I used **all** of my free days.

Pipeline MIPS Emulator

32214362 모바일시스템공학과 조윤서

Introduction

본 프로젝트에서는 MIPS CPU 에뮬레이터를 구현하였다. 이전 과제에서 single-cycle 방식으로 구현한 것과 달리, 이번 프로젝트에서는 MIPS 아키텍처를 pipeline 방식으로 구현하여 처리 속도를 향상시켰다. single-cycle 방식은 모든 명령어가 가장 느린 명령어의 실행 시간에 맞추어 처리되어야 하지만, pipeline 방식은 명령어가 연속적으로 입력되므로 처리 효율이 크게 향상된다. 이를 통해 pipeline architecture는 single-cycle 방식에 비해 최대 10배 더 빠른 성능을 제공함을 확인하였다.

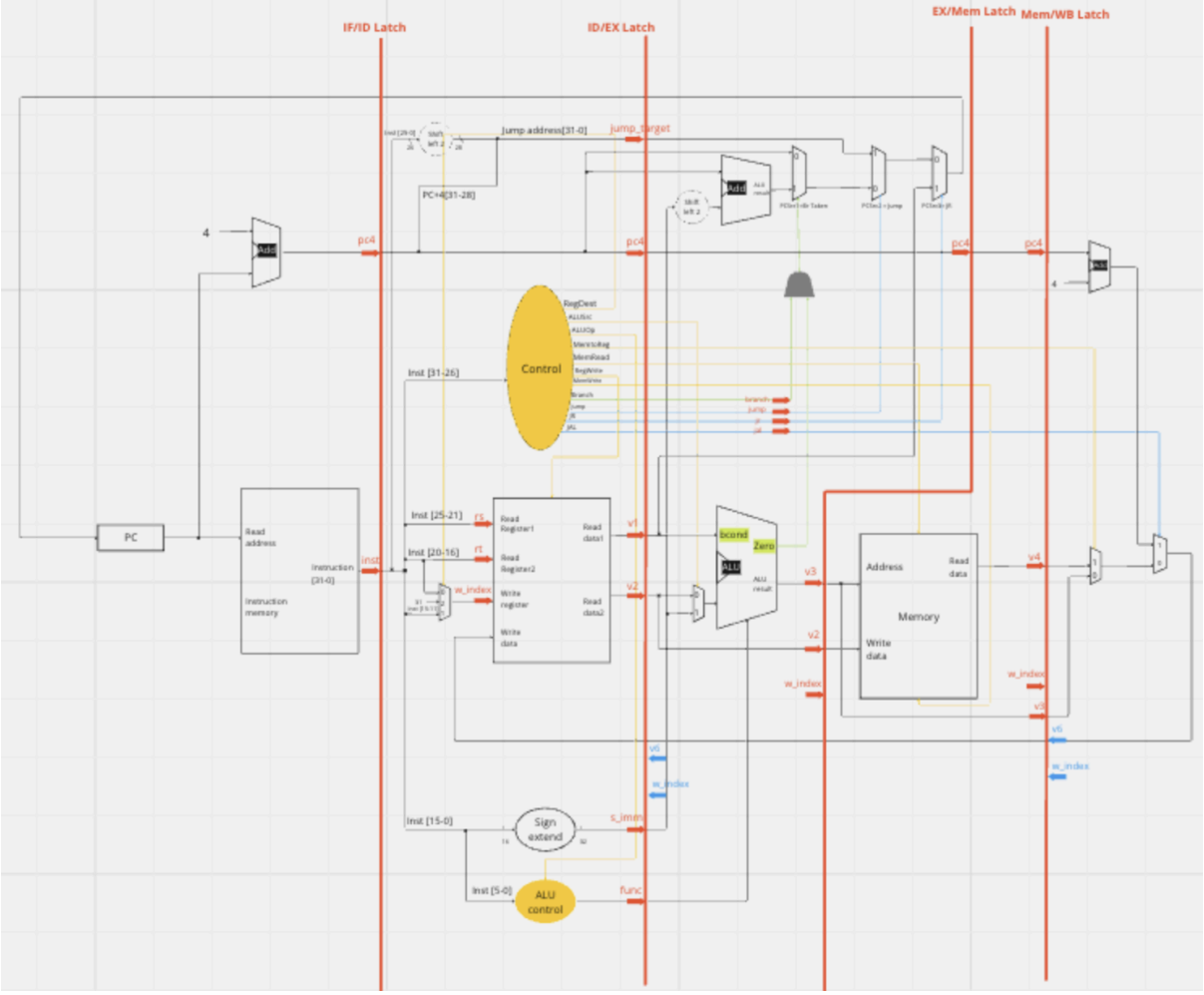
Important Concept

- MIPS 아키텍처
- Pipeline

```
while(is_pipeline_active(&latch)){//n<80
    write_back(&latch);
    print_writeBack(&latch);

    access_memory(&latch);
    print_memoryAccess(&latch);

    execute(&latch);
    print_execute(&latch);

    decode(&latch);
    print_decode(&latch);

    fetch(&latch);
    print_fetch(&latch);

