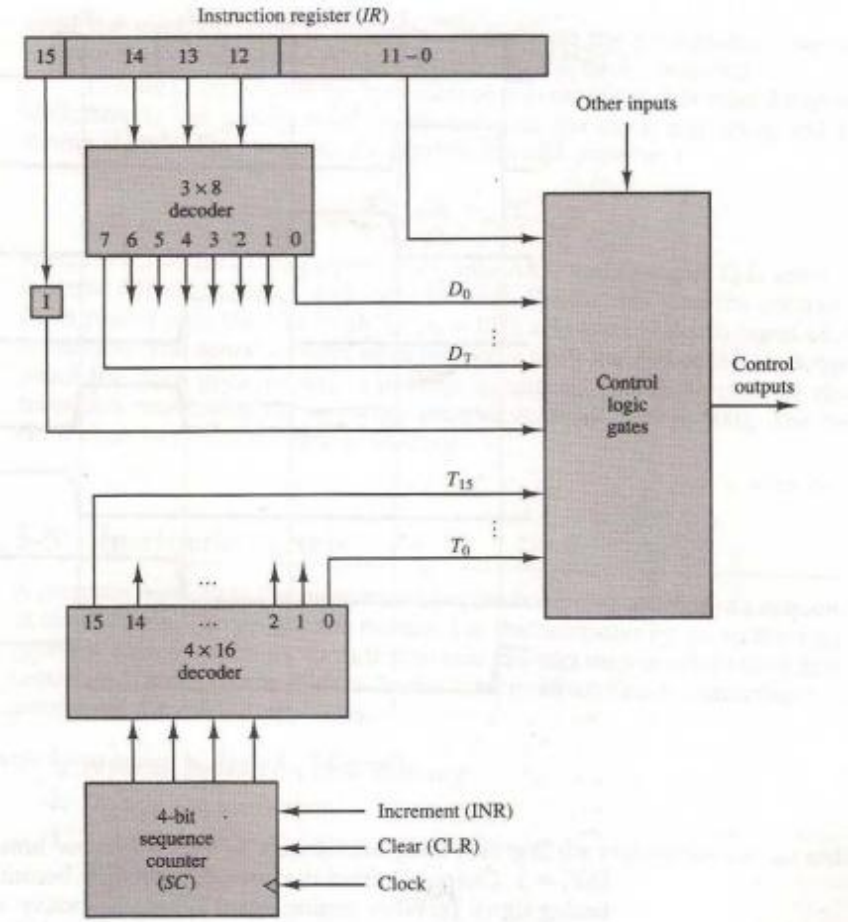


Figure 5-6 Control unit of basic computer.

TABLE 5-5 Input-Output Instructions

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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$	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

Symbol	Hexadecimal code $I = 0$	$I = 1$	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

