Chapter 7

Microprogrammed Control

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Alon Schclar, Tel-Aviv College, 2009

Instruction implementation a different perspective

- What is a assembly program?
  - A **sequence** of instructions
  - The instructions are executes **one after the other**
    - Except in branches
  - **Labels** mark beginnings of Subroutines
- The following slide
  - **Regard it as a program** (similar to assembly)
  - Order of execution is determined according to $\boldsymbol{T_i}$
    - Generated by the Sequence Counter (SC)
    - Labels are regarded as starting points of routines

TABLE 5-6 Control Functions and Microoperations for the Basic Computer

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$,
		$AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$
Indirect	$D_7'IT_3$:	$AR \leftarrow M[AR]$
Interrupt:		
$T_0'T_1'T_2'(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, \quad TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, \quad PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, \quad SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC, \quad SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, \quad SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, \quad SC \leftarrow 0$
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 $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

Microroutines

Microinstructions

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Alon Schclar, Tel-Aviv College, 2009

Symbolic microprogram (partial)

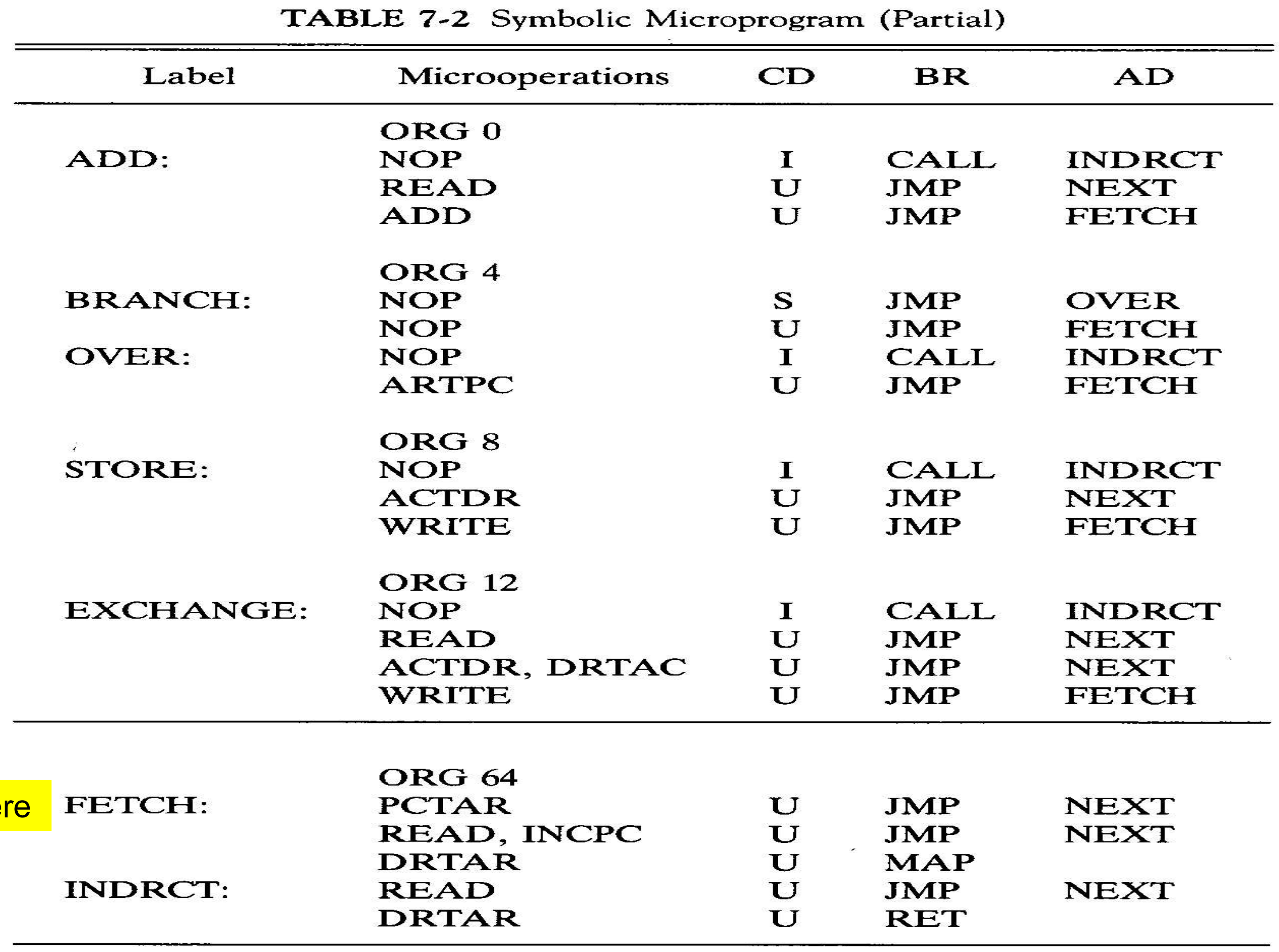


TABLE 7-2 Symbolic Microprogram (Partial)

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

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Example Computer

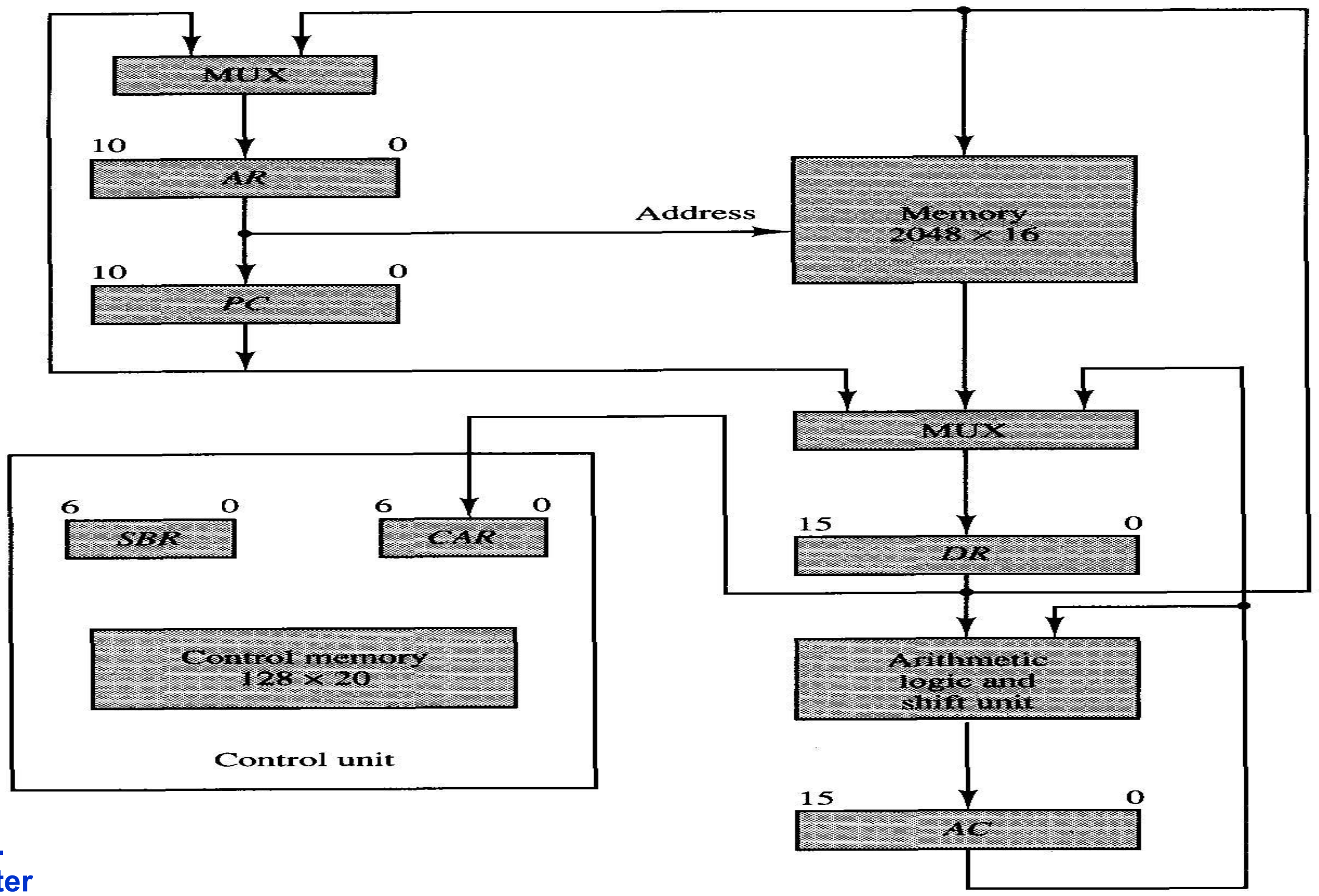


Figure 7-4 Computer hardware configuration.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Instruction implementation a different perspective

- Order of execution
  - Fetch & Decode
    common to all machine instructions
  - Proceed to the appropriate routine **according to the opcode**
  - Every instruction has its own microroutine

What is Microprogramming ?

