INSTRUCTION INTERPRETETION

HARDWIRED CONTROL UNIT

MICRO PROGRAMMED CONTROL UNIT

CPU CONTROL UNIT DESIGN

INSTRUCTION INTERPRETATION

Instruction Cycle

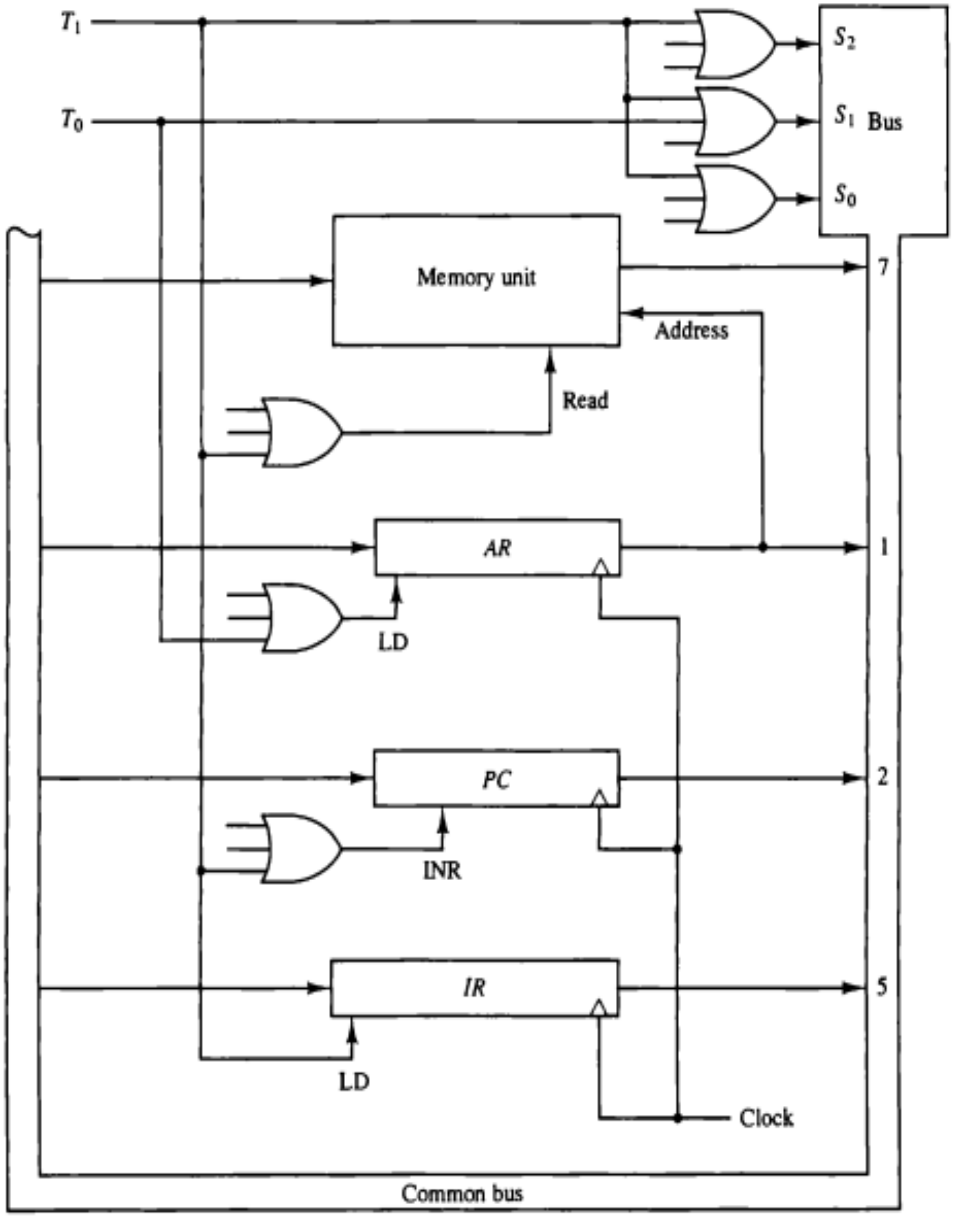
1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