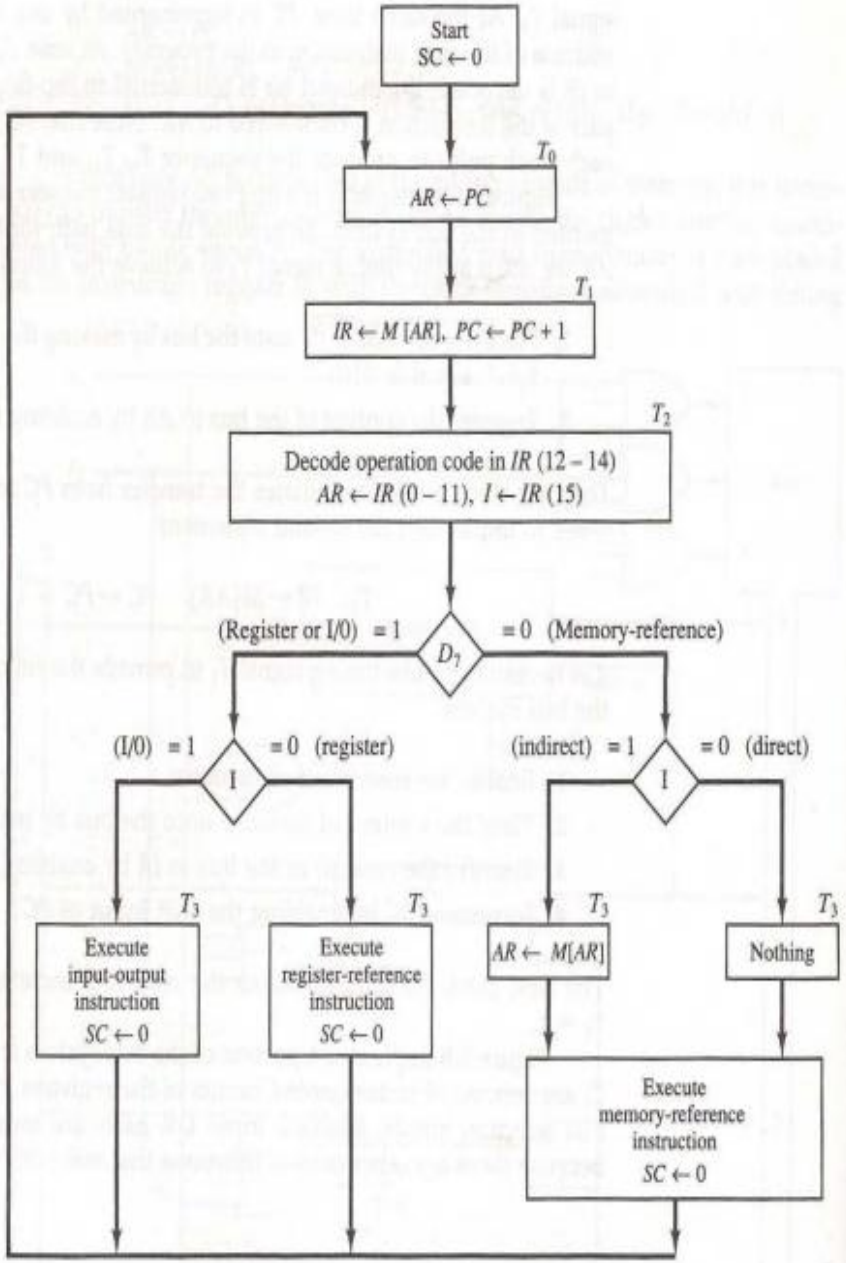


Figure 5-9 Flowchart for instruction cycle (initial configuration).

