컴퓨터 구조

4장 제어 유니트2

안형태

anten@kumoh.ac.kr

디지털관 139호

제어 유니트

3. 마이크로프로그래밍

마이크로프로그래밍

❑ 인출 사이클의 마이크로명령어 루틴

ORG 0						
FETCH:	**PCTAR**	**NONE**	**U**	**JMP**	**NEXT;**	**MAR← PC, 다음 마이크로명령어 실행**
	READ	**INCPC**	**U**	**JMP**	**NEXT;**	**MBR ← M[MAR], PC←PC+1, 다음 마이크로명령어 실행**
	BRTIR	**NONE**	**U**	**MAP**		**IR ← MBR, 해당 실행 사이클 루틴으로 분기**

- **2진 비트 패턴**
  - 주소: 각 마이크로명령어가 저장될 제어 기억장치 주소, μ-ops: 두 개의 마이크로-연산들, CD: 조건 필드, BR: 분기 필드, ADF: 주소 필드

주소	μ-ops	CD	BR	ADF
0000000	001 000	00	00	0000001
0000001	100 001	00	00	0000010
0000010	110 000	00	11	0000000

마이크로프로그래밍

간접 사이클의 마이크로명령어 루틴

ORG 4						
INDRT:	**IRTAR**	**NONE**	**U**	**JMP**	**NEXT;**	**MAR← IR(*addr*), 다음 마이크로명령어 실행**
	READ	**NONE**	**U**	**JMP**	**NEXT;**	**MBR ← M[MAR], 다음 마이크로명령어 실행**
	BRTIR	**NONE**	**U**	**RET**		**IR(*addr*) ← MBR, 실행 사이클 루틴으로 복귀**

- **2진 비트 패턴**

주소	μ-ops	CD	BR	ADF
0000100	010 000	00	00	0000101
0000101	100 000	00	00	0000110
0000110	110 000	00	10	0000000

마이크로프로그래밍

실행 사이클 루틴

- 사상 방식을 이용하여 각 연산 코드에 대한 실행 사이클 루틴의 시작 주소를 결정
  - 각 명령어 실행을 위한 루틴을 작성
- **[예]** 각 연산 코드를 사상함수(1XXXX00)에 대한 사상 결과

명령어	연산 코드	루틴의 시작 주소
NOP	0000	$1000000 = 64_{10}$
LOAD(I)	0001	$1000100 = 68_{10}$
STORE(I)	0010	$1001000 = 72_{10}$
ADD	0011	$1001100 = 76_{10}$
SUB	0100	$1010000 = 80_{10}$
JUMP	0101	$1010100 = 84_{10}$

마이크로프로그래밍

실행 사이클 루틴

- **NOP** 명령어 – 실행사이클의 마이크로서브루틴

ORG 64						
NOP:	**NONE**	**INCPC**	**U**	**JMP**	**FETCH;**	**PC ← PC+1**

- 2진 비트 패턴

Addr	*μ*-OP1	*μ*-OP2	CD	BR	ADF
1000000	000	001	00	00	0000000

마이크로프로그래밍

실행 사이클 루틴

- **LOAD(I)** 명령어 – 실행사이클의 마이크로서브루틴

	ORG 68					
LOAD:	**NONE**	**NONE**	**I**	**CALL**	**INDRT;**	**I=1 이면, 간접 사이클 루틴 호출**
	IRTAR	**NONE**	**U**	**JMP**	**NEXT;**	**MAR ← IR(addr)**
	READ	**NONE**	**U**	**JMP**	**NEXT;**	**MBR ← M[MAR]**
	BRTAC	**NONE**	**U**	**JMP**	**FETCH;**	**AC ← MBR**

- 2진 비트 패턴

Addr	μ-OP1	μ-OP2	CD	BR	ADF
1000100	000	000	01	01	0000100
1000101	010	000	00	00	1000110
1000110	100	000	00	00	1000111
1000111	101	000	00	00	0000000

