Project 2: Simulating Pipelined Execution

**Due 11:59pm, April 30(Sun)**

1. Overview

이 프로젝트의 목적은 MIPS ISA를 가지고 5-단계(5-stages)의 파이프라인을 구현 하는 것이다. 지원하는 명령어는 이전 프로젝트에서 구현 했던 것과 동일하다. 구현할 파이프라인의 모델은 강의 자료와 교과서(컴퓨터 구조 및 설계 5th Edition)의 내용을 따른다.

2. Pipeline Implementation

2.1 Pipeline Stages

우리가 구현 할 파이프라인은 아래의 5-단계를 거친다:

• IF: 새로운 명령어 하나를 메모리에서 읽어 온다.

• ID: 읽어 온 명령어를 해석(decode)하고 레지스터 파일을 읽는다.

• EX: ALU 연산을 실행 한다.

- 산술 논리 연산(Arithmetic and logical operation)을 수행 한다.

- ‘load’와 ‘store’ 명령어에 대한 주소 계산을 한다.

• MEM: ‘load’, ‘store’ 명령어에 대한 메모리 접근(access)을 한다.

• WB: 연산 결과를 레지스터에 적는다(write back).

인접한 파이프라인 단계들 사이에, 파이프라인의 상태 레지스터 (pipeline state register) 들을 구현 해야 한다. 각 단계에서 사용될 연산들(operations)을 제어하기 위해, ID 단계에서 제어 신호 (control signal) 들이 생성 되어야 한다.

2.2 Register Timing at ID and WB

MIPS의 파이프라인처럼, WB은 한 사이클(a cycle)의 첫 절반(the first half of a cycle)에서 레지스터 파일에 기록하고, ID는 해당 사이클의 나머지 절반(the second half of a cycle)에 레지스터 파일을 읽어야 한다. 그렇게 해서, ID와 WB 사이의 구조적 하자드(structural hazard)가 없도록 해야한다.

2.3 Memory

IF에서 메모리를 읽는 동작과 MEM 에서의 읽기/쓰기 동작이 한 사이클에 동시에 이뤄지는 이상적인 듀얼-포트 메모리(dual-ported)를 가정하고 진행 하도록 한다. 즉, IF와 MEM 단계에서 메모리 접근에 대한 구조적 하자드 (structural hazard) 는 없다. 또한, 명령어들이 저장 된 메모리 영역이 중간에 갱신(updated) 되는 경우는 없다고 가정한다. 따라서 같은 사이클에, MEM 단계에 있는 store 가 IF 단계에서 불려진 명령어를 변경하는 경우는 고려할 필요가 없다.

2.4 Forwarding

파이프라인이 적용된 아키텍처(pipelined architecture)는 ‘MEM/WB to EX’, ‘EX/MEM to EX’, ‘MEM/WB to MEM‘데이터 포워딩(data forwarding)을 지원 해야 한다. 또한, ‘load’ 명령어에 이어 곧바로 EX 단계에서 레지스터 값을 사용하는 명령어에 의해 발생하는 데이터 의존성(data dependency)은 데이터 포워딩으로는 해결이 안되는 것을 고려하여 구현한다.

2.5 Control Hazard

무조건 분기 명령어(J, JAL, JR)에 대해, 항상 한 사이클의 stall을 파이프라인에 추가해야 한다.

조건 분기 명령어 (BEQ, BNE)에 대해서는, 아래와 같이 두가지 정적 분기 예측(Always Taken, Always Not Taken)을 사용한다. 조건 분기 명령어의 실제 분기 여부는 MEM stage에서 알 수 있다. (**수업시간에 다룬, ID 스테이지에 comparator를 추가하여, 미리 분기 여부를 결정하는 최적화는 고려하지 않는다**)

- Always Taken: 항상 모든 분기가 일어날 것(taken)이라고 예측 한다. 이를 위해, ID 스테이지에서 분기될 주소(target address)를 계산한다. 분기 예측이 성공하면 1 사이클, 분기 예측이 실패하면 3 사이클 동안 flushing으로 인해 파이프 라인 stall이 일어나게 된다.

- Always Not Taken: 항상 모든 분기가 일어나지 않을 것(not taken)이라고 예측 하고, Stall 없이 바로 다음instruction을 가져 온다. 따라서, 분기 예측이 성공하면 stall이 발생하지 않고, 실패하면 flushing으로 인해 3 사이클 동안 stall이 일어나게 된다.

2.6 Stopping the Pipeline

우리가 작성할 프로그램(simulator)은 모든 명령어가 WB 단계를 끝낸 이후 종료 해야 한다. 첫 번째 사이클에서, 첫 번째 명령어를 메모리에서 불러 오고, 마지막 명령어가 N번째 사이클에서 WB 단계에 있다면, 최종 사이클 카운트(the final cycle count, number of cycles)는 N이다.