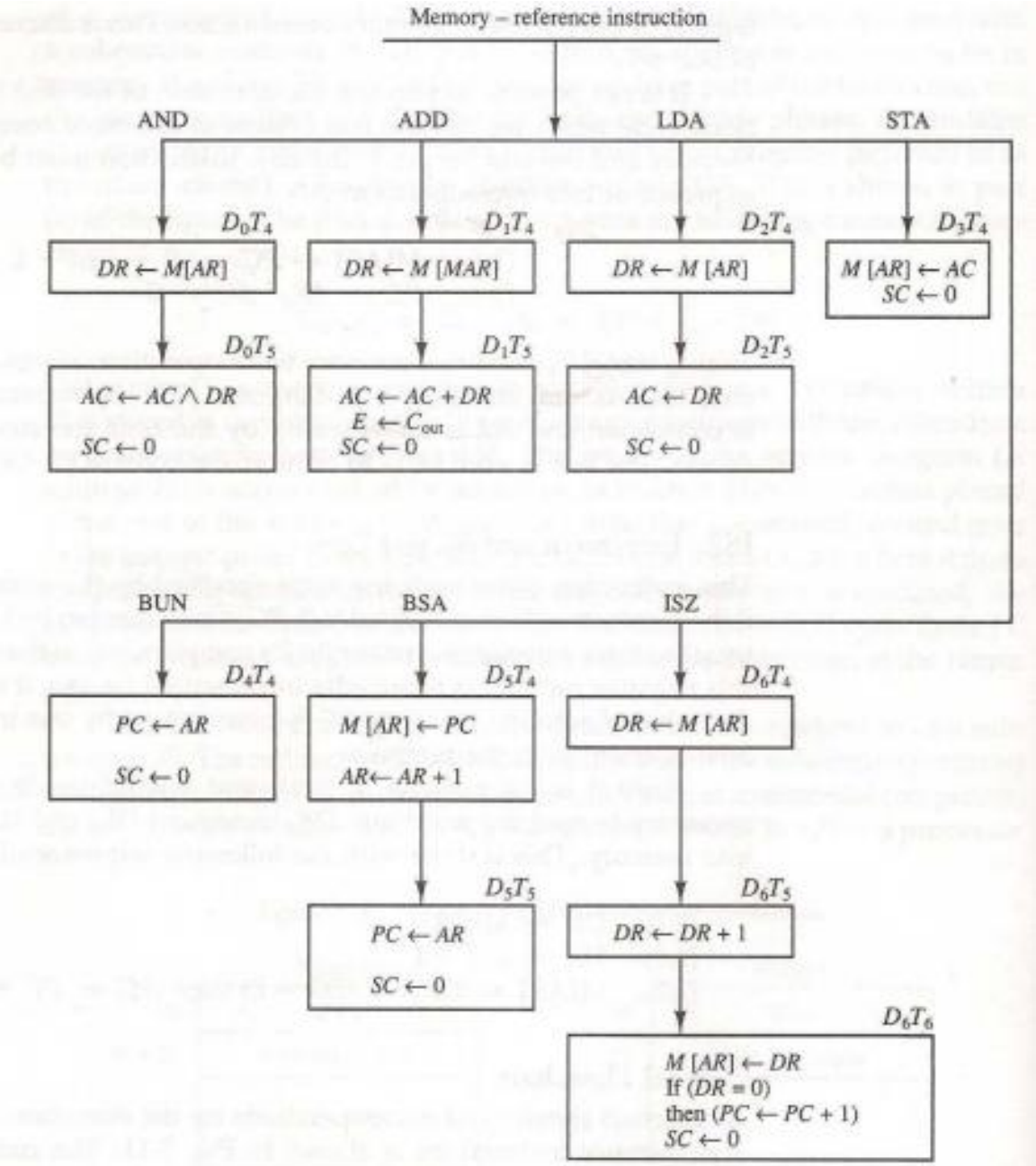
