제4장 제어 유니트

4.1 제어 유니트의 기능

4.2 제어 유니트의 구조

4.3 마이크로 명령어의 형식

4.4 마이크로프로그래밍

4.5 마이크로프로그램의 순서 제어

4.1 제어 유니트의 기능

- 제어 유니트의 기능
  - 명령어 코드의 해독
  - 명령어 실행에 필요한 제어 신호들의 발생
- 마이크로명령어(micro-instruction) : 명령어 사이클의 각 주기에서 실행되는 각 마이크로-연산을 지정해주는 2진 비트들로서, 제어 단어(control word)라고도 함.
- 마이크로프로그램(microprogram) : 마이크로명령어들의 집합
- 루틴(routine) : CPU의 특정 기능을 수행하기 위한 마이크로명령어들의 그룹

[예] 인출 사이클 루틴, 실행 사이클 루틴, 인터럽트 사이클 루틴

4.2 제어 유니트의 구조

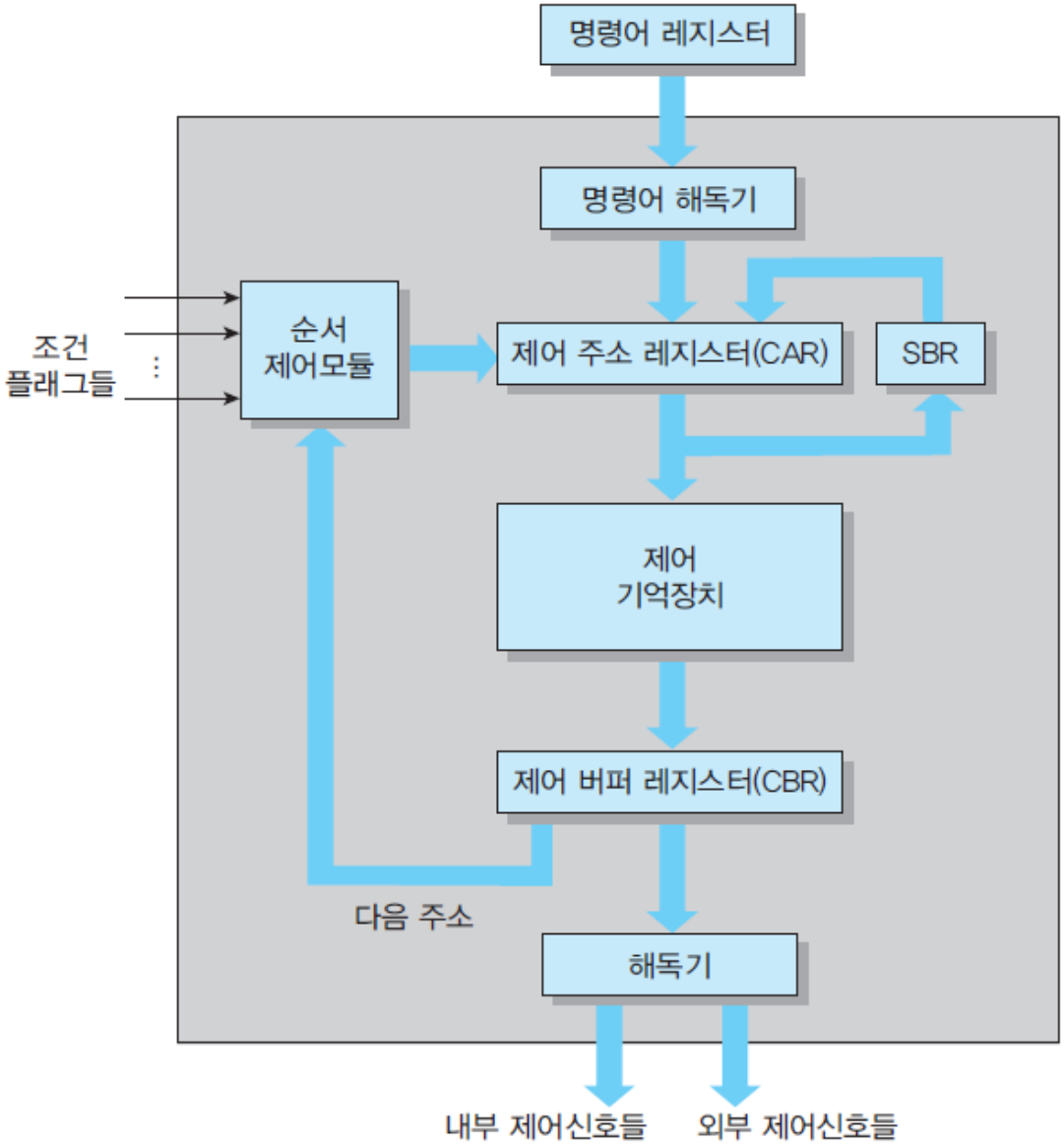
❑ 구성 요소들

- 명령어 해독기(instruction decoder) : 명령어 레지스터(IR)로부터 들어오는 명령어의 연산 코드를 해독하여 해당 연산을 수행하기 위한 루틴의 시작 주소를 결정
- 제어 주소 레지스터(control address register: CAR) : 다음에 실행할 마이크로명령어의 주소를 저장하는 레지스터
  → 이 주소는 제어 기억장치의 특정 위치를 지칭
