Slide 0

8장. 중앙 처리 장치

개요

- 중앙 처리 장치 (Central Processing Unit:CPU)
  - 모든 데이타 처리를 수행함
  - CPU 의 구성 (그림 8-1)

Slide 1

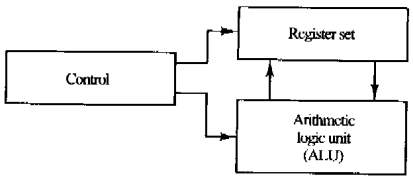


그림 8-1 CPU의 주요소

201

범용 레지스터 구조

Slide 2

- 일시적인 계산 결과를 저장해 놓는곳
- 메모리는 속도가 느림으로 범용레지스터는 많을 수록 좋음
- 7개의 범용 레지스터를 갖는 CPU 구조 (그림 8-2)

범용 레지스터 구조 (계속)

Slide 3

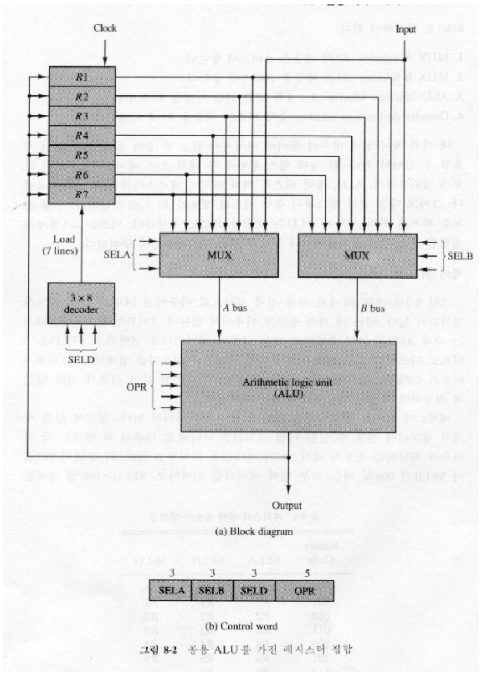


그림 8-2 공용 ALU를 가진 레지스터 집합

범용 레지스터 구조 (계속)

Slide 4

- 2개의 MUX를 이용한 2개의 레지스터 선택
- ALU 의 기능 선택을 통한 연산
- 3 × 8 decoder 를 이용한 결과저장 레지스터 선택
- 예)

  $\boxed{R1 \leftarrow R2 + R3}$

  1. MUX A selector: R2 의 내용을 A 버스에 놓는다
  2. MUX B selector: R3 의 내용을 B 버스에 놓는다
  3. ALU function selector : 덧셈 기능을 선택한다
  4. Decoder destination selector: 출력버스의 내용을 R1 에 전달한다

• 제어 워드

- 제어워드의 구성 (그림 8-2 (b))
- 제어워드의 레지스터 선택 인코딩 (표 8-1)

범용 레지스터 구조 (계속)

Slide 5

표 8-1 레지스터 선택 필드의 인코딩

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

- 제어워드의 연산 인코딩 (표 8-2)

범용 레지스터 구조 (계속)

Slide 6

표 8-2 ALU 연산의 인코딩

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

- 마이크로 연산의 예
  - R1 ←R2 - R3

범용 레지스터 구조 (계속)

Slide 7

필드 :	SELA	SELB	SELD	OPR
기호 :	R2	R3	R1	SUB
제어워드 :	010	011	001	00101

- 몇가지 마이크로 연산예 (표 8-3)

Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
$R1 \leftarrow R2 \quad R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output ← Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

스택 구조

Slide 8

- 스택이란 ?
  - LIFO (last in first out) 구조의 메모리
  - push 동작 : 스택에 새로운 항목을 넣음
  - pop 동작 : 스택의 맨위의 항목을 빼냄
- 레지스터 스택
  - 64워드의 레지스터 스택의 구조 (그림 8-3)

스택 구조 (계속)