마이크로프로그래밍

실행 사이클 루틴

- **STORE(I)** 명령어 – 실행사이클의 마이크로서브루틴

	ORG 72					
STORE:	**NONE**	**NONE**	**I**	**CALL**	**INDRT;**	**I=1 이면, 간접 사이클 루틴 호출**
	IRTAR	**NONE**	**U**	**JMP**	**NEXT;**	**MAR ← IR(addr)**
	NONE	**ACTBR**	**U**	**JMP**	**NEXT;**	**MBR ← AC**
	WRITE	**NONE**	**U**	**JMP**	**FETCH;**	**M[MAR] ← MBR**

- 2진 비트 패턴

Addr	μ-OP1	μ-OP2	CD	BR	ADF
1001000	000	000	01	01	0000100
1001001	010	000	00	00	1001010
1001010	000	010	00	00	1001011
1001011	111	000	00	00	0000000

마이크로프로그래밍

실행 사이클 루틴

- **ADD** 명령어 – 실행사이클의 마이크로서브루틴

	ORG 76					
ADD:	**IRTAR**	**NONE**	**U**	**JMP**	**NEXT;**	**MAR ← IR(addr)**
	READ	**NONE**	**U**	**JMP**	**NEXT;**	**MBR ← M[MAR]**
	ADD	**NONE**	**U**	**JMP**	**FETCH;**	**AC ← AC + MBR**

- 2진 비트 패턴

Addr	μ-OP1	μ-OP2	CD	BR	ADF
1001100	010	000	00	00	1001101
1001101	100	000	00	00	1001110
1001110	011	000	00	00	0000000

제어 유니트

4. 마이크로프로그램의 순서제어

마이크로프로그램의 순서제어

- 제어 유니트가 마이크로명령어의 실행을 제어
  - 제어 기억장치에 저장된 해당 마이크로명령어들을 순서대로 인출
    - 인출된 마이크로명령어의 연산 필드에 있는 비트들을 출력 → 해당 비트들 자체가 제어신호들이 됨

- **순서제어(sequencing):** 제어 유니트에서 다음에 실행될 마이크로명령어의 주소를 결정
  - CAR는 순서제어의 핵심 역할을 수행하며, 초기값 = 0
    - 인출 사이클 루틴의 첫 번째 마이크로명령어의 주소