T_0: $AR \leftarrow PC$

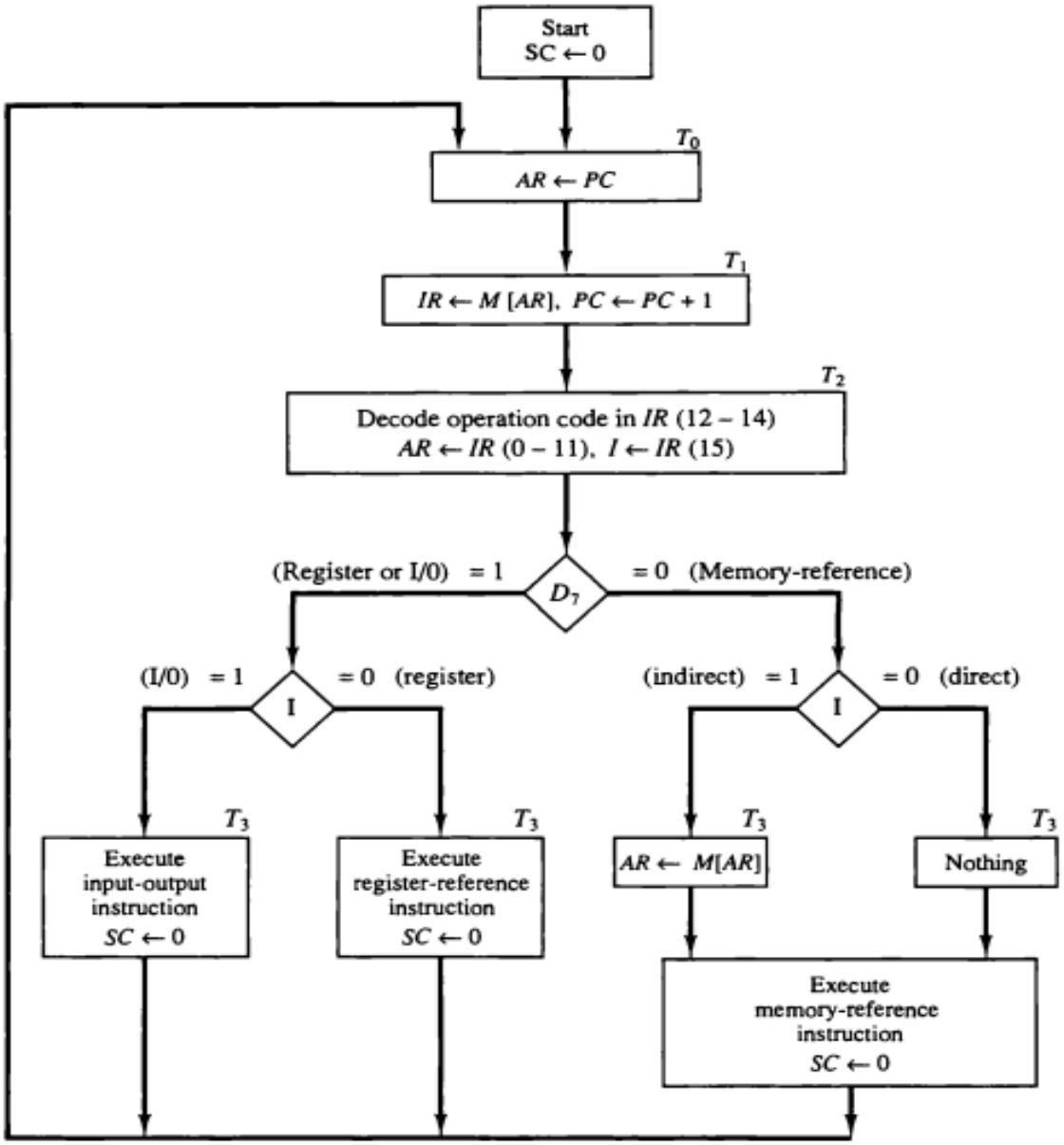
T_1: $IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$

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

INSTRUCTION CYCLE



INSTRUCTION CYCLE



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:		
	$T_0'T_1'T_2'(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	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:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
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:

$D_7 I' T_3 = r$ (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:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
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:

$D_7 I T_3 = p$ (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:	$IEN \leftarrow 0$

