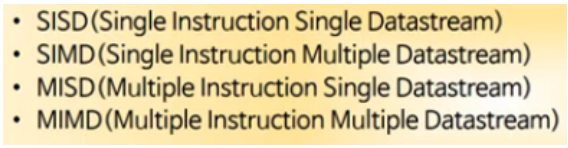
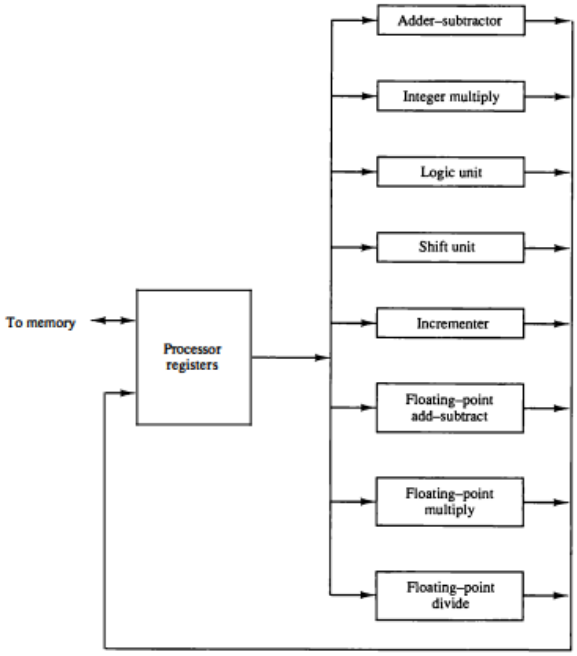
제 9장 Part-1

- 클라우딩 컴퓨팅에 기본이 되는 기술(파이프라인, 병렬 처리)

병렬처리 (Parallel Processing)

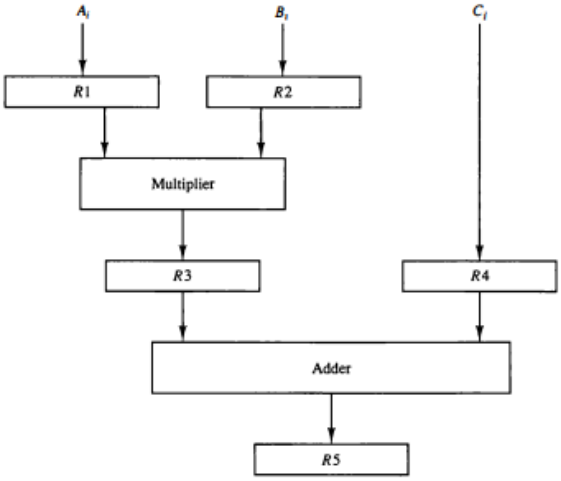
- 다중 기능 장치를 가지는 프로세서
  - **공통 프로세서 레지스터**와 **다중 연산 유닛**
  - **동시에 2개 이상의 연산 처리**
- 병렬처리의 종류(Flynn의 분류)
  - 

  - 대부분 SIMD, MIMD
- Pipelining
  - 다단계 데이터 처리 방식
  - 동시 다중 데이터 처리
-



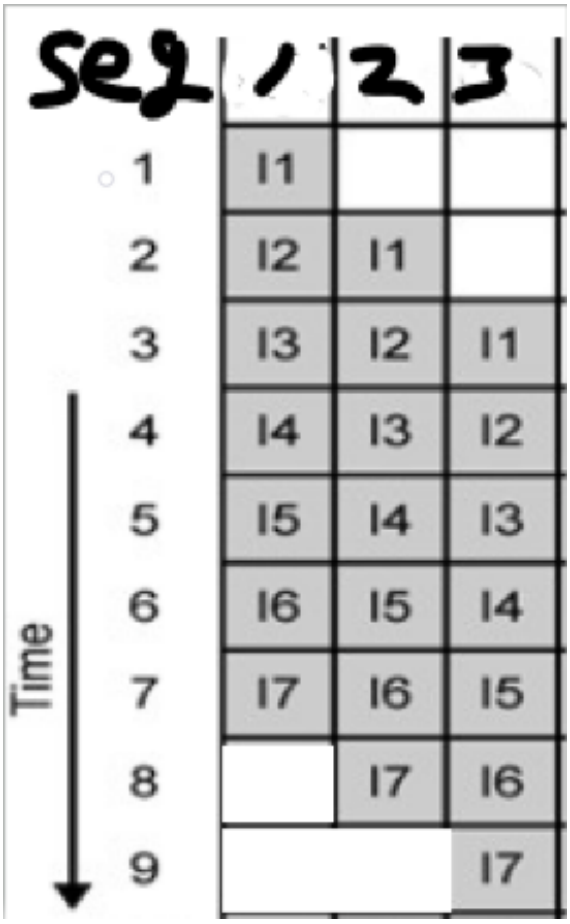
파이프라인 (Pipelining)

