[JO2122] 컴퓨터구조

2022년 1학기

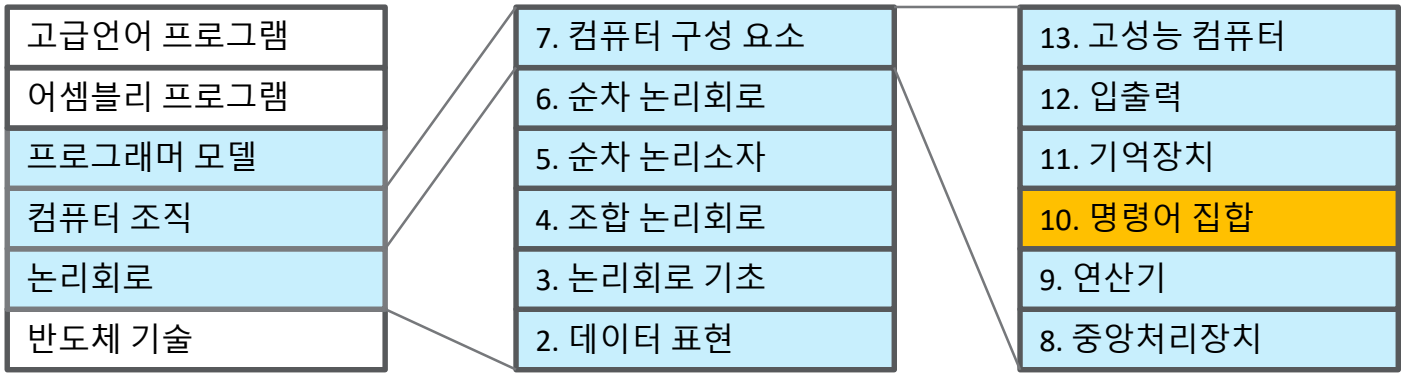
상명대학교 소프트웨어학과 박희민

10.1 명령어 특성
10.2 주소의 수
10.3 주소지정방식
10.4 오퍼랜드 저장
10.5 명령어 종류
10.6 요약

2022-05-18

CHAP10 명령어집합

제10장 명령어 집합



- 학습 목표
  - 오퍼랜드 필드가 지정하는 유효 데이터를 구할 수 있다.
  - 기억장치에 저장되어 있는 단어의 값을 해석할 수 있다.
- 내용
  - 10.1 명령어 특성
  - 10.2 주소의 수
  - 10.3 주소지정방식
  - 10.4 오퍼랜드 저장
  - 10.5 명령어 종류
  - 10.6 요약

10.1 명령어 특성

- 명령어
  - 하드웨어와 소프트웨어를 연결하는 요소
- 학습목표
  - 명령어 구조를 이해한다.
  - 2진수로 명령어를 표현하는 방법을 이해한다.
- 내용

10.1.1 명령어 구성 요소

10.1.2 명령어 형식: 명령어 구성 요소 배치 방법

10.1.1 명령어 구성 요소

- 명령어 종류
  - 데이터 전달 명령어: A ← B
  - 데이터 처리 명령어: A ← B op C
  - 프로그램 제어 명령어: PC ← 목적지 주소
- 명령어 구성 요소: 동작코드 + 오퍼랜드
- 동작코드(opcode, operation code)
  - 중앙처리장치가 실행할 동작을 2진수로 표현한 코드
  - 니모닉 코드(mnemonic code): 의미를 나타내는 기호 코드
  - 예) ADD, SUB, MUL, DIV, BR, PUSH, POP
- 오퍼랜드(operand)
  - 동작의 대상: 소스 오퍼랜드, 목적지 오퍼랜드, 분기목적지 주소
  - 실제: 레지스터 번호, 기억장치 주소, 입출력 포트

10.1.2 명령어 형식

- 명령어 형식
  - 명령어 비트 영역에 명령어 구성 요소를 배치한 형태
  - 동작 코드 필드, 오퍼랜드 필드

- 프로세서의 명령어 형식: 고정 길이/가변 길이 명령어 형식
- 고정 길이 (fixed-length) 명령어 형식
  - 명령어의 길이가 모두 같다.
  - 해석이 쉽고, 제어장치가 간단하다.
  - RISC (Reduced Instruction Set Computer)
- 가변 길이 (variable-length) 명령어 형식
  - 명령어 종류별로 길이가 다르다.
  - 해석이 어렵고, 제어장치가 복잡하다.
  - CISC (Complex Instruction Set Computer)

PowerPC 명령어 형식

6비트	5비트	5비트	16비트	
branch	long immediate			A/L
cond. br.	options	CR Bit	displacement	A/L

(a) 분기 명령어

cond. reg.	dest. bit	src. bit	src. bit	add, or, xor, etc.	X

(b) 조건부 레지스터 논리 명령어

ld/st indirect	dest. reg.	base reg.	displacement		
ld/st indirect	dest. reg.	base reg.	index reg.	size, sign, update	X

(c) 적재/저장 명령어

arithmetic	dest. reg.	src. reg.	src. reg.	sub-opcode
arithmetic	dest. reg.	src. reg.	immediate value	