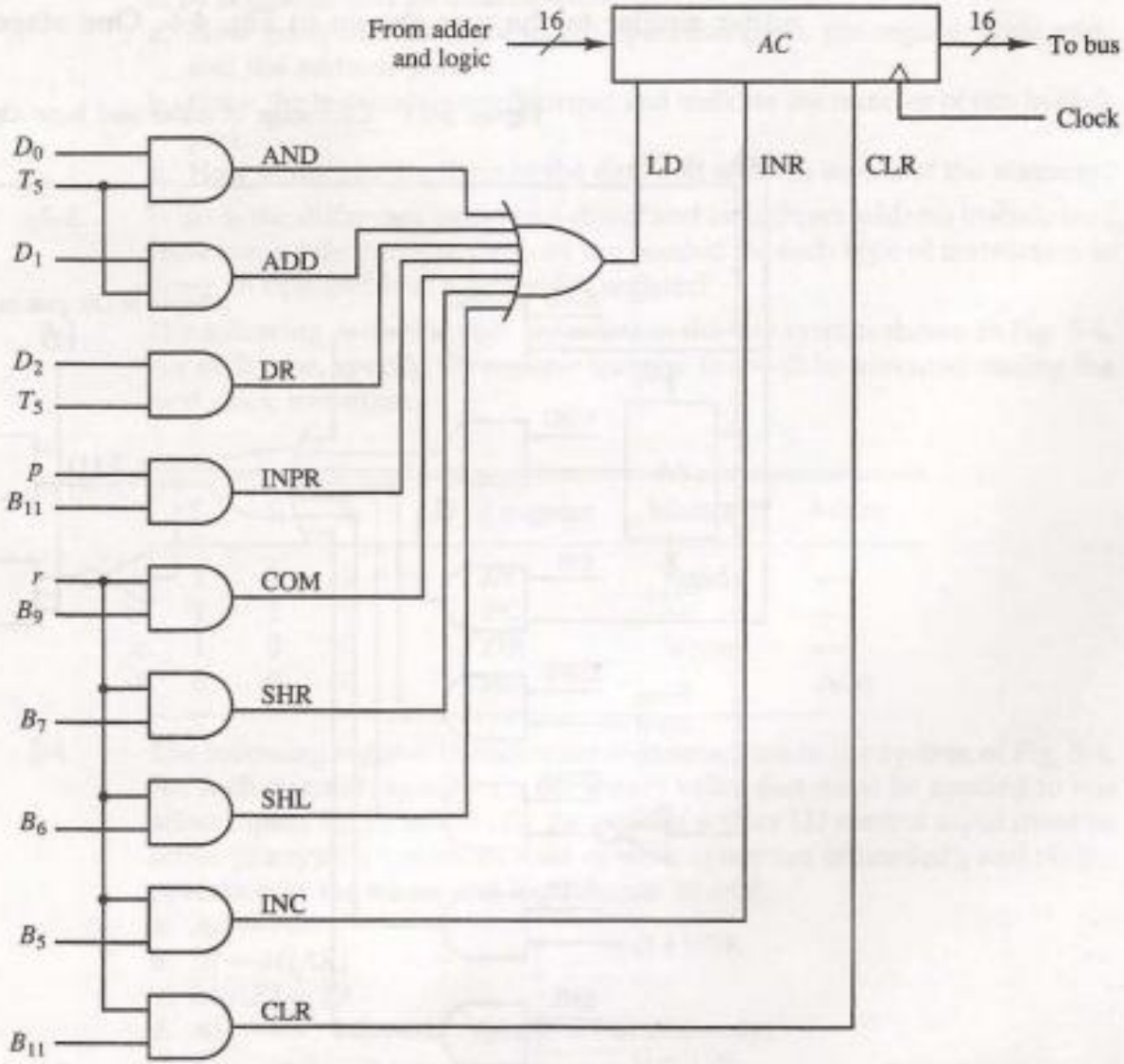


