Computer System Architecture

(7. 마이크로 프로그램된 제어)

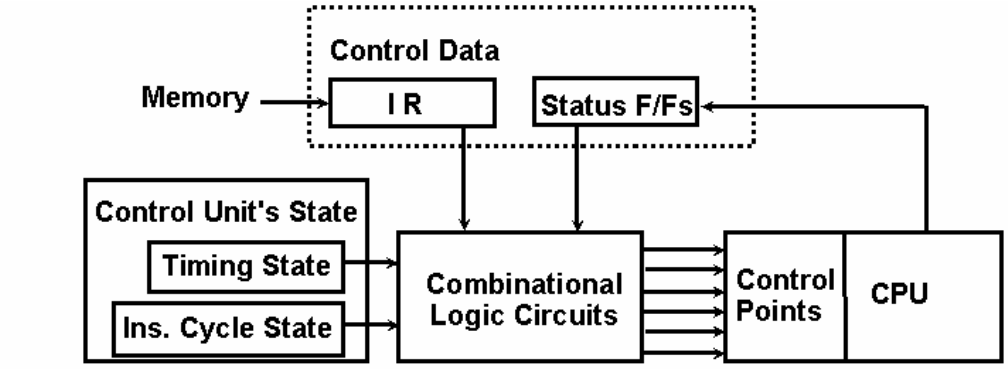
마이크로 프로그램된 제어

- 제어 메모리
- 주소 시퀀싱
- 마이크로 프로그램의 예
- 제어 장치의 설계
- 마이크로 명령어 형식
- **Nanostorage and Nanoprogram**

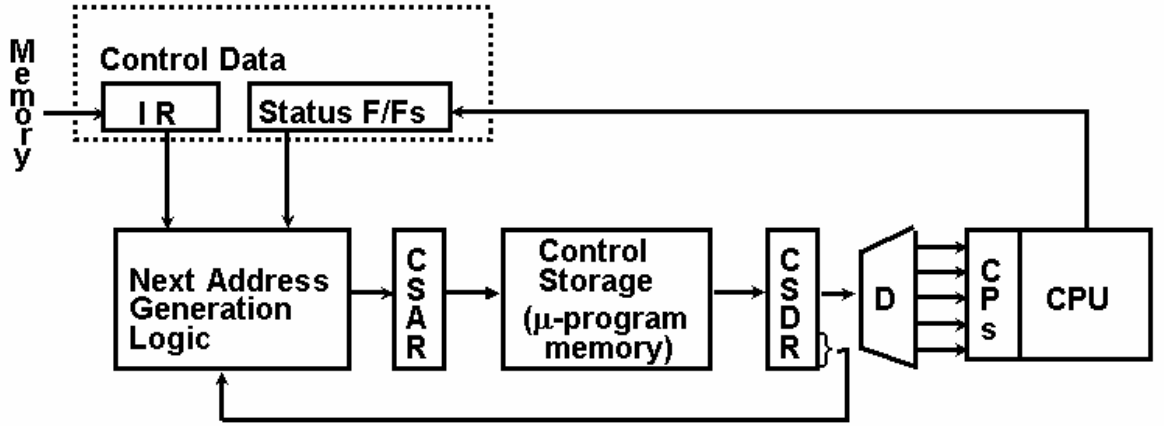
제어 방식의 비교

- 제어 방식

Combinational Logic Circuits (Hard-wired)



Microprogram



전 문 용 어

Microprogram
- 메모리 안에 저장된 프로그램은 모든 제어 신호들이 바르게 실행되도록 요청한다.
- 마이크로 명령어가 존재한다.

Microinstruction – 마이크로 명령어
- 제어 명령과 시퀀서 명령을 가지고 있다.
  제어 명령 - 모든 명령어들이 한클럭에 수행된다.
  시퀀서 명령 - 다음 마이크로 연산 주소에의해 수행되는 정보를 가지고 있다.
- 마이크로 명령어에 의해 표현된다.

Control Memory(Control Storage: CS) – 제어 메모리
- 제어 명령어들이 마이크로 프로그램되어 저장되어 있다. (주로 **ROM**)

Writeable Control Memory(Writeable Control Storage:WCS)
- **CS** 의 내용은 수정 가능
  -> 마이크로 프로그램의 수정가능
  -> **Instruction set** 수정 가능

동적인 마이크로 프로그래밍
- **Computer system** 의 제어장치는 **WCS**의 마이크로 프로그램으로 구현이 된다.
- **Microprogram**은 시스템 프로그래머나 사용자에 의해 수정 가능

