Slide 0

9장. 파이프라인과 벡터 처리

병렬처리

Slide 1

- 병렬처리란?
  - 처리 속도 향상을 목적으로 동시에 여러 데이타를 처리하는 기술
  - 병렬 처리의 여러 단계
    - 레지스터 레벨의 병렬성: 직렬 시프트 레지스터 →병렬 로드를 갖는 레지스터
    - CPU 기능 레벨의 병렬성: 다른 동작을 동시에 수행하는 여러개의 장치를 둠 (그림 9-1)

병렬처리 (계속)

Slide 2

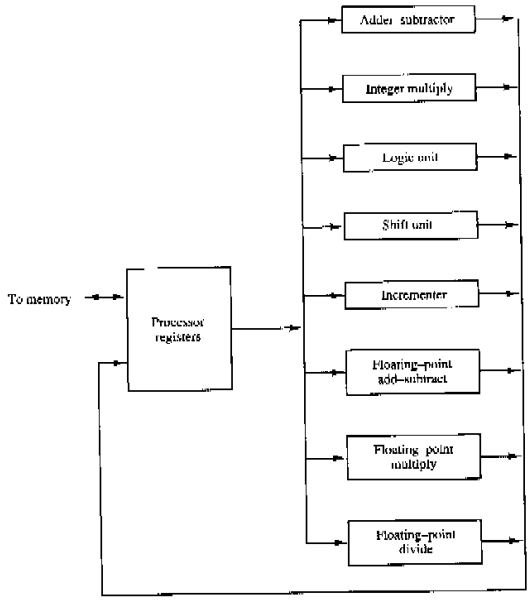


그림 9-1 다중 기능 장치를 가진 프로세서

병렬처리 (계속)

Slide 3

* 프로세서 레벨의 병렬성: 여러개의 프로세서를 두고 동시에 수행
* 시스템 레벨의 병렬성: 여러 시스템이 동시에 수행하면서 통신을 통하여 정보전달

– M. J. Flynn 의 컴퓨터 분류
* 명령어 흐름 (instruction stream) 과 데이타 흐름 (data stream) 을 이용
* 4가지 분류

단일 명령어 흐름, 단일 데이타 흐름 (SISD)
단일 명령어 흐름, 다중 데이타 흐름 (SIMD)
다중 명령어 흐름, 단일 데이타 흐름 (MISD)
다중 명령어 흐름, 다중 데이타 흐름 (MIMD)

– SISD 구조
* 제어장치, 처리장치, 메모리장치 를 갖는 단일 컴퓨터 구조
* 병렬처리는 다중 기능 장치나 파이프 라인 처리에 의하여 구현

– SIMD 구조

병렬처리 (계속)

Slide 4

  * 공통의 제어장치 여러개의 처리 장치를 갖는 컴퓨터 구조
  * 모든 프로세서는 동일한 명령어에 대하여 다른 데이타 항목을 실행
  * 모든 프로세서가 동시에 메모리 접근할수 있는 장치 필요

- MISD 구조
  * 이론적으로만 연구가 됨
- MIMD 구조
  * 여러프로그램을 동시에 수행하는 컴퓨터 시스템
  * 대부분의 다중 프로세서와 다중 컴퓨터 시스템

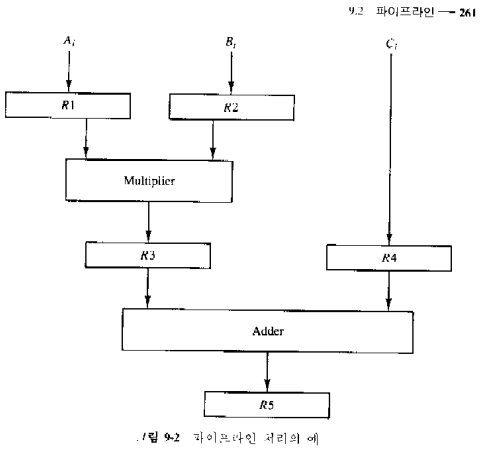
파이프 라인

Slide 5

- 파이프라인 이란
  - 하나의 프로세스를 여러 서브 프로세스로 나누어
  - 각 서브 프로세서가 동시에 다른 데이타를 처리하게 하는 기법
  - 모든 서브 프로세서를 통과하면 최종적인 연산 결과를 얻음
  - 각 단계에서 레지스터를 두어 중간 결과를 보관
  - 파이프라인 처리의 예 (그림 9-2)