- 제어 기억장치(control memory) : 마이크로명령어들로 이루어진 마이크로프로그램을 저장하는 내부 기억장치

제어 유니트의 구조 (계속)

- 제어 버퍼 레지스터(control buffer register: CBR) : 제어 기억장치로부터 읽혀진 마이크로명령어 비트들을 일시적으로 저장하는 레지스터
- 서브루틴 레지스터(subroutine register: SBR) : 마이크로 프로그램에서 서브루틴이 호출되는 경우에 현재의 CAR 내용을 일시적으로 저장하는 레지스터
- 순서제어 모듈(sequencing module) : 마이크로명령어의 실행 순서를 결정하는 회로들의 집합

제어 유니트의 내부 구성도



CPU의 명령어 세트 설계 과정

① 명령어들의 종류 및 비트 패턴 정의

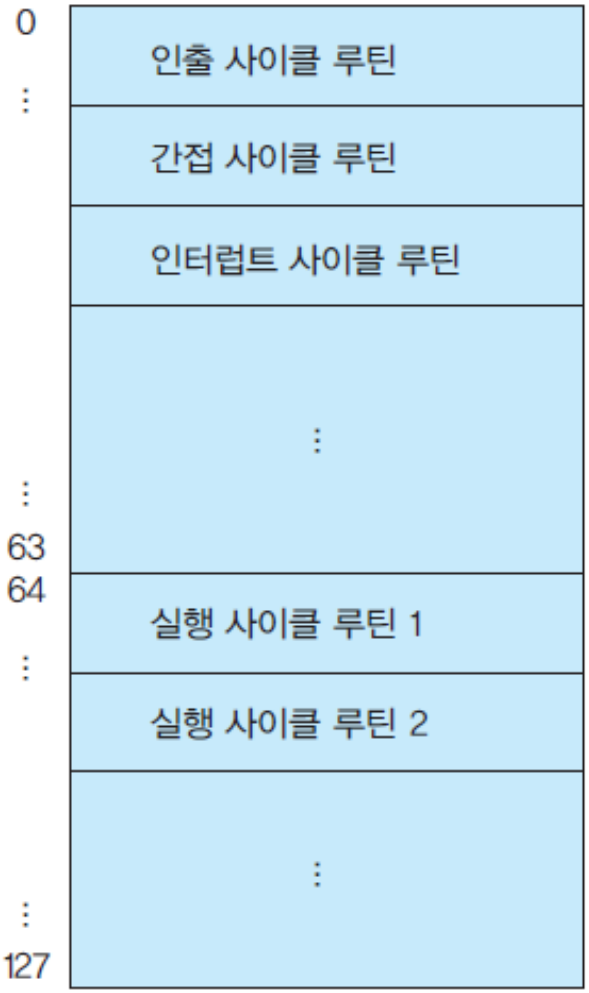
② 명령어들의 실행에 필요한 하드웨어 설계

③ 각 명령어를 위한 실행 사이클 루틴 작성(마이크로프로그래밍)

④ 마이크로프로그램 코드들을 제어 기억장치에 저장

제어 기억장치의 내부 구성

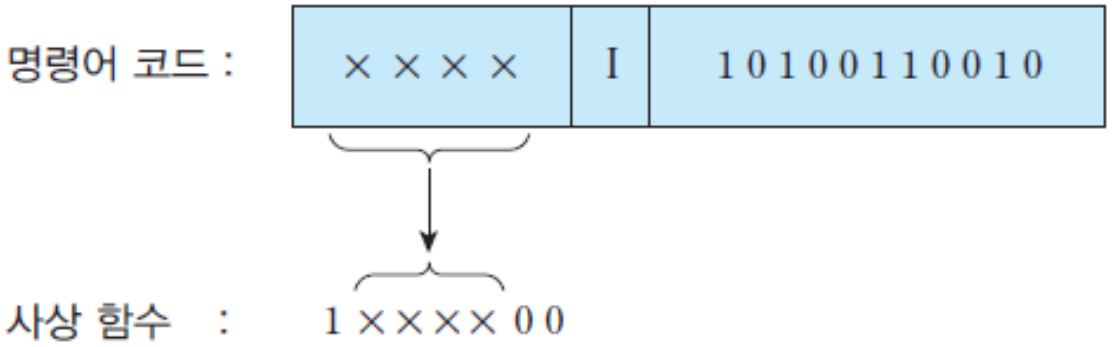
- 마이크로프로그램 루틴들을 제어 기억장치에 저장한 예
  - 제어 기억장치 용량 = 128 단어
  - 전반부 (0 ~ 63번지) : 공통 루틴들 저장
  - 후반부 (64 ~ 127번지) : 각 명령어의 실행 사이클 루틴들 저장