2.7 Other Conditions (same as the last project)

**상태 정보(States)**: 프로그램은 시스템의 상태를 유지 해야 한다. 필요한 레지스터 집합(R0-R31, PC)과 메모리는 프로그램이 시작 될 때 생성 해야 한다.

**입력 바이너리 파일의 로드**: 주어진 입력 바이너리 파일에 대해서, 로더(the loader)는 텍스트 영역(text section)과 데이터 영역(data section)의 크기를 인식 해야 한다. 텍스트 영역은 메모리 주소 0x400000부터 로드되고, 데이터 영역은 메모리 주소 0x10000000부터 로드 된다.

이 프로젝트에서는, 로더가 스택(stack) 영역을 만들지 않는다.

**초기 상태(Initial states)**:

- PC: PC의 초기 값은 0x400000 이다.

- Registers: 레지스터 0~31 의 값은 모두 0이다.

- Memory: 로드 된(loaded) 텍스트와 데이터 영역의 값을 제외 하고, 나머지 값들은 모두 0이다.

**지원하는 명령어 집합(Supported Instruction Set)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ADDIU | ADDU | AND | ANDI | BEQ | BNE | J |
| JAL | JR | LUI | LW | LA\* | NOR | OR |
| ORI | SLTIU | SLTU | SLL | SRL | SW | SUBU |

3. Pipeline Register States

각 스테이지 사이에 파이프라인 레지스터 상태(pipeline register states)를 추가해야 한다. 다음은 가능한 레지스터 내용들(contents)인데, 이외에도 다른 상태 정보들(states)을 추가해야 한다.

- IF\_ID.Instr: 32비트 명령어

- IF\_ID.NPC: 다음 32비트 PC 값(PC+4)

- ID\_EX.NPC: 다음 32bit PC 값

- ID\_EX.REG1: REG1 값

- ID\_EX.REG2: REG2 값

- ID\_EX.IMM: 상수 값(Immediate value)

- EX\_MEM.ALU\_OUT: ALU 출력(output)

- EX\_MEM.BR\_TARGET: 분기 주소(Branch target address)

- MEM\_WB.ALU\_OUT: ALU 출력

- MEM\_WB.MEM\_OUT: 메모리 출력(memory output)

어떤 필드(field)가 각각의 파이프라인 레지스터에 필요 한 지에 대한 설계를 유의해서 해야 한다. 그리고 **추가한 각각의 필드(field)의 의미에 대해 “README” 파일에 설명을 적어 제출 시 같이 내도록** 한다.

4. Simulation Option and Output

4.1 Options

Usage: proj2 <-atp | -antp> [-nobp] [-m addr1:addr2] [-d] [-p] [-n num\_instr] inputBinary

• -atp: Always Taken 분기 예측을 사용한다.

• -antp: Always Not Taken 분기 예측을 사용한다.

• -m: 시뮬레이션이 끝나고 나서 메모리 덤프를 수행 한다.

• -d: 레지스터 파일 내용과 현재 PC의 값을 **매 사이클 마다** 출력 한다.

• -p: **매 사이클 마다** 각 파이프라인 단계의 명령어들(PCs)을 출력 한다.

e.g. CYCLE N: 0x3004 | 0x3003 | 0x3002 | 0x3001 | 0x3000

4.2 Formatting Output

선택적으로 메모리 내용을 추가 하여, PC와 레지스터 내용을 출력 해야 한다.

• 레지스터 내용

• CYCLES: 입력 프로그램을 종료 하는데 필요 한 사이클의 수

• -p 옵션이 사용되었을 경우, 매 사이클 마다 파이프라인 상태를 출력 한다. 위의 ‘4.1’ 참고.

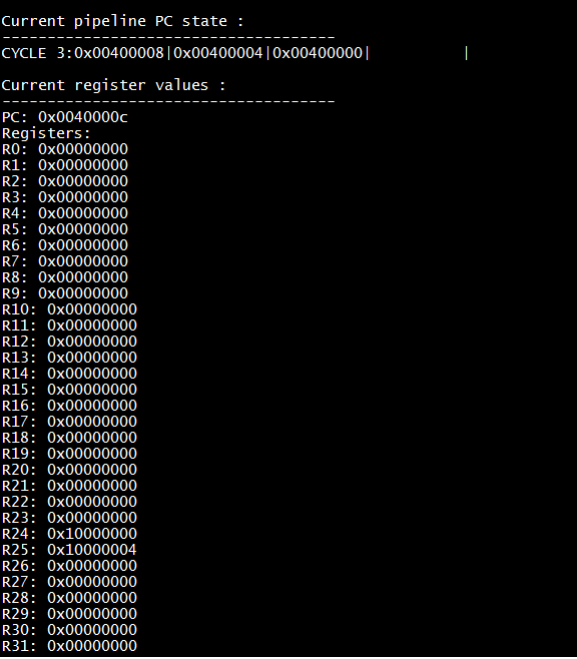
• -d 옵션이 사용되었을 경우, 매 사이클 마다 레지스터의 상태를 출력 한다. 위의 ‘4.1’ 참고.