용 어 설 명

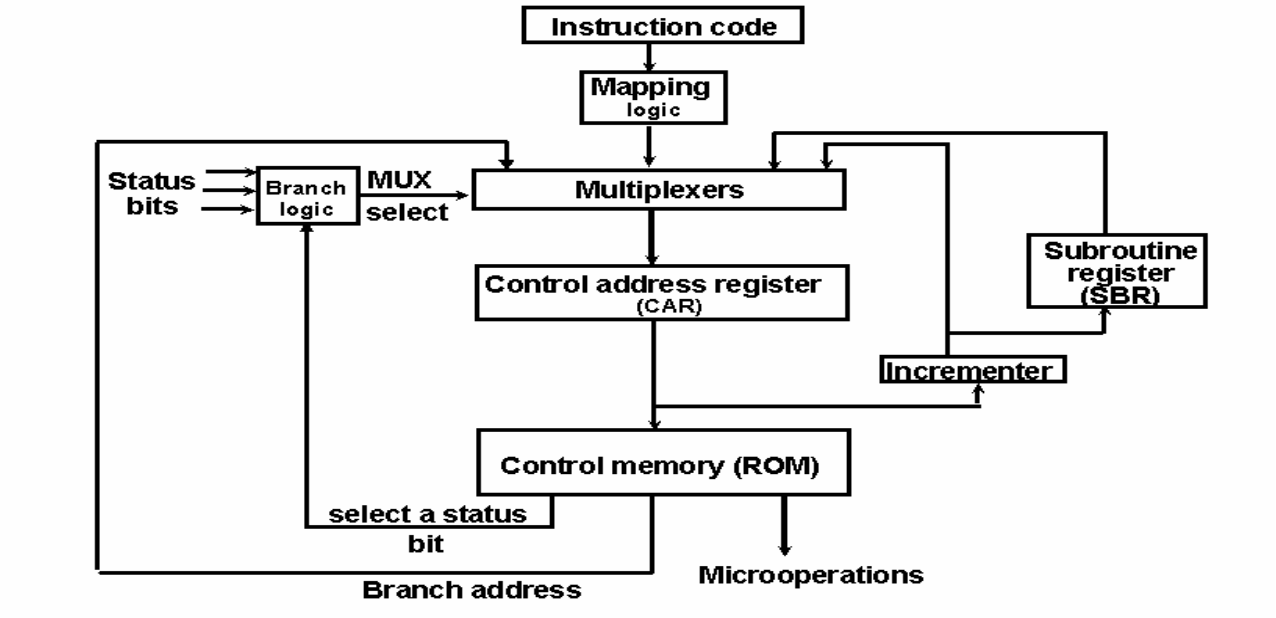
Sequencer (Microprogram Sequencer) - 순차기

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

마이크로 프로그램 제어 장치 에서다음 클럭의 지시하는 주소의 내용을 실행한다.

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

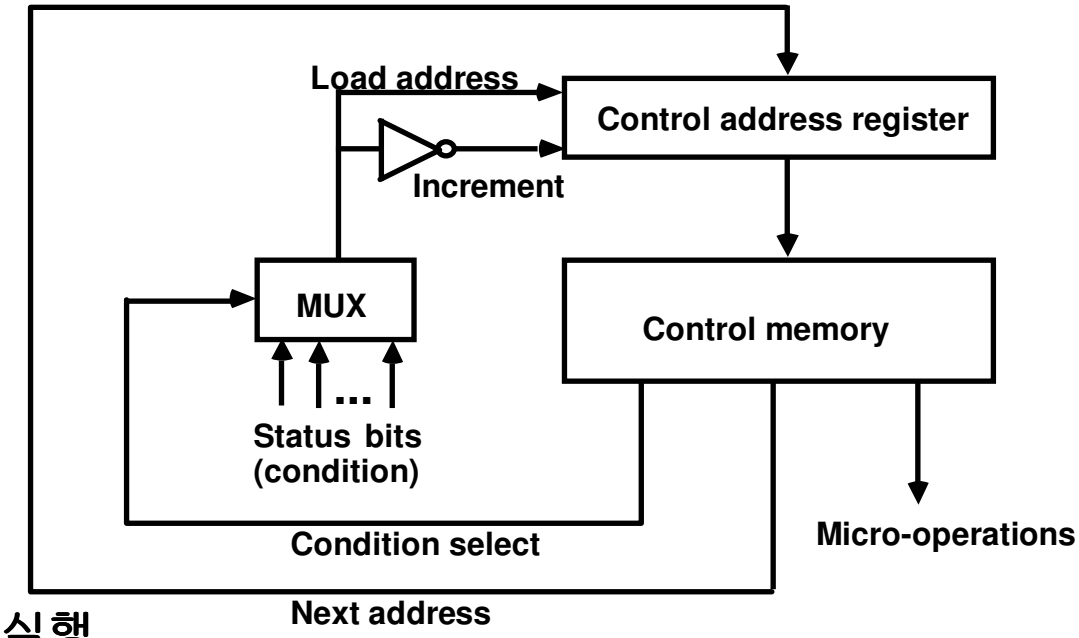
MICROINSTRUCTION SEQUENCING



제어 저장소에 **Sequencing Capabilities** 가 요구된다.