Slide 9

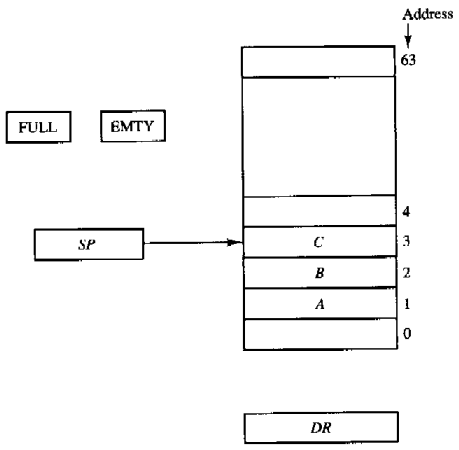


그림 8-3 메모리 스택의 블럭도

  - SP (stack pointer) : 스택의 top을 가리킴

스택 구조 (계속)

Slide 10

- FULL = 1 스택이 꽉 차있음을 0 이면 꽉차지 않음을 표시
- EMPTY = 1 스택이 비었음을 표시 0 이면 비어있지 않음을 표시
- 초기상태
  - SP = 0
  - EMPTY = 1
  - FULL = 0
- push 동작의 기술

SP ←SP + 1	SP 증가
M[SP] ←DR	스택의 top 에 데이타 저장
if(SP=0) then (FULL ←1)	스택이 full 임을 표시
EMPTY ←0	스택이 empty 가 아님을 표시

- pop 동작의 기술

스택 구조 (계속)

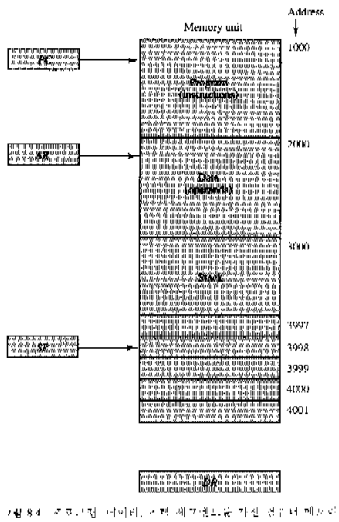
Slide 11

DR ←M[SP]	스택으로부터 데이타 읽음
SP ←SP - 1	SP 를 하나 감소
if(SP=0) then (EMPTY ←1)	스택이 empty 임을 표시
FULL ←0	스택이 full 이 아님을 표시

- 메모리 스택
  - RAM 메모리를 이용 스택을 구현 (그림 8-4)

스택 구조 (계속)

Slide 12



스택 구조 (계속)

Slide 13

- push 동작

  SP ←SP - 1
  M[SP] ←DR

- pop 동작

  DR ←M[SP]
  SP ←SP + 1

- 대부분의 컴퓨터는 full 과 empty 를 검사하는 하드웨어가 없음

  소프트웨어로 검사

- 역 polish 표기
  - 수식의 표현시 infix 표현 방법 → A * B + C * D
  - 수식의 세가지 표현 방법

스택 구조 (계속)

Slide 14

A + B	infix 개념
+ AB	prefix 또는 polish 개념
AB +	postfix 또는 reverse polish 개념

- 역 polish 개념은 스택으로 구현하기 적합한 형태이다

  A*B + C*D →AB* CD* +

- 역 polish 로의 변환
  - (A + B) * [C * (D + E) + F]
    →AB + DE + C * F + *

- 산술식의 계산
  - 역 polish 형태로 변환
  - 피연산자는 스택에 push
  - 연산자가 나타나면

스택 구조 (계속)

Slide 15

1. 스택의 top 에 있는 두개의 값을 이용 연산
2. 스택의 값이 pop 되고 연산결과를 top 아래의 피연산자 위치에 저장

- (3*4) + (5*6) 의 계산
- 역 polish 로 변환 →34* 56* + →(그림 8-5)

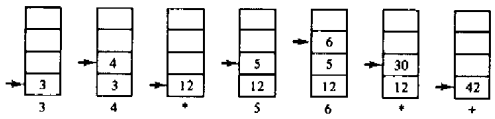


그림 8-5 3*4+5*6을 계산하는 스택 동작

명령어 형식

- 명령어 코드의 필드
  1. 수행해야할 연산을 명시한 연산 코드 필드
  2. 메모리의 주소나 레지스터를 지정하는 주소 필드
  3. 피연산자나 유효 주소가 결정되는 방법을 나타내는 모드 필드
- 명령어 주소 필드의 구성