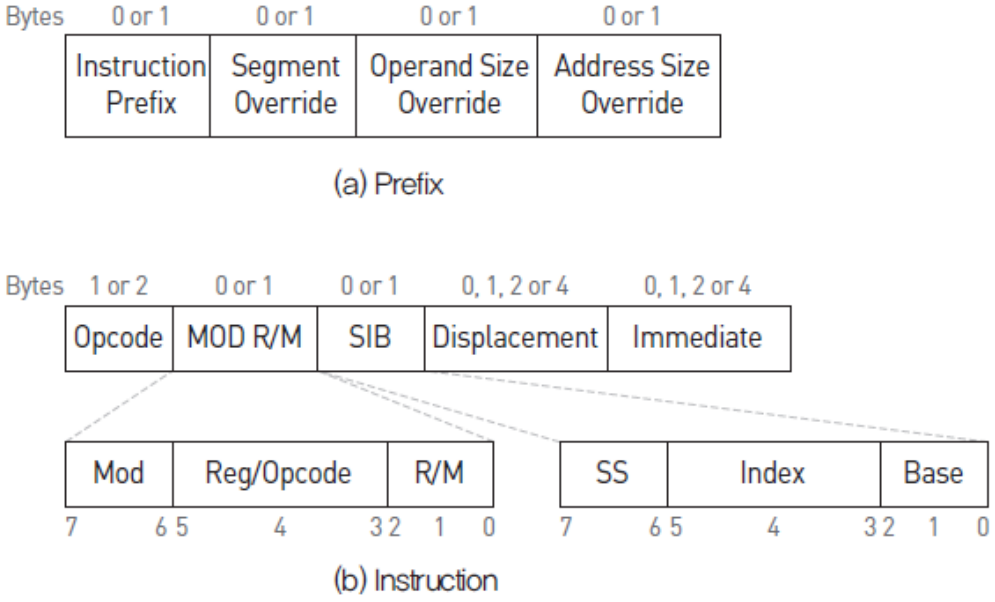
(d) 산술 연산 명령어

float sgl/dbl	dest. reg.	src. reg.	src. reg.	src. reg.	sub-op	R

(e) 실수 연산 명령어

[그림 10-1] PowerPC 프로세서의 명령어 형식

Pentium 명령어 형식



[그림 10-2] Pentium 프로세서의 명령어 형식

10.2 주소의 수

- 명령어 종류 별 오퍼랜드
  - 데이터 전달 명령어: 목적지, 소스
  - 데이터 처리 명령어: 목적지, 소스1, 소스2
  - 프로그램 제어 명령어: 분기 목적지
- 학습목표
  - 명령어에 표현된 오퍼랜드를 해석할 수 있다.
- 내용

10.2.1 3-주소 명령어 형식 (3-address instruction format)

10.2.2 2-주소 명령어 형식 (2-address instruction format)

10.2.3 1-주소 명령어 형식 (1-address instruction format)

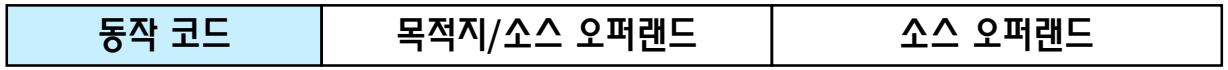
10.2.4 0-주소 명령어 형식 (0-address instruction format) : 스택 연산

10.2.1 3-주소 명령어 형식

동작 코드	목적지 오퍼랜드	소스 오퍼랜드1	소스 오퍼랜드2

- 목적지 ← 소스1 op 소스2
- Y = A×B + C×D
  - MUL R1, A, B // R1 ← A × B
  - MUL R2, C, D // R2 ← C × D
  - ADD Y, R1, R2 // Y ← R1 + R2
- 특징
  - 오퍼랜드를 3개 모두 표현한다.
  - 명령어 길이가 길어진다.
  - 프로그램을 구현하는 명령어 수가 적다.
  - RISC 프로세서에서 주로 사용한다.

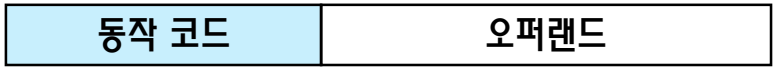
10.2.2 2-주소 명령어 형식



- 목적지 ← 목적지 op 소스
- Y = A×B + C×D
  - LD R1, A // R1 ← A
  - MUL R1, B // R1 ← R1 x B
  - LD R2, C // R2 ← C
  - MUL R2, D // R2 ← R2 x D
  - ADD R1, R2 // R1 ← R1 + R2
  - ST Y, R1 // Y ← R1

- 특징
  - 소스 오퍼랜드 하나를 생략한다.
  - 명령어 길이가 짧아진다.
  - 프로그램을 구현하는 명령어 수가 증가한다.
  - CISC 프로세서에서 주로 사용한다.

10.2.3 1-주소 명령어 형식



- AC ← AC op 오퍼런드
- Y = A×B + C×D
  - LD A // AC ← A
  - MUL B // AC ← AC × B
  - ST T // T ← AC
  - LD C // AC ← C
  - MUL D // AC ← AC × D
  - ADD T // AC ← AC + T
  - ST Y // Y ← AC