- **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**

조 건 분 기



조건 분기

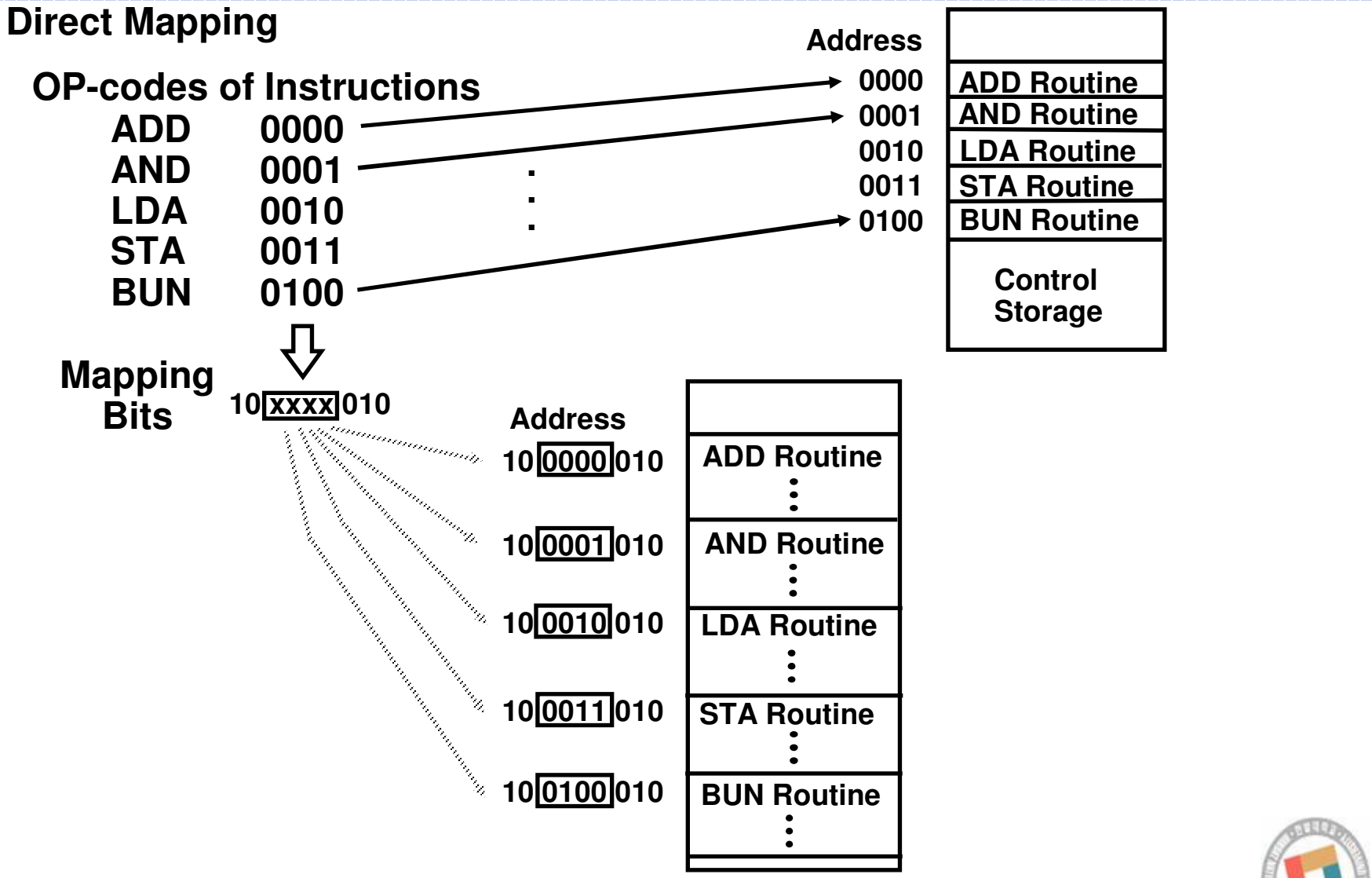
- 조건이 참이라면, 분기 실행
  (address from the next address field of the current microinstruction)
- 거짓이라면 실행 안함.