Slide 16

  - 주소필드는 CPU 내부 레지스터 구성형식에 따라 좌우된다
  - CPU 내부 레지스터의 구성 형식
    1. 단일 누산기 구조
    2. 범용 레지스터 구조
    3. 스택 구조
  - 단일 누산기 구조에서의 주소필드 구성
    * 모든 명령어의 수행은 누산기 레지스터를 이용
    * 예) ADD X → AC ← AC + M[X]
  - 범용 레지스터 구조에서의 주소필드 구성

명령어 형식 (계속)

    * 레지스터이용
    * 여러가지 예)
      1. ADD R1, R2, R3 → R1 ← R2 + R3
      2. ADD R1, R2 → R1 ← R1 + R2
      3. MOV R1, R2 → R1 ← R2
      4. ADD R1, X → R1 ← R1 + M[X]

Slide 17

  - 스택 구조에서의 주소필드 구성
    * 스택 이용
    * 예)
      1. PUSH X → M[SP-1] ← M[X]
      2. ADD → M[SP+1] ← M[SP] + M[SP+1]
  - 대부분의 컴퓨터는 세가지중 하나사용, 몇몇 컴퓨터는 복합적으로 사용
- 주소필드의 숫자가 프로그램에 미치는 영향
  - 다음의 명령어를 0, 1, 2, 3 개의 주소명령어로 구현

명령어 형식 (계속)

Slide 18

X = (A + B) * (C + D)

- 3-주소 명령어

```
ADD R1, A, B     R1 ←M[A] + M[B]
ADD R2, C, D     R2 ←M[C] + M[D]
MUL X, R1, R2    M[X] ←R1 * R2
```

- 3-주소 명령어는 수식 계산할때 프로그램의 길이를 짧게 함
- 2-주소 명령어

명령어 형식 (계속)

Slide 19

```
MOV R1, A     R1 ←M[A]
ADD R1, B     R1 ←R1 + M[B]
MOV R2, C     R2 ←M[C]
ADD R2, D     R2 ←R2 + M[D]
MUL R1, R2    R1 ←R1 * R2
MOV X, R1     M[X] ←R1
```

- 1-주소 명령어 (누산기에 의해 이루어짐)

명령어 형식 (계속)

Slide 20

LOAD A	AC ←M[A]
ADD B	AC ←AC + M[B]
STORE T	M[T] ←AC
LOAD C	AC ←M[C]
ADD D	AC ←AC + M[D]
MUL T	AC ←AC * M[T]
STORE X	M[X] ←AC

- 무주소 명령어 (스택에의해 이루어짐)

명령어 형식 (계속)

Slide 21

PUSH A	TOS ←A
PUSH B	TOS ←B
ADD	TOS ←(A + B)
PUSH C	TOS ←C
PUSH D	TOS ←D
ADD	TOS ←(C + D)
MUL	TOS ←(C + D) * (A + B)
POP X	M[X] ←TOS

- RISC 명령
  - 전형적인 RISC 프로세서는 load, store 외의 모든 명령어가 레지스터참조 명령어임

명령어 형식 (계속)

Slide 22

LOAD R1, A	R1 ←M[A]
LOAD R2, B	R2 ←M[B]
LOAD R3, C	R3 ←M[C]
LOAD R4, D	R4 ←M[D]
ADD R1, R1, R2	R1 ←R1 + R2
ADD R3, R3, R4	R3 ←R3 + R4
MUL R1, R1, R3	R1 ←R1 * R3
STORE X, R1	M[X] ←R1

어드레싱 모드

Slide 23

- 실제 피연산자의 위치는 어드레싱 모드에 따라 정해짐
- 어드레싱 모드 사용의 장점
  1. 포인터, 카운터 인덱싱, 프로그램 relocation 등의 편의 제공
  2. 명령어의 주소필드의 비트를 줄여줌
- 어드레싱 모드의 종류
  - implied 모드
    - 피연산자의 주소가 묵시적으로 정해짐
    - 예) 누산기를 사용하는 명령어들, 스택 명령어들
  - immediate 모드
    - 피연산자가 명령어 자체에 있음
    - 주소필드에 피연자가가 들어있음
  - 레지스터 모드
    - CPU 내의 레지스터에 피연산자가 있음