- Each row in Table 5-6 is called a **microinstruction**
- ***Microprogram*** *- The collection of all microinstructions*
- Store the instruction implementation in special memory – ***Control unit memory***
  - Every microinstruction in a different control memory address

What is Microprogramming ? – cont.

- Each instruction is executed by performing the appropriate microinstructions
  - FETCH + microroutine(s)
- Special hardware will be constructed to facilitate this way of execution
  - Need a Microprogram Counter (similar to **PC**)
    - **Control Address Register (CAR)**
  - Need a mechanism to determine the address of the next instruction CAR++/Branch
    - **Sequencer**
  - Subroutine call and return mechanisms

Chap. 7 Microprogrammed Control*(Control Unit)*

7-1 Control Memory

Control Unit

- Initiate sequences of microoperations
  - » Control signal *(that specify microoperations)* in a bus-organized system
    - groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units
- Two major types of Control Unit
  - » Hardwired Control : *in Chap. 5*
    - The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
    - + Fast operation, - Wiring change(if the design has to be modified)
  - » Microprogrammed Control : *in this Chapter*
    - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
    - + Any required change can be done by updating the microprogram in control memory,
      - Slow operation

Control Word

- The control variables at any given time can be represented by a string of 1's and 0's. The control values for the LD/CLR/INR of the CPU components

Microprogrammed Control Unit

- A control unit whose binary control variables are stored in memory (*control memory*).

- **Microinstruction** : *Control Word in Control Memory*
  - The microinstruction specifies one or more microoperations
- **Microprogram**
  - A sequence of microinstruction
    - » Dynamic microprogramming : *Control Memory* = RAM
      - RAM can be used for writing (*to change a writable control memory*)
      - Microprogram is loaded initially from an auxiliary memory such as a magnetic disk
    - » Static microprogramming : *Control Memory* = ROM
      - Control words in ROM are made permanent during the hardware production.
- **Microprogrammed control Organization** : ***Fig. 7-1***
  - 1) Control Memory
    - » A memory is part of a control unit : *Microprogram*
    - » Computer Memory (*employs a microprogrammed control unit*)
      - Main Memory : for storing user program (*Machine instruction/data*)
      - Control Memory : for storing microprogram (*Microinstruction*)
  - 2) Control Address Register
    - » Specify the address of the microinstruction
  - 3) Sequencer (= *Next Address Generator*)
    - » Determine the address sequence that is read from control memory
    - » Next address of the next microinstruction can be specified several way depending on the sequencer input : *p. 217, [1, 2, 3, and 4]*

$$\begin{tikzcd}
\text{User Program} \arrow[d] \\
\text{Machine Instruction} \arrow[d] \\
\text{Microprogram} \arrow[d] \\
\text{Microinstruction} \arrow[d] \\
\text{Microoperation}
\end{tikzcd}$$

- 4) Control Data Register (=*Pipeline Register*)
  - » Hold the microinstruction read from control memory
  - » Allows the execution of the microoperations specified by the control word ***simultaneously*** with the generation of the next microinstruction

- RISC Architecture Concept
  - RISC(Reduced Instruction Set Computer) system use hardwired control rather than microprogrammed control : ***Sec. 8-8***

7-2 Address Sequencing **of microprogram**

- Address Sequencing = Sequencer : Next Address Generator
  - Selection of address for control memory
- Routine $\Rightarrow$ *Subroutine : program used by other ROUTINES*
  - Microinstruction are stored in control memory *in groups*
- Mapping
  - Instruction Code $\longrightarrow$ Address in control memory(*where routine is located*)
- Address Sequencing Capabilities : ***control memory address***
  - 1) Incrementing of the control address register
  - 2) Unconditional branch or conditional branch, depending on status bit conditions
  - 3) Mapping process (*bits of the instruction ~~address~~ for control memory*) New instruction
  - 4) A facility for subroutine return

Address sequencing

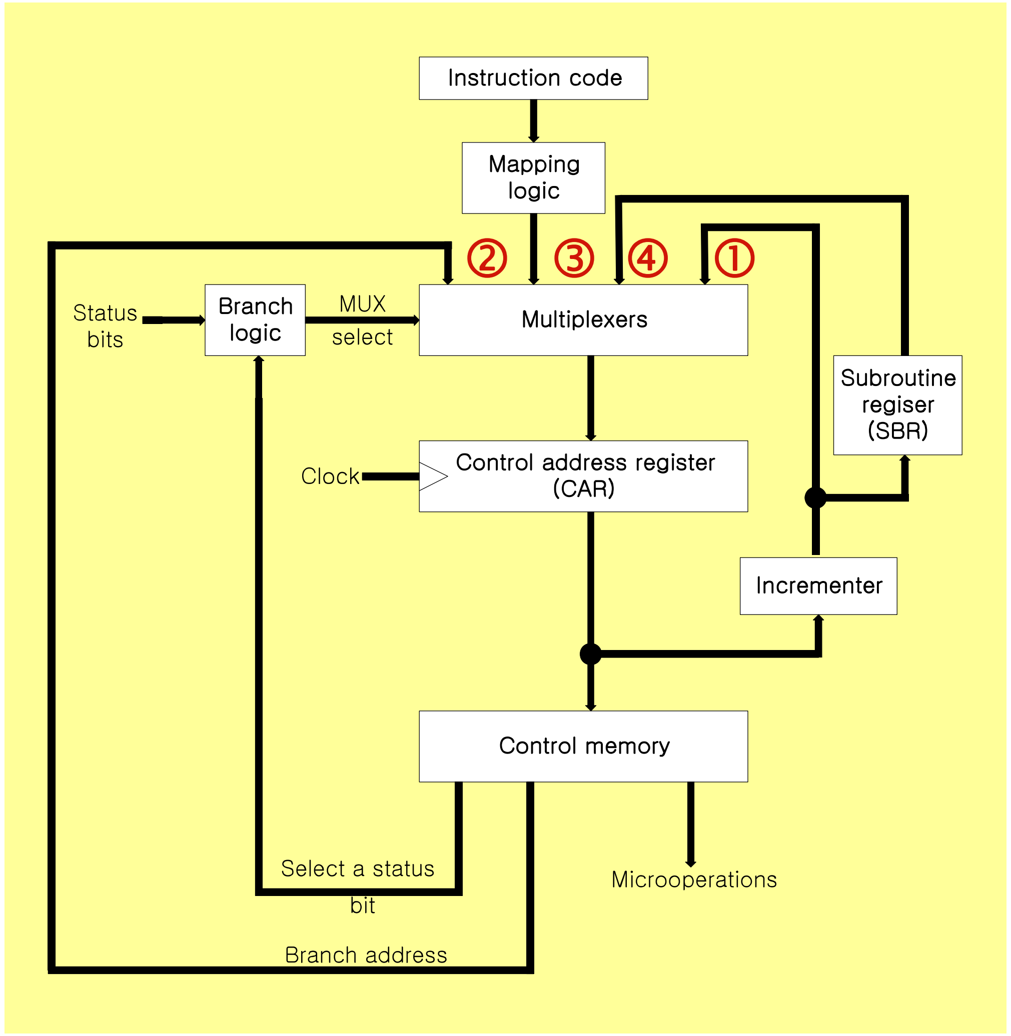
- **Fetch**
  - When **power is turned on** in the computer
    - An **initial address** is loaded into the control address register (CAR)
    - usually the address of the first microinstruction of the **fetch** routine/microprogram.
  - Fetch routine may be **sequenced** by incrementing the control address register
    - through the rest of its microinstructions. **3 microinstructions in total**
  - At the end of the fetch routine, the **first machine instruction (MI)** is in the **instruction register** of the computer.

Address sequencing – cont.