HARDWIRED CONTROL UNIT

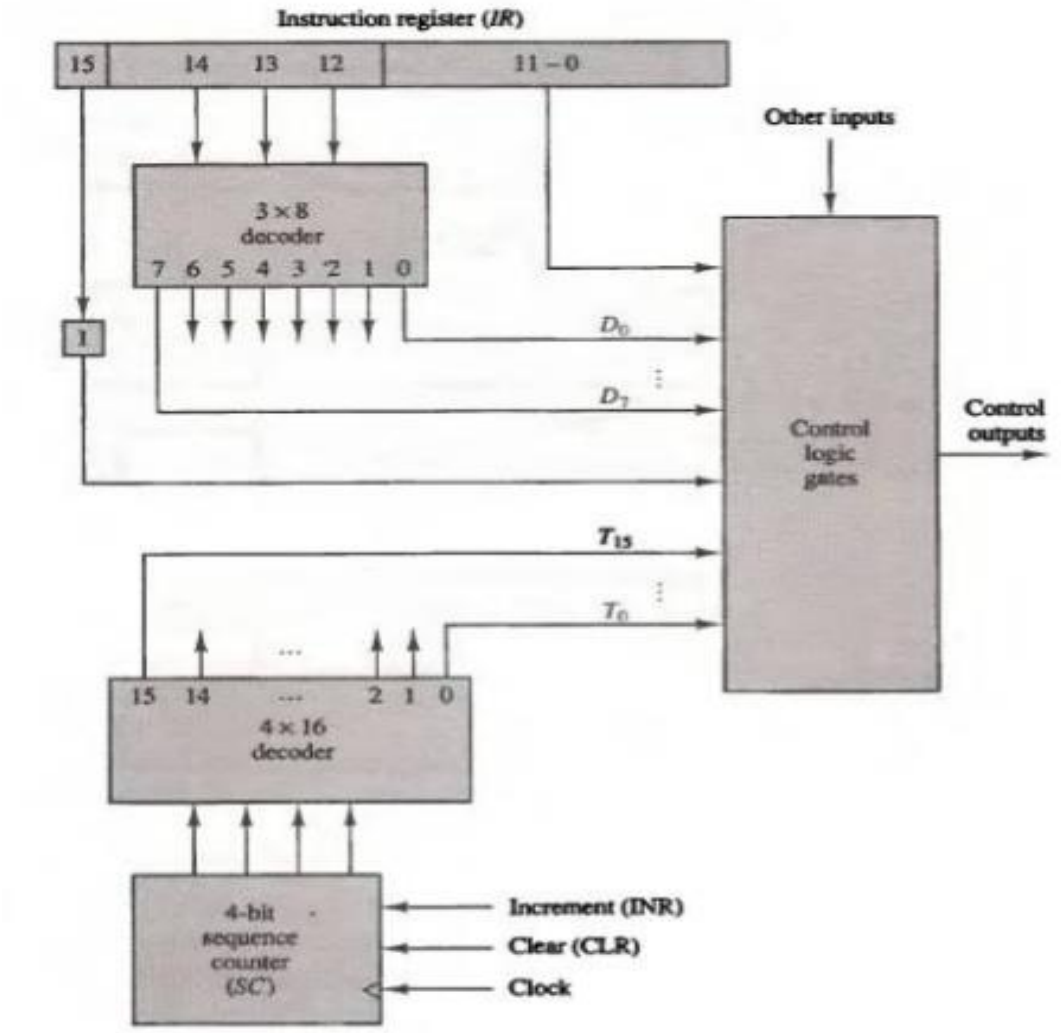


Figure 5-6 Control unit of basic computer.

HARDWIRED CONTROL UNIT

D_3T_4: $SC \leftarrow 0$

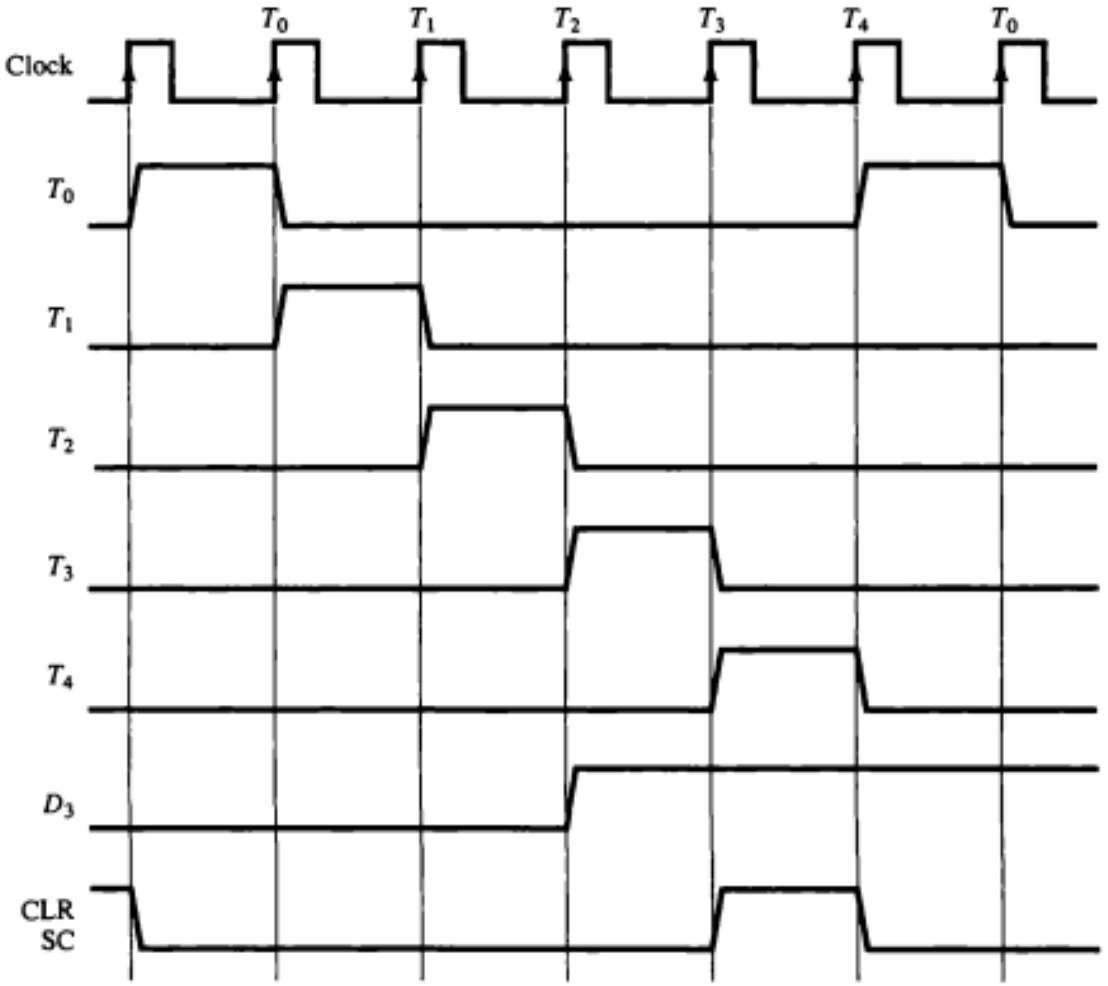


Figure 5-7 Example of control timing signals.

