제 13강. 4.1 서론, 4.2 논리 설계 관리, 4.3 데이터패스 만들기

Chapter 4. 프로세서

4.1. 서론

컴퓨터의 고전적인 5가지 구성 요소

컴퓨터 성능을 결정하는 요소

1. **명령어 개수**

   ㄴ 컴파일러와 명령어 집합 구조가 결정

2. **클럭 사이클 시간**

3. **명령어당 클럭 사이클 수** (**CPI**, clocks per instruction)

   ㄴ 2, 3은 프로세서의 구현 방법에 따라 결정됨.

앞으로 배울 것

- 프로세서를 구현하는 데 사용되는 원리와 기법들
- 실제와 가까운 파이프라인 MIPS 구현
- x86 같이 좀 더 복잡한 명령어 집합을 구현하는 데 필요한 개념 설명
- 4.6 - 파이프라이닝의 기본 개념 - 명령어에 대한 상위 수준 해독과 명령어가 프로그램 성능에 미치는 영향
- 4.11 - 최근 동향
- 4.12 - Intel Core i7과 ARM Cortex-A53 구조 소개
- 4.13 - 명령어 수준 병렬성을 이용하여 3.8절의 행렬 곱셈 성능을 어떻게 2배 이상 증가실 수 있는지

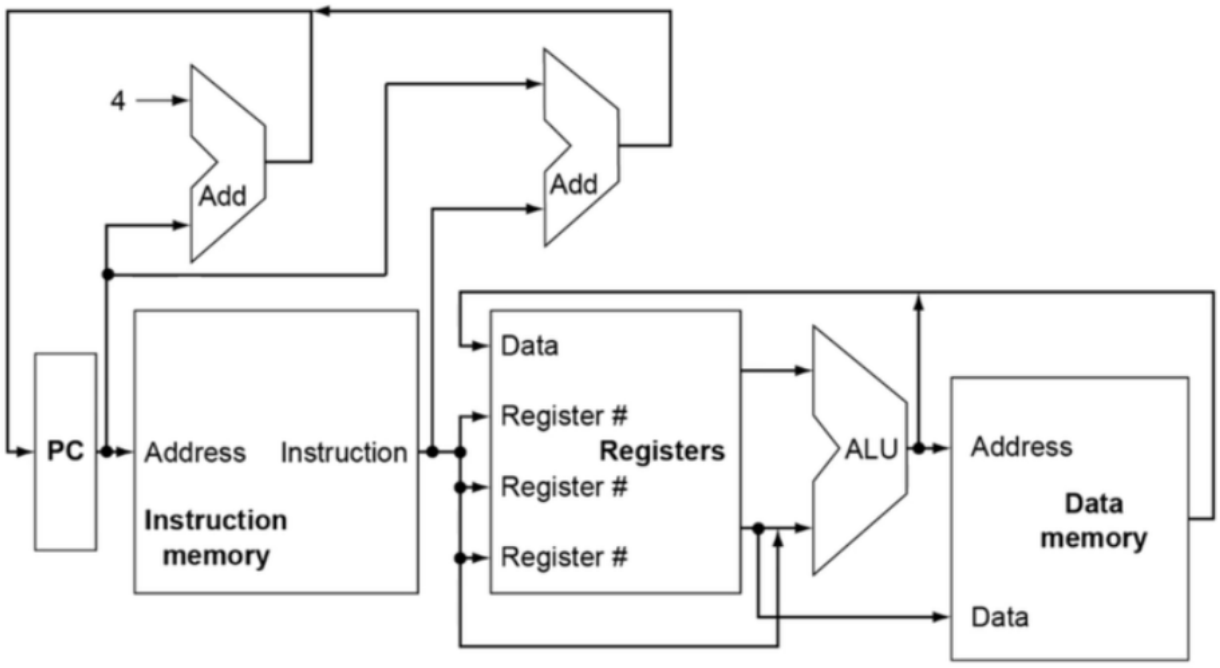
기본적인 MIPS 구현

- 메모리 참조 명령어 : lw(load doubleword) , sw(store doubleword)
- 산술/논리 명령어 : add, sub, and, or, slt
- 조건부 분기 명령어 : beq(branch equal), j(jump)