- **Determine effective address**
  - May be a microsubroutine
    - Branch microinstruction
  - At the end of its execution – AR contains the effective address
- **Execute**
  - Generate **microoperations** (RTL)
  - **Depend** on **MI bits** (opcode etc.
  - Individual routines stored at given locations of control memory (**one routine for each instruction**)
  - Its address obtained by ***mapping*** (different from branch)
  - **When finished** – start the fetch routine again
    - Unconditional branch

◆ Selection of address for control memory : *Fig. 7-2*

- Multiplexer
  - ① CAR Increment
  - ② JMP/CALL
  - ③ Mapping
  - ④ Subroutine Return
- CAR : Control Address Register
  - » CAR receive the address from 4 different paths
    1) Incrementer
    2) Branch address from control memory
    3) Mapping Logic
    4) SBR : Subroutine Register
- SBR : Subroutine Register
  - » Return Address can not be stored in ROM
  - » Return Address for a subroutine is stored in SBR



Selection of address for control memory

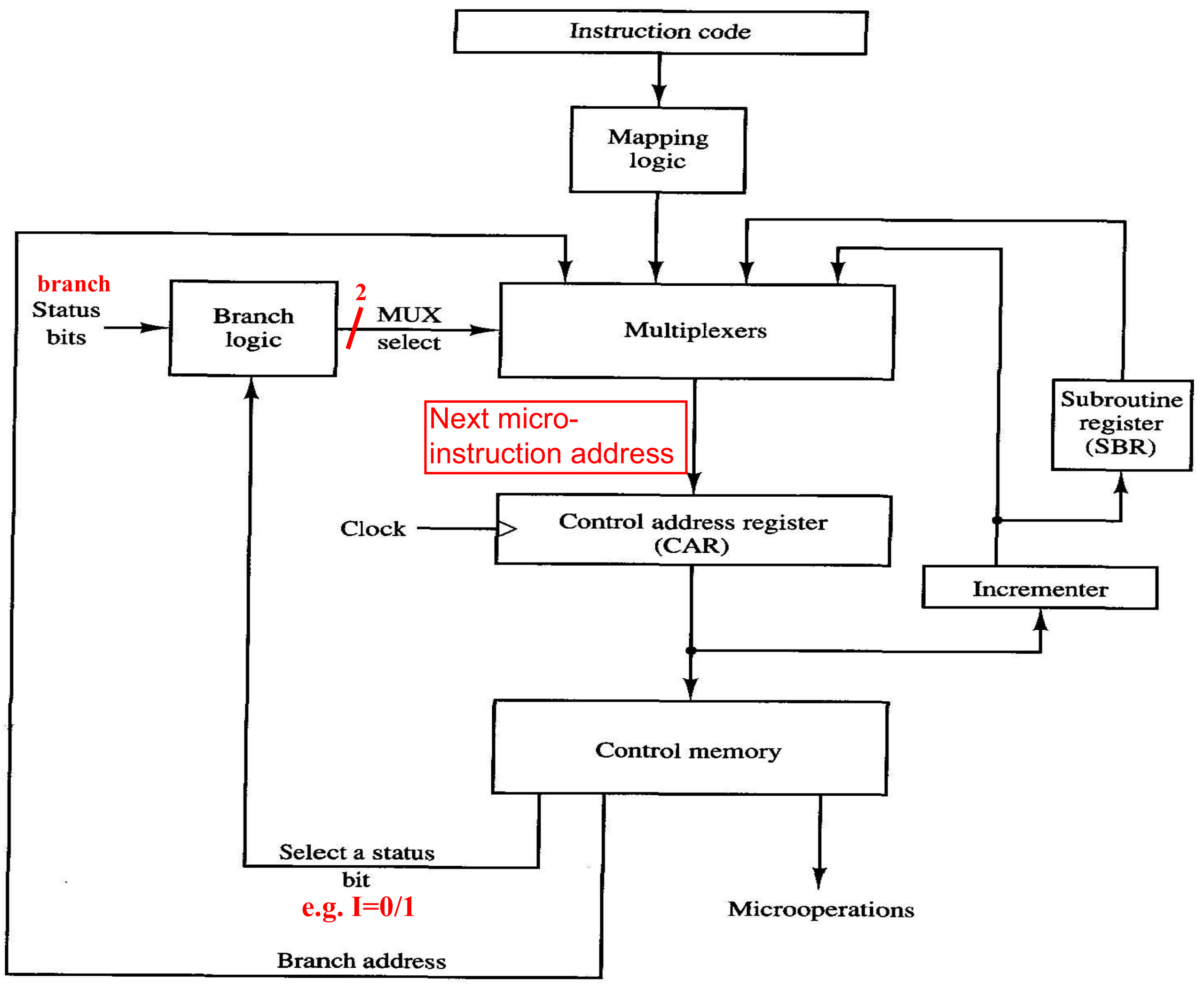


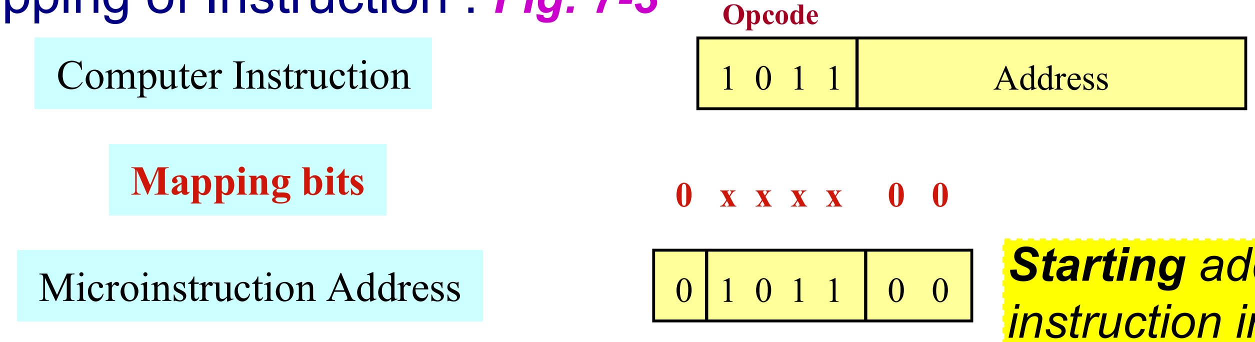
Figure 7-2 Selection of address for control memory.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Conditional Branching

- Status Bits
  - » Control the conditional branch decisions generated in the ***Branch Logic***
- Branch Logic
  - » Test the specified condition and Branch to the indicated address if the condition is met ; otherwise, the control address register is just incremented. *if (AC<0) then PC⟵EA*
- Status Bit Test Branch Logic ***Fig. 7-8***
  - » 4 X 1 Mux Input Logic(***Tab. 7-4***)

Mapping of Instruction : ***Fig. 7-3***

- 

***Starting** address of micro-instruction in control memory*

- 4 bit Opcode = specify up to 16 distinct instruction
- Mapping *Process* : Converts *the 4-bit Opcode* to *a 7-bit control memory address*
  - » 1) Place a “0” in the most significant bit of the address
  - » 2) Transfer 4-bit Operation code bits
  - » 3) Clear the two least significant bits of the CAR (*4* *Microinstruction*)
- Mapping *Function* : Implemented by *Mapping ROM* or *PLD*
- Control Memory Size : 128 words (= 2^7)

Subroutine

- Subroutines are programs that are used by other routines
  - » Subroutine can be called from any point within the main body of the microprogram
- Microinstructions can be saved by subroutines that use common section of microcode
  - » Memory Reference Operand: Effective Address Subroutine
    - p. 228, ***Tab. 7-2*** **INDRCT** (*FETCH* *INDRCT* *Subroutine*)
    - Subroutine ORG 64, 1000000 - 1111111 (*Routine* *0000000 - 0111111*)
- Subroutine must have a provision for
  - » storing the return address during a subroutine call *Subroutine register (SBR)*
  - » restoring the address during a subroutine return
    - Last-In First Out(LIFO) Register Stack : ***Sec. 8-7***

7-3 Microprogram Example

Computer Configuration : ***Fig. 7-4***

- 2 Memory : Main memory(*instruction*/*data*), Control memory(*microprogram*)
  - » Data written to memory come from DR, and Data read from memory can go *only to DR*
- 4 CPU Register and ALU : DR, AR, PC, AC, ALU
  - » DR can receive information from AC, PC, or Memory (*selected by MUX*)
  - » AR can receive information from PC or DR (*selected by MUX*)
  - » PC can receive information only from AR
  - » ALU performs microoperation with data from AC and DR (*AC*)
- 2 Control Unit Register : SBR, CAR