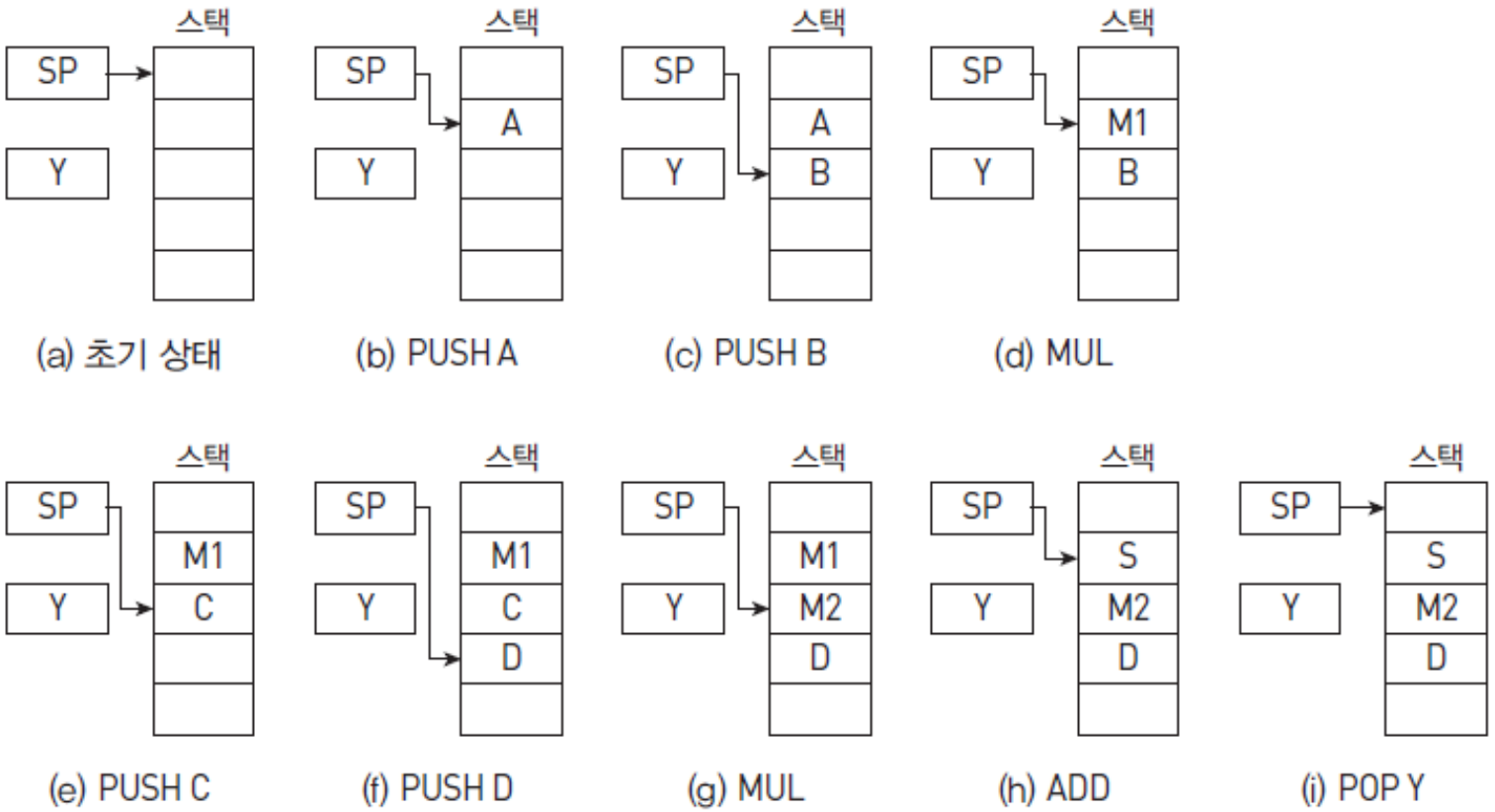
- 특징
  - 누산기(AC)를 사용한다.
  - 명령어 표현에서 누산기를 생략한다.
  - 명령어의 길이가 짧다.
  - 프로그램을 구현하는 명령어의 수가 가장 많다.
  - 초기 컴퓨터에서 사용하였으나, 지금은 찾아보기 힘들다.

10.2.4 0-주소 명령어 형식

- 특수 계산기에서 사용하는 명령어 형식
- 스택 연산
  - 계산식 표현
  - 후위 표기식: $Y = A \times B + C \times D \Rightarrow (Y = AB \times CD \times +)$
- 계산 과정
  - 변수: PUSH Data // 데이터를 푸시
  - 연산자: POP R1 // 두 개의 데이터를 팝하고
    POP R2
    R3 ← R1 op R2 // 계산 후
    PUSH R3 // 결과를 푸시
  - 계산 끝: POP Y // 계산 결과를 변수에 저장

스택 연산

- 후위 표기식: (Y = AB×CD×+)



[그림 10-6] 스택 연산

10.1 주소의 수 요약

- 3-주소 명령어 형식: C ← A op B, 오퍼랜드 3개 표현
- 2-주소 명령어 형식: A ← A op B, 오퍼랜드 표현 생략
- 1-주소 명령어 형식: AC ← AC op A, 누산기 표현 생략
- 0-주소 명령어 형식: 스택 연산

10.3 주소지정방식

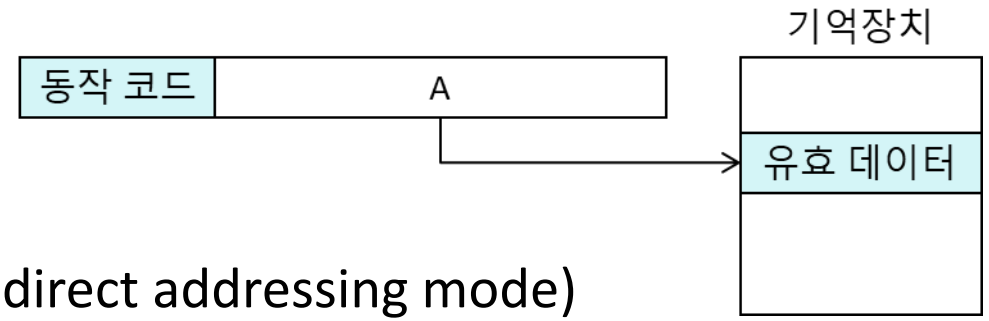
- 용어
  - 오퍼랜드 필드: 명령어에 표현되어 있는 연산의 대상
  - 유효 데이터(effective data): 실제로 처리되는 데이터
  - 유효 주소(effective address): 유효 데이터가 저장되어 있는 기억장치 주소
- 학습 목표
  - 명령어의 오퍼랜드 필드에서 유효 데이터를 찾을 수 있다.
- 내용

