컴퓨터 구조

제어 유니트
(제 7주 차)

서울사이버대학교

오 창 환

학습 목표

- 제어 유니트의 기능, 제어 유니트의 구조 등을 설명할 수 있다.
- 마이크로명령어의 형식을 설명할 수 있다.
- 마이크로프로그래밍, 마이크로프로그램의 순서제어 등을 설명할 수 있다.

학습 내용

- 제어 유니트의 기능, 제어 유니트의 구조
- 마이크로명령어의 형식
- 마이크로프로그래밍, 마이크로프로그램의 순서제어

제어 유니트의 기능 (1)

- ALU 및 레지스터들과 더불어 CPU의 주요 구성요소들 중의 하나인 제어 유니트는 다음과 같은 기능을 수행함.
  * 명령어 코드의 해독
  * 명령어 실행에 필요한 제어 신호들의 발생
- 제어 유니트는 명령어 사이클이 적절히 수행되도록 모든 동작들을 제어하는 장치임.
- 각 명령어 사이클은 부 사이클인 인출 사이클, 간접 사이클 및 실행 사이클로 이루어지며, 각 사이클에서는 여러 개의 마이크로-연산들이 수행됨.
  예를 들어서, 인출 사이클에서 수행되는 마이크로-연산들은 아래와 같음.
  t_0 : MAR ← PC
  t_1 : MBR ← M[MAR], PC ← PC + 1
  t_2 : IR ← MBR

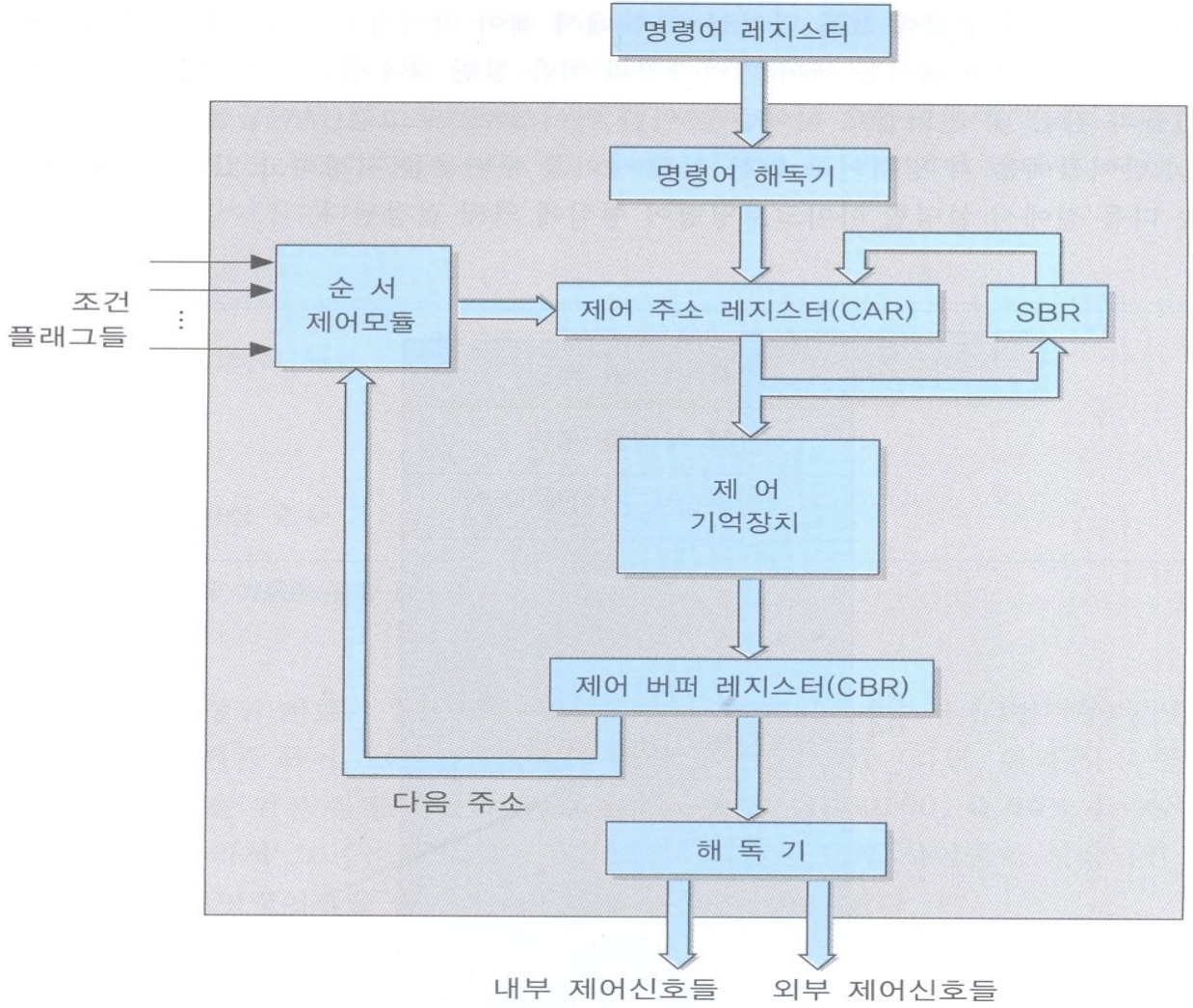
제어 유니트의 기능 (2)