파이프 라인 (계속)

Slide 6



* 수식 : $\boxed{A_i \times B_i + C_i}$
* 각 서브 프로세서에서의 부연산

파이프 라인 (계속)

Slide 7

R1 $\leftarrow A_i$, R2 $\leftarrow B_i$	A_i 와 B_i 의 입력
R3 $\leftarrow$R1 $\times$ R2, R4 $\leftarrow C_i$	곱셈과 C_i 의 입력
R5 $\leftarrow$R3 + R4	곱셈 결과와 C_i의 덧셈

* 매 클럭마다 레지스터 값의 변화 (표 9-1)

파이프 라인 (계속)

Slide 8

표 9-1 파이프라인 예에서 레지스터의 내용

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

- 일반적인 고찰
  - 네 세그먼트를 갖는 파이프라인 (그림 9-3)

파이프 라인 (계속)

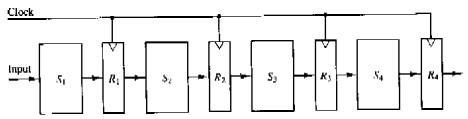


그림 9-3 네 세그먼트 파이프라인

Slide 9

- 네 세그먼트를 갖는 파이프라인의 공간-시간 (space-time) 표 (그림 9-4)

9.2 파이프라인 ··· 263

Segment:	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

Clock cycles

그림 9-4 파이프라인에 대한 공간-시간표

- k 개의 세그먼트 파이프라인에서 n 개의 태스크 수행시 필요한 시간
  - 첫번째 태스크 $\rightarrow kt_p$ clock 필요

파이프 라인 (계속)

Slide 10

  - * 나머지 태스크 $\rightarrow t_p$ clock 필요
  - * 그러므로 n 개의 태스크 수행시 필요한 clock $\rightarrow (k + (n-1))t_p$
- 비파이프라인 장치에서 $\rightarrow nt_n$
- 파이프라인의 속도 증가율 $\rightarrow \boxed{S = \frac{nt_n}{(k+n-1)t_p}}$
- n 이 $k-1$ 에 비해 클경우 속도증가율 $\rightarrow \boxed{S = \frac{t_n}{t_p}}$
- 비파이프라인과 파이프라인에서 태스크 하나 처리시간이 같다면 $\rightarrow t_n = kt_p$
- 결국 속도증가율은 $\rightarrow \boxed{S = \frac{kt_p}{t_p} = k}$
- 처리 속도 증가율은 세그먼트의 갯수 k 와 같음
- k 개의 병렬 비파이프라인 다중장치와 (그림 9-5) k 세그먼트 파이프라인의 성능이 거의 같음

파이프 라인 (계속)

Slide 11

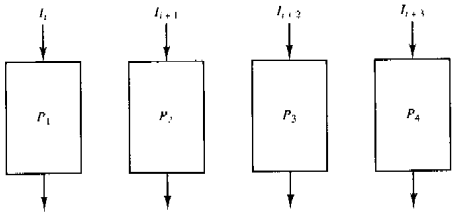


그림 9-5 병렬적인 다중 기능 장치

- 실제적으로 이론적인 최대 성능은 얻을수 없음
  - * 각 부연산의 수행시간이 다르다
  - * 클럭 사이클이 최대 전파지연시간을 갖는 부연산에 맞춰져야함
  - * 중간값을 저장하는 레지스터에서의 지연시간이 있다
- 파이프라인의 종류
  - * 산술 파이프 라인 : 산술 연산을 부연산으로 나눔
  - * 명령어 파이프라인 : fetch, 디코드, 실행, 저장 단계를 나눔