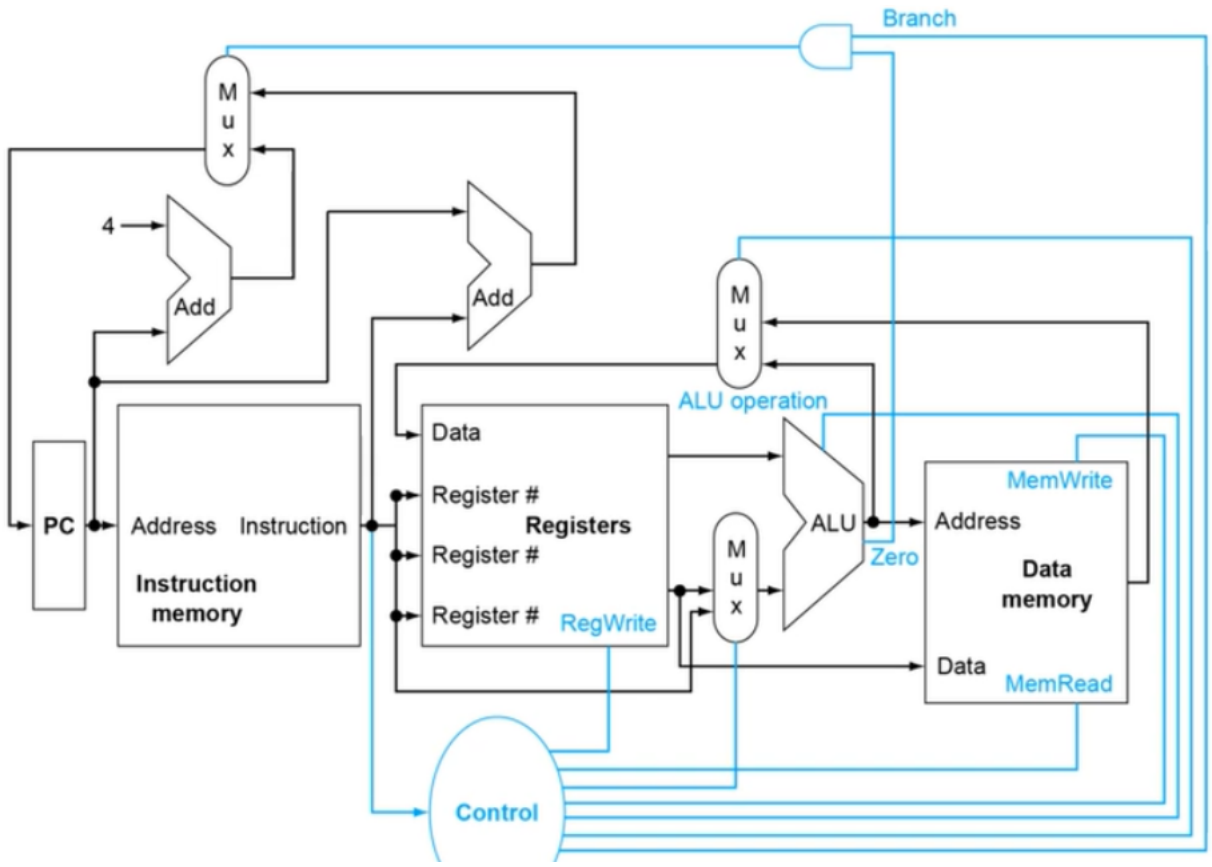
구현에 대한 개요

1. 프로그램 카운터(PC)를 프로그램이 저장되어 있는 메모리에 보내서 메모리로부터 명령어를 가져온다. fetch instruction
2. 읽을 레지스터를 선택하는 명령어 필드를 사용하여 하나 또는 2개의 레지스터를 읽는다. 워드 적재 명령어는 레지스터 하나만 읽으면 되지만 대부분의 다른 명령어는 레지스터 2개를 읽는다.
- 연산에 따라 다르다
    - 계산을 위해선 ALU를 사용

Multiplexers



- 하나로 합칠 수 없다.
  - Use multiplexers 먹스를 사용해야 된다.
- 위 MIPS 부분집합 구현 그림에서 빠진 2가지
  1. 멀티플렉서가 빠져 있다 : 다수의 근원지 중 하나를 선택하여 그것을 목적지로 보내는 구성 요소 (멀티플렉서,multiplexor = 데이터 선택기, data selector)
  2. 어떤 유닛들은 데이터 종류에 따라 다르게 제어되어야 함