Figure 5-11 Flowchart for memory-reference instructions.

Figure 5-20 Gate structure for controlling the LD, INR, and CLR of *AC*.



Uses three 8-bit register's A, B, and C. Supports two instructions I1 and I2.

Micro-operation	Control Signal	
	I_1	I_2
T_1	A_{in}, B_{out}	A_{in}, A_{out}
T_2	B_{in}, C_{out}	C_{in}, A_{out}
T_3	B_{in}, B_{out}	C_{in}, C_{out}
T_4	A_{in}, A_{out}	A_{in}, B_{out}

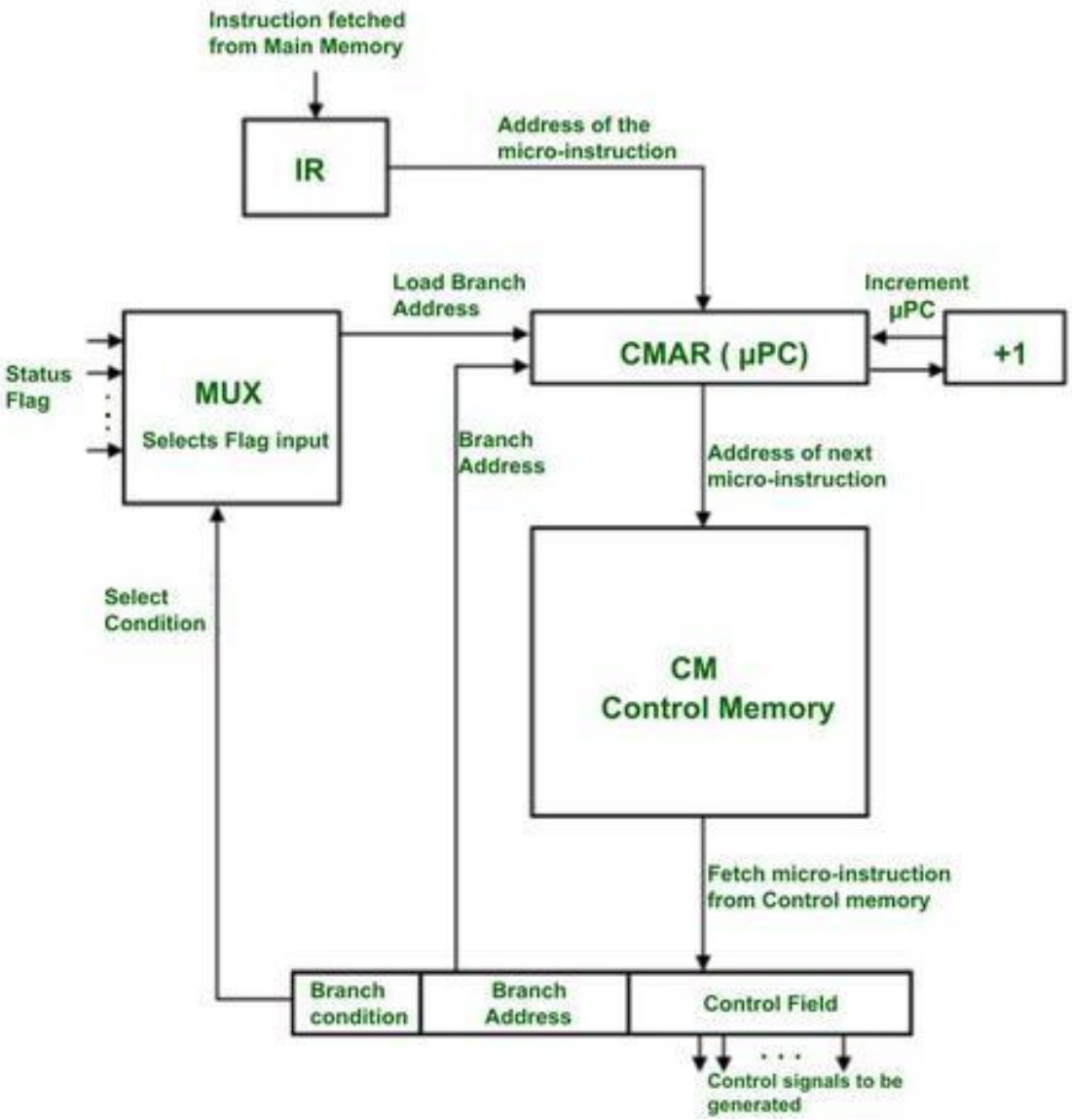
Difference between Hardwired and Micro-Programmed

Characteristics	Hardwired	Micro-programmed control
Speed	Fast	Slow
Implementation	Hardware	Software
Complex instruction	Difficult	Easier
Design Process	Difficult for more operation	Easy
Memory	Not used	Control Memory used
Used in	RISC	CISC

Micro programmed control unit:

- Introduced by Maurice Wilkies (1951).
- Advantage is the simplicity of its structure
- Microprograms were organized as a sequence of microinstructions and stored in a special control memory
- In this the binary patterns of control signals are stored in control m/y. After accessing word from the control memory, hardware is used to generate the control signals.
- Design is flexible and it is used in CISC System
- Types-