- CPU 클럭 주기마다 서로 다른 마이크로-연산이 수행되며,
  결과적으로 명령어 인출 동작은 세 주기만에 종료되고,
  두 번째 주기에서는 두 개의 마이크로-연산들이 동시에 수행됨.
- 각 마이크로-연산이 실제 수행되기 위해서는 2진 비트들로 표현되어야 하는데,
  이를 마이크로명령어 혹은 제어 단어라고 부르고,
  마이크로명령어들의 집합을 마이크로프로그램(microprogram)이라고 함.
- 마이크로명령어들은 명령어 인출과 같은 CPU의 특정 기능을 위해
  그룹 단위로 작성되는데, 이러한 각 그룹을 루틴(routine)이라고 부름.
- 결과적으로 명령어사이클을 위한 마이크로프로그램은 인출 사이클 루틴, 간접 사이클 루틴, 실행 사이클 루틴 등으로 구성됨.

제어 유니트의 구조 (1)

- 제어 유니트 구성 요소들의 각 기능은 다음과 같음.
  * 명령어 해독기(instruction decoder) : 명령어 레지스터(IR)로부터 들어오는 명령어의 연산 코드를 해독하여 해당 연산을 수행하기 위한 루틴의 시작 주소를 결정함.
  * 제어 주소 레지스터 (control address register : CAR) : 다음에 실행할 마이크로명령어의 주소를 저장하는 레지스터인데 이 주소는 제어 기억장치의 특정 위치를 가리킴.
  * 제어 기억장치(control memory) : 마이크로명령어들로 이루어진 마이크로프로그램을 저장하는 내부 기억장치임.
  * 제어 버퍼 레지스터(control buffer register : CBR) : 제어 기억장치로부터 읽혀진 마이크로명령어 비트들을 일시적으로 저장하는 레지스터임.
  * 서브루틴 레지스터(subroutine register : SBR) : 마이크로프로그램에서 서브루틴이 호출되는 경우에 현재의 CAR 내용을 일시적으로 저장하는 레지스터임.
  * 순서제어 모듈(sequencing module) : 마이크로명령어의 실행 순서를 결정하는 회로들의 집합임.

제어 유니트의 구조 (2)

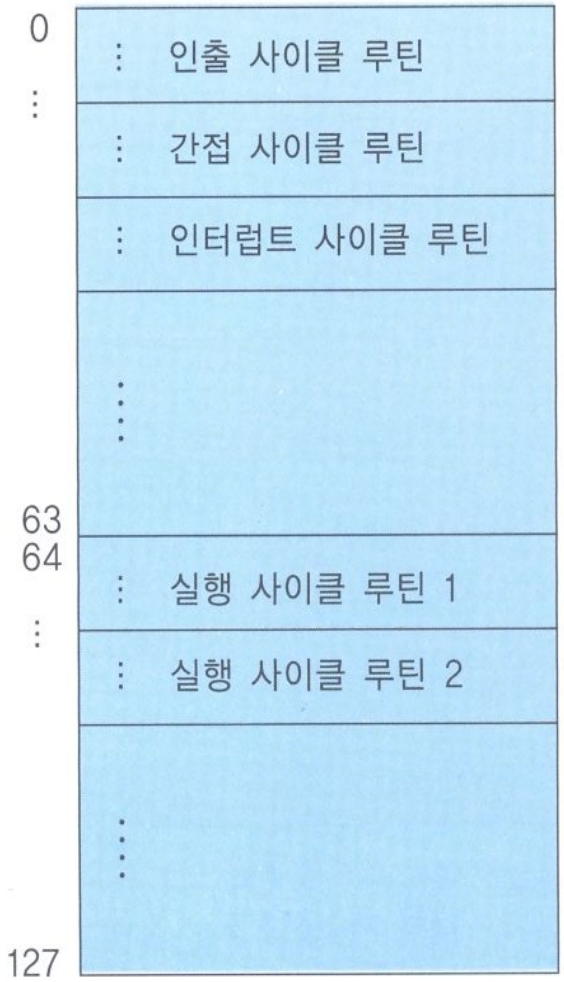


• 제어 유니트의 내부 구성도

제어 유니트의 구조 (3)