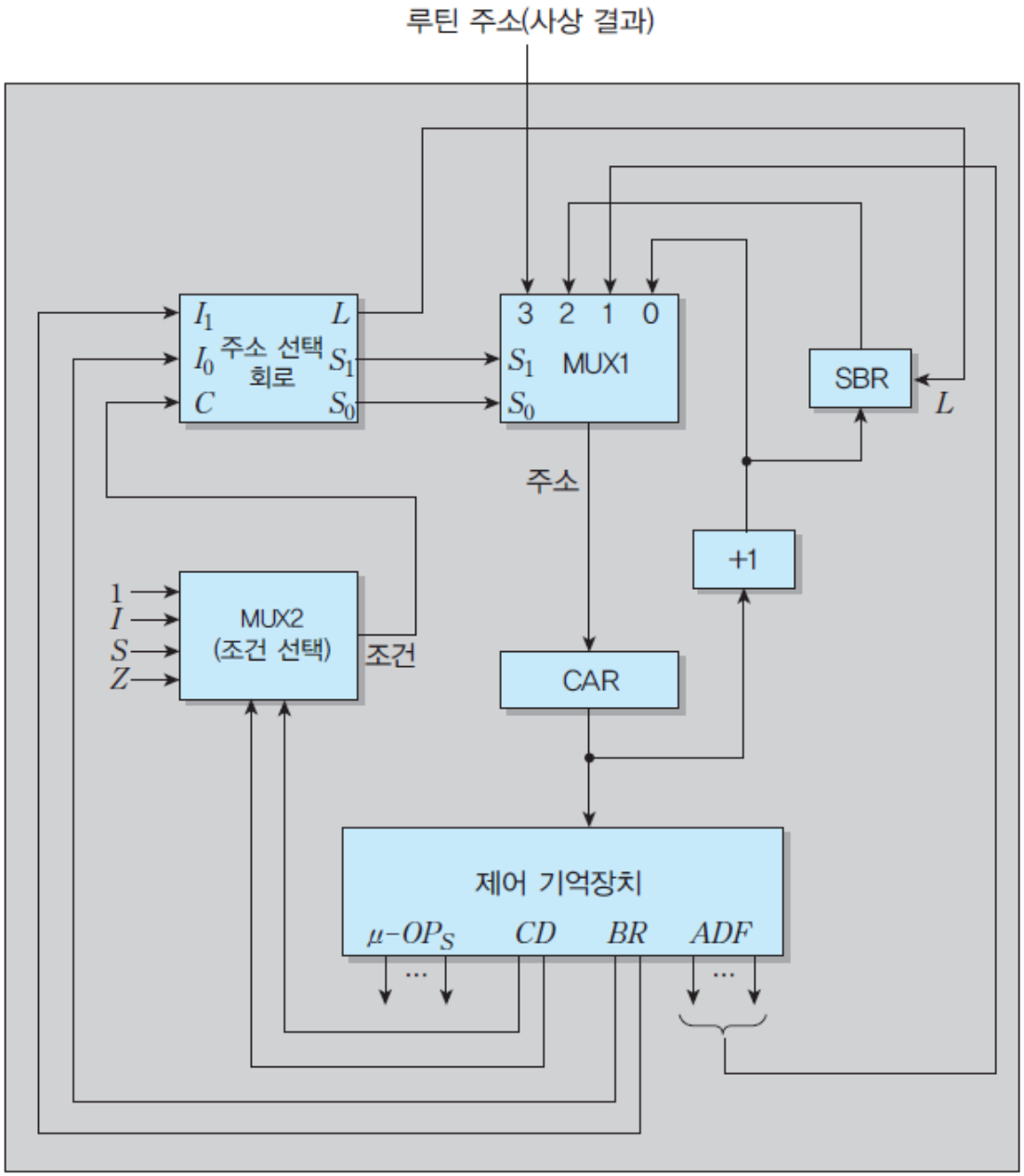
마이크로프로그램의 순서제어

- 멀티플렉서(multiplexer, MUX)
  - 선택선들의 값에 따라서 입력들 중의 하나를 출력으로 보내주는 조합 회로

- **MUX1**에서는 주소 선택 회로의 선택 신호에 따라서 다음에 실행할 마이크로명령어의 주소를 4개의 입력 중 하나로 결정
  - CAR+1, ADF, RET, MAP
  - 결정된 마이크로명령어의 주소는 CAR로 적재됨

- **MUX2**에서는 분기 조건으로 사용될 조건 플래그를 선택하여 주소 선택 회로로 그 값을 전송

순서제어 회로가 포함된 제어 유니트의 구성도

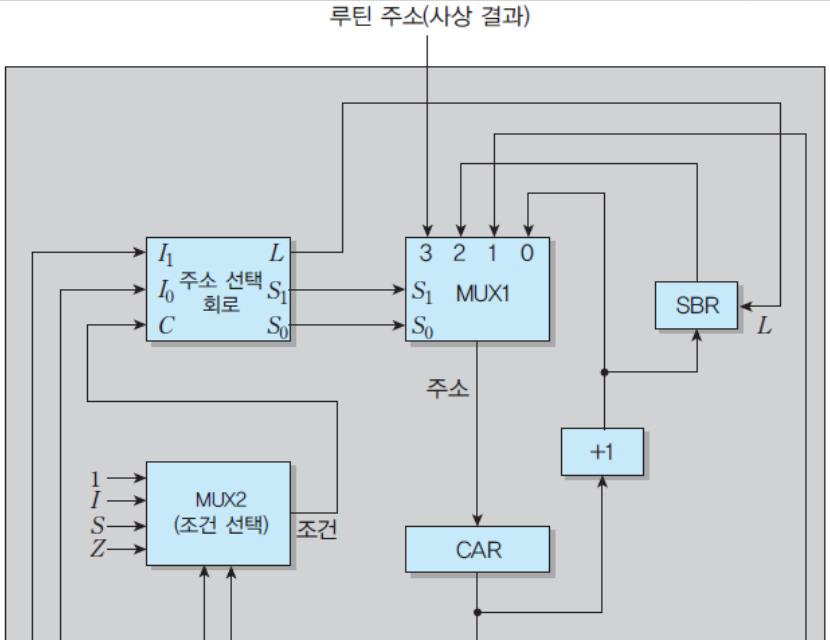


주소 선택 방법

❑ 주소 선택 방법

- ***BR* = *00*** (JUMP) 혹은 ***01*** (CALL)일 때,
  - ***C* = *0***, 다음 위치의 마이크로명령어 선택
  - ***C* = *1***, 주소 필드(ADF)가 지정하는 위치로 점프(jump) 혹은 호출(call) (단, 호출 시에는 CAR 내용을 SBR에 저장)
- ***BR* = *10*** (RET)일 때는 SBR 내용을 CAR로 적재 → 복귀
- ***BR* = *11*** (MAP)일 때는 사상 결과를 CAR에 적재