MICROPROGRAMMED CONTROL

- Control Memory
- Sequencing Microinstructions
- Microprogram Example
- Design of Control Unit
- Microinstruction Format
- Nanostorage and Nanoprogram

TERMINOLOGY

Microprogram
- Program stored in memory that generates all the control signals required to execute the instruction set correctly
- Consists of microinstructions

Microinstruction
- Contains a control word and a sequencing word
  - Control Word - All the control information required for one clock cycle
  - Sequencing Word - Information needed to decide the next microinstruction address
- Vocabulary to write a microprogram

Control Memory(Control Storage: CS)
- Storage in the microprogrammed control unit to store the microprogram

Writeable Control Memory(Writeable Control Storage:WCS)
- CS whose contents can be modified
  - -> Allows the microprogram can be changed
  - -> Instruction set can be changed or modified

Dynamic Microprogramming
- Computer system whose control unit is implemented with a microprogram in WCS
- Microprogram can be changed by a systems programmer or a user

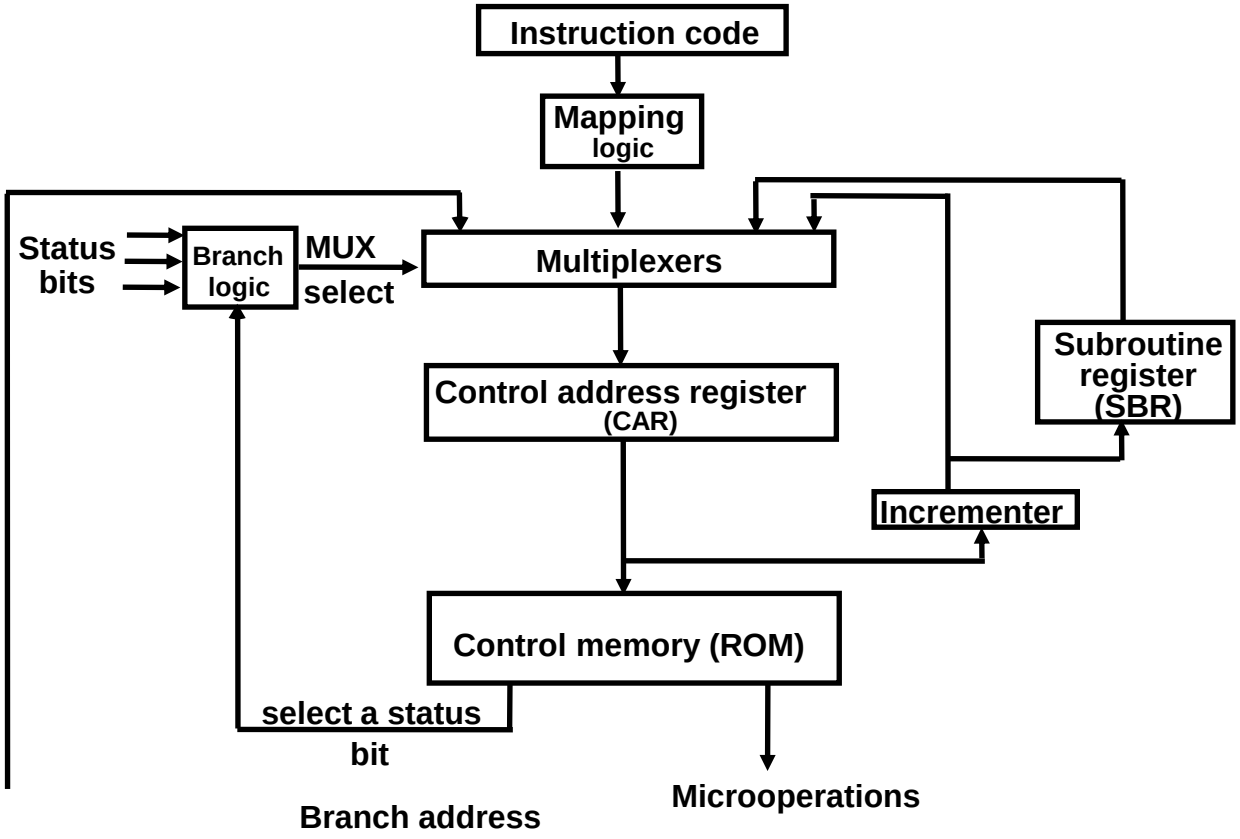
TERMINOLOGY

Sequencer (Microprogram Sequencer)

A Microprogram Control Unit that determines the Microinstruction Address to be executed in the next clock cycle

- **In-line Sequencing**
- **Branch**
- **Conditional Branch**
- **Subroutine**
- **Loop**
- **Instruction OP-code mapping**