- CPU의 명령어 세트를 설계한다는 것은 (1) 명령어들의 종류와 비트 패턴을 정의하고, (2) 그 명령어들의 실행에 필요한 하드웨어를 설계하며, (3) 각 명령어를 위한 실행 사이클 루틴을 마이크로프로그래밍 한다는 것을 의미함.
  마이크로프로그램은 그러한 루틴들의 집합이므로 CPU 설계 단계에서 확정되고,
  그 이후에는 변하지 않으며
  마이크로프로그램을 저장하는 제어 기억장치는 ROM으로 만들어져 CPU 칩 내에 둠.
- 루틴들의 길이와 제어 기억장치에 저장되는 위치는 CPU마다 다른데,
  예를 들어서 제어 기억장치의 용량이 128 단어인 CPU에서
  제어 기억장치의 처음 절반(64 단어) 부분에 공통 루틴들,
  즉 인출과 간접 및 인터럽트 사이클을 위한 마이크로프로그램 루틴들을 저장하고,
  하반부 64 단어들에는 각 명령어를 위한 실행 사이클 루틴들을 저장하고 있음.

제어 유니트의 구조 (4)



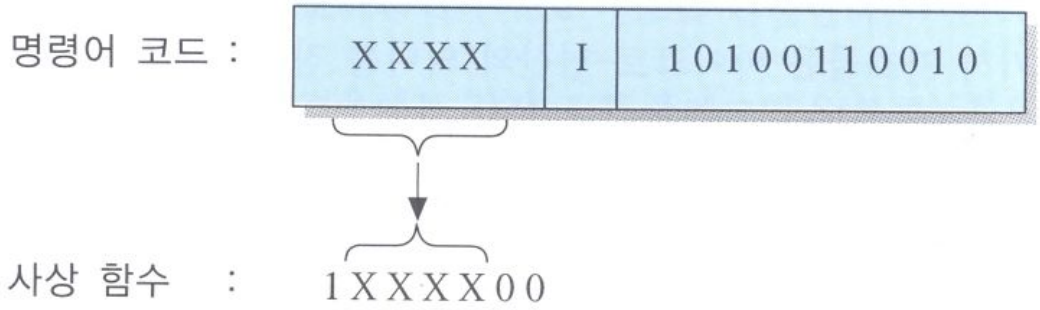
• 제어 기억장치의 내부 구성

제어 유니트의 구조 (5)

- 인출 사이클 동안에 명령어 레지스터로 적재된 명령어 비트들 중에서
  연산 코드 부분은 제어 유니트의 명령어 해독기로 들어오는데
  이 연산 코드가 지정하는 연산은
  실행 사이클 동안에 제어 기억장치에 저장된 해당 루틴을 실행함으로써 수행됨.
- 따라서 명령어 해독기는 연산 코드를 이용하여
  제어 기억장치 내 해당 실행 사이클 루틴의 시작 주소를 찾아야 하는데
  그 방법에는 여러가지가 있으나,
  여기서는 사상(mapping)을 이용하는 방식에 대해 설명함.

제어 유니트의 구조 (6)

- 실행 사이클 루틴들이 제어 기억장치의 64번지부터 저장되어 있고, 각 루틴은 최대 네 개씩의 마이크로명령어들로 구성된다고 가정할 때에, 만약 16 비트 길이의 명령어가 4비트의 연산코드, 1비트의 간접 주소지정(I)비트 및 11비트의 주소로 구성되어 있다면 사상 과정은 아래 그림과 같음.



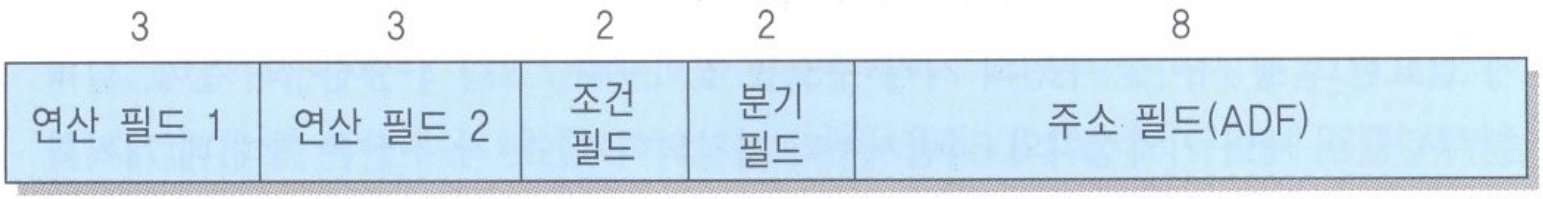
• 연산 코드를 이용한 루틴 주소의 사상

- 사상 함수의 최상위 비트가 1이므로, 사상에 의해 결정되는 주소가 64번지부터 시작할 수 있게 됨. 예를 들어서 LOAD 명령어의 연산 코드가 0001이라면 위의 사상 함수에 의해 이 연산을 위한 실행 사이클 루틴의 시작 주소는 1000100, 즉 64+4=68번지로 결정됨.