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

무조건 분기

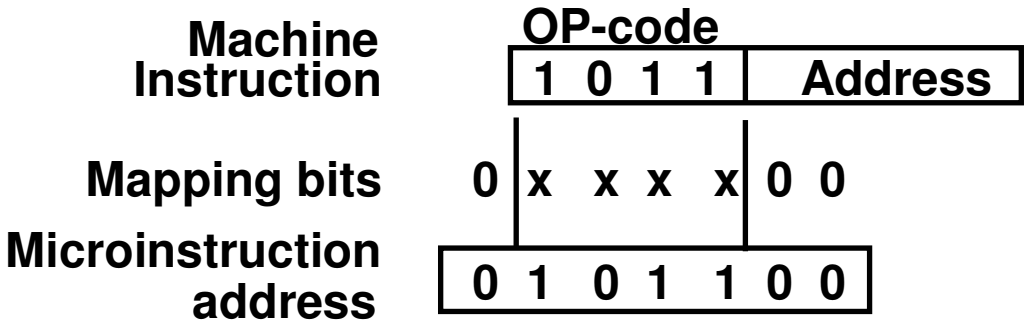
Fixing the value of one status bit at the input of the multiplexer to 1

INSTRUCTION의 맵핑

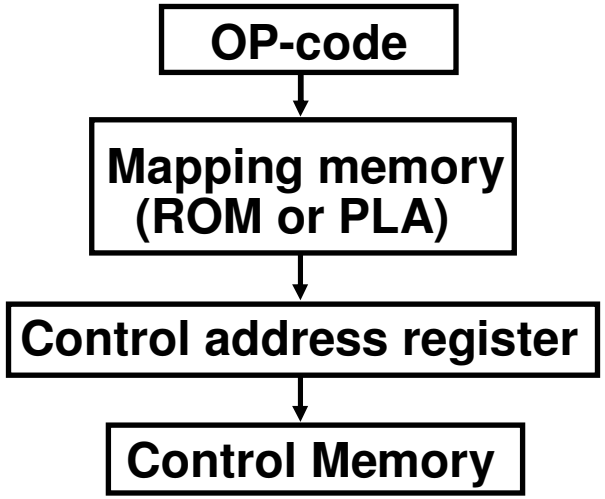


MAPPING OF INSTRUCTIONS TO MICROROUTINES

마이크로 프로그램 실행은 시작되는 마이크로 명령어의 **OP-code** 로부터 맵핑함으로 이루어 진다.