명령어 해독

- 명령어의 연산 코드가 지정하는 연산을 위한 실행 사이클 루틴의 시작 주소를 결정하는 동작
- 사상(mapping)을 이용한 해독 방법
  - 명령어의 연산 코드를 특정 비트 패턴과 조합

[예] 16-비트 길이의 명령어가 4 비트의 연산 코드, 1 비트의 간접 주소지정(I) 비트 및 11 비트의 주소로 구성된 경우



- 연산 코드 = 0001→ 실행 사이클 루틴의 시작 주소 = 1000100 (68_{10})
- 연산 코드 = 0110 → 실행 사이클 루틴의 시작 주소 = 1011000 (88_{10})

4.3 마이크로명령어의 형식

- 연산 필드가 두 개이면, 두 개의 마이크로-연산들을 동시에 수행 가능
- 조건(CD) 필드는 분기에 사용될 조건 플래그를 지정
- 분기(BR) 필드는 분기의 종류와 다음에 실행할 마이크로명령어의 주소를 결정하는 방법을 명시
- 주소 필드(ADF)의 내용은 분기가 발생하는 경우에 목적지 마이크로명령어의 주소로 사용

마이크로연산들에 대한 2진 코드 및 기호 [예]

❑ ‘연산필드 1’에 위치할 마이크로-연산들

코드	마이크로-연산	기호
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

마이크로 연산들에 대한 2진 코드 및 기호 [예] (계속)

❑ '연산필드 2'에 위치할 마이크로-연산들

코드	마이크로-연산	기호
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

조건 필드의 코드 지정

- 조건 필드 : 두 비트로 구성되며, 분기의 조건으로 사용

  U : 무조건 분기

  I : 만약 **I = 1**이면, 간접 사이클 루틴을 호출

  S : 누산기에 저장된 데이터의 부호가 **1**이면, 분기

  Z : 누산기에 저장된 데이터가 **0 (Z=1)**이라면, 분기

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

분기 필드의 코드 지정

- **분기 필드** : 두 비트로 구성되며, 분기 동작을 지정.
  - 조건 필드의 조건이 만족되면, ADF 필드의 내용을 CAR로 적재
    → 그 주소로 분기 (JUMP 혹은 CALL)
  - RET : 서브루틴으로부터 복귀(SBR에 저장된 내용을 CAR로 적재)
  - MAP : 사상 방식에 의하여 분기 목적지 주소 결정

코드	기호	설명
00	JMP	만약 조건 = 1이면, CAR ← ADF 만약 조건 = 0이면, CAR ← CAR + 1
01	CALL	만약 조건 = 1이면, SBR ← CAR + 1, CAR ← ADF 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

4.4 마이크로프로그래밍

4.4.1 인출 사이클 루틴의 마이크로명령어 루틴

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

- 2진 비트 패턴

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

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

4.4.2 간접 사이클 루틴

```
        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
```

❑ 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

4.4.3 실행 사이클 루틴

- 사상 방식을 이용하여 각 연산 코드에 대한 실행 사이클 루틴의 시작 주소를 결정하고, 각 명령어 실행을 위한 루틴을 작성
- 각 연산 코드에 대한 사상의 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	$1010100 = 84_{10}$