2 교시

마이크로명령어의 형식 (1)

• 아래 그림은 제어 기억장치에 저장되는 마이크로 명령어 형식의 한 예를 보여줌.



• 마이크로 명령어 형식의 예

• 이 예에서 마이크로 명령어는 길이가 18비트이고 다섯 개의 필드로 구성되며, 연산 필드가 두 개이므로 두 개의 마이크로-연산들이 동시에 수행될 수 있음.

• 조건 필드는 분기에 사용될 조건 플래그를 지정하고, 분기 필드는 분기의 종류와 다음에 실행할 마이크로 명령어의 주소를 결정하는 방법을 명시함.

• 주소 필드는 분기가 발생하는 경우를 위하여 목적지 마이크로 명령어의 주소를 가지고 있음.

마이크로명령어의 형식 (2)

• 아래는 마이크로프로그램의 예시를 보여줌.

(a) 연산 필드1에 위치할 마이크로-연산들

코드	마이크로-연산	기 호
000	None	NOP
001	MAR ← PC	PCTAR
010	MAR ← IR(*addr*)	IRTAR
011	AC ← AC + MBR	ADD
100	MBR ← M[MAR]	READ
101	AC ← MBR	BRTAC
110	IR ← MBR	BRTIR
111	M[MAR] ← MBR	WRITE

(b) 연산 필드2에 위치할 마이크로-연산들

코드	마이크로-연산	기 호
000	None	NOP
001	PC ← PC + 1	INCPC
010	MBR ← AC	ACTBR
011	MBR ← PC	PCTBR
100	PC ← MBR	BRTPC
101	MAR ← SP	SPTAR
110	AC ← AC − MBR	SUB
111	PC ← IR(*addr*)	IRTPC

• 마이크로-연산들에 대한 2진 코드 및 기호의 예

마이크로명령어의 형식 (3)

- 두 비트로 구성되는 조건 필드는 다음에 위치한 분기 필드를 위한 조건을 나타냄.
- 이 필드의 값이 00이면, 현재의 마이크로 명령어의 실행이 완료된 다음에는 무조건 분기하는데, 분기될 목적지 마이크로 명령어 주소는 이 형식의 마지막에 있는 주소 필드의 값이 됨. 즉, 주소 필드의 내용이 CAR로 적재됨.
- 만약 조건 필드의 값이 01이면, I 비트의 값에 따라 분기 여부가 결정되는데 여기에서 I 비트는 간접 주소지정 방식을 나타내는 비트임. 만약 I비트가 1이라면, 간접 사이클 루틴을 호출하여 기억장치로부터 오퍼랜드의 유효 주소를 인출함.
- 조건 필드의 값이 10 혹은 11인 경우에는 각각 조건 플래그인 S(부호) 혹은 Z(영) 플래그의 값이 1이면 분기가 일어남.

코드	조건	기호	설명
00	1	U	무조건 분기
01	I 비트	I	간접 주소지정
10	AC(S)	S	누산기(AC)에 저장된 데이터의 부호
11	AC=0	Z	AC에 저장된 데이터=0

• 조건 필드의 코드 지정

마이크로명령어의 형식 (4)

- 분기 필드도 두 비트로 구성되는데, 먼저 이 필드의 값이 00이면, 조건부 점프를 나타냄. 따라서 만약 조건 필드가 가리키는 조건이 만족되면, 주소 필드(ADF) 값이 CAR로 적재되어서 그 주소의 마이크로명령어로 점프하게 됨.
  그러나 만약 조건이 만족되지 않으면 분기는 일어나지 않고 CAR이 1 증가되어 다음에 위치한 마이크로명령어를 실행하게 됨.
- 분기 필드의 값이 01이면, 조건부 호출을 나타내는데, 이 경우에 만약 조건이 만족된다면, 현재의 CAR 내용에 1을 더한 값이 서브루틴 레지스터(SBR)에 저장되고, 주소 필드의 값이 CAR로 적재되어 다음 사이클에서 그 주소의 마이크로명령어가 실행됨. 그러나 만약 조건이 만족되지 않는다면, 원래 순서대로 다음에 위치한 마이크로명령어가 실행됨.
- 호출되는 루틴의 마지막 마이크로명령어의 분기 필드에는 반드시 복귀(return)를 나타내는 필드값인 10이 들어있어야 함. 그러면 SBR에 저장되어 있던 주소가 CAR로 다시 적재됨으로써 호출되기 전의 프로그램 실행 순서로 되돌아가게 됨.