입력으로 주어진 MIPS 바이너리(이전 프로젝트에서 생성한 .o 파일)에 대해, 시뮬레이터는 MIPS ISA 실행을 흉내 내야 한다. 출력 형식의 예는 다음 장에 있다.

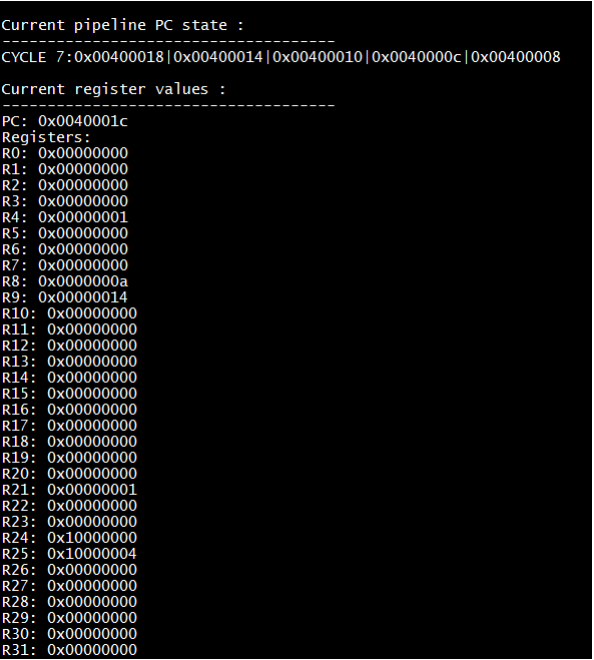
5. Hand in

zip 또는 tar 형식으로 소스 코드와 “README” 파일을 압축하여 제출 하도록 한다(3. Pipeline Register States 참고). 압축 파일의 이름은 팀 이름으로 한다(team\_name.tar 또는 team\_name.zip). 예를 들어, 팀14 일 경우, “team14.zip” 또는 “team14.tar”가 압축 파일의 이름이 된다.

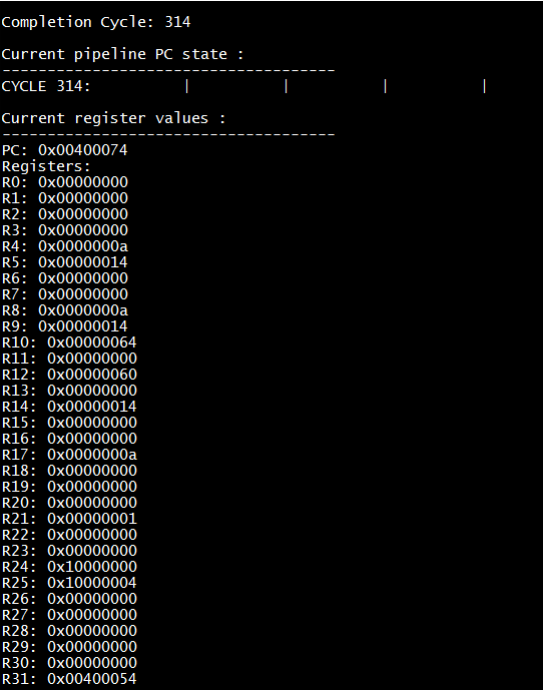
과제 제출은 압축 파일을 포함하여, [nickeys@dgist.ac.kr](mailto:nickeys@dgist.ac.kr) 로 보내도록 한다(**기한 내 제출 필수!**).



[Figure 1. Output format at cycle 3]



[Figure 2. Output format at cycle 7]



[Figure 3. Output format at the program completion]

6. ETC

6.1 Available language

**사용 가능한 언어는 C, C++, Python** 3가지 이며, 어떤 언어를 사용해도 평가상의 불이익은 전혀 없다.

6.2 Output

**이번 프로젝트는**, 이전 프로젝트처럼, **출력에 사용하는 함수가 제공되지 않는다**. 따라서 ‘출력 형식의 예’를 참고하여, 출력 하는 코드를 작성하여 사용 하도록 한다.