- 하나의 프로세서를 서로 다른 기능을 가진 세그먼트로 분할
- 각 세그먼트가 동시에 서로 다른 데이터 처리
- $A_i * B_i + C_i \quad \text{for } i = 1, 2, 3, \ldots, 7$
- $R1 \leftarrow A_i, \quad R2 \leftarrow B_i$ — Input A_i and B_i
  $R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i$ — Multiply and input C_i
  $R5 \leftarrow R3 + R4$ — Add C_i to product
- Figure 9-2 Example of pipeline processing.



- TABLE 9-1 Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$

  - segment1 - load, segment2 - multiply, segment3 - adder
  - 모든 segment가 다 사용되는 경우를 파이프라인이 다 찼다라고 표현
  - 파이프라인을 안쓰면 21clock 필요
  - 파이프라인을 쓰면 9clock 이 필요
  - 

- 일반적인 파이프라인 구조

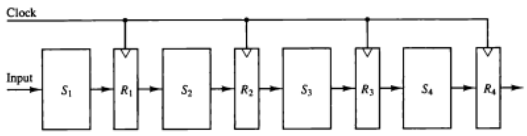
- 

Figure 9-3 Four-segment pipeline.

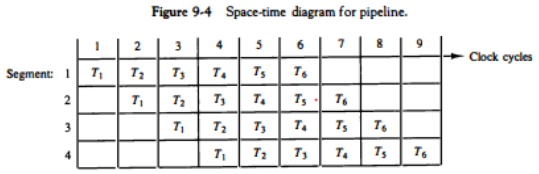
- 파이프라인 공간-시간표
  - 

Figure 9-4 Space-time diagram for pipeline.

  - 파이프라인의 효율
    - $$S = \frac{nt_n}{(k + n - 1)t_p}$$
    - 6 x 4 / 9 = 24 / 9 (원래 필요한 clock / 파이프라인 사용시 필요한 clock)
  - 최대 효율
    - $$S = \frac{kt_p}{t_p} = k$$
      - full 상태일 때 (원래 필요한 clock/ 파이프 라인사용시 필요한 clock)
    - t_n : 각 태스크를 완료하는시간
      t_p : clock 사이클 시간
      k : 세그먼트의 수
      n : 태스크의 수

제 9장 Part-2

산술 파이프라인 (Arithmetic Pipeline)

- 실수의 가산
  - X = A x 2^a
  - Y = B x 2^b
- 세그먼트별 연산
  - 지수의 비교
  - 가수의 정렬
  - 가수의 연산
  - 결과의 정규화
-

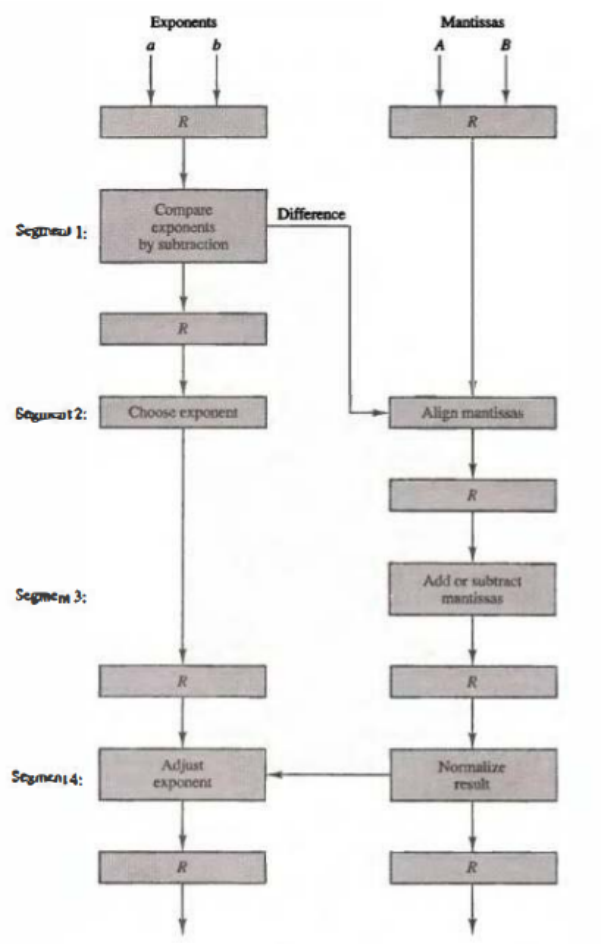


Figure 9-6 Pipeline for floating-point addition and subtraction.