마이크로명령어의 형식 (5)

- 분기 필드값이 11이면 사상(mapping)의 결과로 얻어지는 루틴의 시작주소가 CAR로 적재되어 그 주소로 분기하게 됨. 즉 인출 사이클 루틴의 실행이 종료된 다음에, 명령어의 연산 코드에 따라 적절한 실행 사이클 루틴으로 분기되도록 해 줌.
- 아래 그림의 마지막 코드 부분에는 앞에서 설명한 사상 방법이 실제로 적용되어 있는데 CAR의 첫 번째(최상위) 비트에는 항상 1이 적재되고, 두 번째부터 다섯 번째 비트까지는 연산 코드(op code) 비트들이 들어가며, 마지막 두 비트들은 항상 0임.

코드	기호	설 명
00	JMP	만약 조건 = 1이면, CAR ← ADF 만약 조건 = 0이면, CAR ← CAR + 1
01	CALL	만약 조건 = 1이면, CAR ← ADF, SBR ← CAR + 1 만약 조건 = 0이면, CAR ← CAR + 1
10	RET	CAR ← SBR (서브루틴으로부터의 복귀)
11	MAP	CAR(1) ← 1, CAR(2-5) ← IR(op), CAR(6,7) ← 0

• 분기 필드의 코드 지정

3 교시

마이크로프로그래밍 (1)

(1) 인출 사이클 루틴

• 인출 사이클의 마이크로명령어 루틴은 아래와 같으며, 이 루틴이 제어 기억장치의 0번지부터 저장된다고 가정하였으므로 ORG 0으로 그 위치를 표기하였음.

```
        ORG 0
FETCH:  PCTAR          U  JMP  NEXT   ; MAR ← PC, 다음 마이크로명령어 실행
        READ, INCPC    U  JMP  NEXT   ; MBR ← M[MAR], PC = PC + 1,
                                        다음 마이크로명령어 실행.
        BRTIR          U  MAP         ; IR ← MBR, 해당 실행 사이클 루틴으로 분기.
```

• 이 루틴은 세 개의 마이크로명령어들로 구성되는데, 두 번째 마이크로명령어에서는 두 개의 마이크로-연산들이 동시에 수행되고 마지막 단계에서는 사상을 이용하여 CAR에 실행 사이클 루틴의 시작 주소를 적재함으로써 다음 사이클에서 해당 실행 루틴으로 분기가 일어나게 되며, 2진 비트 패턴으로 변환하면 아래와 같음.

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

마이크로프로그래밍 (2)

(2) 간접 사이클 루틴

- 어떤 명령어가 간접 주소지정 방식을 사용하는 경우에는 명령어 내의 I비트가 1로 세트되는데, 이 경우에는 실행 사이클의 시작 부분에서 간접 사이클 루틴을 호출하여 기억장치로부터 실제 오퍼랜드 주소를 읽어와야 함.
- 일반적으로 간접 사이클 루틴은 인출 사이클 루틴의 다음 위치인 4번지부터 저장되며, 아래와 같은 마이크로명령어들로 이루어짐.

```
        ORG 4
INDRT:  IRTAR   U  JMP  NEXT    ; MAR ← IR(addr), 다음 마이크로명령어 실행
        READ    U  JMP  NEXT    ; MBR ← M[MAR], 다음 마이크로명령어 실행
        BRTIR   U  RET          ; IR(addr) ← MBR, 실행 사이클 루틴으로 복귀
```

- 즉 IR에 저장되어 있는 명령어의 주소 필드가 가리키는 기억장치 위치로부터 실제 주소를 인출하여 다시 IR의 주소 필드에 적재하는 것임. 이 루틴의 마지막 마이크로명령어가 실행된 다음에는 이 간접 사이클을 호출하였던 실행 사이클 루틴으로 복귀해야 함.

마이크로프로그래밍 (3)

(3) 실행 사이클 루틴

• 어떤 CPU가 아래와 같이 열거된 명령어들을 가지고 있을 때, 이들에 대한 실행 사이클 루틴들을 마이크로프로그래밍 해 볼 수 있음.

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	$1010100 = 84_{10}$

• 연산 코드들에 대한 사상의 결과

• 명령어 NOP은 아무런 연산도 수행하지 않고 PC만 1 증가시키는 명령어이지만, 특정 목적을 위하여 거의 모든 CPU들의 명령어 세트에 포함되어 있음.

• 실행 사이클 루틴들의 마지막 마이크로명령어에서는 다음에 인출 사이클 루틴(FETCH)으로 점프되도록 지정함으로써, 각 실행 사이클 루틴의 수행이 종료된 다음에는 인출 사이클부터 다시 시작하도록 함.