10.3.1 즉치 주소지정방식(immediate addressing mode)
10.3.2 직접 주소지정방식(direct addressing mode)
10.3.3 간접 주소지정방식(indirect addressing mode)
10.3.4 레지스터 주소지정방식(register addressing mode)
10.3.5 레지스터 간접 주소지정방식(register indirect addressing mode)
10.3.6 변위 주소지정방식(displacement addressing mode)

10.3.1 즉치 주소지정방식

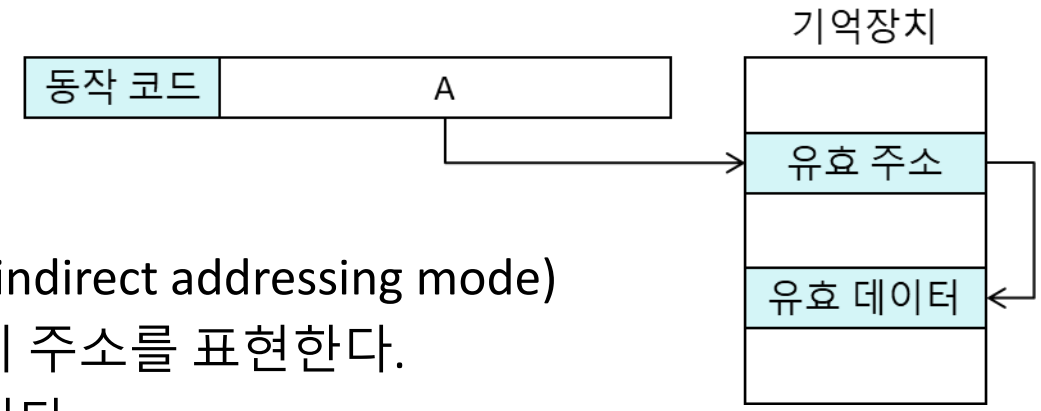
- 즉치 데이터(immediate data)
  - 명령어에 유효 데이터 포함
  - 명령어를 실행할 때 기억장치를 액세스할 필요가 없다.
  - 유효 데이터의 표현 범위가 오퍼랜드 필드 크기에 제한 받는다.
- 명령어 예
  - 1-주소지정방식 ADDI #20 // AC ← AC + 20
  - 2-주소지정방식 LDI R0, #12 // R0 ← 12
- [예제 10-1]
  - 명령어 길이가 16비트이다.
  - 동작코드를 표현하는데 5비트를 사용하고
  - 나머지를 즉치 데이터(immediate data)를 표현하는데 사용한다.
  - 데이터 필드를 부호 없는 수로 해석할 때 수의 표현 범위는?

10.3.2 직접 주소지정방식



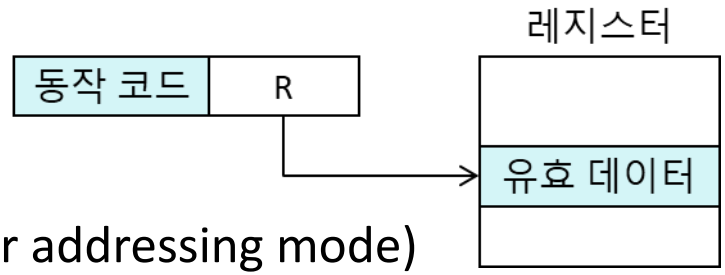
- 직접 주소지정방식 (direct addressing mode)
  - 명령어에 기억장치 주소(A)가 표현된다.
  - A는 부호 없는 수이다.
  - 유효 주소 = A
- 명령어 예
  - 1-주소지정방식 ADD #20 // AC ← AC + Mem(20)
  - 2-주소지정방식 LD R0, #1200 // R0 ← Mem(1200)
- [예제 10-2]
  - 기억장치 용량이 64KB이고 바이트 단위로 주소를 지정한다.
  - 유효 데이터를 직접 주소지정방식으로 표현하기 위하여
  - 주소 필드에 몇 비트를 할당해야 하는가?

10.3.3 간접 주소지정방식



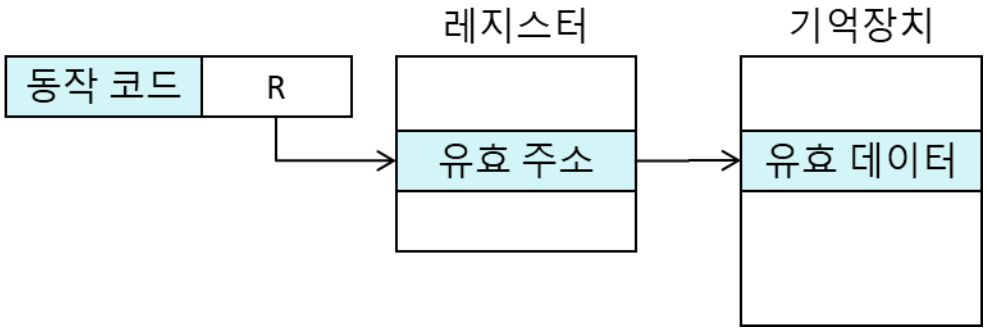
- 간접 주소지정방식(indirect addressing mode)
  - 명령어에 기억장치 주소를 표현한다.
  - A는 부호 없는 수이다.
  - A의 길이는 기억장치 주소의 길이와 같다.
  - 유효 주소 = mem(A)
  - 명령어 실행 단계에서 기억장치를 2번 액세스한다.
    - 유효 주소 인출/유효 데이터 인출
- 명령어 예
  - 1-주소지정방식 ADDID #20 // AC ← AC + Mem(Mem(20))
  - 2-주소지정방식 LDID R0, #1200 // R0 ← Mem(Mem(1200))