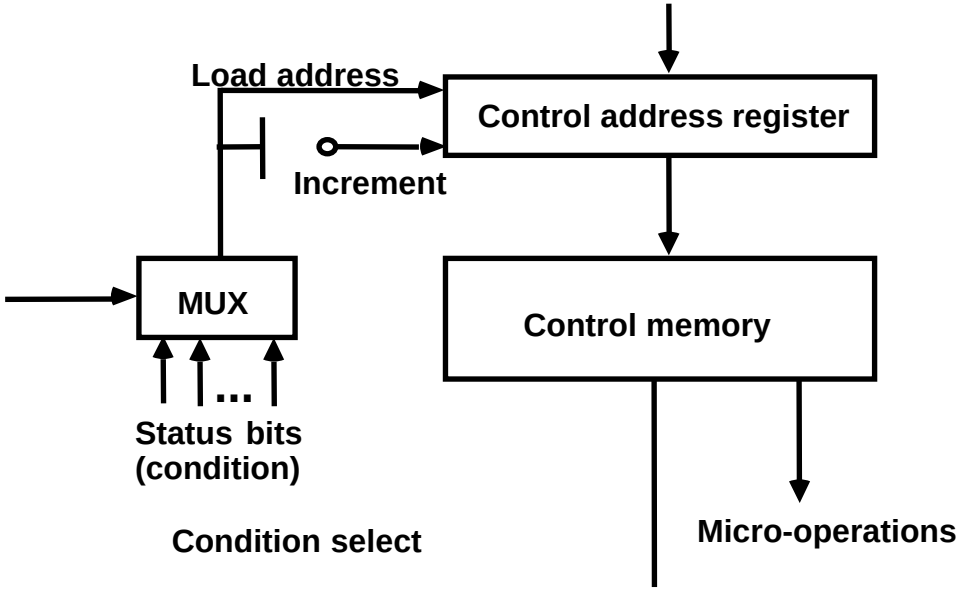
MICROINSTRUCTION SEQUENCING



Sequencing Capabilities Required in a Control Storage

- Incrementing of the control address register
- Unconditional and conditional branches
- A mapping process from the bits of the machine instruction to an address for control memory
- A facility for subroutine call and return

CONDITIONAL BRANCH



Conditional Branch

If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

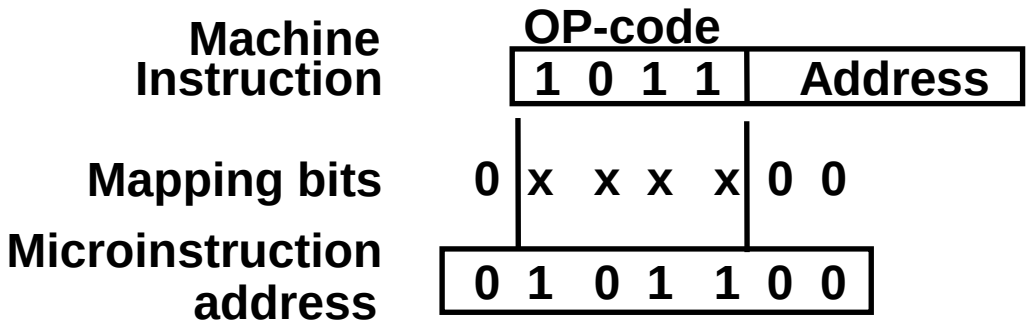
Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.

Unconditional Branch

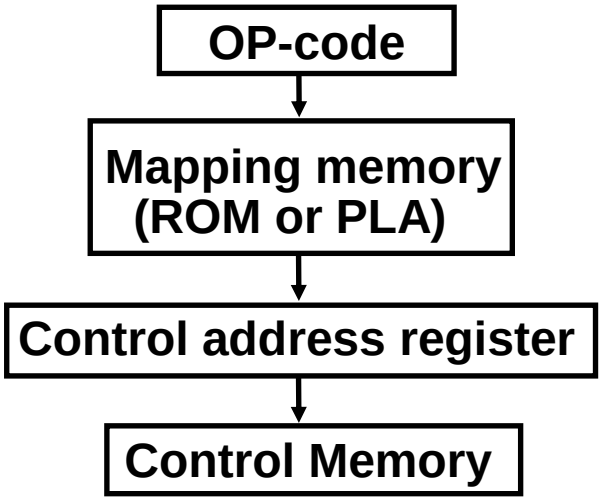
Fixing the value of one status bit at the input of the multiplexer to 1