마이크로프로그래밍 (4)

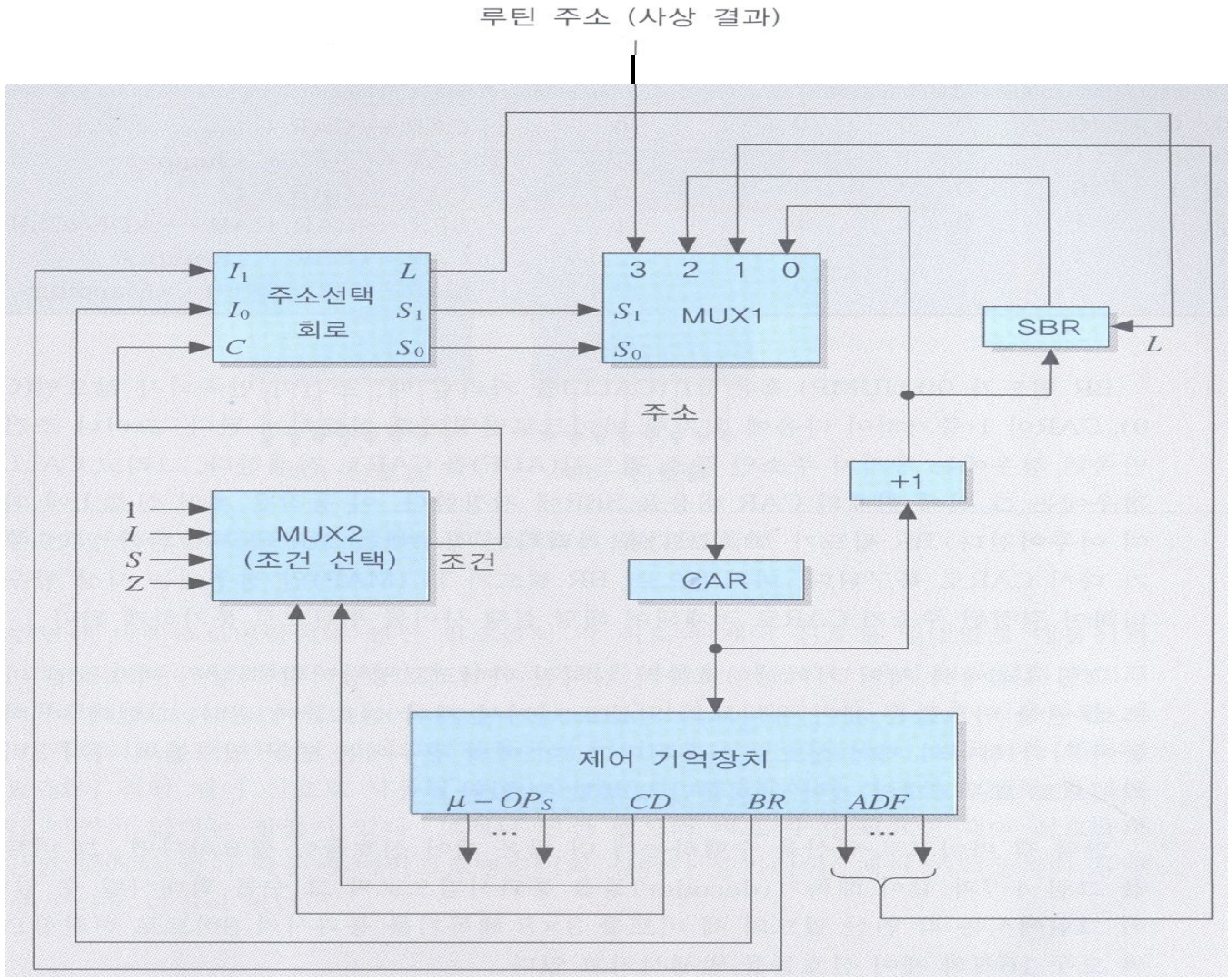

```
        ORG 64
NOP:    INCPC   U   JMP   FETCH    ; PC ← PC + 1

        ORG 68
LOAD:   NOP     I   CALL  INDRT    ; I = 1이면, 간접 사이클 루틴 호출
        IRTAR   U   JMP   NEXT     ; MAR ← IR(addr)
        READ    U   JMP   NEXT     ; MBR ← M[MAR]
        BRTAC   U   JMP   FETCH    ; AC ← MBR

        ORG 72
STORE:  NOP     I   CALL  INDRT    ; I = 1이면, 간접 사이클 루틴 호출
        IRTAR   U   JMP   NEXT     ; MAR ← IR(addr)
        ACTBR   U   JMP   NEXT     ; MBR ← AC
        WRITE   U   JMP   FETCH    ; M[MAR] ← MBR

        ORG 76
ADD:    IRTAR   U   JMP   NEXT     ; MAR ← IR(addr)
        READ    U   JMP   NEXT     ; MBR ← M[MAR]
        ADD     U   JMP   FETCH    ; AC ← AC + MBR

        ORG 80
SUB:    IRTAR   U   JMP   NEXT     ; MAR ← IR(addr)
        READ    U   JMP   NEXT     ; MBR ← M[MAR]
        SUB     U   JMP   FETCH    ; AC ← AC − MBR

        ORG 84
JUMP:   IRTPC   U   JMP   FETCH    ; PC ← IR(addr)
```