명령어 파이프라인 (Instruction Pipeline)

- 명령 실행의 순차

  - 1. 메모리에서 명령어 fetch
  - 2. 명령어 디코딩
  - 3. 유효주소의 계산
  - 4. 메모리에서 피연산자 fetch
  - 5. 명령어 실행
  - 6. 연산 결과의 저장

- 4 세그먼트 CPU 파이프 라인

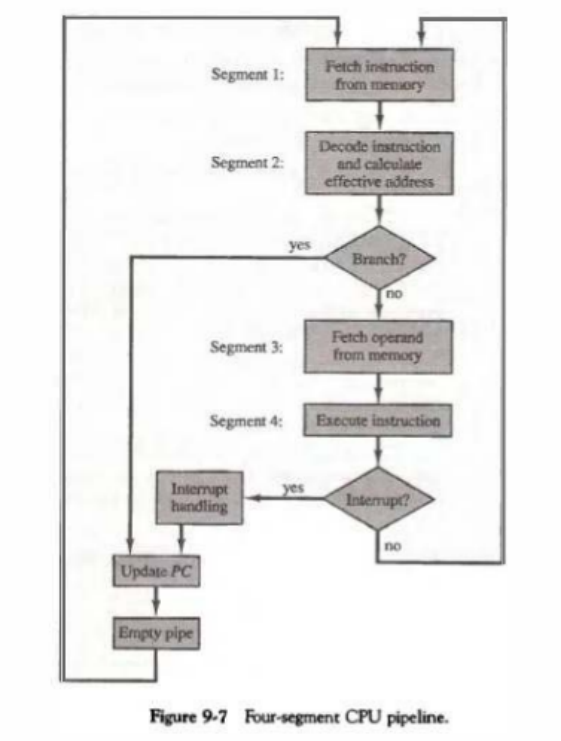
  - 

Figure 9-7 Four-segment CPU pipeline.

- 명령어 파이프라인의 4 세그먼트

  - 1. FI (Fetch Instruction)
  - 2. DA(Decode and Address)
  - 3. FO(Fetch Operand)
  - 4. EX(Execution)

- 명령어 파이프라인의 지연

- 원인
  - 자원 충돌(Resource conflict)
  - 데이터 의존성 (data dependency)
  - 분기 곤란 (Branch difficulty)
-

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

Figure 9-8 Timing of instruction pipeline.

RISC 파이프라인 (RISC Pipeline)

- 3 세그먼트 명령어 파이프라인
  - I : 명령어의 fetch
  - A: ALU 동작
  - E: 명령어의 실행
- 지연된 Load
-

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$

- 데이터 의존성 발생(a) -> 지연된 코드로(b) 문제 해결
-

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

(a) Pipeline timing with data conflict

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

(b) Pipeline timing with delayed load

Figure 9-9 Three-segment pipeline timing.

- 지연된 분기
  - Load from memory to *R*1
    Increment *R*2
    Add *R*3 to *R*4
    Subtract *R*5 from *R*6
    Branch to address *X*
  - 해결
    - 정교한 컴파일러 사용 -> RISC 컴파일러의 특징
    - 실행 순서 변경
    - 데이터 의존성 회피
  -

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

Figure 9-10 Example of delayed branch.

제 9장 Part-3

백터 처리 (Vector Processing)

- **벡터 처리**
  - 행렬 데이터 처리
  - 병렬 데이터 처리
- **벡터 처리가 중요한 분야**
  - 장기 기상 예보
  - 석유 탐사
  - 지진 데이터 분석
  - 의학 검진, 분석
  - 기계 역학, 비행 시뮬레이션
  - 인공지능, 전문가 시스템
  - 유전자 분석

- 2/3 차원 이미지 처리

- **벡터 연산과 벡터 명령어**

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

  -
  ```
        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
  ```

  - $C(1:100) = A(1:100) + B(1:100)$

  - Figure 9-11 Instruction format for vector processor.