산술 파이프라인

Slide 12

- 부동 소숫점 가산기 파이프라인
  - 정규화된 부동 소숫점 이진수

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

  - 각 세그먼트에서 수행되는 부연산
    1. 지수의 비교 : 큰 지수가 결과 지수
    2. 가수의 정렬 : 작은 지수를 큰 지수에 맞추고 가수를 조절
    3. 가수의 덧셈이나 뺄셈 : 두 가수를 더하거나 뺌
    4. 결과의 정규화 : 결과를 정규화
  - 예)

$$X = 0.9504 \times 10^3$$
$$Y = 0.8200 \times 10^2$$

$$X = 0.9504 \times 10^3$$
$$Y = 0.0820 \times 10^3$$

$$Z = 1.0324 \times 10^3 \qquad Z = 0.10324 \times 10^4$$

명령어 파이프라인

Slide 13

- 명령어 파이프라인의 단계
  1. 메모리에서 명령어를 fetch
  2. 명령어를 디코딩
  3. 유효주소를 계산
  4. 메모리에서 피연산자 fetch
  5. 명령어 실행
  6. 결과 저장
- 명령어 파이프라인에서의 어려운점
  - 분기명령시 파이프라인은 모두 비워져야함
  - 명령에 따라서 세그먼트를 거치지 않는것이 있음
  - 두개 이상의 세그먼트가 동시에 메모리 참조 요구
- 예) 네개의 세그먼트를 갖는 명령어 파이프라인

명령어 파이프라인 (계속)

Slide 14

– 구성 (그림 9-7)

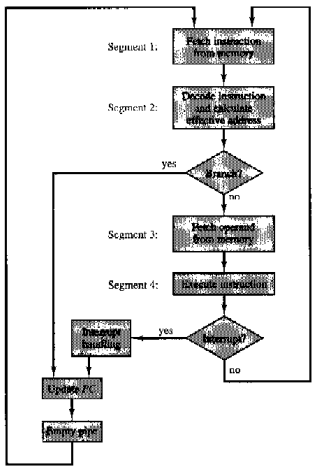


그림 9-7 네 세그먼트 CPU 파이프라인

는 명령어의 결과는 프로세서 레지스터에 저장되므로 명령어의 실행과 결과의

명령어 파이프라인 (계속)

Slide 15

– 분기 명령어에 의한 지연 (그림 9-8)

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

그림 9-8 명령어 파이프라인의 시간 관계

– 명령어 파이프라인이 정상적인 동작을 벗어나게 하는 원인

1. 자원충돌 (resource conflict)
   * 두개의 세그먼트가 동시에 메모리 접근시 발생
   * 명령어 메모리와 데이타 메모리를 분리함으로 대부분 해결
2. 데이타 의존성 (data dependency)
   * 계산되지 않은 결과를 참조할때 발생

명령어 파이프라인 (계속)

3. 분기 곤란 (branch difficulty)
   * 분기명령어 같이 PC 를 변경시키는 명령어에 의해 발생

- 데이타 의존성
  - 데이타 의존성 : 이전 명령어에 의한 결과를 다음 명령어에서 참조
  - 주소 의존성 : 주소 계산에서 필요한 정보들이 이전 명령어에 의해서 준비되지 않음

Slide 16

  - 해결방법
    * 하드웨어 인터락 (hardware interlock)
    * 오퍼랜트 포워딩 (operand forwarding)
    * 지연된 로드 (delayed load)
  - 하드웨어 인터락
    * 이전 명령어의 결과와 어떤 명령의 피연산자가 일치하는지 검사 (인터락)
    * 일치하면 피연자가 준비되지 않은 명령어를 지연시킴
  - 오퍼랜트 포워딩

명령어 파이프라인 (계속)

    * 특별한 하드웨어 사용 충돌을 감지하고 별도의 통로를 통해 파이프라인 세그먼트에 연결
  - 지연된 로드
    * 컴파일러가 충돌을 감지하고 no-operation 명령을 삽입함으로서 해결