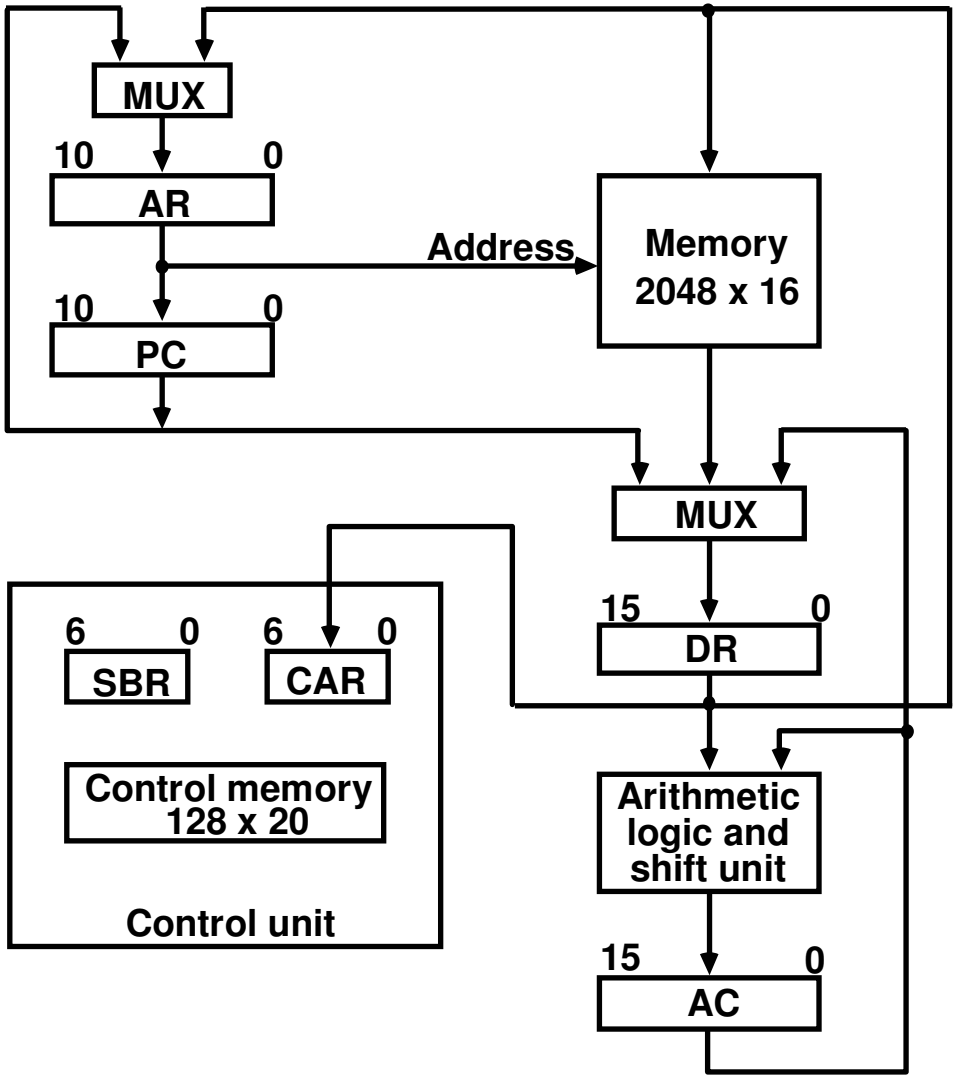


Mapping function implemented by ROM or PLA



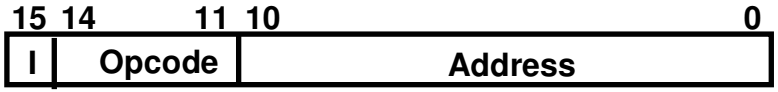
MICROPROGRAM EXAMPLE

Computer Configuration



MACHINE INSTRUCTION FORMAT

Machine instruction format

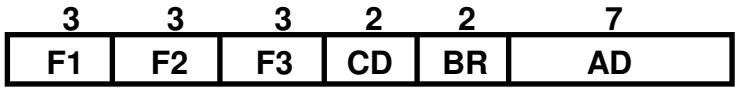


Sample machine instructions

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

MICROINSTRUCTION FIELD DESCRIPTIONS - F1,F2,F3

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

MICROINSTRUCTION FIELD DESCRIPTIONS - CD, BR

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

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

SYMBOLIC MICROINSTRUCTIONS

- 마이크로 명령어에서는 심볼로써 어셈블러 가 쓰여 진다.
- 심볼화된 마이크로 프로그램은 마이크로 프로그램 어셈블러에 의해 이진으로 표현 된다.

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

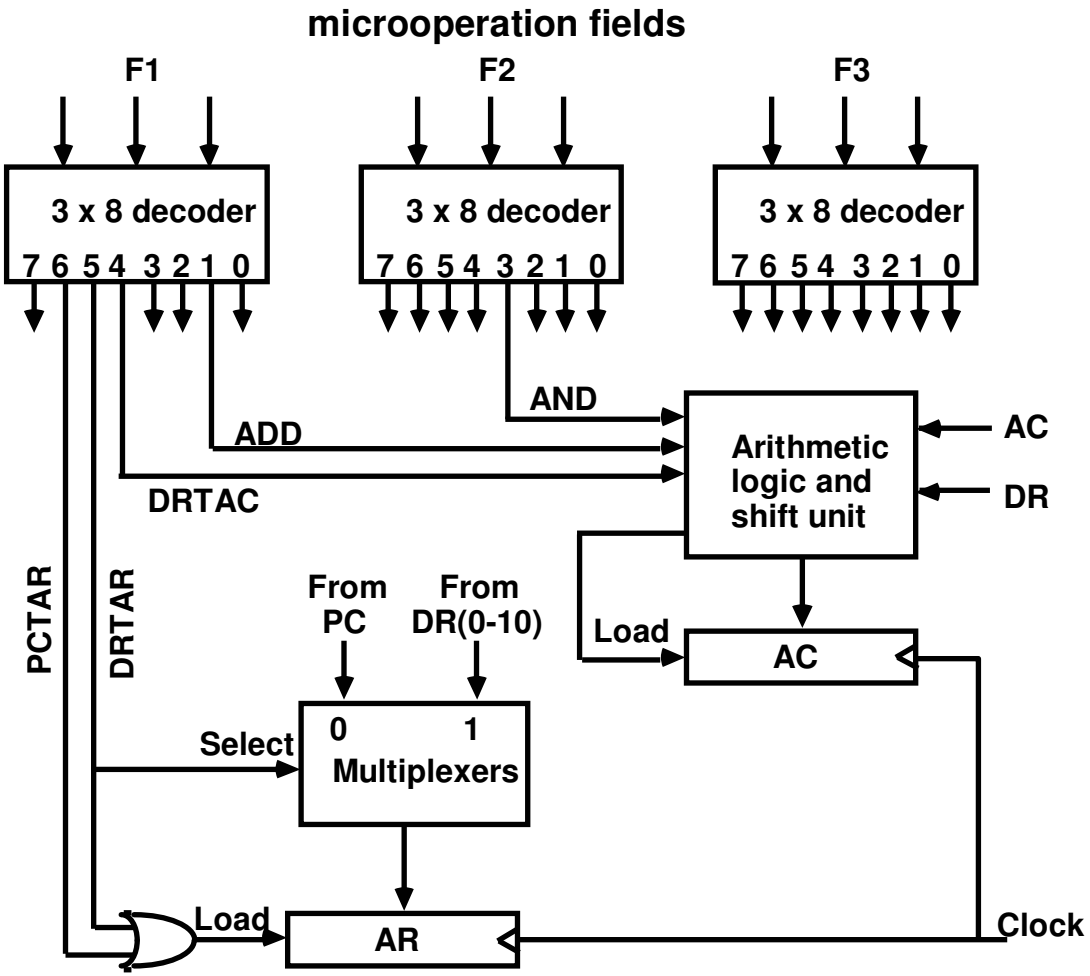
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	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