10.3.4 레지스터 주소지정방식



- 레지스터 주소지정방식(register addressing mode)
  - 명령어는 레지스터 번호 표현
  - 유효 주소 개념이 없음.
  - R 필드이 길이는 주소(A) 필드 길이보다 많이 짧다.
- 명령어 예
  - 2-주소지정방식 LD R0, #1200 // R0 ← Mem(1200)
  - 2-주소지정방식 ADD R0, R1 // R0 ← R0 + R1
  - 3-주소지정방식 ADD R0, R1, R2 // R0 ← R1 + R2
- [예제 10-3]
  - 레지스터 R0~R7, 8개인 프로세서에서
  - 명령어에 레지스터를 표현하는데 필요한 비트 수는?

10.3.5 레지스터 간접 주소지정방식



- 레지스터 간접 주소지정방식(register indirect addressing mode)
  - 명령어는 레지스터 번호를 표현
  - 유효 주소 = (R)
  - 유효 데이터 = mem((R))
  - 실행단계에서 유효 데이터를 인출하기 위하여 기억장치를 한 번 액세스
  - 명령어의 길이가 짧으면서, 기억장치를 액세스할 수 있다.

레지스터 간접 주소지정방식

- [예제 10-4] 적재 명령어와 저장 명령어의 동작이 다음과 같다.
  - 레지스터 간접 적재 명령어: LDR Rd, Mem(Rs+1:Rs) // Rd ← Mem(Rs+1:Rs)
  - 레지스터 간접 저장 명령어: STR Mem(Rd+1:Rd), Rs // Mem(Rd+1:Rd) ← Rs

적재 명령어는 Rs만 표현하고 Rs+1을 표현하지 않고, 저장 명령어는 Rd만 표현하고 Rd+1을 표현하지 않는다.

예를 들어, 적재 명령어의 Rs=6이면, 기억장치 주소는 (R7:R6)이다.

1) LDR R0, Mem(R7:R6) 명령어를 실행한 후 변하는 레지스터의 값을 구하라.
2) STR Mem(R7:R6), R1 명령어를 실행한 후 변하는 기억장치의 값을 구하라.

5 비트	3 비트	3 비트	3 비트	2 비트
LDR: 00110	Rd	Rs	XXX	00
STR: 00110	Rd	Rs	XXX	01

(a) 명령어 형식

번호	데이터
R7	01
R6	23
...	...
R1	3A
R0	00

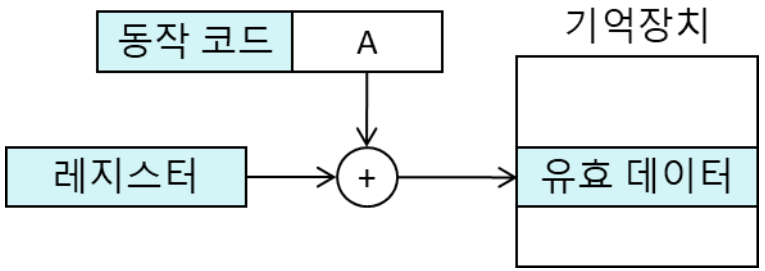
(b) 레지스터 상태

주소	데이터
...	...
125	78
124	AB
123	0D
122	1A

(c) 기억장치 상태

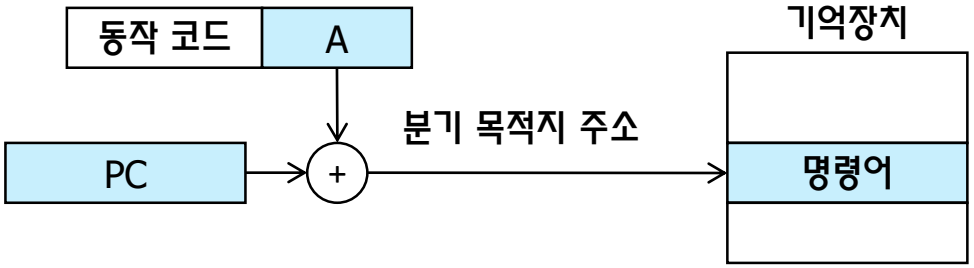
[그림 10-12] 명령어 형식과 상태

10.3.6 변위 주소지정방식



- 유효 주소 = (R) + A // 부호 확장 덧셈
- 부호 확장 덧셈 예
  - 기억장치 주소: 16 비트 (기억장치 공간 64 Kbytes)
  - 레지스터 길이: 16 비트
  - 명령어 주소 필드 A: 12 비트
  - 유효 주소 = $(R)_{(16비트)} + A_{(12비트)}$
- 종류
  - 상대 주소지정방식(relative addressing mode)
  - 베이스 레지스터 주소지정방식(base-register addressing mode)
  - 인덱싱(indexing)