Instruction Format

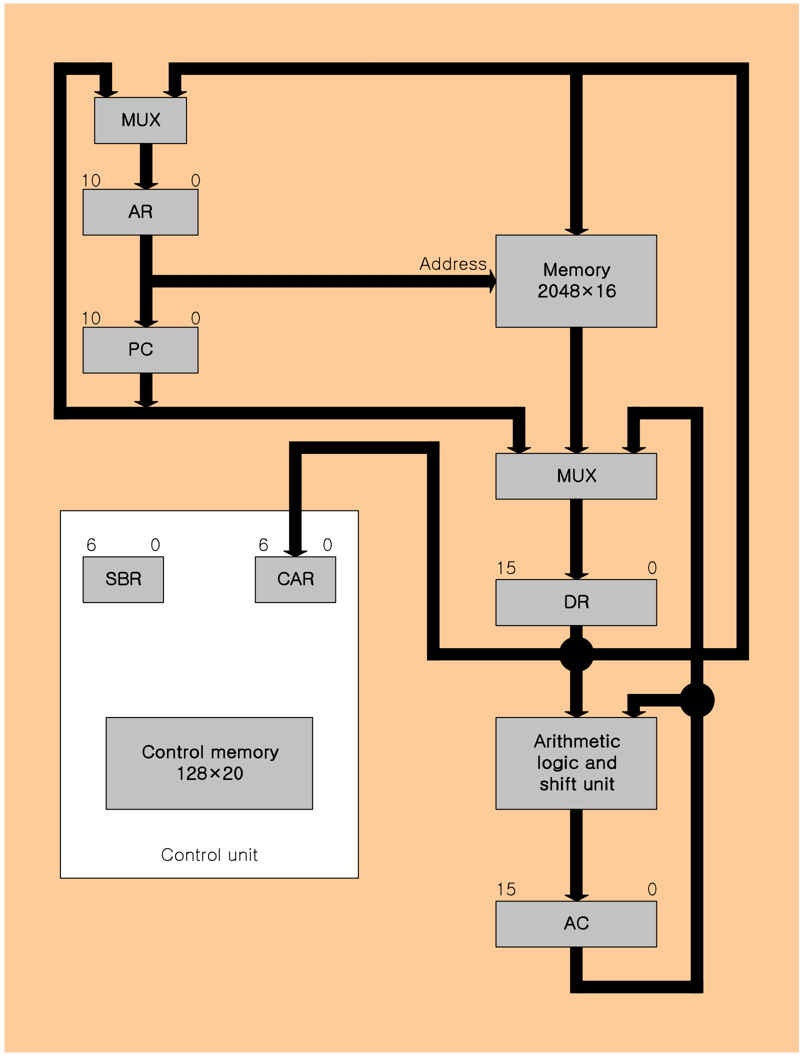
- Instruction Format : ***Fig. 7-5(a)***
  - » I : 1 bit for indirect addressing
  - » Opcode : 4 bit operation code
  - » Address : 11 bit address for system memory
- Computer Instruction : ***Fig. 7-5(b)***
  - » *For simplicity: Only 4 out of the 16*

Microinstruction Format : ***Fig. 7-6***

- 3 bit Microoperation Fields : F1, F2, F3
  - » *Possible 21 Microops* ***Tab. 7-1***
  - » *at most 3 per microinstruction*

    000(no operation)
  - » two or more conflicting microoperations can not be specified simultaneously
    - ■ 010 001 000

      Clear AC to 0 and subtract DR from AC at the same time
  - » Symbol **DRTAC**(F1 = 100) *Reg **T** Reg*
    - ■ stand for a transfer from DR to AC (*T* = *to*)



Example Computer

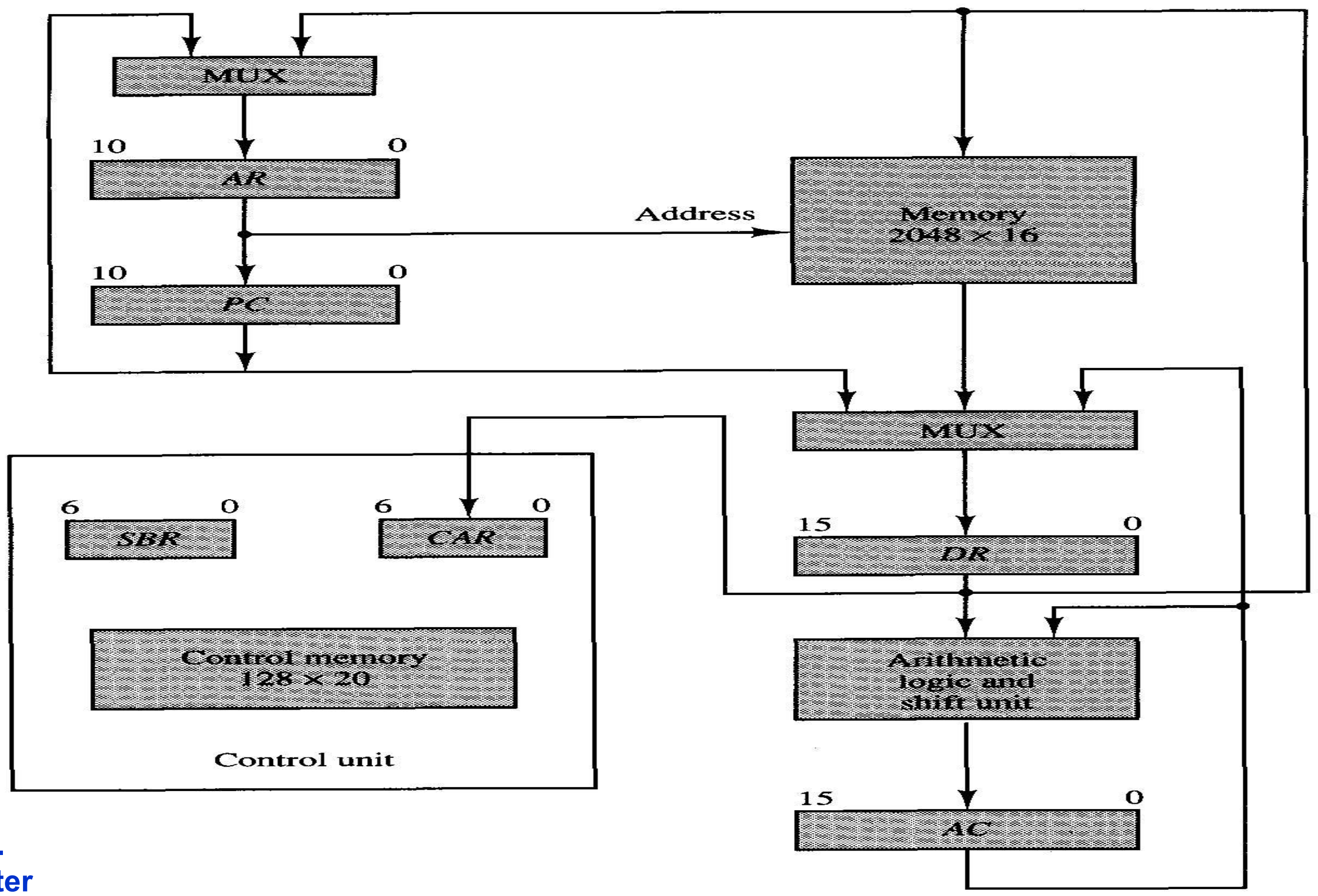


Figure 7-4 Computer hardware configuration.

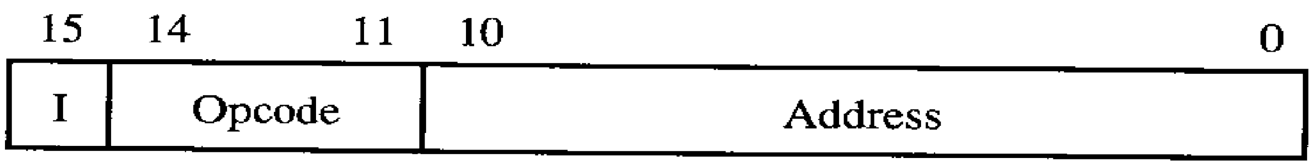
Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Mapping of instruction

- Each **MI** has its own microprogram/routine
- **Mapping**: Derive (fast) the microprogram starting address of every **MI**
- Assume control memory **size**: **128 = 2^7 words**
  - Address is 7 bits
- Derivation methods:
  - Mapping **process: 0 | xxxx | 00**
    - 4 microinstructions for each **MI**
    - If not enough – use some of addresses **1000000-1111111**
  - Mapping **function:** a ROM holds the starting addresses
    - **Input** is the **opcode**
    - **Output** is the **7 bit** control memory **starting address**

Computer Instructions

Figure 7-5 Computer instructions.



(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	AC(15)=1 If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

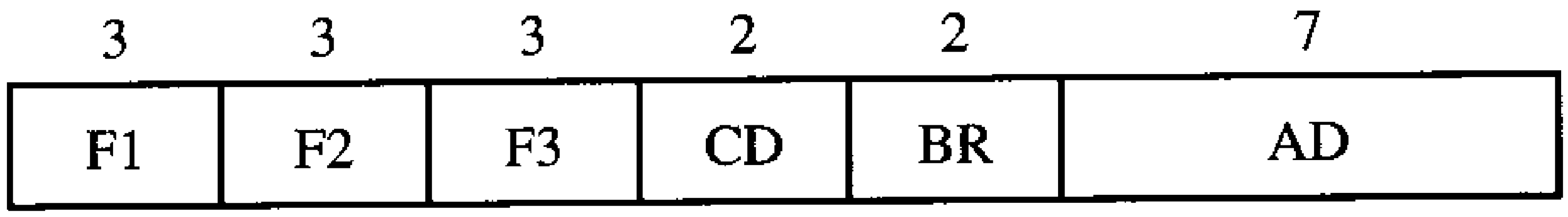
EA is the effective address

(b) Four computer instructions

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Alon Schclar, Tel-Aviv College, 2009