4.2 논리 설계 관례

MIPS를 구성하는 데이터패스 요소

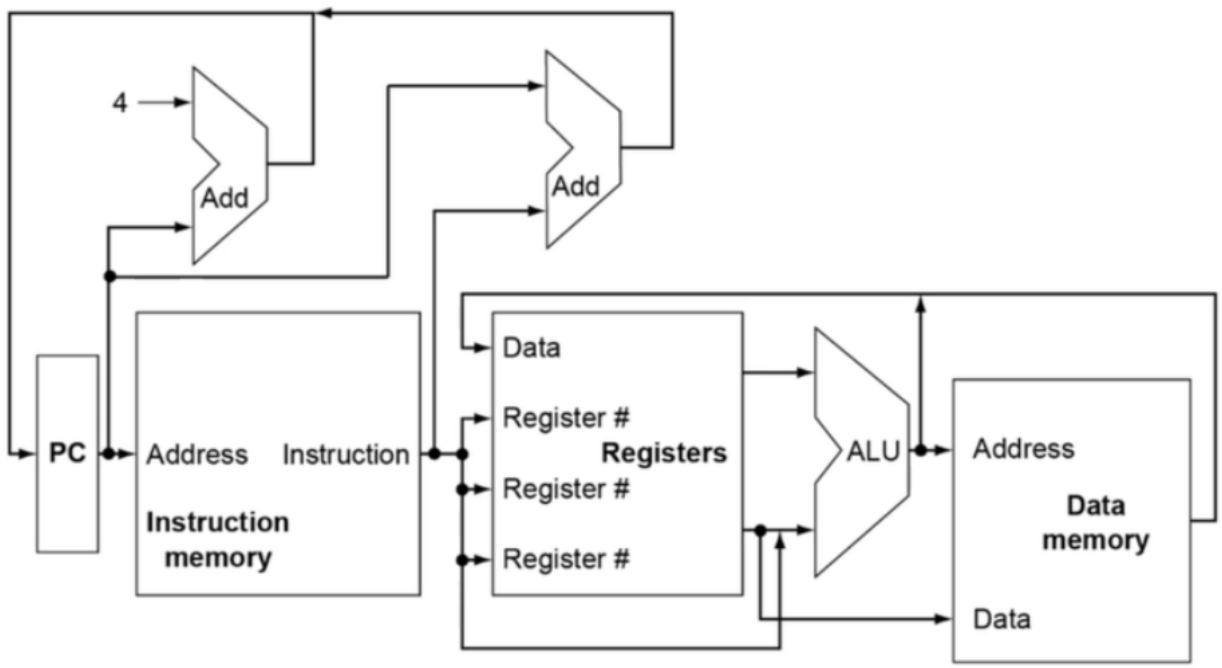
1. 조합 소자 (combinational element)
2. 상태 소자 (state element)

조합 소자

- 데이터 값에 대해 연산을 수행하는 소자
- 출력이 현재 입력에 의해서만 결정됨
  - Input이 결정되면 Output이 결정됨
- 조합 소자의 예 : ALU, AND-gate, Adder, Multiplexer, Arithmetic/Logic Unit

상태 소자

- 소자에 내부 기억 장소가 있으면 상태를 갖게 됨
- 컴퓨터의 전원 플러그를 빼더라도 플러그를 빼기 전의 값을 상태 소자에 넣어 주면 전과 똑 같은 상태에서 다시 시작할 수 있음
- 상태 소자의 예 : 명령어 메모리, 데이터 메모리, 레지스터