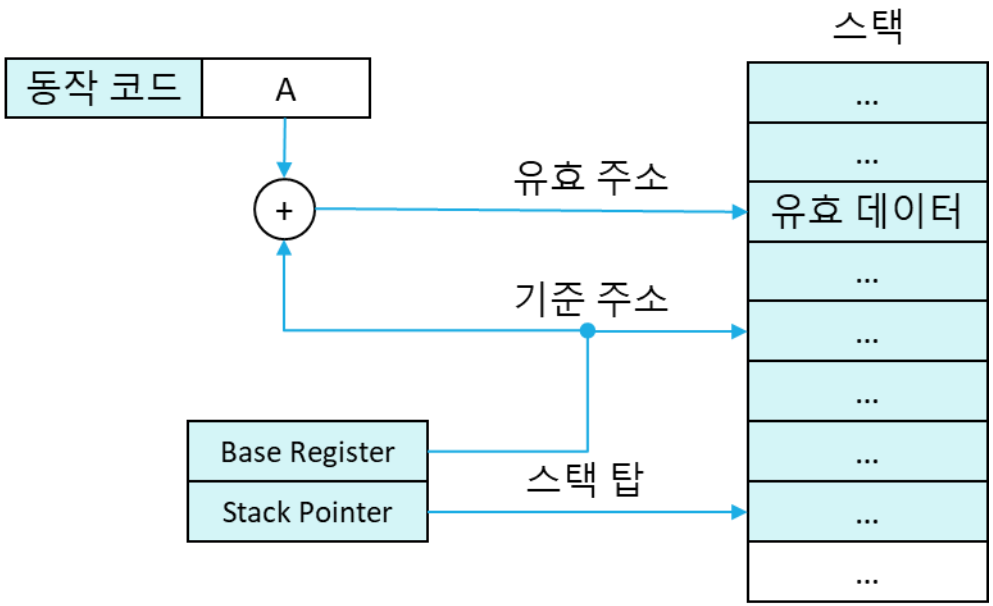
상대 주소지정방식



- 상대 주소지정방식 (relative addressing mode)
  - 분기 목적지 = PC + A (정수, 부호 확장)
  - (조건) 분기 명령어, 서브루틴 호출 명령어
- [예제 10-5]
  - 기억장치 용량: 64 KB, 바이트 단위 주소
  - 명령어 크기 16비트 (2바이트)
  - 현재 프로세서는 204_{10}번지에 있는 무조건 분기(BR) 명령어 실행 중

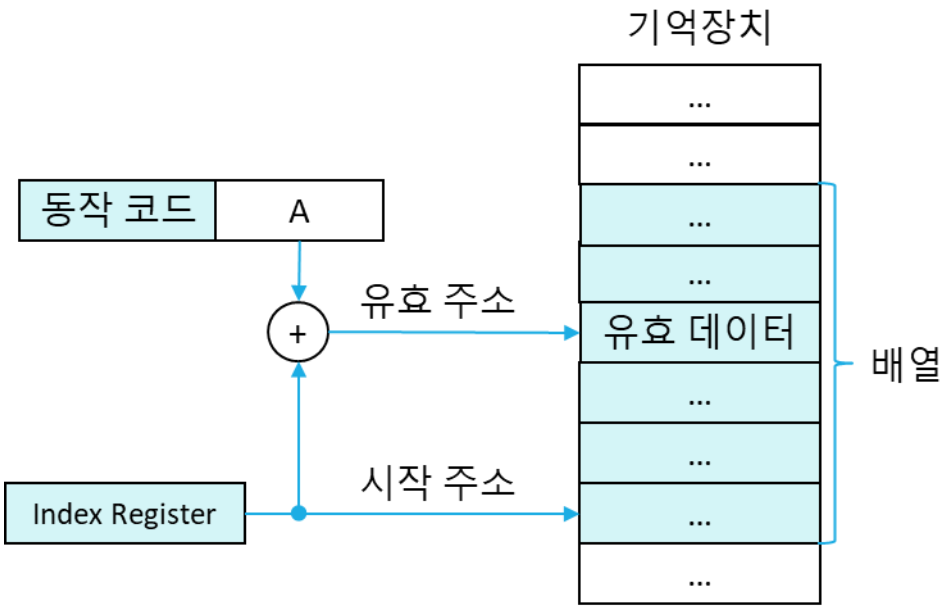
1) 현재 PC의 값은? PC = 다음 명령어의 주소 = 204 + 2 = 206
2) 명령어가 BR + 64일 때 명령어 실행 후 PC는? PC = 206 + 64 = 270
3) 명령어가 BR – 66일 때 명령어 실행 후 PC는? PC = 206 – 66 = 140

베이스 레지스터 주소지정방식



- Base Register: 스택의 기준점 지정
- 유효 주소 = (BR) + A (정수, 부호 확장 계산)
- 기준점의 위 아래 액세스 가능

인덱싱



- 유효 주소 = (Index Register) + A (부호 없는 수, 부호 확장 계산)
- 자동 인덱싱 (auto indexing)
  - Pre-indexing: 인덱스 레지스터 갱신 후 기억장치 액세스
  - Post-indexing: 기억장치 액세스 후 인덱스 레지스터 갱신
- 사용 예: 배열 영역 액세스
  - LD R0, (IX+4)
  - LD R0, (IX+R1)
  - LD R0, (IX++)