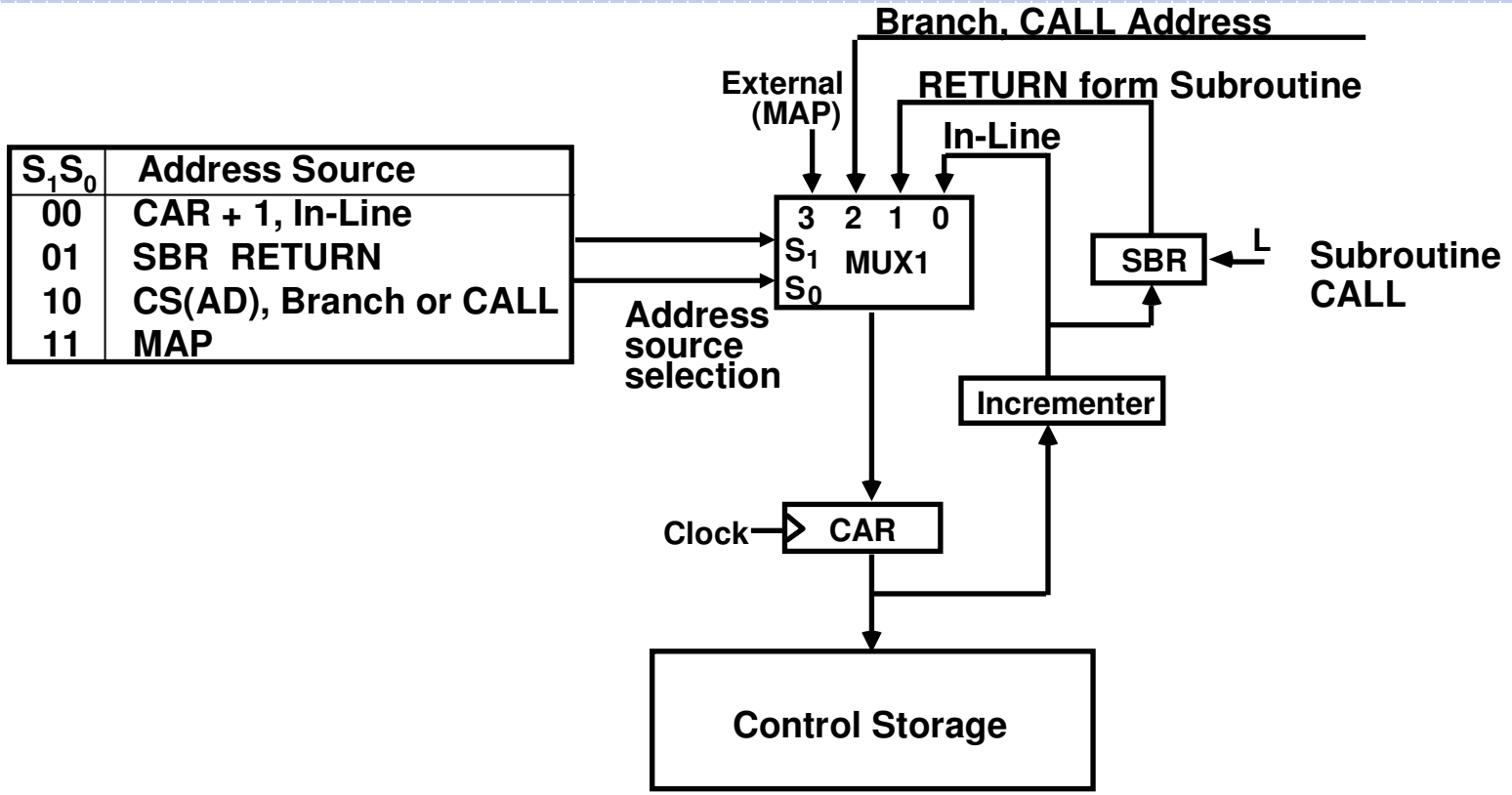
	Address		Binary Microinstruction					
Micro Routine	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	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