어드레싱 모드 (계속)

  * 주소필드에는 특정 레지스터를 선택하는 정보가 들어감
- 레지스터 간접 모드 (register indirect mode)
  * CPU 내의 레지스터에 피연산자의 주소가 있음
  * 주소필드에는 특정 레지스터를 선택하는 정보가 들어감
- 자동증가 또는 자동 감소 모드
  * 레지스터 간접 모드로 동작

Slide 24

  * 레지스터 액세스시 레지스터 값이 자동 감소 혹은 자동 증가됨
  * 연속적인 메모리 액세스시 편리
- 직접 주소 모드 (direct address mode)
  * 명령어의 주소부분에 해당하는 메모리에 피연산자가 있음
- 간접 주소 모드 (indirect address mode)
  * 명령어의 주소부분에 해당하는 메모리에 피연산자의 주소가 있음
- 상대 주소 모드 (relative address mode)
  * 명령어의 주소부분의 값과 PC 를 더한결과의 주소에 피연산자가 있음
  * 가까운 곳으로 분기시 편리

어드레싱 모드 (계속)

- 인덱스드 어드레싱 모드 (indexed addressing mode)
  * 명령어의 주소부분의 값과 인덱스 레지스터값을 더한 결과의 주소에 피연산자가 있음
  * 인덱스 레지스터 값 변경하여 액세스
  * 배열 액세스시 편리

Slide 25

- 베이스 레지스터 어드레싱 모드 (base register addressing mode)
  * 명령어의 주소부분의 값과 베이스 레지스터값을 더한 결과의 주소에 피연산자가 있음
  * 인덱스 어드레싱과 유사 사용시 차이
  * 명령어의 주소부분의 값 변경하여 액세스
  * 인덱스 레지스터 →명령어주소에 대하여 상대적위치를 가짐
  * 베이스 레지스터 →명령어 주소의 베이스 주소를 가짐
- 실제 예) →(그림 8-7 및 표 8-4)

어드레싱 모드 (계속)

Slide 26

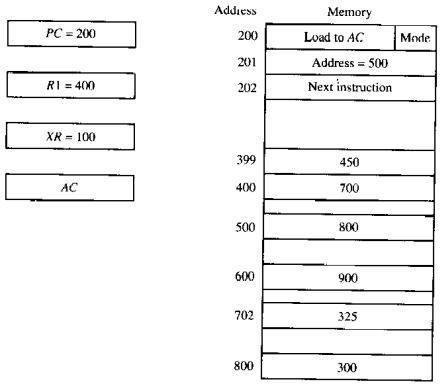


그림 8-7 어드레싱 모드에 대한 실제 예

어드레싱 모드 (계속)

Slide 27

8.6 데이타 전송과 처리 — 225

표 8-4 실제 예의 결과표

Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

데이타 전송과 처리

Slide 28

- 대부분의 컴퓨터가 지니고 있는 기본적인 명령어 집합
  1. 데이타 전송 명령어들
  2. 데이타 처리 명령어들
  3. 프로그램 제어 명령어들
- 데이타 전송 명령어
  - 데이타 변경없이 한장소에서 다른 장소로 위치를 옮김
    - 메모리와 레지스터사이
    - 레지스터와 입출력 장치사이
    - 레지스터와 레지스터사이
  - 8개의 데이타 전송명령어 (표 8-5)

데이타 전송과 처리 (계속)

Slide 29

. 저장(store) 명령어는 레지스터에서 메모리로의 정보 전송을 나타낸다. **move** 령어는 레지스터간의 정보 전송을, **exchange** 명령어는 두 개의 레지스터 사이

표 8-5 대표적인 데이타 전송 명령어

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

  - 전송명령어는 어드레싱모드와 연관적으로 사용됨 (표 8-6)

데이타 전송과 처리 (계속)

Slide 30

8.6 데이타 전송과 처리 — 227

표 8-6 로드 명령에 대한 8개의 어드레싱 모드

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

- 데이타 처리 명령어
  - 데이타 처리 명령어의 분류
    1. 산술 명령어
    2. 논리 및 비트 처리 명령어
    3. 시프트 명령어
  - 산술 명령어 (표 8-7)