Sequential Elements

Register : stores data in a circuit

현재 상태에 따라 clock signal에 따라 저장된 값이 바뀌는 것

Edge-triggered

레지스터, 메모리를 구현하는 데 사용

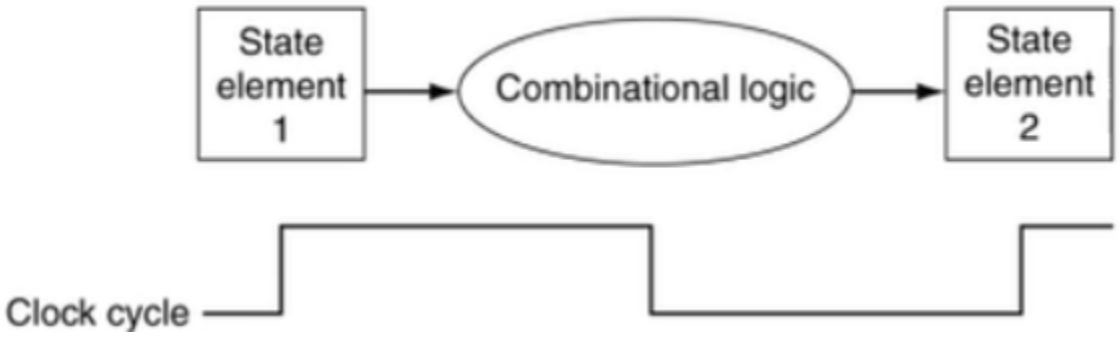
클러킹 방법론

클러킹 방법론 은 신호를 읽고 쓰는 시점을 정의한다.

에지 구동 클러킹(edge-triggered clocking)으로 알아보는 클러킹 방법론

순차 논리 소자에 저장된 값은 클럭 에지에서만 바꿀 수 있다.

- 클럭 에지 : 낮은 값에서 높은 값, 혹은 그 반대로의 빠른 변이를 말한다.



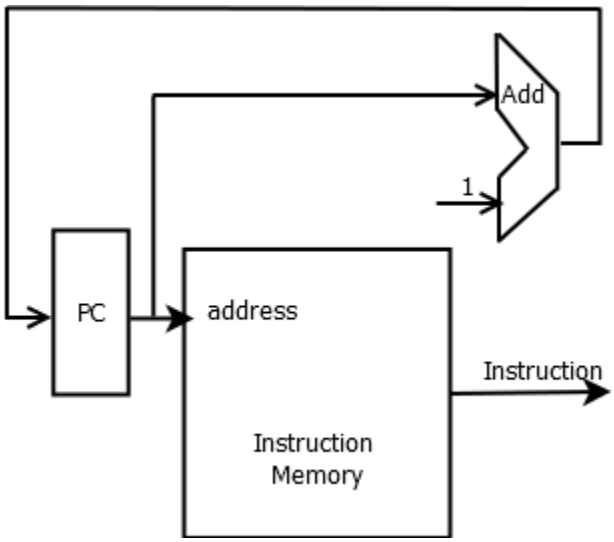
- 상태 소자들만이 데이터 값을 저장할 수 있기 때문에 모든 조합 논리는 상태 소자에서 입력을 받고 상태 소자로 출력을 내보냄
- 모든 신호가 한 클럭 이내에 상태 소자1에서 나와 조합 회로를 거쳐 상태 소자 2까지 전달되어야 함
- 신호가 상태 소자 2에 도달하는 데 필요한 시간 : 클럭 사이클의 길이
- 클럭 신호와 쓰기 제어 신호는 상태 소자의 입력
- 쓰기 제어 신호가 인가되고 활성화 클럭 에지일 때만 상태 소자가 변화하게 됨
- 에지구동클러킹은 레지스터 내용을 읽고 그 값을 조합 회로에 보내고 같은 레지스터에 쓰는 작업 모두가 한 클럭 사이클에 일어나는 것을 허용

4.3 데이터패스 만들기

데이터패스란?

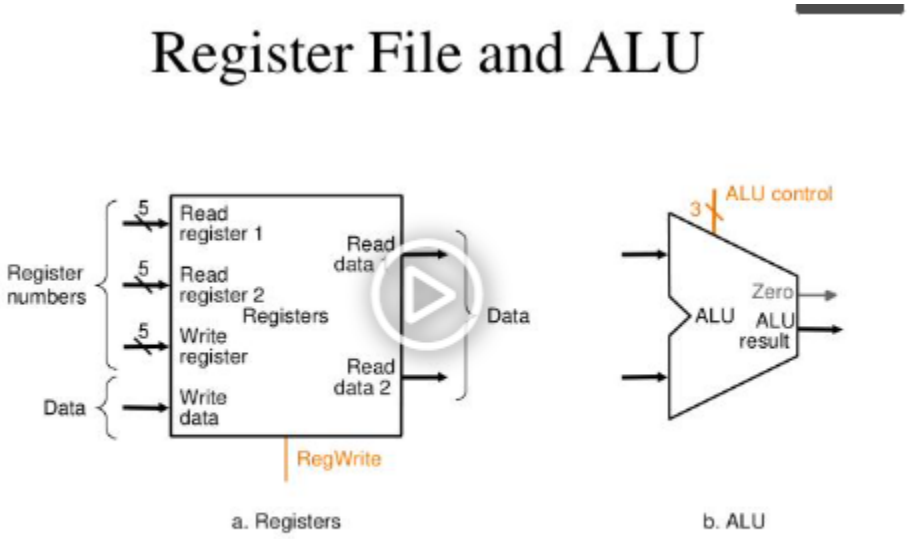
- CPU에서 데이터와 주소를 프로세싱하는 요소
  - registers, ALUs, mux’s, memories
- MIPS datapath를 순차적으로 만들 것