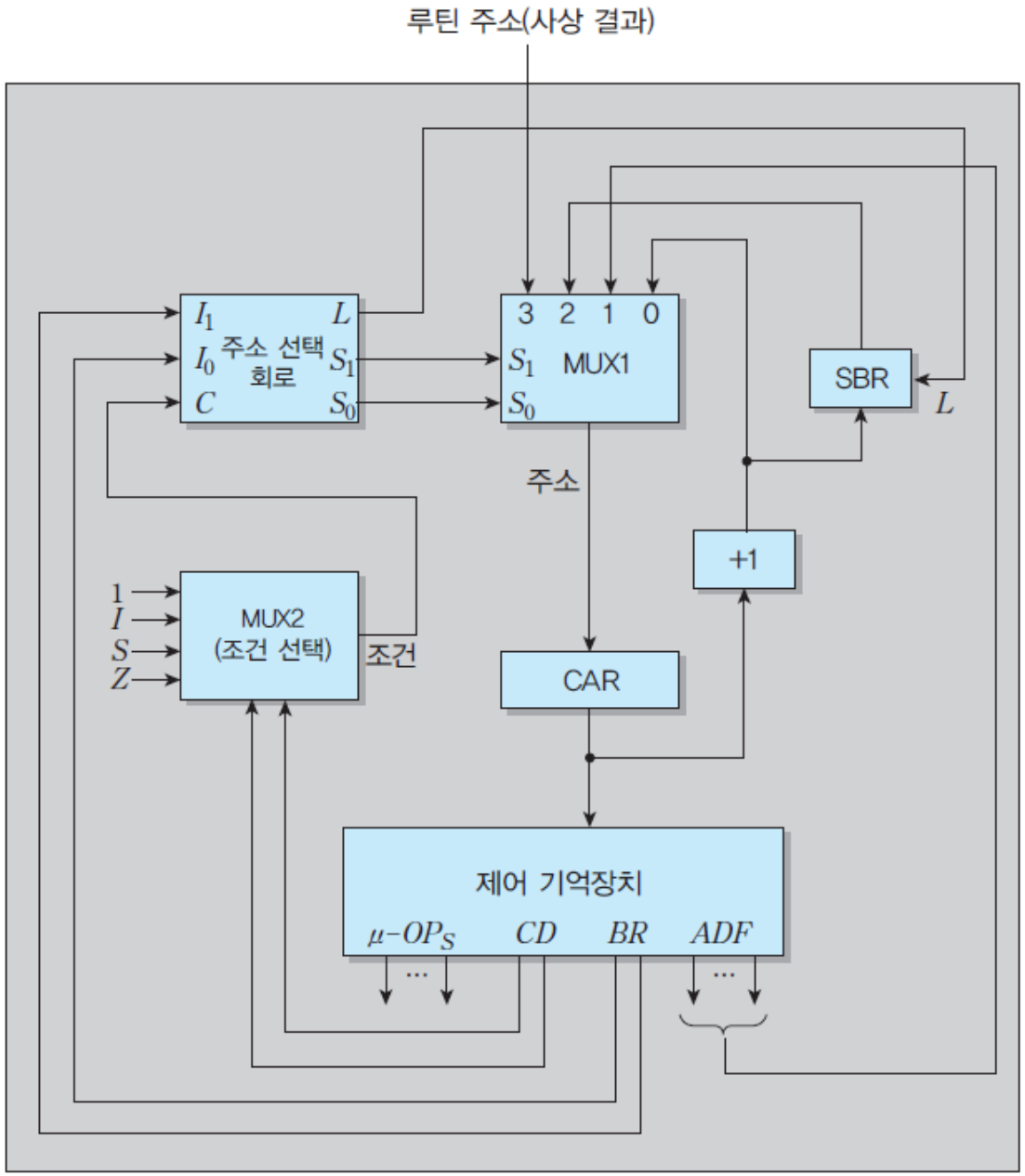
각 명령어에 대한 실행 사이클 루틴들

```
        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)
```

4.5 마이크로프로그램의 순서제어

- 순서제어(sequencing) : 다음에 실행할 마이크로명령어의 주소 결정
- CAR의 초기값 = 0 (인출 사이클 루틴의 첫 번째 마이크로명령어의 주소)
- MUX1 : 다음에 실행할 마이크로명령어의 주소 선택
- MUX2 : 조건 플래그를 선택하여 주소선택 회로로 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