주소 선택 회로의 입력 및 출력 신호들



BR		조건	MUX1 선택		SBR	CAR로 적재될 MUX1의 입력	설명
I_1	I_0	C	S_1	S_0	L		
0	0	0	0	0	0	0	CAR ← CAR + 1
0	0	1	0	1	0	1	CAR ← ADF ⟨Jump⟩
0	1	0	0	0	0	0	CAR ← CAR + 1
0	1	1	0	1	1	1	SBR ← CAR + 1, CAR ← ADF
1	0	x	1	0	0	2	CAR ← SBR ⟨Return⟩
1	1	x	1	1	0	3	CAR ← 1XXXX00 ⟨Mapping⟩

제어 신호의 생성

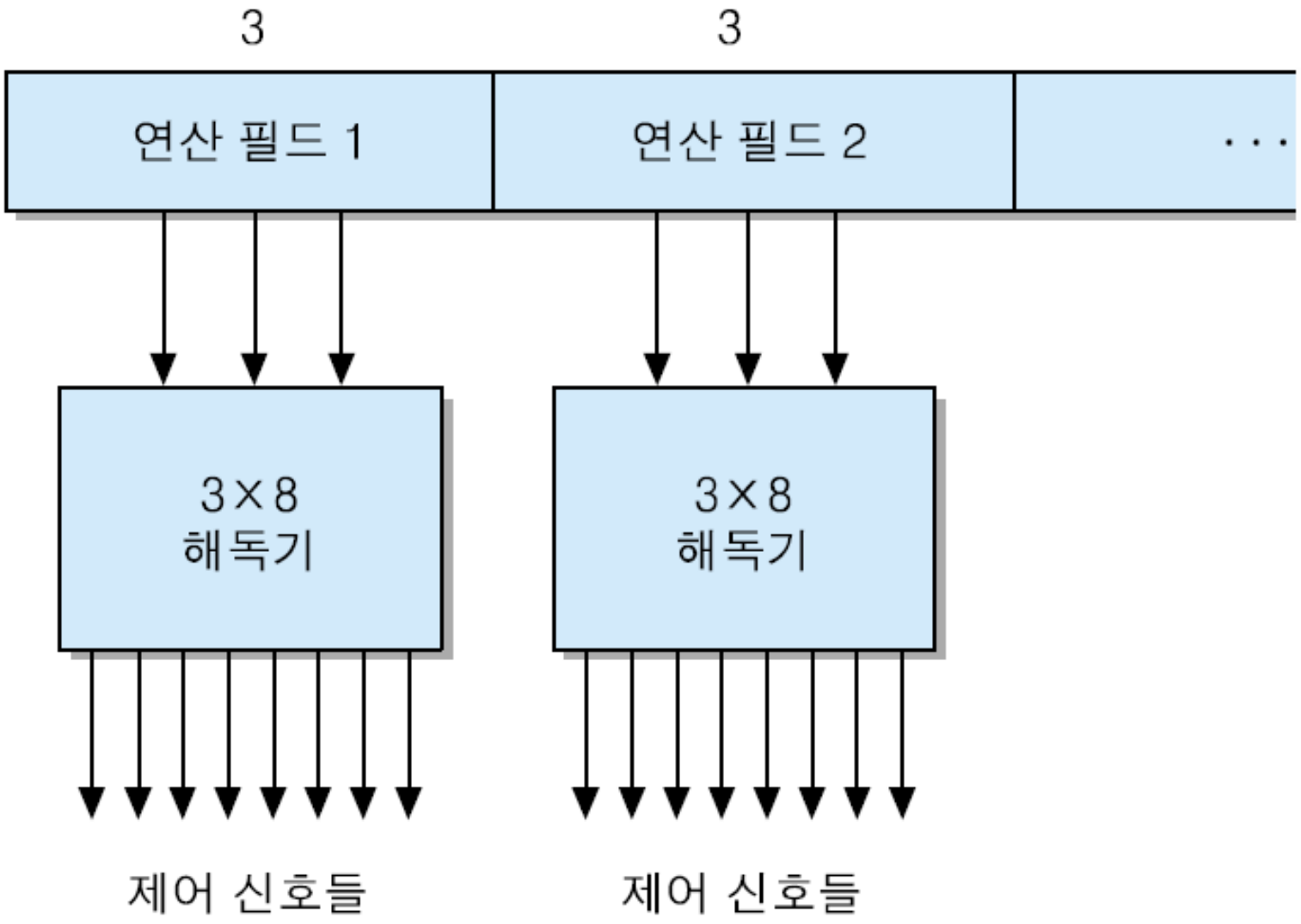
- 제어 기억장치로부터 인출된 마이크로명령어 내 연산 필드의 비트들이 제어 유니트의 외부로 출력되어, 각각 제어 신호로 사용됨
  - 수직적 마이크로프로그래밍
  - 수평적 마이크로프로그래밍

