CSE305
Project 3. Simulating Pipelined Execution

201611057 김준우

1. Introduction

이 프로그램은 MIPS(Big-endian) 어셈블리 코드를 바이너리 코드로 변환된 'object.o'파일을 실행시켜주는 MIPS ISA 에뮬레이터에서 5단계의 pipeline을 구현한 것이다.

2. Environment

VMWare Workstation 16을 사용하여 Ubuntu 20.04.4를 구동하였으며, Ubuntu 내부에서는 VSCode로 코딩을 한 후, 터미널을 통해 구동을 확인하였다. 사용한 언어는 C++이며 컴파일 환경은 c++ 9.4.0 version이다. 프로그램은 Ubuntu의 Terminal 콘솔창에서 c++ -o runfile project3.cpp 명령어로 컴파일 후 실행이 가능하다.

3. Explanation

해당 프로그램의 코드는 main함수 영역과 연산처리 과정을 돕는 함수로 구성되어있으며, 사용한 STL은 [iostream, fstream, vector, bitset, string, algorithm, vector, sstream, stdlib.h, cstring, cmath] 이다.

각각의 MIPS operator에 따른 연산처리는 project2에서 사용했던 같은 함수를 활용했고, 아래의 세 함수를 통해 연산이 처리된다.

void r_type_oper(string str)	string type인 str을 input으로 받아, operator를 구분하여 각 함수에서 적절한 연산처리를 한다.
void I_type_oper(string str)	
void j_type_oper(string str)	

그리고 연산된 결과는 메모리 또는 레지스터에 저장되는데, 이는 vector구조로 구현하였다.
또한, 연산에 도움을 주는 간단한 함수를 아래와 같이 정의하였다.

int complement2_16bit(string bits)	16비트인 2의 보수를 정수로 변환
string hex_to_bi(string hexnum)	16진수를 2진수의 string으로 변환
string detect_oper(string input_str)	바이너리코드를 받아 operator를 구분해주는 함수

그리고 pipeline을 구현하기 위해, 아래와 같은 stage 함수를 만들었다.

IF(string str)	PC에 알맞은 Instruction을 pipeline에 fetch해준다.
ID(string str)	ID단계 연산처리를 하며, load명령어의 stall과 branch의 주소계산
EX(string str)	EX단계 연산, ALU연산, 연산 후 pipeline state 레지스터에 저장
MEM(string str)	MEM단계, Branch의 분기 결과에 따른 stall 관리
WB(string str)	WB 메모리또는 레지스터에 저장

메인함수에서 바이너리 코드의 파일을 line별로 구별하여 각각의 알맞은 vector로 분류를 실행한 후, operator의 연산이 진행이 된다. pipeline의 경우 IF, ID, EX, MEM, WB가 순차적으로 실행이 되며, 이전 stage에서 처리된 instruction이 다음 stage로 전달되는 것을 구현하

기 위해서 if_inst, id_inst 등과 같은 변수를 활용하여 instruction을 다음으로 넘겨줄 수 있도록 구현했다.

[Data Hazard] 파이프라인 상태 레지스터 또한 각각의 변수로 선언한 후, 각 단계에서 파이프라인 레지스터에 저장하도록 구현하였으며, 앞선 instruction과의 data 의존성이 발생하는 경우, 데이터 전방전달이 될 수 있도록 구현했다. load 명령어의 경우 ID단계에서 한 사이클의 stall이 실행될 수 있도록 구현하였다.

[Control Hazard] Jump instruction의 경우, ID단계에서 판단하여 한 사이클의 지연이 실행될 수 있도록 구현했다. 그리고 Branch instruction은 ID단계에서 주소를 계산한 후, MEM 단계에서 분기 예측기의 조건에 맞춰 예측결과에 맞는 적절한 stall이 발생할 수 있도록 구현했다.

이때 stall을 구현하기 위해서 if_stall, id_stall과 같은 변수를 사용했으며, stall변수 값이 1인 경우 stall을 발생시키고, stall_manage()함수를 이용하여 이전 cycle에서 발생한 stall이 다음 cycle에서 다음 stage에 stall이 전달될 수 있도록 구현하였다.

[출력] '-d' 옵션이 있는 경우 및 '-m' 옵션이 있는 경우를 고려하여 코드를 구성했다. 또, '-d'옵션이 없는 경우에 연산과정이 종료된 이후에 레지스터와 해당하는 메모리 값을 출력하도록(-m옵션이 있는 경우) 설정하였다. 또한 '-p' 옵션이 있는 경우, pipeline의 상태를 출력하도록 설정했다. 또한, 분기예측기를 command에서 선언하지 않는 경우, "Error : You have to command with atp or antp!"와 같은 에러 문구를 출력하며 프로그램이 종료되도록 구성했다. 덧붙여 input binary file이 입력되지 않았을 때는 "Error : You have to command with input file!"라는 에러 문구와 함께 프로그램이 종료되도록 구성했다.

프로그램은 우분투의 Terminal 콘솔창에서 c++ -o runfile project3.cpp로 컴파일 후 실행이 가능하다. 컴파일을 진행한 후 [$./runfile -atp -m 0x40000:0x40040 -p -n 25 sample.o], [$./runfile -antp -m 0x40000:0x40040 -d -p -n 25 sample.o] 와같은 명령어로 실행할 수 있다. 이때 각 실행 옵션에 따라 알맞은 출력결과가 콘솔창에 나타난다.

4. Results

프로그램은 정상적으로 실행되었다.