(1) 수직적 마이크로프로그래밍

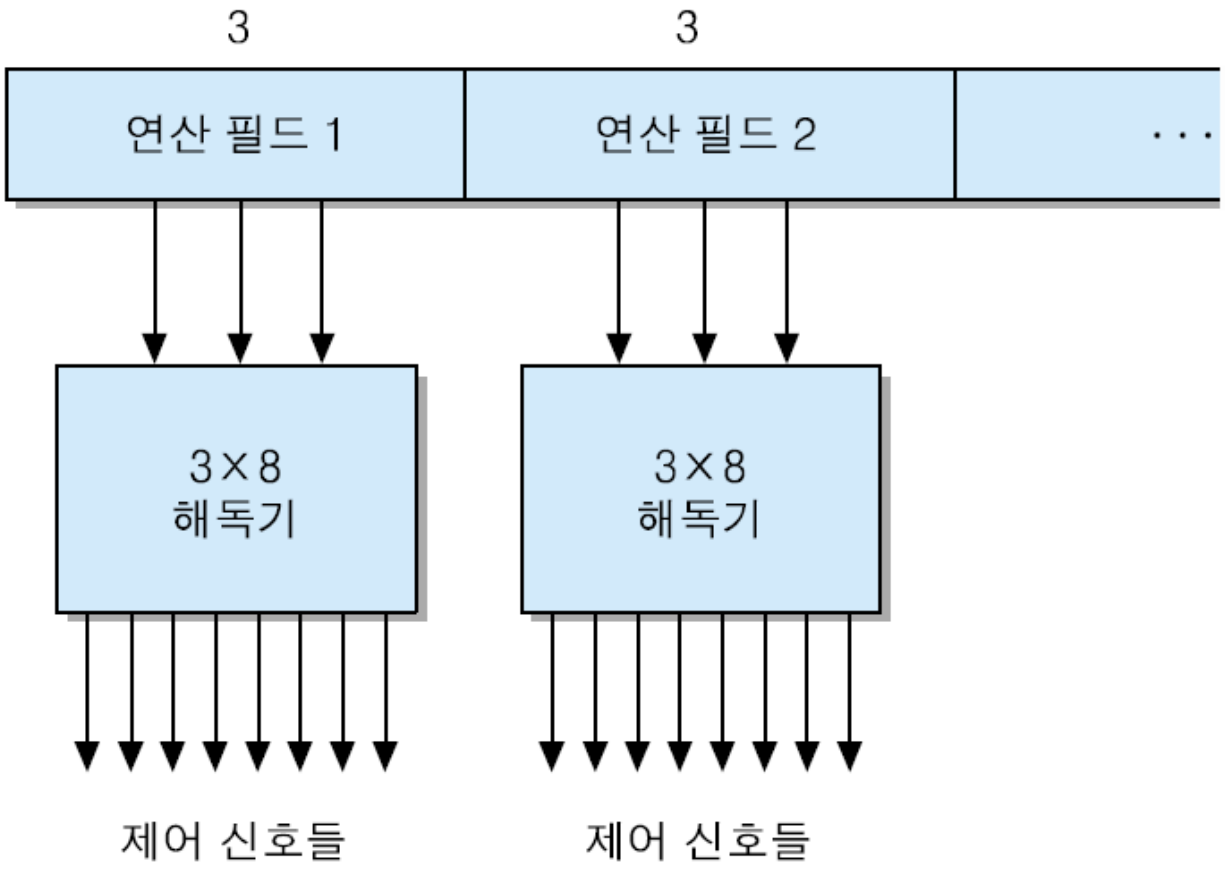
- Vertical microprogramming
- 마이크로명령어의 연산 필드에 적은 수의 코드화된 비트들(encoded bits)을 포함시키고, 해독기를 이용하여 그 비트들을 필요한 수 만큼의 제어 신호들로 확장하는 방식

[장점] 마이크로명령어의 길이(비트 수) 최소화

→ 제어 기억장치 용량 감소

[단점] 해독 동작에 걸리는 만큼의 지연 시간 발생

수직적 마이크로프로그래밍에서의 제어 신호 발생 방법



(2) 수평적 마이크로프로그래밍

- Horizontal microprogramming
- 연산 필드의 각 비트와 제어 신호를 1:1로 대응시키는 방식
- 필요한 제어 신호 수만큼의 비트들로 이루어진 연산필드 비트들이 마이크로명령어에 포함되어야 함

[장점] 하드웨어가 간단하고, 해독에 따른 지연 시간이 없음

[단점] 마이크로명령어의 비트 수가 길어지기 때문에 제어 기억장치의 용량이 증가

들을 접속하여 원하는 수만큼의 제어 신호로 확장하는 방식을 **수직적 마이크로프로그래밍(vertical microprogramming)**이라고 한다. 그리고 이 방식에서 사용되는 마이크로명령어는 **수직적 마이크로명령어(vertical microinstruction)**라고 부른다. 이 방식은 마이크로명령어의 길이가 짧기 때문에 제어 기억장치의 용량이 적게 필요하다는 장점이 있지만, 해독기를 통과하는데 걸리는 시간만큼 지연이 발생한다는 단점이 있다.