MAPPING OF INSTRUCTIONS TO MICROROUTINES

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram

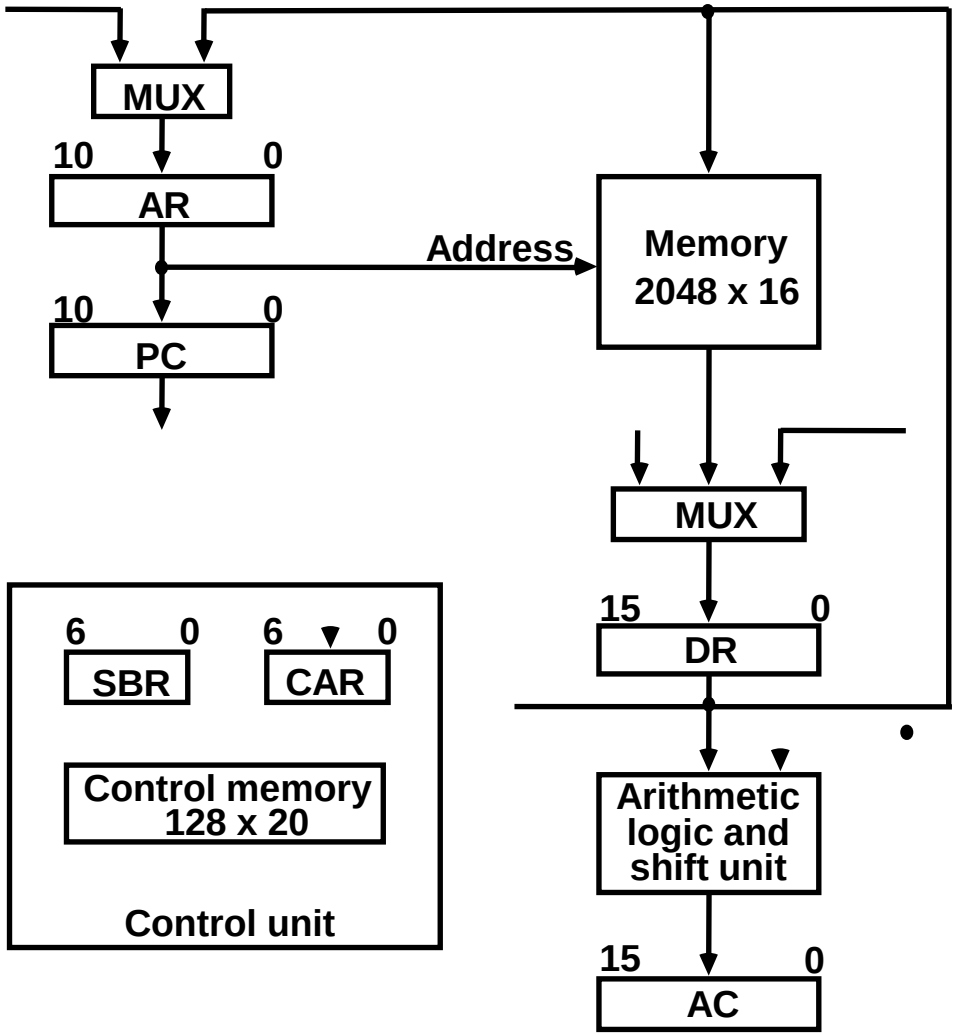


Mapping function implemented by ROM or PLA



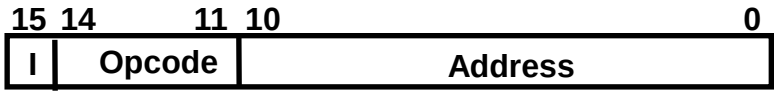
MICROPROGRAM EXAMPLE

Computer Configuration



MACHINE INSTRUCTION FORMAT

Machine instruction format

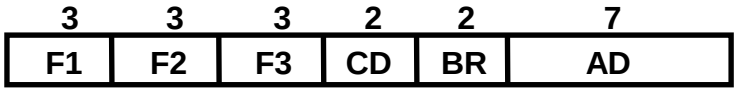


Sample machine instructions

Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Microinstruction Format



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

SYMBOLIC MICROINSTRUCTIONS

- **Symbols are used in microinstructions as in assembly language**
- **A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.**

Sample Format

five fields: label; micro-ops; CD; BR; AD

Label: may be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z}, where
- **U: Unconditional Branch**
- **I: Indirect address bit**
- **S: Sign of AC**
- **Z: Zero value in AC**

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty}

SYMBOLIC MICROPROGRAM - FETCH ROUTINE

During FETCH, Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the fetch cycle:

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 for the fetch cycle:

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

Binary equivalents translated by an assembler

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

- **Control Storage: 128 20-bit words**
- **The first 64 words: Routines for the 16 machine instructions**
- **The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)**
- **Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60**

Partial Symbolic Microprogram

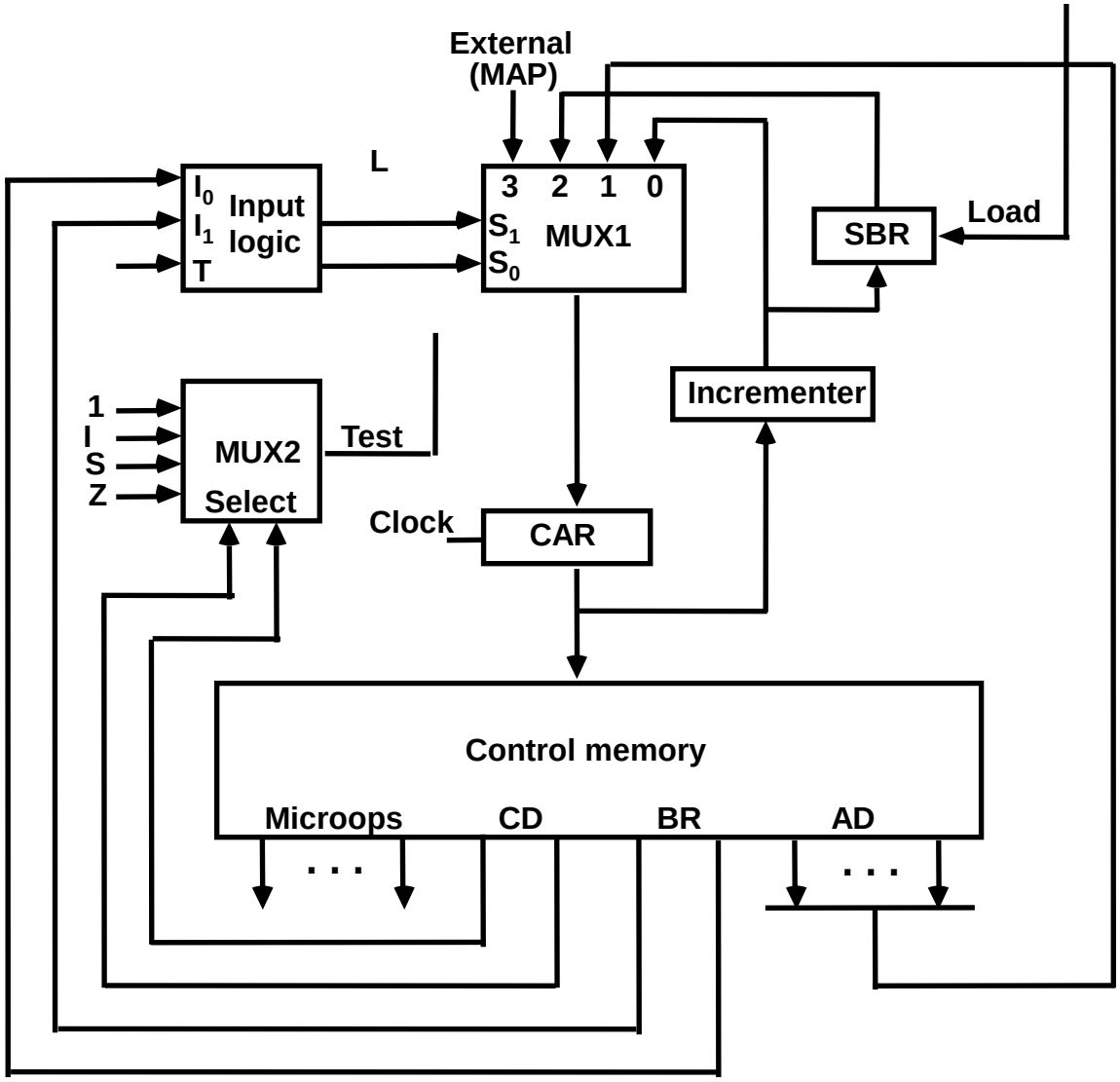
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