Instruction Fetch



R 형식 명령어 (= 산술/논리 명령어)

- 2개의 레지스터를 읽고 레지스터 내용에 ALU 연산을 수행한 후 그 결과를 레지스터에 쓴다.



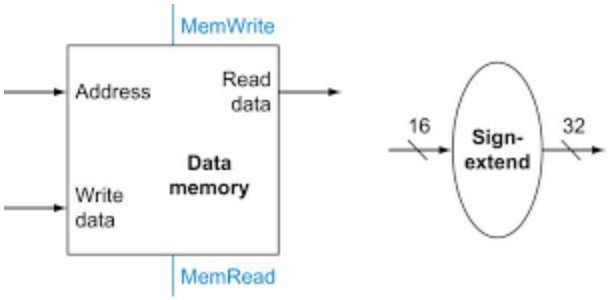
- 레지스터 피연산자를 3개 가지고 있음 ⇒ 매 명령어마다 레지스터 파일에서 데이터 워드 2개를 읽고 데이터 워드 하나를 써야 함
- 레지스터에서 데이터 워드를 읽기 위해서는 레지스터의 입력과 출력 하나씩 필요
- 데이터 워드를 쓰기 위해서는 입력이 2개 필요

- 입력 1 ← 레지스터 번호를 지정
- 입력 2 ← 레지스터에 쓸 데이터 값 제공

적재/ 저장 명령어

- 워드 적재 명령어, 워드 저장 명령어
- 베이스 레지스터($t2)와 명령어에 포함돼 있는 16비트 부호있는 변위 필드를 더하여 메모리 주소를 계산
- 저장 명령어이면 저장할 값을 레지스터 파일에서 읽어 와야 하는데 이 값은 $t1에 있음
- 적재 명령어이면 메모리로부터 읽어들인 값을 지정된 레지스터($t1)에 써야 함
- 위 그림의 레지스터 & ALU 모두 필요

분기 명령어



a. Data memory unit b. Sign extension unit

- 16비트 변위 필드 값을 32비트 부호있는 값으로 부호확장하기 위한 유닛이 필요하며 읽고 쓸 데이터 메모리가 필요
- beq 명령어는 비교할 레지스터 2개와 16비트 변위의 세 피연산자를 가짐
- 변위는 분기 목적지 주소(branch target address)를 분기 명령어 주소에 대한 상대적인 값으로 표현하는 데 사용
- 명령어 형태 : beq $t1, $t2, offset

분기 명령어의 정의 중 주의해야 할 점

1. 분기 주소 계산의 베이스 주소는 분기 명령어 다음 명령어의 주소이다.