DESIGN OF CONTROL UNIT
- DECODING ALU CONTROL INFORMATION -



MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -

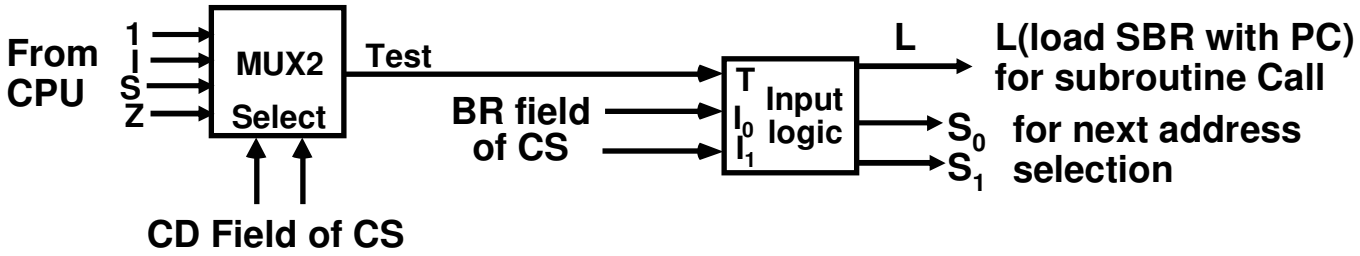


S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP

MUX-1 selects an address from one of four sources and routes it into a CAR

- In-Line Sequencing → CAR + 1
- Branch, Subroutine Call → CS(AD)
- Return from Subroutine → Output of SBR
- New Machine instruction → MAP