Microinstruction Format



F1, F2, F3: Microoperation fields

CD: Condition for branching

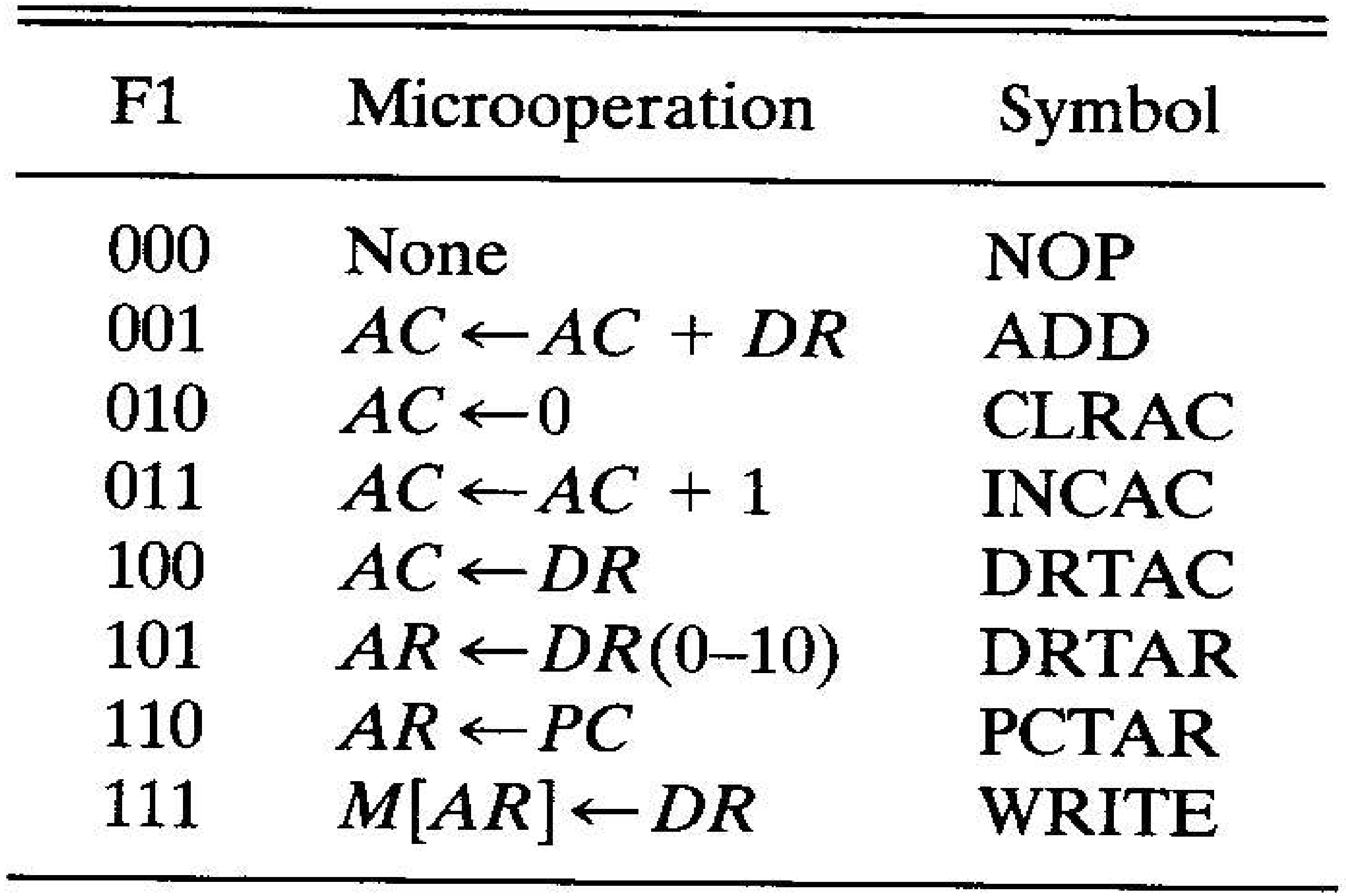
BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

F1 possible functions



F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

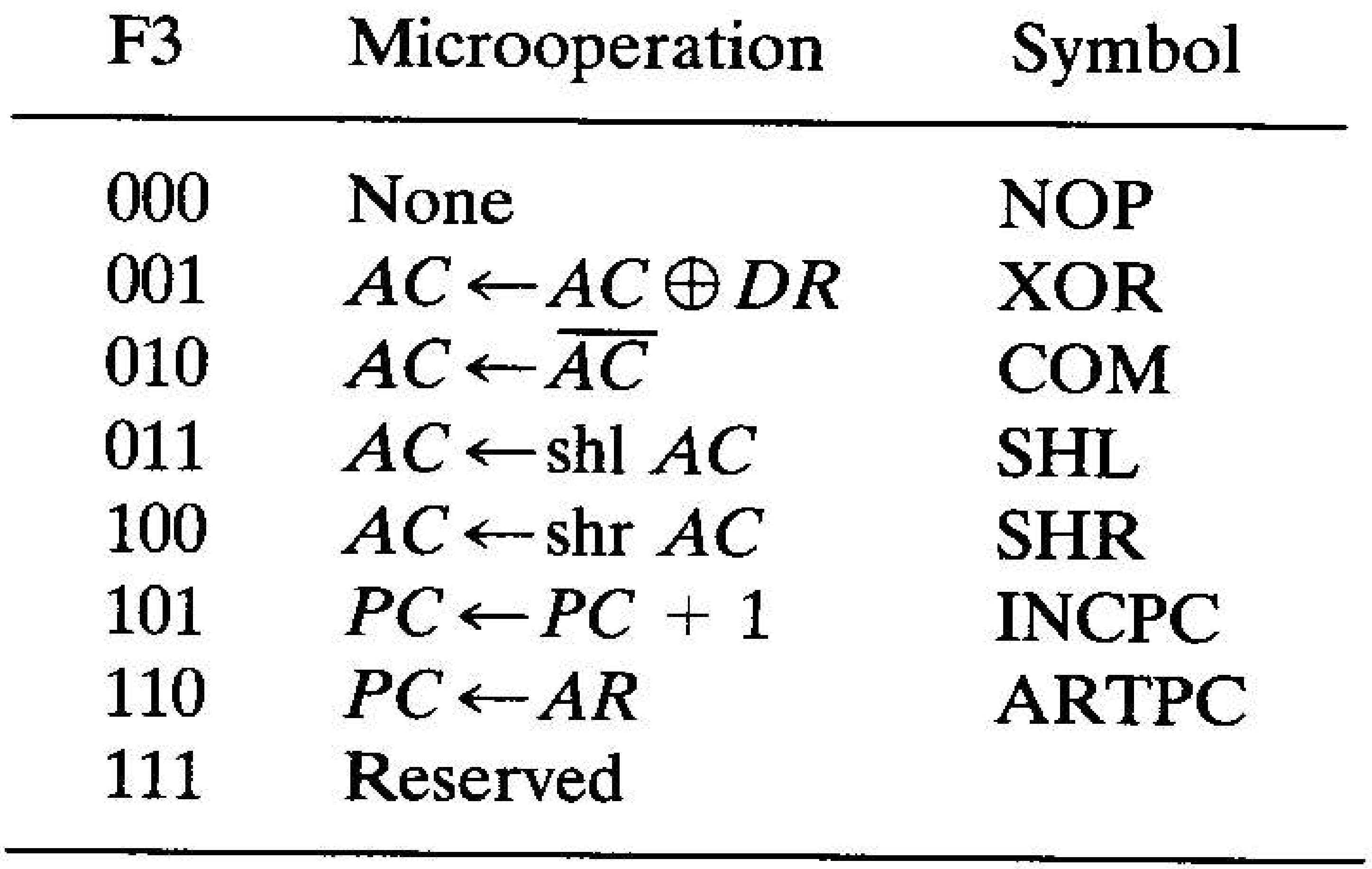
Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

F2 possible functions

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

F3 possible functions



F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Simultaneous microoperations

- Only possible when not contradicting
  - DR $\leftarrow$ M[AR] with F2 = 100
  - PC $\leftarrow$ PC + 1 with F3 = 101
  - F1F2F3 = 000 100 101
- Impossible
  - F1F2F3 = 010 001 000
  - clear AC to 0 and
  - subtract DR from AC at the same time.