    update_latch(&latch);
    init_latch(&latch);
    // print_register();
    print_cycle();
    // n++;
}
```

Implementation Process

1. Data Path Design
    a. 본격적인 코드 작업에 앞서 데이터 패스를 먼저 설계하였다. 이는 프로세서의 데이터 처리 과정을 명확히 하기 위한 초기 단계이다.

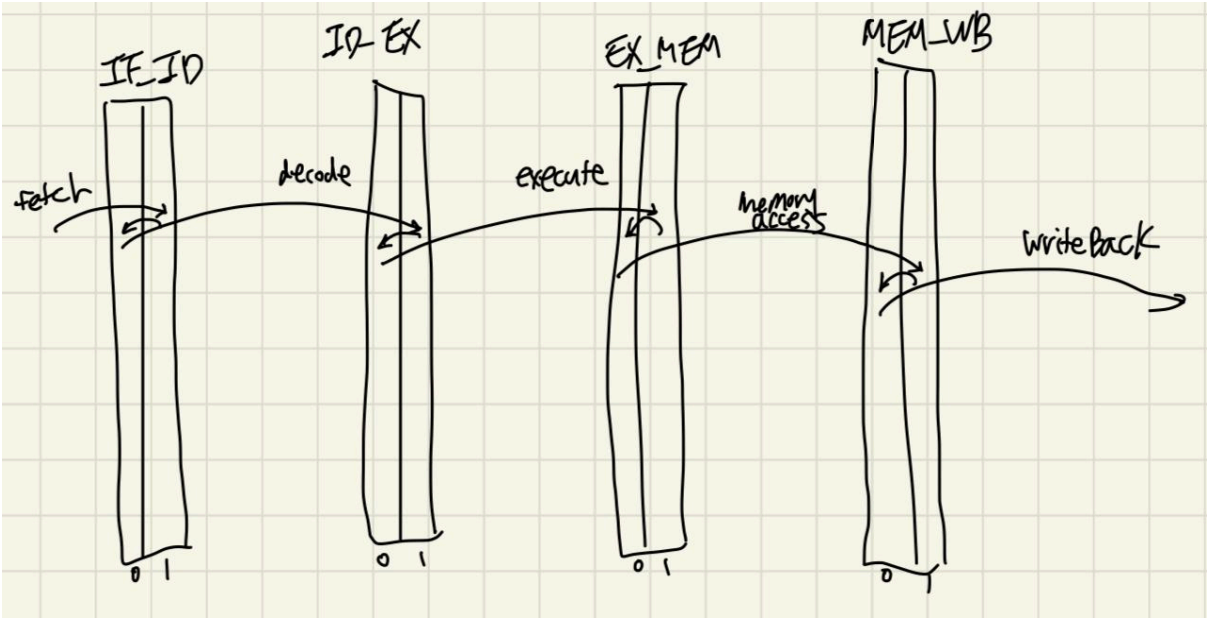


<Data Path Design>

2. Implementation:
    a. 헤더 파일 및 소스파일 분리
        i. Single-cycle과제에서는 모든 기능을 main.c 파일 내에 구현하였으나, 이번 프로젝트에서는 헤더 파일과 소스파일을 분리하여 가독성을 향상시켰다.
    b. Latch 추가
        i. 각 처리 단계 사이에 래치(Latch), 즉 상태를 저장하는

플립플롭(flip-flop) 하드웨어 구조를 구현하였다.

ii. 래치 배열을 활용하여 현재 상태와 다음 상태를 관리하였다. 여기서 인덱스 0은 현재 상태를, 인덱스 1은 다음 상태를 나타낸다. 각 사이클의 시작에서 인덱스 1의 데이터를 인덱스 0으로 이동함으로써 다음 사이클의 상태를 현재 상태로 업데이트하였다.



c. pipeline 종료조건

i. pc = -1이어도 writeBack stage까지 실행을 해주어야하므로 별도의 isPipelineActive() 함수를 구현하여 처리하였다.