반면에 마이크로명령어 내 연산 필드들의 각 비트와 제어 신호를 일대일로 대응시켜서, 그 수만큼의 비트들로 이루어진 마이크로명령어들을 사용하는 방식을 **수평적 마이크로프로그래밍(horizontal microprogramming)**이라고 한다. 이 방식을 이용하면 제어 기억장치로부터 인출된 마이크로명령어의 마이크로-연산 비트들이 해독기를 통과할 필요 없이 직접 제어 신호로 사용될 수 있기 때문에, 하드웨어가 간단하고 해독에 따른 시간 지연이 없다는 장점이 있다. 그러나 연산 필드의 비트 수가 필요한 제어 신호들의 수만큼 제공되어야 하기 때문에 마이크로명령어의 길이가 증가하여, 제어 기억장치의 용량이 커진다는 단점이 있다.

기 본 문 제

4.1 제어 유니트에서 다음에 실행할 마이크로명령어의 주소를 가지고 있는 레지스터는 어느 것인가?

가. CBR　　나. CAR
다. IR　　라. PC

4.2 제어 기억장치는 다음 중의 어느 기억장치 소자를 이용하여 구현되는가?

가. RAM　　나. CAM
다. ROM　　라. DISK

4.3 다음 중에서 서브루틴 레지스터(SBR)가 사용되지 않는 마이크로-연산은 어느 것인가?

가. 호출(CALL)　　나. 복귀(RET)
다. 점프(JUMP)　　라. 조건부 호출(conditional CALL)

4.4 용량이 128 단어인 제어 기억장치의 처음 절반 부분에 실행 사이클 루틴들을 저장한다고 하자. 각 루틴을 최대 8개의 마이크로명령어로 구성될 수 있도록 하려면, 사상 함수는 다음 중 어느 것과 같아야 하는가?

가. 0xxx000　　나. 0xxxx00
다. 1xxxx000　　라. 1xxx000

4.5 조건부 호출 마이크로-연산이 실행될 때 조건이 만족되었다면, CAR에는 어떤 값이 적재되는가?

가. CAR ← CAR+1　　나. CAR ← ADF
다. CAR ← SBR　　라. CAR ← SBR+1

4.6 복귀(RET) 마이크로-연산이 실행되면 CAR에는 어떤 값이 적재되는가?

가. CAR ← CAR+1　　나. CAR ← ADF
다. CAR ← SBR　　라. CAR ← SBR+1

4.7 다음 중 CAR에 적재될 수 없는 값은 어느 것인가?

가. SBR의 내용
나. 사상(mapping)의 결과값
다. 연산 필드 비트들
라. 주소 필드(ADF)의 값

4.8 두 개의 연산 필드가 각각 4비트씩으로 구성되어 있다. 수평적 마이크로프로그래밍 방식이 사용되는 경우라면 최대 몇 개의 제어 신호들이 발생될 수 있는가?

가. 4개
나. 8개
다. 16개
라. 32개

4.9 두 개의 연산 필드가 각각 4비트씩으로 구성되어 있다. 수직적 마이크로프로그래밍 방식이 사용되는 경우라면 최대 몇 개의 제어 신호들이 발생될 수 있는가? 단, 해독기는 두 개만 사용한다고 가정한다.

가. 8개
나. 16개
다. 32개
라. 64개

4.10 다음 중에서 수평적 마이크로프로그래밍의 장점에 해당하는 것은?

가. 마이크로명령어의 길이가 짧아진다.
나. 제어 기억장치의 용량이 줄어든다.
다. 제어 신호의 수를 확장시키는 것이 용이하다.
라. 제어 신호의 발생을 위한 추가적 하드웨어가 필요하지 않다.

연 습 문 제

4.1 그림 4-2의 제어 기억장치의 용량을 256 단어로 확장하고, 각 실행 사이클 루틴이 최대 8개의 마이크로-연산들로 구성될 수 있도록 하려면, 그림 4-3의 사상 함수는 어떻게 바뀌어야 하는가?

4.2 제어 기억장치의 전체 용량이 128 단어이고, 실행 사이클 루틴들이 처음 절반 부분에 저장된다고 하자. 각 루틴이 최대 네 개씩의 마이크로명령어들로 구성된다면, 그림 4-3의 사상 함수는 어떻게 변경되어야 하는가?

4.3 제어 유니트에서 명령어 실행 사이클 루틴들을 제어 기억장치의 절반 하반부에 저장하고자 한다. 각 루틴은 최대 8개의 마이크로명령어들로 구성될 수 있도록 하며, 연산 코드는 5비트이다.
(1) 명령어 해독을 위한 사상 함수를 제시하라.
(2) 제어 기억장치의 0번지부터 상반부에는 인출 사이클을 비롯한 공통 루틴들을 저장한다면, 제어 기억장치의 전체 용량은 몇 단어(word)가 되어야 하는가?