MICROPROGRAM SEQUENCER
- CONDITION AND BRANCH CONTROL -

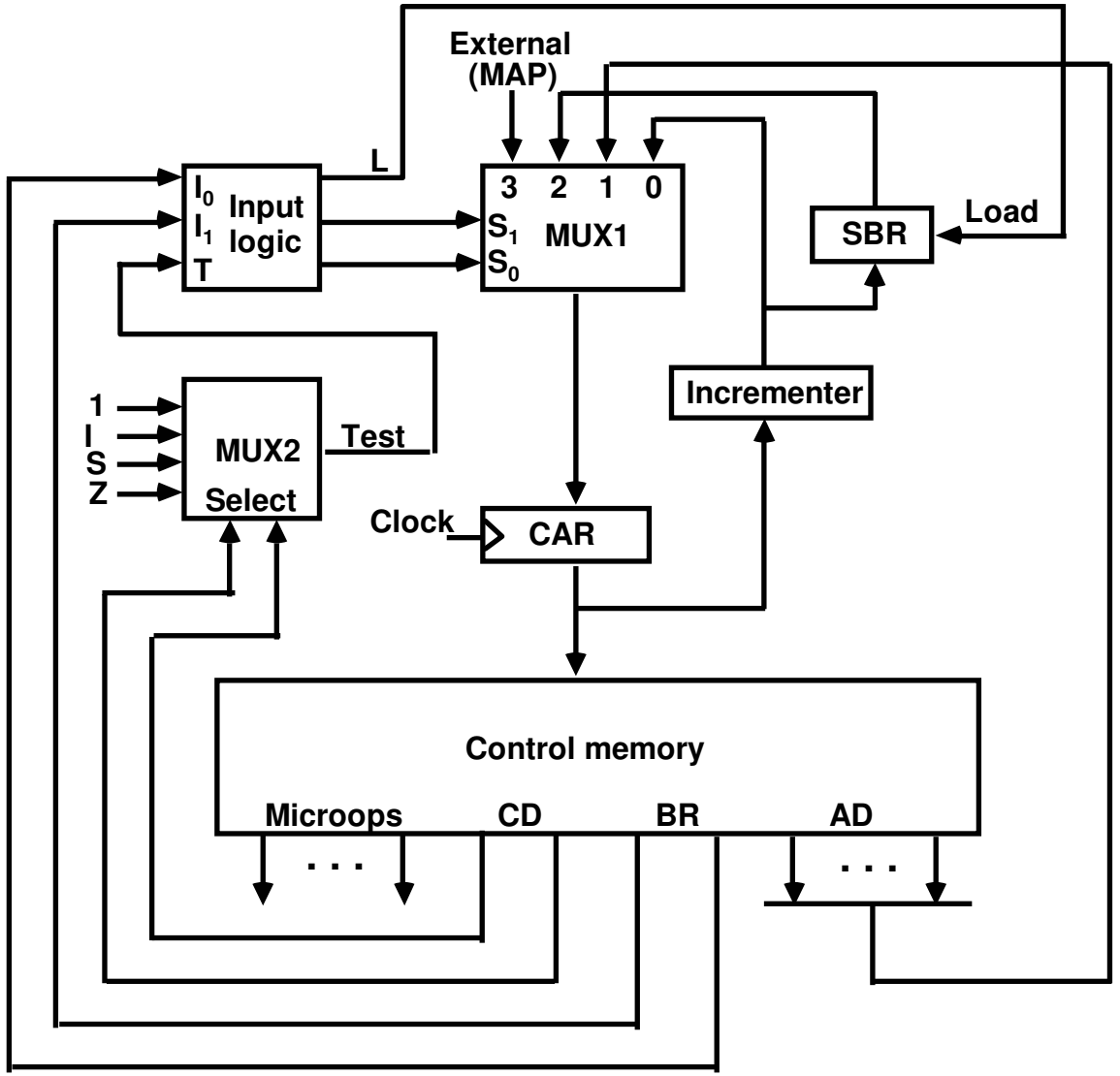


Input Logic

I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$

MICROPROGRAM SEQUENCER



MICROINSTRUCTION FORMAT

Microinstruction이 담고 있는 정보들

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

시스템에서 쓰여질때 정보가 유용해 진다.

이러한 정보들은 다음의 목적을 위해 구성되어질 필요가 있다.

- **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

각각의 비트는 직접적으로 **micro-operation** 이나 **control point** 제어한다.

Horizontal 는 긴 **microinstruction word**를 의미한다.

장 점 : 병렬로 많은 **component**들을 제어 할수 있다.

--> **Advantage of efficient hardware utilization**

단 점 : **Control word bit**들은 모든비트를 사용하지 못한다.

--> **CS becomes large --> Costly**

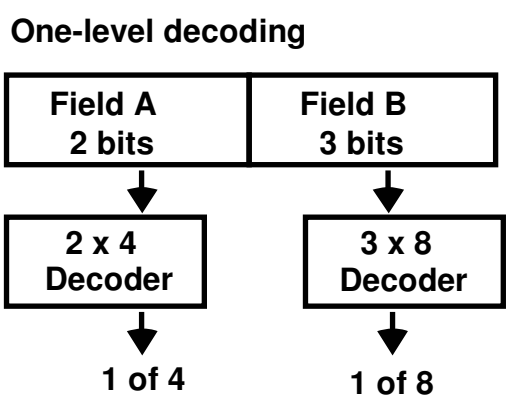
Vertical Microinstructions

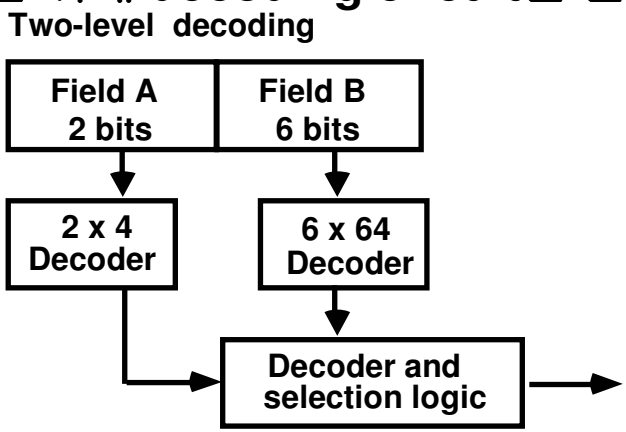
Microinstruction format이 **horizontal**이 아니다.

Vertical 는 짧은 **microinstruction word** 를 의미한다.

Encoded Microinstruction fields

--> 한단계 또는 두단계 **decoding**을 위해 **decoding circuit**를 필요로 한다.





NANOSTORAGE AND NANOINSTRUCTION

vertical microprogram storage organization의 decoder circuit들은
ROM에의해 대체될수 있다.

=> 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

- Microinstruction field들을 해석한다.

그래서, converts a vertical microinstruction format을
horizontal nanoinstruction format으로 바꾼다.

보통,

Nanoprogram이 짧은 word의 긴 nanoinstructions을 포함 할때
Microprogram은 긴 word의 짧은 microinstruction들로 구성된다.

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)