- 분기 명령어의 처리

Slide 17

  - 순차적인 명령어와 함께 분기의 목표가 되는 명령어를 미리 가져오는 방법
  - 분기 명령어와 분기 목표 명령어를 어소시어티브 메모리에 저장 (BTB) 하고 다음에 이용
  - 프로그램 루프의 명령어및 분기 목표 명령어를 루프버퍼 (loop buffer) 에 저장 이용
  - 분기를 미리 예측 (branch prediction) 하고 미리 명령어 읽음, 예측 맞으면 정상적 동작
  - 대부분의 RISC 프로세서는 지연된 분기 (delayed branch) 기법을 사용
    * 분기 명령어를 찾아내고 유효한 명령어 삽입 기계어 코드 순서 재배치

명령어 파이프라인 (계속)

Slide 18

* 예로 분기 명령어 다음에 no-operation 명령어 삽입

RISC 파이프라인

Slide 19

- RISC 가 파이프라인에 적합한 이유
  - 명령어가 단순하여 한 클럭에 수행되는 부연산 구현이 쉬움
  - 고정된 명령어 형식을 사용함으로 디코딩과 레지스터 선택이 용이
  - 모든 피연산자가 레지스터에 있음으로 메모리 참조가 필요없음
  - 충돌과 분기 명령어 문제점을 컴파일러로 해결
- 예) 세 세그먼트 명령어 파이프라인
  - 세그먼트의 구성

    I : 명령어의 fetch
    A : ALU 동작
    E : 명령어의 실행
- 지연된 로드
  - 실행될 명령어

RISC 파이프라인 (계속)

Slide 20

1. LOAD : R1 ←M[주소1]
2. LOAD : R2 ←M[주소2]
3. ADD : R3 ←R1 + R2
4. STORE: M[주소3] ←R3

- 데이타의 충돌 발생 (그림 9-9 (a))
- 컴파일러가 no-operation 명령을 삽입 해결 (그림 9-9 (b))

RISC 파이프라인 (계속)

Slide 21

Clock cycles:	1	2	3	4	5	6
1. Load *R1*	I	A	E			
2. Load *R2*		I	A	E		
3. Add *R1* + *R2*			I	A	E	
4. Store *R3*				I	A	E

(a) Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load *R1*	I	A	E				
2. Load *R2*		I	A	E			
3. No-operation			I	A	E		
4. Add *R1* + *R2*				I	A	E	
5. Store *R3*					I	A	E

(b) Pipeline timing with delayed load

그림 9-9 세 세그멘트 파이프라인의 시간 관계

RISC 파이프라인 (계속)

Slide 22

→지연된 로드

- 지연된 분기
  - 분기의 예를 보여주는 프로그램

```
메모리에서 R1 으로 로드
R2 를 하나 증가
R3 를 R4 로 더함
R6 에서 R5 를 뺌
주소 X 로 분기
```

  - 컴파일러가 분기명령 다음에 2개의 no-operation 명령을 삽입 (그림 9-10 (a))
  - 덧셈과 뺄셈 명령어를 분기 명령어 다음으로 옮김 (그림 9-10 (b))

RISC 파이프라인 (계속)

Slide 23

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

(a) Using no-operation instructions

Clock cycles:	1	2	3	4	5	6	7	8
1. Load	I	A	E					
2. Increment		I	A	E				
3. Branch to *X*			I	A	E			
4. Add				I	A	E		
5. Subtract					I	A	E	
6. Instruction in *X*						I	A	E

(b) Rearranging the instructions

그림 9-10 지연된 분기의 예

벡터 처리

Slide 24

- 방대한 양의 수계산을 하는 문제는 보통 벡터나 행렬로 수식화 됨
- 이러한 벡터연산을 빠르게 하기위한것
- 벡터 연산
  - 예제 프로그램

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

  - 기계어 프로그램

벡터 처리 (계속)

Slide 25

```
      Initialize I = 0
20    Read A(I)
      Read B(I)
      Store C(I)= A(I) + B(I)
      Increment I = I + 1
      If I ≤ 100 go to 20
      Continue
```

- 프로그램 루프에서 명령어를 fetch 하고 실행시키는 오버헤드 시간을 없앰
- 벡터 명령어 형식 → C(1:100) = A(1:100) + B(1:100) (그림 9-11)

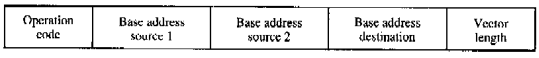


그림 9-11 벡터 프로세서의 명령어 형식

벡터 처리 (계속)