BINARY MICROPROGRAM

	Address		Binary Microinstruction					
Micro Routine	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

This microprogram can be implemented using ROM

MICROPROGRAM SEQUENCER



MICROINSTRUCTION FORMAT

Information in a Microinstruction

- **Control Information**
- **Sequencing Information**
- **Constant**

  Information which is useful when feeding into the system

These information needs to be organized in some way for

- **Efficient use of the microinstruction bits**
- **Fast decoding**

Field Encoding

- **Encoding the microinstruction bits**
- **Encoding slows down the execution speed due to the decoding delay**
- **Encoding also reduces the flexibility due to the decoding hardware**

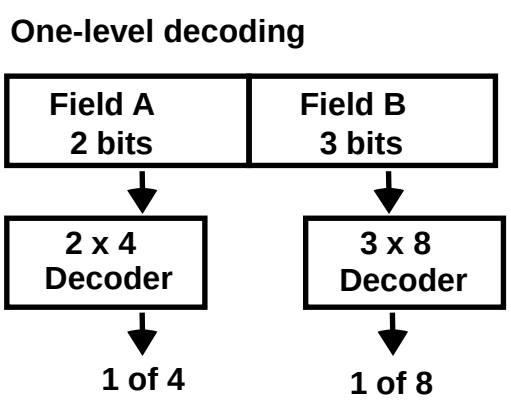
HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

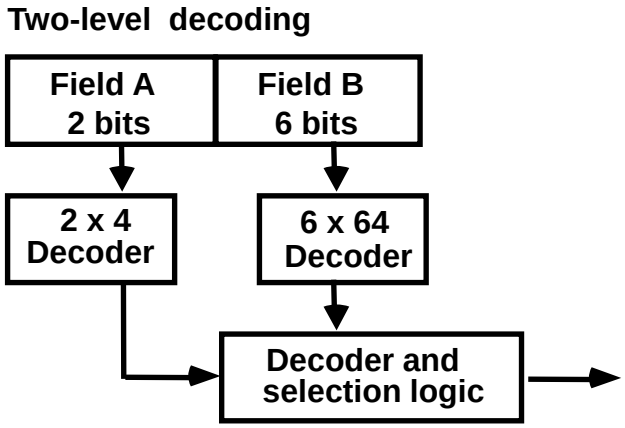
Horizontal Microinstructions

- **Each bit directly controls each micro-operation or each control point**
- ***Horizontal* implies a long microinstruction word**
- **Advantages: Can control a variety of components operating in parallel.**
  --> Advantage of efficient hardware utilization
- **Disadvantages: Control word bits are not fully utilized**
  --> CS becomes large --> Costly

Vertical Microinstructions

- **A microinstruction format that is not horizontal**
- ***Vertical* implies a short microinstruction word**
- **Encoded Microinstruction fields**
  --> Needs decoding circuits for one or two levels of decoding





NANOSTORAGE AND NANOINSTRUCTION

The decoder circuits in a vertical microprogram storage organization can be replaced by a ROM
=> Two levels of control storage
First level - *Control Storage*
Second level - *Nano Storage*

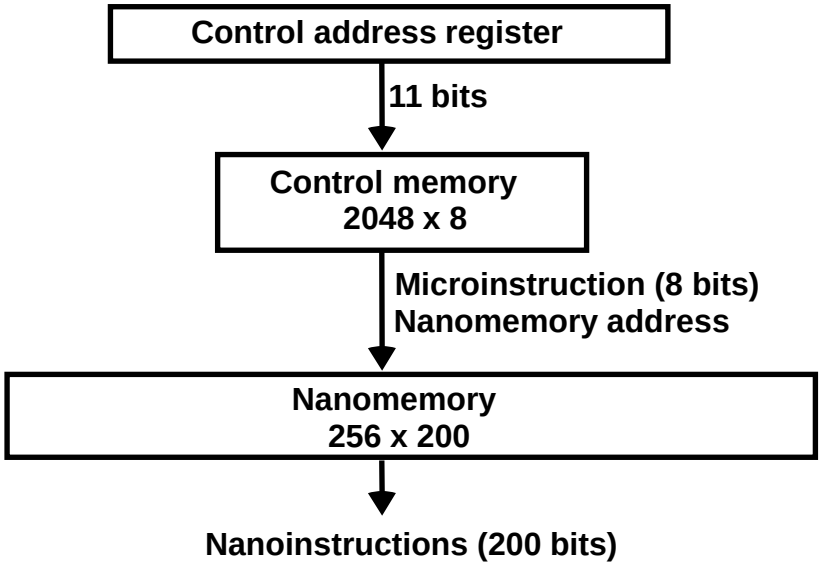
Two-level microprogram

First level
-*Vertical* format Microprogram
Second level
-*Horizontal* format Nanoprogram
- Interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nanoinstruction format.