• 실행 사이클 루틴들

마이크로프로그램의 순서제어 (1)

- 제어 유니트가 명령어의 실행을 제어한다는 것은 제어 기억장치에 저장된 해당 마이크로명령어들을 순서대로 인출하는 것이라고 할 수 있음.
  즉 각 마이크로명령어를 읽어서 연산 필드에 있는 비트들을 출력시키면 그 자체가 제어 신호들이 되는 것임.
- 아래 그림에서 제어 기억장치를 제외한 상단의 모듈들은 다음에 실행할 마이크로명령어의 주소를 결정하는 회로들이며, 그 기능을 순서제어(sequencing)라 함.
- 이 회로에서의 핵심 요소는 다음에 인출할 마이크로명령어의 제어 기억장치 주소를 가지고 있는 CAR임.
- 제어 기억장치로부터 출력된 그 마이크로명령어의 비트들 중에 CD 필드의 두 비트들은 MUX2로 보내져서 4개의 조건 비트들 중의 하나를 선택하게 되고, 이 출력은 상단의 주소 선택 회로의 한 입력(C)으로 들어감.
  주소 선택 회로의 다른 두 입력들로는 BR 필드의 두 비트들이 들어감.
  그 세 입력들이 조합되면 MUX1의 선택 신호들과 SBR의 적재(L) 신호가 생성되며, 그에 따라 다음 주소들이 결정되어 CAR로 적재됨.

마이크로프로그램의 순서제어 (2)



서울사이버대학교 SEOUL CYBER UNIVERSITY

• 순서제어를 위한 세부 회로가 포함된 제어 유니트의 구성도

마이크로프로그램의 순서제어 (3)

BR I_1	BR I_0	조건 C	MUX1 선택 S_1	MUX1 선택 S_0	SBR L	CAR에 적재되는 MUX1의 입력	설 명
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, CAR ← ADF <Call>
1	0	x	1	0	0	2	CAR ← SBR <Return>
1	1	x	1	1	0	3	CAR ← 1XXXX00 <Mapping>

• 주소 선택 회로의 입력 및 출력 신호들

• BR 필드가 00(JUMP) 혹은 01(CALL)을 가리킬 때, 조건이 만족되지 않으면(C = 0), CAR이 1 증가하여 다음에 위치한 마이크로명령어를 실행하게 됨. 그러나 조건이 만족된 경우에는 목적지 주소인 주소 필드값(ADF)을 CAR로 적재함.
그리고 CALL의 경우에는 그 전에 현재의 CAR 내용을 SBR에 저장하는데 이 동작은 제어 신호 L에 의해 이루어짐.
BR 필드가 10(RET)을 가리키는 경우에는 SBR에 저장되어 있던 주소가 다시 CAR로 복구되고, BR 필드가 11(MAP)인 경우에는 사상 함수에 의해 결정된 주소가 CAR로 적재되어 해당 실행 사이클 루틴으로 분기하게 됨.

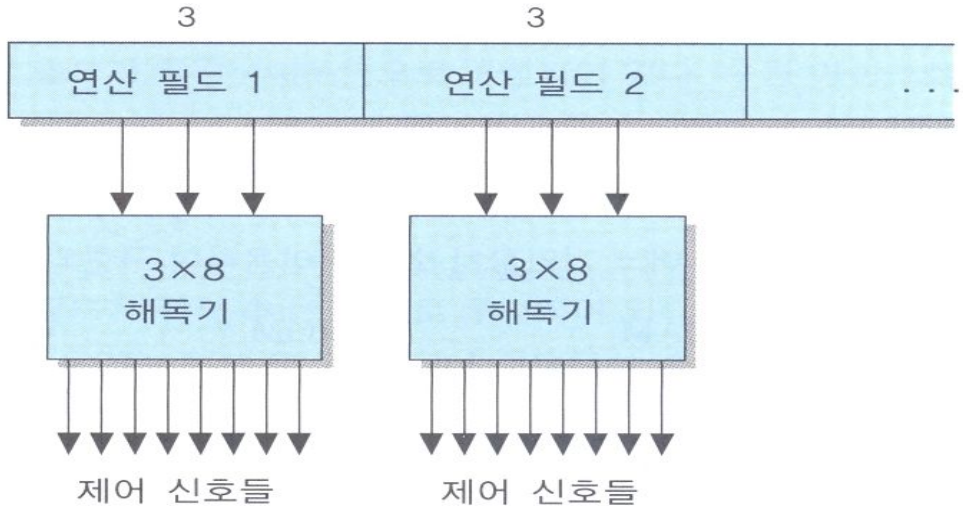
마이크로프로그램의 순서제어 (4)