- 2 bit Condition Fields : CD
  - » 00 : Unconditional branch, **U** = 1
  - » 01 : Indirect address bit, **I** = DR(15)
  - » 10 : Sign bit of AC, **S** = AC(15)
  - » 11 : Zero value in AC, **Z** = AC = 0
- 2 bit Branch Fields : BR
  - » 00 : **JMP**
    - Condition = 0 : 1 $CAR \leftarrow CAR+1$
    - Condition = 1 : 2 $CAR \leftarrow AD$
  - » 01 : **CALL**
    - Condition = 0 : 1 $CAR \leftarrow CAR+1$
    - Condition = 1 : 2 $CAR \leftarrow AD, SBR \leftarrow CAR+1$
  - » 10 : **RET** 3 $CAR \leftarrow SBR$
  - » 11 : **MAP** 4 $CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Save Return Address

Restore Return Address

- 7 bit Address Fields : AD
  - » 128 word : 128 X 20 bit

◆ Symbolic Microinstruction

① Label Field : Terminated with a colon (:)

② Microoperation Field : one, two, or three symbols, separated by commas

③ CD Field : U, I, S, or Z

④ BR Field : JMP, CALL, RET, or MAP

F1, F2, F3

① Label	② Microoperat	③ CD	④ BR	⑤ AD
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	0

⑤ AD Field

a. Symbolic Address : Label (= Address)

b. Symbol “NEXT” : next address

c. Symbol “RET” or “MAP” : AD field = 0000000

- ORG : Pseudoinstruction(*define the origin, or first address of routine*)

Fetch (Sub)Routine

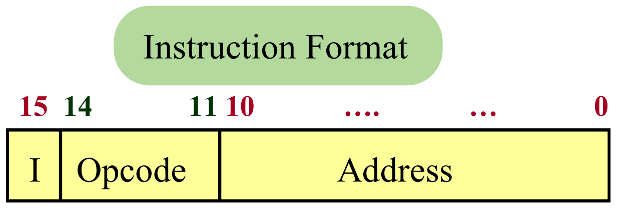
- Memory Map(*128 words*) : ***Tab. 7-2, Tab. 7-3***
  - » Address 0 to 63 : Routines for the 16 instruction *each no more than 4 micro-instructs*
  - » Address 64 to 127 : Any other purpose(*Subroutines : FETCH, INDRCT*)
- Microinstruction for FETCH Subroutine
  - » $AR \leftarrow PC$
  - » $DR \leftarrow M[AR], PC \leftarrow PC+1$
  - » $AR \leftarrow DR(0-10), CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Opcode Fetch

Opcode Decode

Operand Address

Mapping



- Fetch Subroutine : address *64*

Label	Microoperat	CD	BR	AD
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	0

Fetch subroutine

$AR \leftarrow PC$

$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$

$AR \leftarrow DR(0\text{–}10), \quad CAR(2\text{–}5) \leftarrow DR(11\text{–}14), \quad CAR(0,1,6) \leftarrow 0$

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

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

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Symbolic Microprogram : *Tab. 7-2*

- The execution of MAP microinstruction in FETCH subroutine
  - » Branch to address 0**xxxx**00 (*xxxx = 4 bit Opcode*)
    - ADD : 0 **0000** 00 = 0
    - BRANCH : 0 **0001** 00 = 4
    - STORE : 0 **0010** 00 = 8
    - EXCHANGE : 0 **0011** 00 = 12, *(16,20,...,60)*
- Indirect Address : **I** = 1
  - » Indirect Addressing : $AR \leftarrow M[AR]$
    - $M[AR]$->DR->AR
  - » INDRCT subroutine

Label	Microoperat	CD	BR	AD
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	0

$DR \leftarrow M[AR]$
$AR \leftarrow DR$

CAR←SBR

Symbolic microprogram (partial)

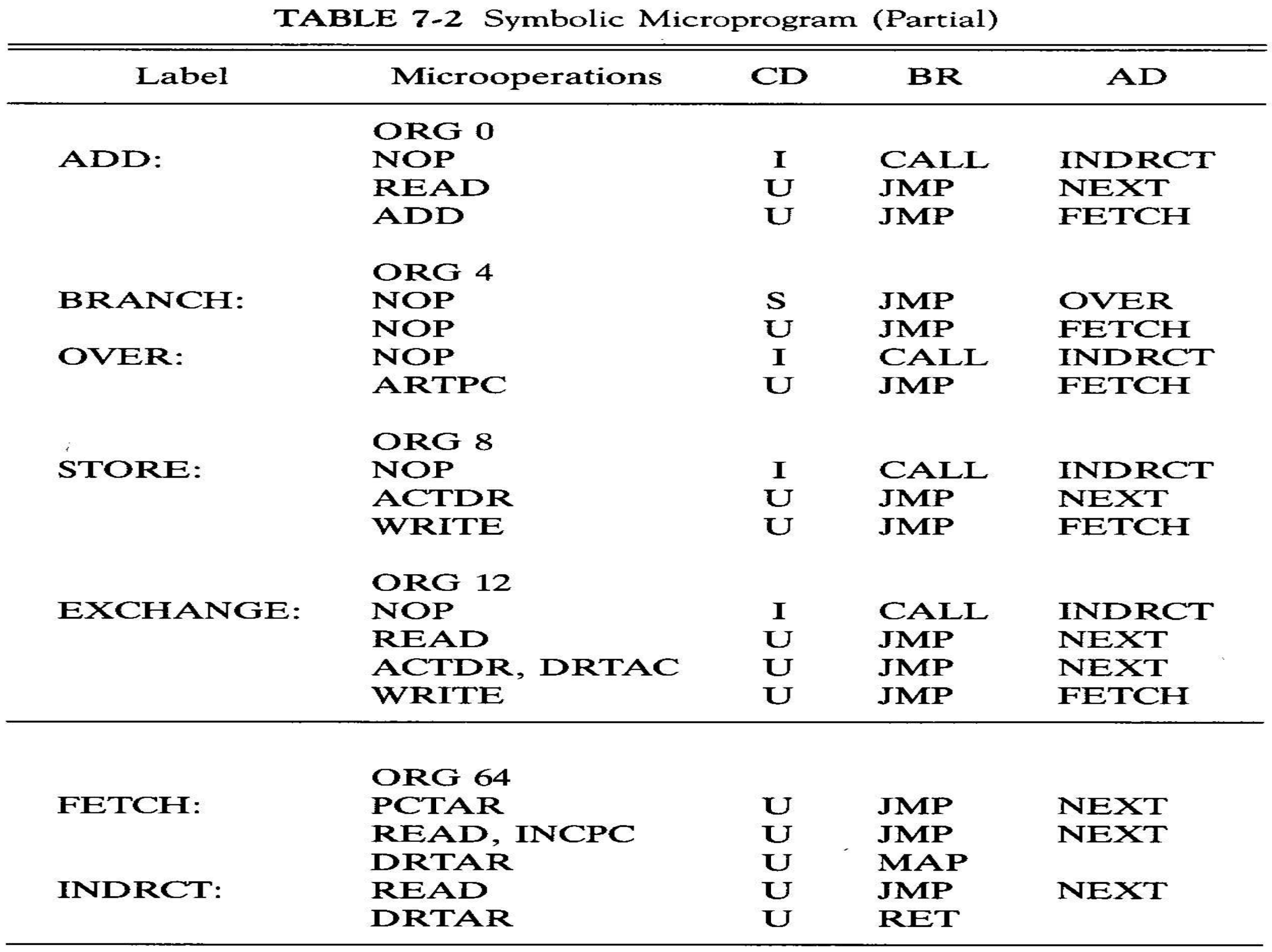


TABLE 7-2 Symbolic Microprogram (Partial)

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

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Binary Microprogram for Control Memory (Partial)