d. 출력 함수 분리

i. 각 pipeline stage 별로 printf 함수를 구현하여, 기능 수행 함수와 출력 함수를 분리하였다.

e. Data dependency

i. EX/MEM -> ID/EX 로의 data forwarding

ii. MEM/EX -> ID/EX 로의 data forwarding

iii. data forwarding을 모두 execute stage에서 처리하였다

iv. 분기 명령어(예: Branch, Jump)를 실행 단계에서 처리할 경우, 목표 PC 값이 결정되고, 이에 따라 이전 IF 및 ID 단계에 로드된 명령어들을 초기화한 후, 새로운 PC 값에 해당하는 명령어를 파이프라인에 주입할 수 있도록 invalidate_pipeline_stages()를 구현하였다

v. 하지만 data dependency를 완벽하게 해결하지 못했다.

f. **Control dependency**

i. 구현 못함

Build Configuration / Environment

- Development Environment: **Visual Studio Code**
- Programming Language: **C**

- Build Configuration:
  - 컴파일: **gcc main.c src/alu.c src/control_signal.c src/cpu.c src/data_forwarding.c src/executionStats.c src/latch.c src/memory.c src/opcode.c**
  - 실행 파일과 'simple.bin' 함께 실행: **./a.out ./test_prog/simple.bin > a.log**
  - 실행 파일과 'simple2.bin' 함께 실행: **./a.out ./test_prog/simple2.bin > a.log**
  - 실행 파일과 'fib.bin' 함께 실행: **./a.out ./test_prog/fib.bin > a.log**
  - 실행 파일과 'linearSearch.bin' 함께 실행: **./a.out ./test_prog/linearSearch.bin > a.log**

cf) makefile을 사용하여 컴파일하는 것 일반적이나, 시간 제약으로 인해 직접 컴파일을 수행하였다.

Screen Capture / Working Proofs

```
============================PROGRAM RESULT============================
Return value R[2] : 0
Total Cycles : 13
Total # of Instructions: 12
Total # of Memory Operation Instructions: 2
Total # of Register Operation Instructions: 8
Total # of Branch Instructions: 0
Total # of Not-Taken Branches: 9
Total # of Jump Instructions: 0
======================================================================
```

<simple.bin의 실행 결과**>**

```
============================PROGRAM RESULT============================
Return value R[2] : 0
Total Cycles : 15
Total # of Instructions: 12
Total # of Memory Operation Instructions: 4
Total # of Register Operation Instructions: 9
Total # of Branch Instructions: 0
Total # of Not-Taken Branches: 11
Total # of Jump Instructions: 0
======================================================================
```

<simple2.bin의 실행결과**>**

```
============================PROGRAM RESULT============================
Return value R[2] : 0
Total Cycles : 34
Total # of Instructions: 56
Total # of Memory Operation Instructions: 12
Total # of Register Operation Instructions: 20
Total # of Branch Instructions: 1
Total # of Not-Taken Branches: 29
Total # of Jump Instructions: 2
======================================================================
```

<fib.bin의 실행결과**>**

```
============================PROGRAM RESULT==============================
Return value R[2] : 0
Total Cycles : 39
Total # of Instructions: 56
Total # of Memory Operation Instructions: 18
Total # of Register Operation Instructions: 22
Total # of Branch Instructions: 3
Total # of Not-Taken Branches: 32
Total # of Jump Instructions: 0
========================================================================
```

<linearSearch.bin의 실행결과**>**

Trial & Errors

Error 1: data dependency 구현 전, pipelining이 원활히 작동이 안됨
Trial 1: 한 사이클을 돌고 latch를 update할 때, 배열 전체를 초기화하는 것이 아닌 1번 배열만 초기화하였다.
Trial 2: 각 instruction 별로 pc값을 update해주었다. pc를 전역변수로 선언하되, 각 pc값을 latch에 저장하였다.

Error2: Return value가 제대로 업데이트되지 않는다.
Trial 2: data dependency가 제대로 처리가 안되었다는 얘긴데 시간부족으로 완성하지 못하였다.

Lesson

이번 프로젝트에서는 헤더파일과 소스파일을 분리함으로써 전체적인 코드의 가독성을 향상시켰으나 기능 구현에 신경을 많이 쓰지 못했다. 가장 핵심기능이었던

data dependency와 control dependency를 제대로 구현하지 못한 것에 많은 아쉬움이 남는다. 다음 프로젝트에서는 가독성과 기능구현 두 마리 토끼를 모두 잡을 것이다.