Slide 26

- 행렬 곱셈
  - $n \times n$ 행렬의 곱셈 $\rightarrow n^2$ 의 곱셈과 n^3 의 곱셈-덧셈필요
  - 3×3 행렬에서 내부곱은 9개 곱셈덧셈은 27개가 필요
  - 일반적인 내부곱의 표현

    $$\boxed{C = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + \cdots + A_kB_k}$$
  - 내부곱을 계산하기위한 파이프라인 (그림 9-12)

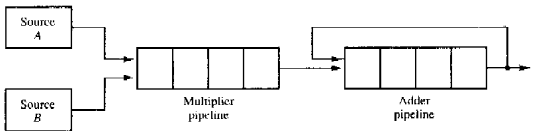


그림 9-12 내부곱을 계산하기 위한 파이프라인

- 메모리 인터리빙 (그림 9-13)

벡터 처리 (계속)

Slide 27

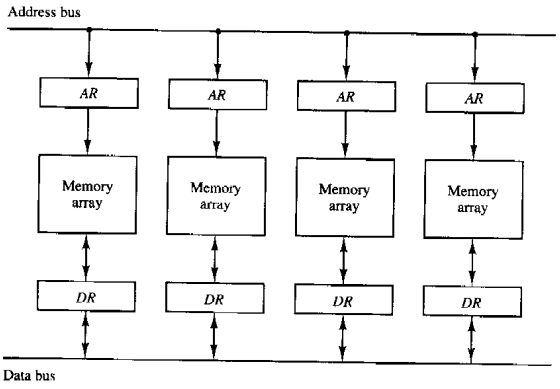


그림 9-13 다중 모듈 메모리 구조

  - m 개의 독립적인 모듈을 두고 연속적으로 명령어나 데이타를 가져옴
- 슈퍼 컴퓨터
  - 벡터명령어와 파이프라인된 부동 소숫점 연산을 제공하는 컴퓨터

벡터 처리 (계속)

- 부품들을 밀접하게 배치 열을 제거하는 특별한 기술 필요
- 일반명령어 와 벡터 처리 명령어 벡터와 스칼라 처리 명령어가 있음
- 빠른 계산 속도, 큰 메모리 시스템, 병렬처리 기능
- 가격 비쌈

Slide 28

- 배열 프로세서 (array processor)
  - 대규모 배열에 대한 계산을 수행
  - 종류
    - 부가 배열 프로세서 (attached array processor)
    - SIMD 배열 프로세서 (SIMD array processor)
  - 부가 배열 프로세서
    - 호스트 컴퓨터에 주변장치로 설계된것
    - back-end machine 으로서 동작 (그림 9-14)

벡터 처리 (계속)

Slide 29

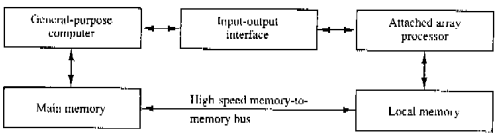


그림 9-14 호스트 컴퓨터를 가진 부가 배열 프로세서

- 고속의 벡터 처리능력 제공
- SIMD 배열 프로세서 (그림 9-15)

벡터 처리 (계속)

Slide 30

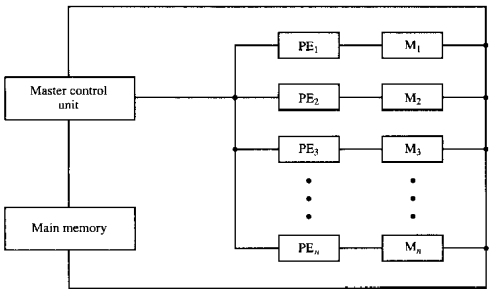


그림 9-15 SIMD 배열 프로세서 구조

* 병렬 다중 처리 장치를 갖음
* 프로세싱 요소 (PE) 와 로컬 메모리 (M)를 갖음
* 스칼라 명령어는 master control unit 에서 실행
* 벡터 명령어는 각 벡터 연산자를 로컬메모리에 미리 넣고 동일 명령 수행