- 제어 기억장치로부터 읽혀진 마이크로명령어의 최상위 비트들인
  마이크로-연산 비트들은 제어 유니트의 외부로 나가서 제어 신호들이 되는데,
  이 비트들이 각각 하나의 제어 신호로 사용한다면 각각 3비트씩이므로 모두 6개의
  제어 신호들만 발생할 수 있음.
- 만약 아래 그림과 같이 연산 필드를 3 x8 해독기의 입력으로 사용해서 회로를 구성하면
  각각 8개씩 모두 16개의 제어 신호들을 발생시킬 수 있음.
- 이와 같이 외부에 해독기들을 접속하여 원하는 수만큼의 제어신호로 확장시키는 방식을
  수직적 마이크로프로그래밍이라 하고 이 방식에서 사용되는 마이크로명령어를 수직적
  마이크로명령어라고 부르는데,
  이 방식은 마이크로명령어의 길이가 짧기 때문에
  제어 기억장치의 용량이 적게 필요하다는 장점이 있지만,
  해독기를 통과하는데 걸리는 시간만큼 지연이 발생한다는 단점이 있음.

마이크로프로그램의 순서제어 (5)

- 반면에 마이크로명령어 내 연산 필드들의 각 비트와 제어 신호를 일대일로 대응시켜서 그 수만큼의 비트들로 이루어진 마이크로명령어들을 사용하는 방식을 수평적 마이크로 프로그래밍이라고 함.

  이 방식은 해독기를 통과할 필요가 없으므로 하드웨어가 간단하고 해독에 따른 지연시간이 없다는 장점이 있지만, 연산 필드의 비트 수가 필요한 제어 신호들의 수만큼 되어야 하기 때문에 마이크로명령어의 길이가 증가하여 제어 기억장치의 용량이 커진다는 단점이 있음.



- 수직적 마이크로명령어로부터 제어 신호들을 발생하는 방법

셀프 테스트

• 제어 유니트 구성 요소들 중에서 다음에 실행할 마이크로명령어의 주소를 저장하는 레지스터의 영어 약어는 무엇인가?

* CAR

해설) CAR은 Control Address Register의 약어로서 CPU의 PC와 유사한 기능, 즉 다음에 실행할 명령어 주소를 저장함.

• 사상 함수의 최상위 비트가 1이고 하위 두 비트는 00이며 연산코드가 0010일 때에 실행 사이클 루틴의 시작 주소의 번지 수는 얼마인가?

* 72

해설) 최상위비트, 연산코드, 하위비트를 순서대로 연결하면 1001000 이 되므로 이는 72를 나타냄.

• 아무런 연산도 수행하지 않고 PC만 1 증가시키는 마이크로명령어는 무엇인가?

* NOP

해설) NOP는 No Operation의 의미로서 아무런 연산을 수행하지 않고 PC만 증가시키는 마이크로명령어임.

요점 정리

- ALU 및 레지스터들과 더불어 CPU의 주요 구성요소들 중의 하나인 제어 유니트는 명령어 코드의 해독, 명령어 실행에 필요한 제어 신호들의 발생 등을 수행함.
- 각 마이크로-연산이 실제 수행되기 위해서는 2진 비트들로 표현되어야 하는데, 이를 마이크로명령어라고 부르고, 마이크로명령어들의 집합을 마이크로프로그램이라고 함.
- 명령어 해독기는 연산 코드를 이용하여 제어 기억장치 내 해당 실행 사이클 루틴의 시작 주소를 찾는 방법에는 사상(mapping)을 이용하는 방식이 있음.
- 인출 사이클 루틴은 세 개의 마이크로명령어들로 구성되는데, 두 번째 마이크로명령어에서는 두 개의 마이크로-연산들이 동시에 수행되고 마지막 단계에서는 사상을 이용하여 CAR에 실행 사이클 루틴의 시작 주소를 적재함.
- 제어 유니트가 명령어의 실행을 제어한다는 것은 제어 기억장치에 저장된 해당 마이크로명령어들을 순서대로 인출하는 것이라고 할 수 있음.
- 제어 기억장치로부터 읽혀진 마이크로명령어의 최상위 비트들인 마이크로-연산 비트들은 제어 유니트의 외부로 나가서 제어 신호들이 되는데 이러한 제어 신호를 발생시키는 방법에 따라 수직적 마이크로명령어와 수평적 마이크로명령어 등으로 구분됨.