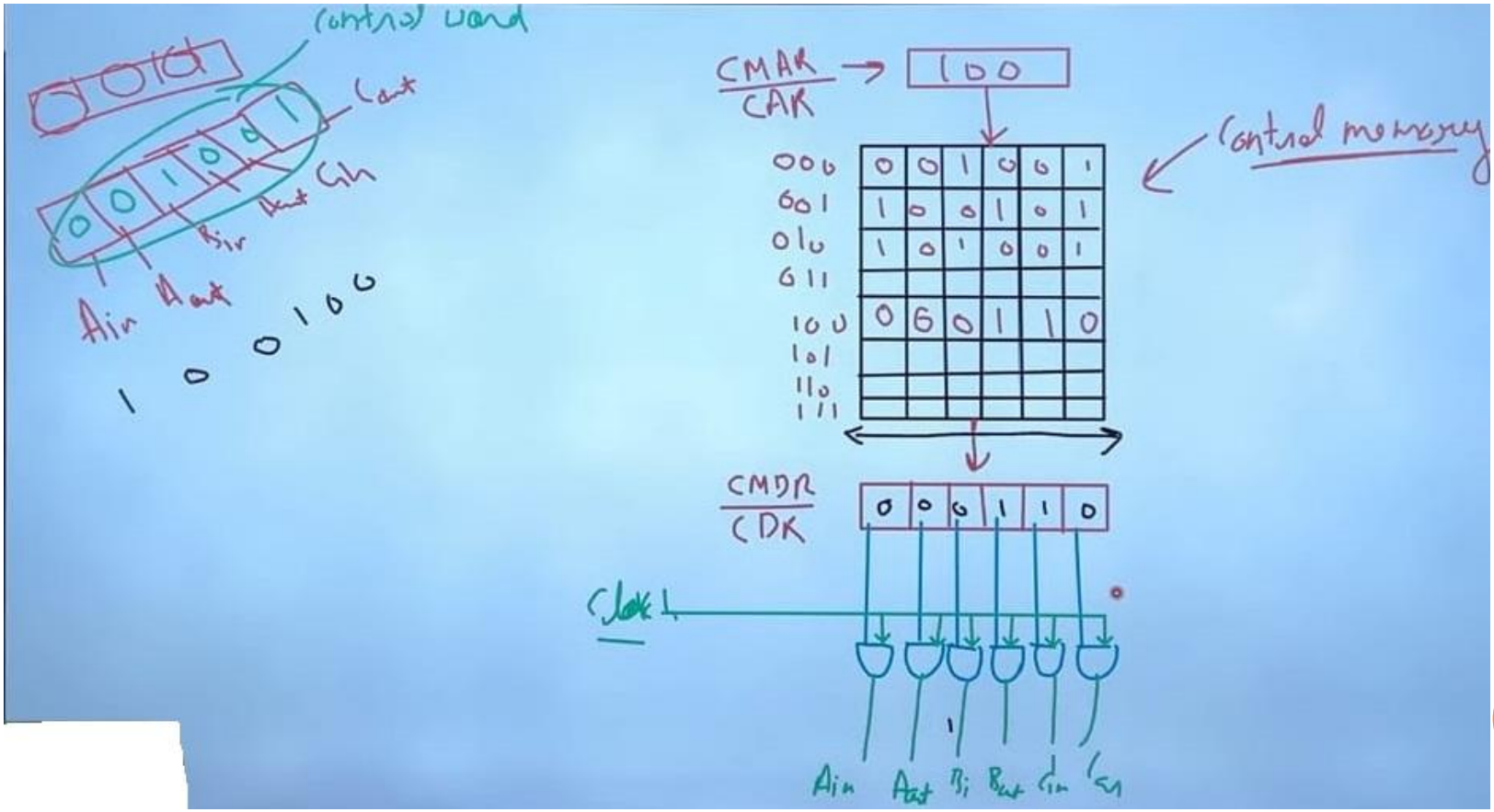
1. Horizontal microprogram
2. Vertical microprogram


Instruction fetched
from Main Memory
IR
Address of the
micro-instruction
Load Branch
Address
Increment
µPC
CMAR (µPC)
+1
Status
Flag
MUX
Selects Flag input
Branch
Address
Address of next
micro-instruction
Select
Condition
CM
Control Memory
Fetch micro-instruction
from Control memory
Branch
condition
Branch
Address
Control Field
Control signals to be
generated