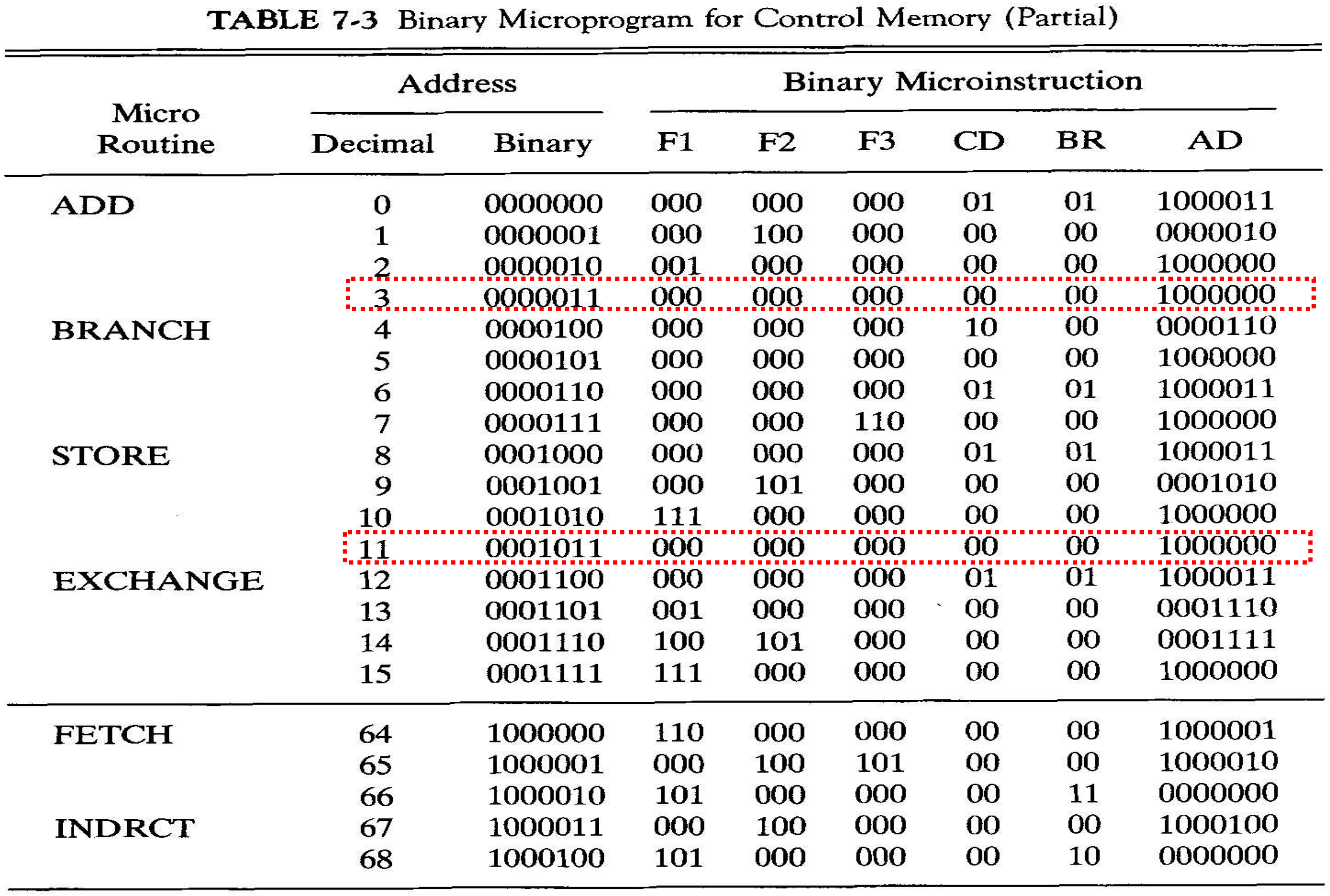


TABLE 7-3 Binary Microprogram for Control Memory (Partial)

Micro Routine	Address		Binary Microinstruction					
	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

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

ADD

- The fetch subroutine reads the instruction into DR
- The MAP process produces 0 **0000** 00
  - **Sequencer continues at control address 0**
- The effective address is calculated at address **0**
  - In case **I=1** the INDRCT subroutine is called
- The execution is done by the microinstructions at addresses 1 and 2.
  1. The first reads the operand from memory into DR.
  2. The second adds the content of DR and AC and then jumps back to the beginning of the fetch routine.

BRANCH

- Branch to effective address if AC < 0.
  - When sign is negative - AC(15) = 1 - status bit S = 1.
- Fetch
- The BRANCH routine
  - if S = 0, no branch occurs
    - the next microinstruction jumps back to the fetch routine without altering the content of PC.
  - If S = 1,
    - the first JMP microinstruction transfers control to location OVER.
    - at this location the INDRCT subroutine is called if **I = 1**.
    - the effective address is transferred from AR to PC
    - the microprogram jumps back to the fetch routine.

STORE

- Fetch
- The INDRCT subroutine is called if **I = 1**
- The content of AC is transferred into DR.
- A memory write stores the content of DR in a location specified by the effective address in AR

EXCHANGE

- Fetch
- The INDRCT subroutine is called if **I = 1**
- Read into DR the operand from in effective address
- Interchange contents of DR and AC
  - third microinstruction.
  - possible when using edge-triggered registers
- DR is stored back in memory.

- Binary Microprogram : ***Tab. 7-3***
  - Symbolic microprogram(***Tab. 7-2***) must be translated to **binary** either by means of an assembler program or by the user
  - Control Memory
    - » Most microprogrammed systems use a ROM for the control memory
      - Cheaper and faster than a RAM
      - Prevent the occasional user from changing the architecture of the system

7-4 Design of Control Unit

- Decoding of Microinstruction Fields : ***Fig. 7-7***
  - F1, F2, and F3 of Microinstruction are decoded with a 3 x 8 decoder

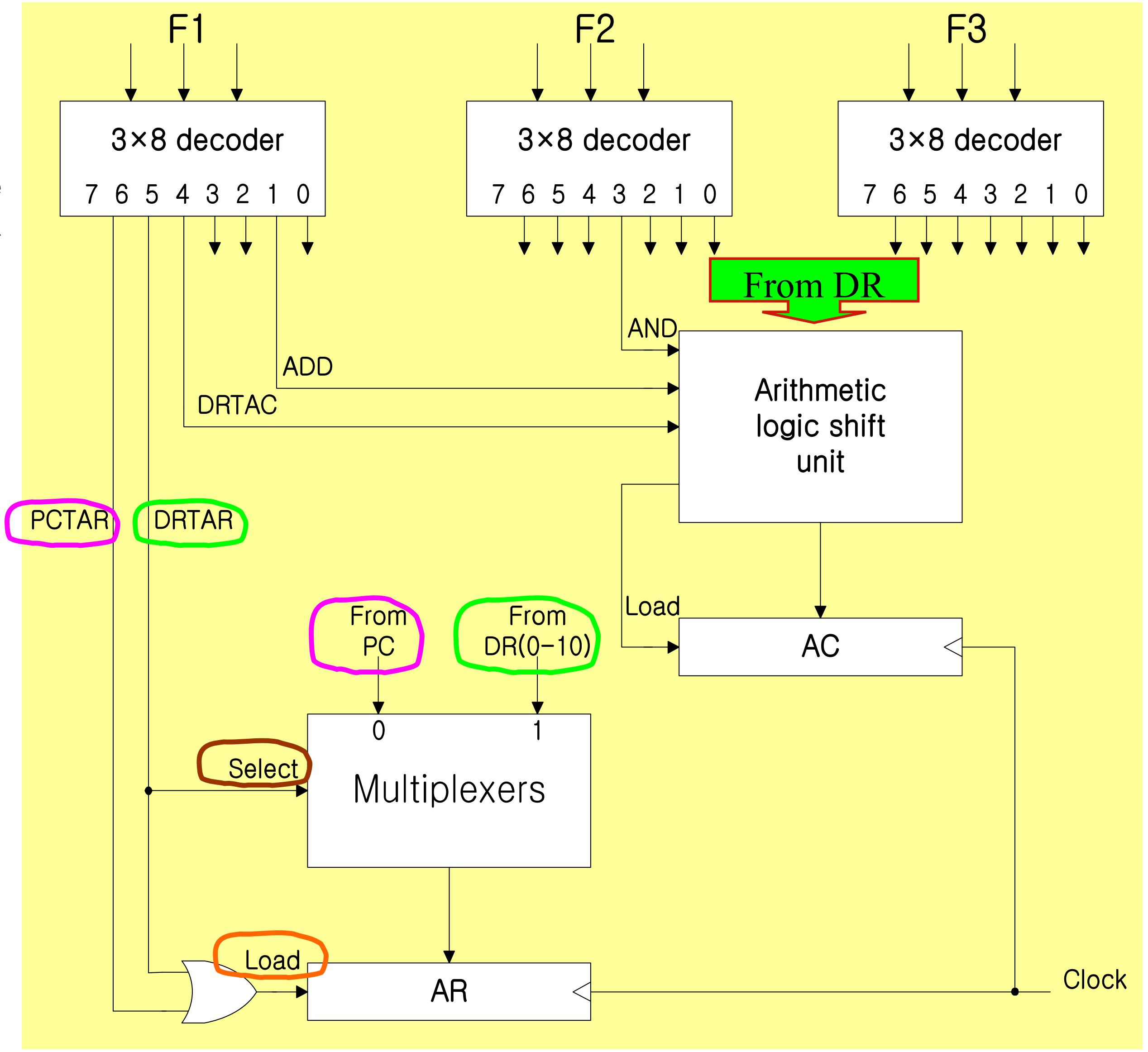
    Simplifies but incurs delays
  - Output of decoder must be connected to the proper circuit to initiate the corresponding microoperation (*as specified in **Tab. 7-1***)