10.3 주소지정방식 요약

- 주소지정방식
  - 오퍼랜드 필드에서 유효 데이터를 구하는 방법 정의
- 주소지정방식의 종류
  - 즉치 주소지정방식: 명령어에 유효 데이터
  - 직접 주소지정명식: 명령어에 유효 주소
  - 간접 주소지정방식: 명령어에 유효 주소의 주소
  - 레지스터 주소지정방식: 레지스터에 유효 데이터
  - 레지스터 간접 주소지정방식: 레지스터에 유효 주소
  - 변위 주소지정방식: 유효 주소 = (레지스터) + 명령어의 주소 필드
  - 상대 주소지정방식: 분기 목적지 주소 지정
  - 베이스 레지스터 주소지정방식: 오퍼랜드 = 스택 영역 액세스
  - 인덱싱: 배열 액세스

10.4 오퍼런드 저장

- 오퍼랜드(동작의 대상)의 두 가지 의미
  - 명령어 형식에서 동작의 대상을 표현하는 필드
  - 동작 코드가 실제로 처리하는 유효 데이터
- 학습 목표
  - 여러 바이트로 구성된 단어를 기억장치에 저장하는 방법을 이해한다.
- 내용

10.4.1 데이터 정렬

10.4.2 바이트 저장 순서

10.4.1 데이터 정렬

- 기억장치 구성
  - 기억장치: 바이트 단위 주소 지정
  - 단어(word)는 n바이트에 분산
  - 단어는 Mem[A+0] ~ Mem[A+(n-1)]번지에 연속적으로 저장되어 있음.

- 단어 크기에 정렬 (word-size aligned)
  - n의 배수로 시작하는 주소에 배정
  - 단어가 배정된 시작 주소 A는 n의 배수 (단, $n=2^k$)

- 단어 액세스 방법
  - 정렬된 n바이트 단어를 한 번에 액세스할 수 있다.
  - 정렬되지 않은 단어는 두 번에 걸쳐 읽은 후 조합해야 한다.

단어 크기에 정렬

- 4GB 용량, 4바이트 단어

A[31:2]	$A_1A_0 = 11_2$	$A_1A_0 = 10_2$	$A_1A_0 = 01_2$	$A_1A_0 = 00_2$
0000_0000h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_0004h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_0008h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_000ch	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_0010h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_0014h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_0018h	D[31:24]	D[23:16]	D[15:8]	D[7:0]
0000_001ch	D[31:24]	D[23:16]	D[15:8]	D[7:0]
…	D[31:0] …			

정렬된 단어

정렬되지 않은 단어

- [예제 10-6] 프로세서가 한 번에 액세스 할 수 있는가?

1) 주소 0000_0004h, 0000_0005h, 0000_0006h, 0000_0007h 번지 단어
2) 주소 1000_000eh, 1000_000fh, 1000_0010h, 1000_0011h 번지 단어
3) 주소 1234_1e08h, 1234_1e09h, 1234_1e0ah, 1234_1e0bh 번지 단어
4) 주소 b800_a003h, b800_a004h, b800_a005h, b800_a006h 번지 단어

10.4.2 바이트 저장 순서

- 바이트 순서(byte order)
  - 한 개의 단어가 한 바이트 이상의 기억장치 공간을 차지할 때,
  - 단어를 구성하는 바이트를 주어진 공간에 배치하는 방법

12345678		A_1A_0 = 11	A_1A_0 = 10	A_1A_0 = 01	A_1A_0 = 00
	little endian	12	34	56	78
레지스터	big endian	78	56	34	12

- Little endian
  - 주소 값이 작은 장소에 무게가 작은 수 저장
  - 예) Intel X86, ARM
- Big endian
  - 주소 값이 작은 장소에 무게가 큰 수 저장
  - 예) Motorola 68XXX, MIPS, 인터넷

바이트 저장 순서 예제

- [예제 10-7] 그림을 보고 질문에 답하라.
  1) 주소 1200h부터 1203h까지 데이터를 리틀 엔디언으로 해석하라.
  2) 주소 1204h부터 1207h까지 데이터를 빅 엔디언으로 해석하라.

주소	데이터
1203h	55
1202h	44
1201h	33
1200h	22

Little endian?

주소	데이터
1207h	66
1206h	77
1205h	88
1204h	99

Big endian?

- [풀이]
  1) 주소 값이 작은 1200h에 무게가 작은 수: _____
  2) 주소 값이 작은 1204h에 무게가 큰 수: _____

10.4 오퍼랜드 저장 요약

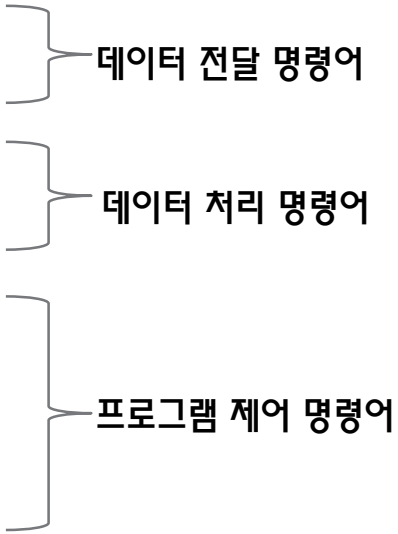
- 단어 크기에 정렬 (aligned to word size)
  - 단어를 기억장치 시작 주소가 단어 크기의 배수인 곳에 저장
  - 한 번에 액세스 가능

- 저장 방법
  - 리틀 엔디언: 주소 값이 작은 곳에 무게가 작은 바이트 저장
  - 빅 엔디언: 주소 값이 작은 곳에 무게가 큰 바이트 저장
  - 표준이 정해지지 않음.