Horizontal microprogram (Example)

Provides higher degree of parallelism, suitable in multi-processor system

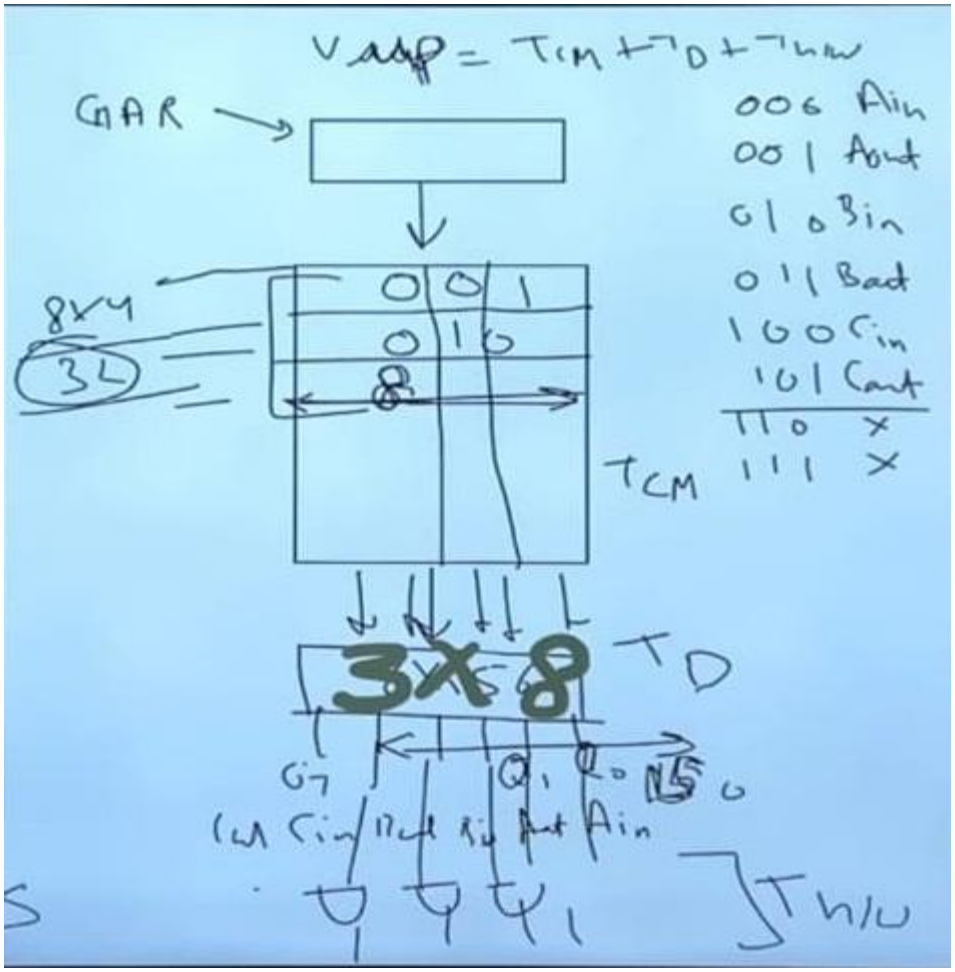
Requires more bits for control word (1 bit for every control signal)



Vertical microprogram (Example)

Reduces the size of control words by encoding.

Maximum degree of parallelism is 1 (due to decoder)



Horizontal	Vertical
Long Format	Short Format
Ability to express high degree of multi-programming	Limited ability to express Parallelism
Little encoding of control information	Considerable encoding
Useful when higher operating speed is desired	Slower operating speed

Note: The Combination of both controlled unit is preferred in most applications

Control Word Sequencing:

- In order to implement micro-program, control words are to be sequentially from control memory. One address instruction is used for performing the control word sequencing.