수직적 마이크로프로그래밍

수직적 마이크로프로그래밍(vertical microprogramming)

- 마이크로명령어의 연산 필드에 적은 수의 코드화(encoded)된 비트들을 포함시키고, **해독기(decoder)**를 이용하여 그 비트들을 필요한 수 만큼의 제어 신호들로 확장하는 방식
  - N개 제어 신호가 필요할 경우 $\lceil \log_2 N \rceil$비트 필요
- **장점**
  - 마이크로명령어의 길이(비트 수) 최소화
  - 제어 기억장치 용량 감소
- **단점**
  - 제어 신호를 생성하는 추가적인 하드웨어 필요
  - 해독 동작에 걸리는 만큼의 지연 시간 발생
  - 낮은 수준의 병렬 처리

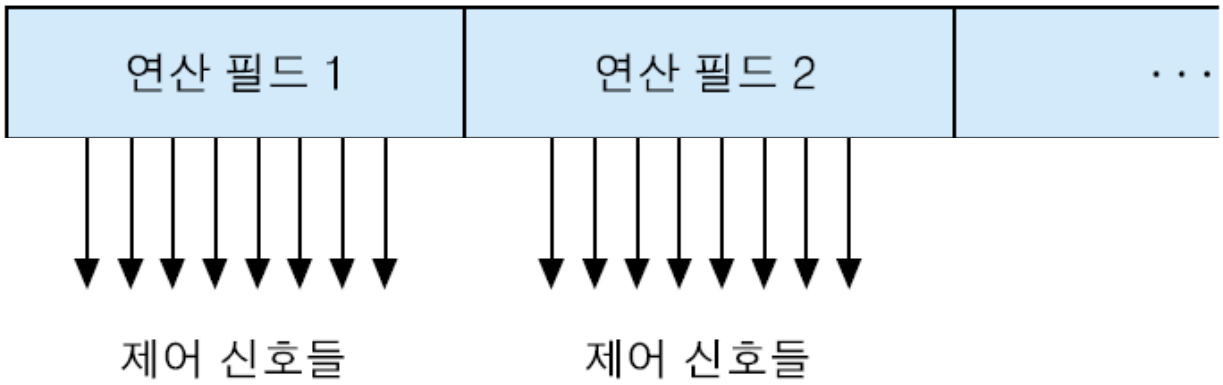
수직적 마이크로프로그래밍에서의 제어 신호 발생 방법



수평적 마이크로프로그래밍

❑ 수평적 마이크로프로그래밍(horizontal microprogramming)

- 연산 필드의 개별 비트와 제어 신호를 1:1로 대응
  - 연산 필드의 개별 비트가 일종의 on/off 스위치로 활용되어 대응되는 제어 신호를 발생시키는 제어방식
- 필요한 제어 신호 수 = 연산 필드의 비트 수
- **장점:** 하드웨어가 간단하고, 해독에 따른 지연 시간이 없음, 높은 수준의 병렬 처리 가능
- **단점:** 마이크로명령어의 비트 수가 길어지기 때문에 제어 기억장치의 용량이 증가



Q & A

End!