4.4 [표 4-1], [표 4-2] 및 [표 4-3]을 이용하여 그림 4-5의 마이크로명령어들을 2진 비트 패턴으로 변환하라. 만약 필요하다면, [표 4-1]에 새로운 마이크로-연산을 추가하여 사용하라.

4.5 각각 세 비트씩으로 구성된 두 개의 연산 필드들로부터 64개의 제어 신호들을 발생하는 방법을 제안하라. [힌트: 다단계 해독 회로 사용]

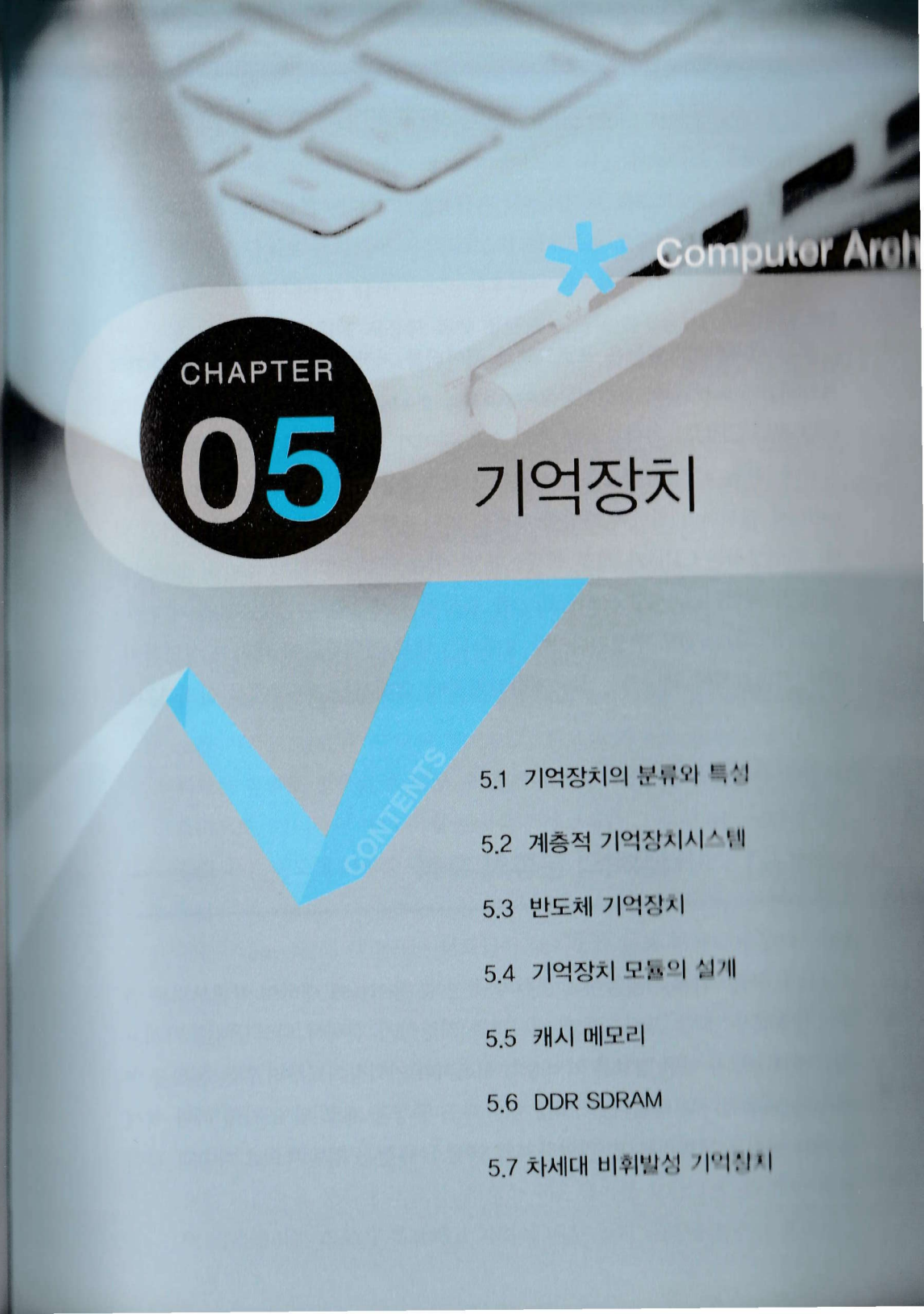
4.6 조건분기 명령어 'JNZ *addr*'는 Z 플래그가 '1'이 아니면 *addr* 번지로 점프하라는 것이다. 점프 목적지 주소(*addr*)는 I 비트에 따라 직접 혹은 간접 주소지정 된다. 이 명령어를 위한 실행 사이클 루틴을 그림 4-5와 같은 형식으로 작성하라.