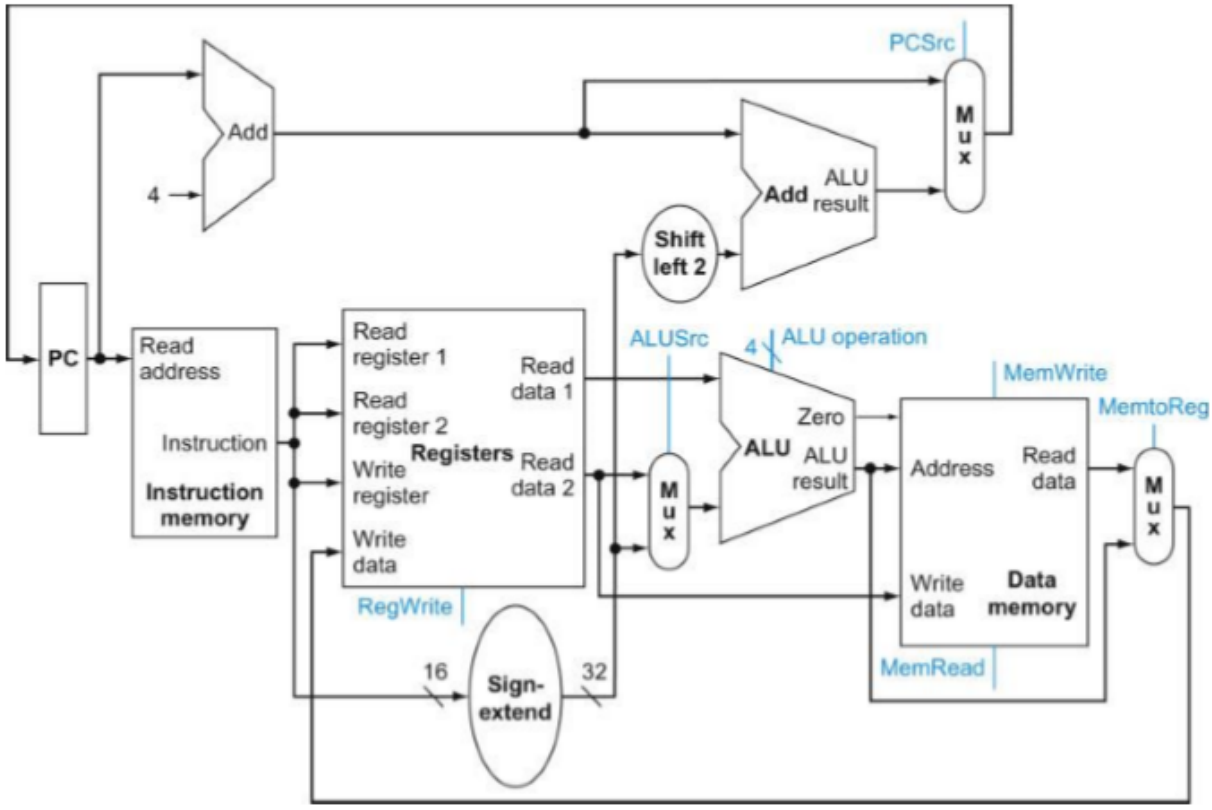
2. 명령어 집합 구조는 변위 필드를 2비트만큼 오른쪽으로 자리이동하여 워드 변위로 만든다.
   → 이렇게 함으로써 변위 필드의 유효 범위를 4배만큼 증가시킴

- 분기 목적지 주소만 계산하면 되는 것이 아니고, 실행할 다음 명령어가 분기 명령어 뒤에 있는 명령어가 될지 분기 목적지 주소에 있는 명령어가 될지를 판단해야 함
- 조건이 사실일 때 (두 피연산자 값이 같을 때) 분기 목적지 주소가 새로운 PC 값이 되며 **분기가 일어났다(branch taken)**라고 한다.
- 두 피연산자 값이 같지 않으면 증가된 PC 값이 새 PC 값이 된다. ⇒ **분기가 일어나지 않았다 (branch not taken)**
- 분기 데이터패스는

  1. 분기 목적지 주소 계산
  2. 레지스터 내용 비교

  이 2가지 일을 수행해야 함

단일 데이터패스 만들기



- 어느 데이터패스 자원도 명령어당 2번 이상 사용될 수 없음
- 2번 이상 사용할 필요가 있는 구성 요소는 필요한 만큼 여러 개를 두어야 함

  ㄴ 데이터 메모리 & 명령어 메모리가 별도로 필요한 이유

정리

단일 사이클 데이터패스가 데이터 메모리와 명령어 메모리를 따로 가져야 하는 이유

1. MIPS에서는 데이터와 명령어의 형식이 달라 다른 메모리가 필요하기 때문
2. 메모리를 따로따로 갖는 것이 저렴하기 때문
3. 프로세서가 명령어를 한 사이클에 실행하는데 단일 포트의 메모리로는 한 사이클에 2개의 서로 다른 접근을 할 수 없기 때문

Combinational element 의 정의는?