데이타 전송과 처리 (계속)

Slide 31

표 8-7 대표적인 산술 명령어

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

* 데이타 형에따라 여러가지 구분이 가능
* 예)

데이타 전송과 처리 (계속)

ADDI	두개의 이진 정수를 덧셈
ADDF	두개의 부동 소숫점 덧셈
ADDD	두개의 십진수 (BCD) 덧셈

– 논리 연산 및 비트 처리 명령어 (표 8-8)

Slide 32

표 8-8 대표적인 논리와 비트 처리 명령어

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

데이타 전송과 처리 (계속)

– 시프트 명령어 (표 8-9)

Slide 33

표 8-9 대표적인 시프트 명령어

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

프로그램 제어

Slide 34

- 프로그램의 수행순서 제어 명령어들 (표 8-10)

표 8-10 대표적인 프로그램 제어 명령어

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

- 상태 비트 조건
  - 조건 코드 (condition code) 혹은 플래그 (flag) 로 불림

프로그램 제어 (계속)

Slide 35

- 상태 레지스터 비트 (그림 8-8)

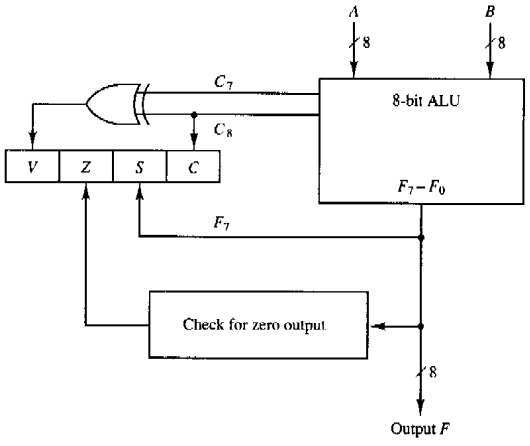


그림 8-8 상태 레지스터 비트

* C (carry bit) : 출력 캐리를 저장

프로그램 제어 (계속)

Slide 36

  * S (sign bit) : 연산결과의 부호를 저장
  * Z (zero bit) : 연산결과가 0 이면 1로 됨
  * V (overflow bit) : 연산결과 overflow 발생하면 1 로됨
- overflow
  * msb 와 msb -1 비트의 exclusive OR 가 1 이면 발생
  * 연산결과가 잘못되었음을 표시

- 조건부 분기 명령어
  - 일반적인 분기 명령어 (표 8-11)

프로그램 제어 (계속)

Slide 37

표 8-11 조건부 분기 명령어

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions ($A - B$)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions ($A - B$)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

프로그램 제어 (계속)

Slide 38

- 상태 비트를 이용하여 분기 여부 결정

• 서브루틴 call 과 return

- 분기명령어: call, jump to subroutine, branch to subroutine, branch and save address 등
- 수행
  1. 현재의 PC 를 메모리에 기억시켜놓음 (돌아올 주소 보관)
  2. 서브루틴으로 점프
- 리턴 주소의 저장 (컴퓨터 마다 다름)
  * 레지스터
  * 특정위치의 메모리
  * 서브루틴의 첫번째 위치
  * 스택 (가장 효율적)
- 서브루틴 call 의 마이크로 연산

프로그램 제어 (계속)

Slide 39

SP ←SP + 1	스택포인터 증가
M[SP] ←PC	PC 의 내용을 스택에 저장
PC ←유효주소	서브루틴 수행

- 리턴의 마이크로 연산

PC ←M[SP]	스택의 top 값을 PC 로 옮김
SP ←SP - 1	SP 감소

• 프로그램 인터럽트

- 정상적인 프로그램의 진행을 벗어나는 여러가지 문제를 다루는 기법
- CPU 내부 외부의 요인에 의하여 발생
- 서비스 프로그램 수행후 원래의 프로그램으로 복귀

프로그램 제어 (계속)