Usually, the microprogram consists of a large number of short microinstructions, while the nanoprogram contains fewer words with longer nanoinstructions.

TWO-LEVEL MICROPROGRAMMING - EXAMPLE

*** Microprogram: 2048 microinstructions of 200 bits each**
*** With 1-Level Control Storage: 2048 x 200 = 409,600 bits**
*** Assumption:**
256 distinct microinstructions among 2048
*** With 2-Level Control Storage:**
Nano Storage: 256 x 200 bits to store 256 distinct nanoinstructions
Control storage: 2048 x 8 bits
To address 256 nano storage locations 8 bits are needed
*** Total 1-Level control storage: 409,600 bits**
Total 2-Level control storage: 67,584 bits (256 x 200 + 2048 x 8)



DESIGN OF CONTROL UNIT

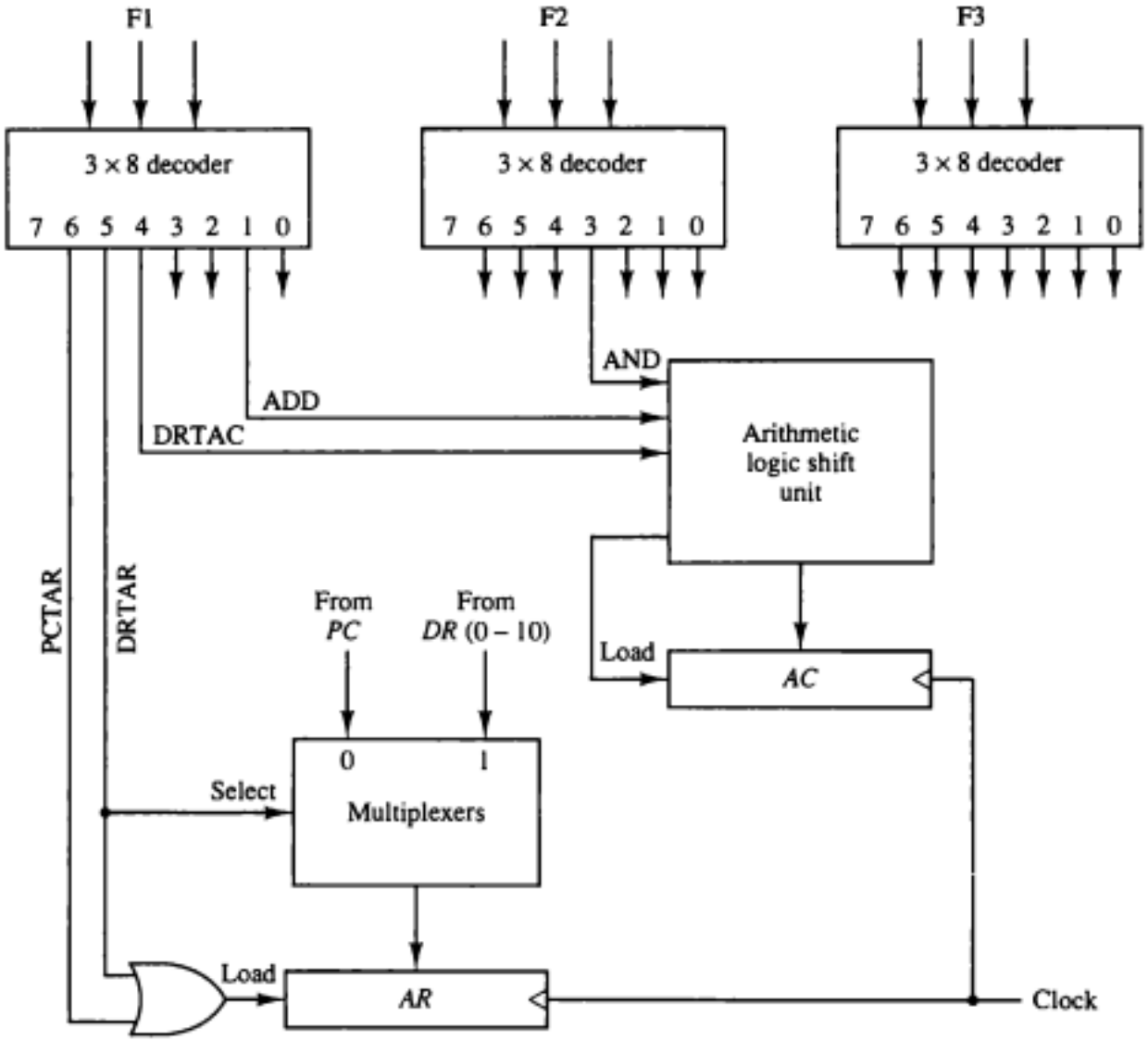


Figure 7-7 Decoding of microoperation fields.

MICROPROGRAM SEQUENER