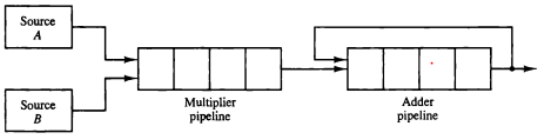
Operation code	Base address source 1	Base address source 2	Base address destination	Vector length

  - 위에서 base address => 1, length => 100

- **메모리 인터리빙(Interleaving)**

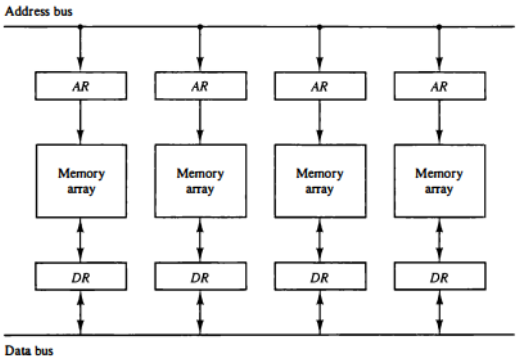
  - 두 개 이상의 명령어가 동시에 메모리를 접근하는 경우
  - 메모리를 여러 모듈로 분할 구성
  - 파이프라인의 자원 충돌 문제 해결

  - Figure 9-12 Pipeline for calculating an inner product.

  -

Figure 9-13 Multiple module memory organization.



- 슈퍼컴퓨터 (Supercomputer)
  - 정의
    - 벡터 명령어 제공
    - 파이프라인된 부동 소수점산술연산제공
    - 상업용 컴퓨터
  - 성능 요소
    - 고속의 연산을 위한 설계
    - 고속 위주의 소재, 부품 사용
  - Flop
    - 초당 처리할 수 있는 floating point 연산의 수
    - MFlop, Gflop 단위
- 대표적인 슈퍼컴퓨터
  - CRAY-1, CRAY-2, CRAY X-MP
  - Fujitsu VP-200, VP-2600

배열 프로세서 (Array Processors)

- 부가 배열 프로세서
  - backend 프로세서 array 사용
    - 대량의 데이터 처리 전담

    - Local memory에 데이터 저장
    - 트랜스퓨터(Transputer)라고도 지칭
- Main 프로세서
    - 데이터 전송 프로그램 실행

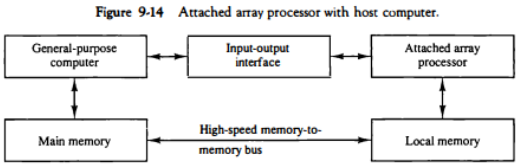
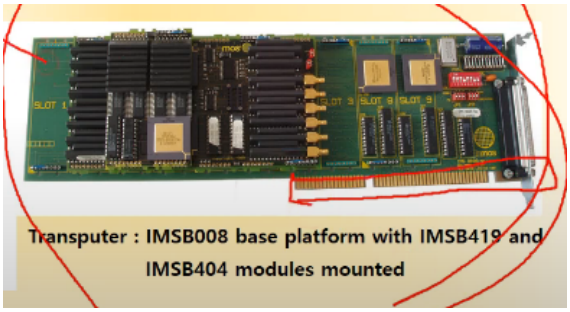


Figure 9-14 Attached array processor with host computer.


Transputer : IMSB008 base platform with IMSB419 and IMSB404 modules mounted

- SIMD 배열 프로세서
    - Main CPU, Main memory에 다수의 PE 연결
    - PE(Processing Element)
        - 자체 프로세서와 local memory 포함
        - Array 프로세서 형태로 구현

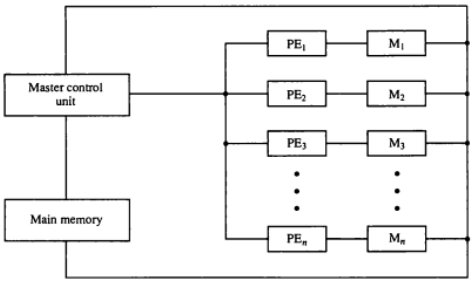


Figure 9-15 SIMD array processor organization.

Illinois/Burroughs's ILLIAC-IV

- ILLIA-IV
  - 대표적인 초기 SIMD 배열 프로세서
  - 미국 일리노이 대학 연구실 개발
  - 초기형 SIMD 슈퍼컴퓨터급 성능