Slide 40

- 서브루틴 call 과 인터럽트의 차이점
  1. 소프트 인터럽트를 제외하고는 모두 내 외부의 신호에의하여 발생
  2. 서비스 프로그램의 주소는 하드웨어에 의하여 결정됨
  3. 인터럽트시 PC 이외의 정보도 대피시킴
- 인터럽트 복귀시 인터럽트 발생 바로 전의 상태로 복귀 하기 위하여 필요한 정보
  1. PC 의 내용
  2. 모든 레지스터의 내용
  3. 상태 조건의 내용
- 모든 상태 비트의 집합을 PSW (program status word) 라고도함
- 인터럽트의 형태
  1. 외부 인터럽트 : 입출력장치, 타이밍 장치, 전원등에서 발생
  2. 내부 인터럽트 : 불법 명령어, 오버플로우, 0으로 나누가, 스택 overflow 등에서 발생
  3. 소프트웨어 인터럽트 : 명령어로 발생

간소화된 명령어 집합 컴퓨터 (RISC)

Slide 41

- CISC (complex instruction set computer)
  - VLSI 기술의 발달로 복잡한 명령어를 하드웨어로 구성
  - 특징
    1. 많은수의 명령어 : 일반적으로 100에서 250개의 명령어
    2. 몇몇 자주사용되지 않는 특별한 명령어
    3. 다양한 어드레싱 모드 : 일반적으로 5 에서 20 가지의 모드
    4. 가변 길이 명령어 형식 (디코딩이 복잡)
    5. 대부분이 memory reference 명령어
    6. microprogrammed 제어
- RISC (reduced instruction set computer)
  - 복잡한 명령어 보다 간단한 명령어만 구현
  - 대신 내부 레지스터의 갯수를 늘림
  - 특징
    1. 적은 수의 명령어

간소화된 명령어 집합 컴퓨터 (RISC) (계속)

2. 적은 수의 어드레싱 모드
3. 대부분이 register reference 명령어들
4. 모든 동작은 CPU 내부에서 수행됨
5. 고정길이의 명령어 (디코딩이 간단)
6. 단일 사이클의 명령어 실행
7. 하드와이어드 제어

Slide 42

– RISC 구조의 다른 특징

1. 많은 레지스터
2. 프로시져의 호출과 복귀의 속도 향상을 위한 중첩된 레지스터 윈도우
3. 효과적인 명령어 파이프라인
4. 고급언어를 효과적으로 기계어로 번역하는 컴파일러

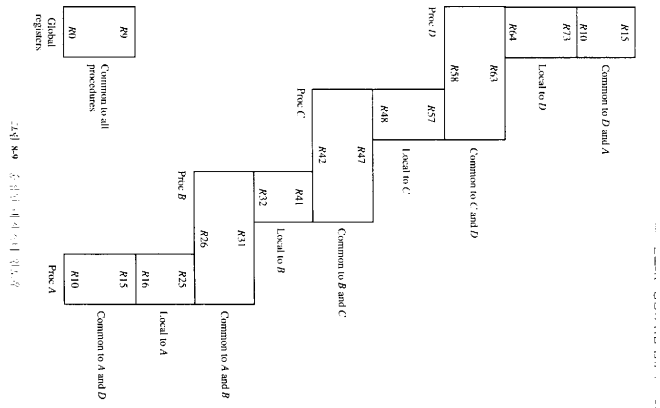
– 중첩된 레지스터 윈도우

* 프로시져 호출과 복귀시 복잡한 스택 동작을 줄이기 위함
* 레지스터 화일내에 중첩된 레지스터 윈도우를 둠
* 각 procedure 마다 레지스터 윈도우를 할당
* 파라메터 전달및 return을 위한 다음 procedure 와의 공통 레지스터가 있음

간소화된 명령어 집합 컴퓨터 (RISC) (계속)

* 중첩된 레지스터 윈도우 (그림 8-9)

Slide 43



– 레지스터 윈도우 구조

간소화된 명령어 집합 컴퓨터 (RISC) (계속)

Slide 44

전역 레지스터 수 = G
각 윈도우에서 지역 레지스터수 = L
두 윈도우에 공통인 레지스터수 = C
윈도우 수 = W

각 윈도우 크기 = L + 2C + G

총 레지스터 수 = (L + C)W + G

Homework

Slide 45

- 7장 연습문제
  - 3, 5, 7, 14, 21
- 8장 연습문제
  - 3, 5, 9, 14, 21, 27, 34