4.7 수직적 마이크로명령어 형식에서 연산-코드 필드가 9비트이다. 지정해야 할 마이크로-연산들의 수가 46개이고 동시에 두 개 이상의 마이크로-연산들이 수행될 수 있도록 하려면, 연산-코드 필드를 어떤 서브필드(subfield)들로 몇 비트씩 나누면 되는가?

4.8 수직적 마이크로프로그래밍과 수평적 마이크로프로그래밍의 장점과 단점을 비교하라.

4.9 어떤 제어 기억장치의 폭(단어 길이)이 26비트이다. 마이크로명령어 형식에서 14비트는 마이크로-연산을 가리키는 연산 필드로 사용된다. 주소 선택 필드는 분기 조건을 규정하며, 그 조건을 결정하는 플래그들은 8개이다. 그리고 분기의 종류는 한 가지뿐이어서 분기 필드는 필요하지 않다고 가정한다.

(1) 주소 선택 필드는 몇 비트가 필요한가?
(2) 주소 필드(ADF)는 몇 비트로 구성될 수 있는가?
(3) 제어 기억장치의 최대 용량(단어 수 × 폭)을 구하라.



제 4 장

4.1 1 X X X X 0 0 0

4.2 0 X X X X 0 0

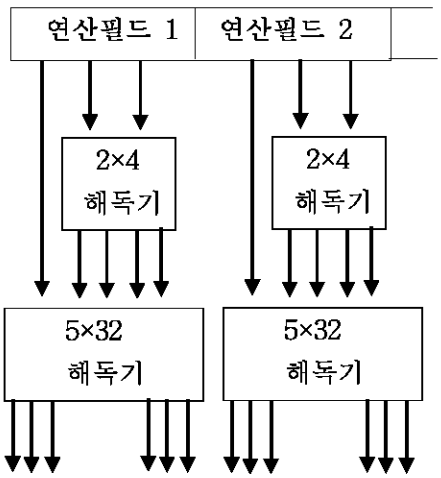
4.3 (1) 1 X X X X X 0 0 0 (2) 512 단어

4.4

	주소	μ-ops	CD	BR	ADF
NOP :	1000000	001 000	00	00	1000001
LOAD :	1000100	000 000	01	01	0000100
	1000101	010 000	00	00	1000110
	1000110	100 000	00	00	1000111
	1000111	101 000	00	00	0000000
STORE :	1001000	000 000	01	01	0000100
	1001001	010 000	00	00	1001010
	1001010	000 010	00	10	1001011
	1001011	111 000	00	00	0000000
ADD :	1001100	010 000	00	00	1001101
	1001101	100 000	00	00	1001010
	1001010	011 000	00	00	0000000
SUB :	1010000	010 000	00	00	1010001
	1010001	100 000	00	00	1010010
	1010010	000 110	00	00	0000000
JUMP :	1010100	000 111	00	00	0000000

(단, [표4-1](b)의 마지막 항목을 IRTPC로 변경)

4.5



. . . . . .

32개의 제어신호 32개의 제어신호

4.6 ORG 88

JNZ : IRTPC Z I JMP INDRT ; PC←IR(addr)

단, [표 4-2]에서 AC ≠ 0 일 때, Z 플래그가 1로 세트되는 것으로 가정한다.

4.7 연산-코드 필드 1 : 5비트 → 최대 32개 연산 지정 가능
연산-코드 필드 2 : 4비트 → 최대 16개 연산 지정 가능
결과적으로, 최대 48개의 연산 지정이 가능.

4.8 수직적 마이크로프로그래밍
장점 : 마이크로명령어 내에 적은 수의 코드화된 비트들을 포함시킴으로써 제어 기억장치의 용량을 줄일 수 있다.
단점 : 외부에 해독기들을 접속하여 하드웨어가 복잡해지며, 해독 시간만큼 지연된다.

수평적 마이크로프로그래밍
장점 : 제어 기억장치로부터 인출된 마이크로명령어의 마이크로-연산 비트들이 해독기를 통과할 필요 없이 직접 제어 신호로 사용될 수 있기 때문에, 하드웨어가 간단하고 해독에 따른 시간지연이 없다.
단점 : 각 마이크로명령어의 길이(비트 수)가 증가하기 때문에 제어 기억장치의 용량이 커진다.

4.9 (1) 2^3 = 8이므로 세 비트가 필요
(2) 주소필드-(연산필드+조건필드+분기필드)
26 - (14 + 3 + 0) = 9 비트
(3) $2^9 \times 26 = 512 \times 26$ 비트