10.5 명령어 종류

- 학습 목표
  - 프로세서가 제공하는 명령어의 동작을 설명할 수 있다.
- 내용

10.5.1 데이터 전달 명령어
10.5.2 입출력 명령어
10.5.3 산술 연산 명령어
10.5.4 논리 연산 명령어
10.5.5 상태 레지스터 조작 명령어
10.5.6 분기 명령어
10.5.7 서브루틴 호출 명령어
10.5.8 인터럽트 명령어
10.5.9 시스템 제어 명령어



10.5.1 데이터 전달 명령어

- 적재(load): Reg ← Mem
- 저장(store): Mem ← Reg
- 이동(move): {Reg, Mem} ← {Reg, Mem}
- 교환(exchange): {Reg or Mem} ↔ {Reg or Mem}
- 스택: PUSH REG, POP REG

10.5.2 입출력 명령어

- 입력(input): Reg ← I/O port
- 출력(output): I/O port ← Reg

- 일반적으로
  - 직접주소지정방식 또는 레지스터 간접 주소지정방식을 사용한다.
  - 복잡한 주소지정방식을 사용하지 않는다.

10.5.3 산술 명령어

- 단항 연산 명령어
  - 절대값(absolute)
  - 음수(negate)
  - 증가(increment)
  - 감소(decrement)
  - 제곱근(square root)
- 이항 연산 명령어
  - 더하기
  - 빼기
  - 곱하기
  - 나누기

- 데이터 형식: 부호 없는 수, 정수, 실수, 이진화 십진 코드(BCD)
- 데이터 크기: 8, 16, 32, 64비트

10.5.4 논리 연산 명령어

- 논리 이항 연산 명령어
  - AND, OR, XOR
  - NAND, NOR, XNOR

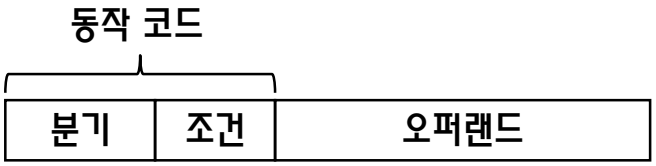
- 논리 단항 연산 명령어
  - NOT
  - Clear bit
  - Set bit
  - Shift: 논리 시프트, 산술 시프트, 회전

10.5.5 상태 레지스터 조작 명령어

- Compare 명령어 compare A, B // A – B
- Test 명령어 test A, B // A AND B

- Set/reset carry flag 명령어
  - stc (set carry flag)
  - clc (clear carry flag)
- Set/reset interrupt flag 명령어
  - sti (set interrupt flag)
  - cli (reset interrupt flag)

10.5.6 분기 명령어



- 오퍼랜드 = 분기 목적지 주소
  - 직접 주소지정방식
  - 레지스터 간접 주소지정방식
  - 상대 주소지정방식
- 종류
  - 무조건 분기 명령어: br / jmp
  - 조건 분기 명령어
    - Branch taken : PC ← 분기 목적지 주소
    - Branch not taken : PC ← 다음 명령어 (변하지 않음)

조건 분기 예

- [예제 10-8]
  - 상태 레지스터 값이 다음과 같다. 다음 명령어를 실행한 후, PC 값을 구하라.

    Z = 0, S = 0, C = 1, P = 0, OV = 1
  - 명령어 길이는 2바이트이고, 현재 실행하는 명령어는 1000번지에 저장되어 있고, 상대 주소지정방식을 사용한다.

1) brz +100 // branch if Z = 1
2) brnz +100 // branch if Z = 0
3) brc +24 // branch if C = 1

- [풀이]

1) branch (taken, not taken) ? PC = _____
2) branch (taken, not taken) ? PC = _____
3) branch (taken, not taken) ? PC = _____

10.5.7 서브루틴 호출 명령어

(a) 서브루틴 호출 명령어

(b) 복귀 명령어

- call [서브루틴 시작 주소]
  - 복귀 주소 저장
  - 스택 동작
    - PUSH PC
    - PC ← 서브루틴 시작 주소
- ret
  - PC 복구
  - 스택 동작
    - POP PC

10.5.8 인터럽트 명령어

(a) 인터럽트 요청 명령어

(b) 복귀 명령어

- 소프트웨어 인터럽트
  - System call
- int n
  - PUSH PC
  - PUSH SR
  - PC ← ISR 시작 주소

- 인터럽트 복귀 명령어
- iret
  - POP SR
  - POP PC

10.5.9 시스템 제어 명령어

- 시스템 관리, 운영체제 기능 지원 명령어
  - Halt　　시스템 정지
  - Wait　　이벤트 대기
  - Nop　　클럭 소모

- 프로세서 동작 모드(operation mode)
  - 시스템 모드(system mode, 커널 모드, 운영체제 모드)
  - 사용자 모드(user mode)

- 특권 명령어(privileged instruction)
  - 시스템 모드에서만 실행할 수 있는 명령어
  - 특수 레지스터 관리, 기억장치 관리, 입출력 명령어 등

10.6 요약

- 10.1 명령어 특성
  - 동작 코드 + 오퍼런드
- 10.2 주소의 수
  - 3-주소, 2-주소, 1-주소, 0-주소 명령어 형식
- 10.3 주소지정방식
  - 즉치, 직접, 간접, 레지스터, 레지스터 간접 주소지정방식
  - 상대, 베이스 레지스터, 인덱싱 주소지정방식
- 10.4 오퍼랜드 저장
  - 데이터 정렬(word-size aligned), 바이트 저장 순서
- 10.5 명령어 종류
  - 데이터 전달, 입출력
  - 산술 연산, 논리 연산,
  - 상태 레지스터 조작, 분기 및 조건분기, 서브루틴 호출 및 복귀, 인터럽트
  - 시스템 제어 명령어