» F1 = 101 (5) : **DRTAR**

F1 = 110 (6) : **PCTAR**

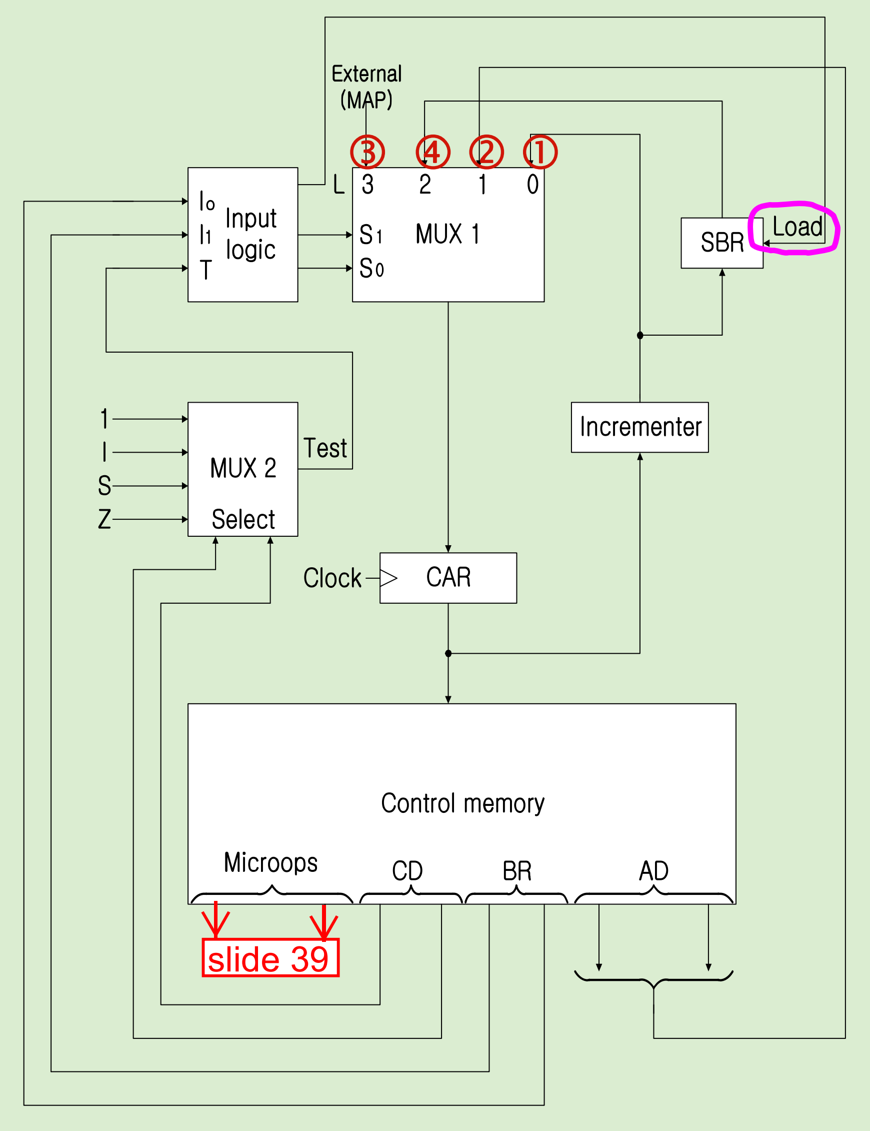
- Output 5 and 6 of decoder F1 are connected to the load input of AR (*two input of OR gate*)
- Multiplexer select the data from DR when output 5 is active
- Multiplexer select the data from AC when output 5 is inactive

- Arithmetic Logic Shift Unit
  - » Control signal of ALU in *hardwired control* : p. 164, ***Fig. 5-19, 20***
  - » Control signal will be now come from the *output of the decoders* associated with the AND, ADD, and DRTAC.



Microprogram Sequencer : *Fig. 7-8*

- Microprogram Sequencer select the next address for control memory
- MUX 1
  - » Select an address source and route to CAR
    - ① CAR + 1
    - ② JMP/CALL
    - ③ Mapping
    - ④ Subroutine Return
  - » JMP & CALL
    - JMP : AD -> MUX 1 -> CAR
    - CALL : AD -> MUX 1 -> CAR
      CAR+1 (return address) -> SBR,
      LOAD(SBR) = 1
- MUX 2
  - » Test a status bit and the result of the test is applied to an input logic circuit
  - » One of 4 Status bit is selected by Condition bit (**CD**)
- Design of Input Logic Circuit
  - » Select one of the source address(**S_0, S_1**) for CAR
  - » Enable the load input(**L**) in SBR



Microprogram sequencer for a control memory

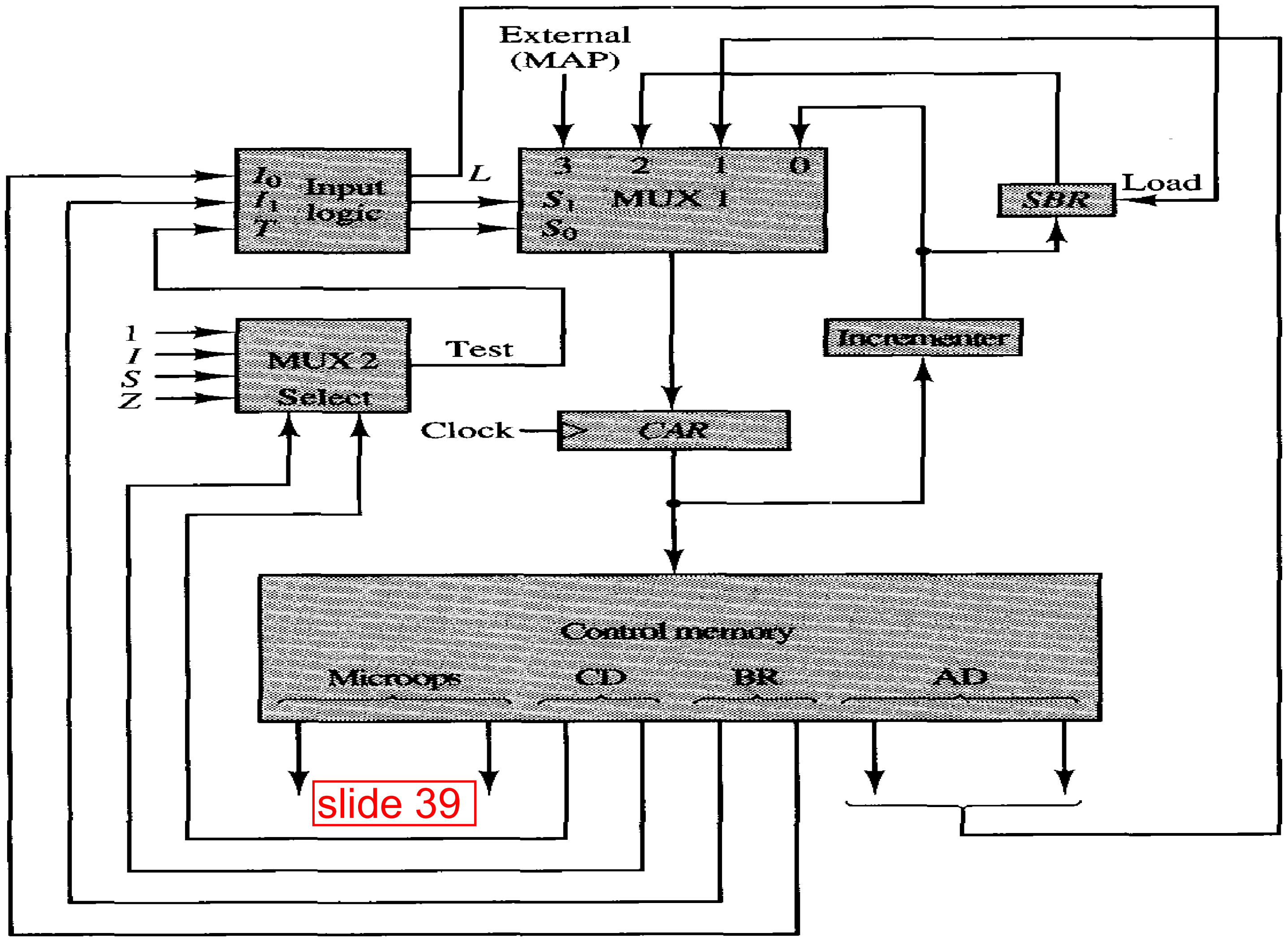


Figure 7-8 Microprogram sequencer for a control memory.

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

Alon Schclar, Tel-Aviv College, 2009

TABLE 7-4 Input Logic Truth Table for Microprogram Sequencer

BR Field		Input			MUX 1		Load SBR	
		I_1	I_0	T	S_1	S_0	L	
0	0	0	0	0	0	0	0	JMP
0	0	0	0	1	0	1	0	
0	1	0	1	0	0	0	0	CALL
0	1	0	1	1	0	1	1	
1	0	1	0	×	1	0	0	RET
1	1	1	1	×	1	1	0	MAP

Taken from: **M. Mano/Computer Design and Architecture 3rd Ed.**

- Input Logic Truth Table : ***Tab. 7-4***
  - » Input :
    - I_0, I_1 from Branch bit (**BR**)
    - T from MUX 2 (**T**)
  - » Output :
    - MUX 1 Select signal (**S_0, S_1**)

      $S1 = I_1I_0' + I_1I_0 = I_1(I_0' + I_0) = I_1$

      $S0 = I_1'I_0'T + I_1'I_0T + I_1I_0$

      $= I_1'T(I_0' + I_0) + I_1I_0$

      $= I_1'T + I_1I_0$
    - SBR Load signal (**L**)

      $L = I_1'I_0T$

BR Field		Input			MUX 1		Load SBR
		I1	I0	T	S1	S0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	x	1	0	0
1	1	1	1	x	1	1	0