Project 2: Building a Simple MIPS Emulator

Submitter: Jiwon Jang (202211167)

1. 컴파일 및 실행 방법

언어는 C++를 사용하였고, 컴파일러는 g++ 9.5.0을 이용하였다. 다음 명령어를 치면 컴파일할 수 있다.

```
g++ -o homework2 homework2.cpp
```

이후 파일 실행은 다음과 같이 진행했다. ./homework2 sample.o 순서로 입력하면, 프로그램이 binary file 읽어와 모든 인스트럭션이 수행되고난 후의 결과를 출력을 해준다. 이 외에도 -m, -n, -d 옵션을 입력하면 assignment에서 요구한 출력이 나온다.

-m begin_memory:end_memory: begin_memory 부터 end_memory에 담겨있는 내용 출력

-n num_of_instruction: num_of_instructruction 만큼의 instruction 수행 후 레지스터 값 출력

-d: instruction이 수행될 때마다 레지스터 값 출력

주의 사항

-n 0 option과 -d option이 함께오면 아무것도 출력되지 않는다.

-m option과 -d option이 함께오면 매 출력마다 메모리가 출력된다.

2. 클래스 및 함수 설명

이번 과제에서 구현한 클래스와 함수에 대한 설명이다. 코드의 흐름을 설명하기 위한 배경 설명이다.

<library>

총 5개의 library를 이용하였다.

iostream: 표준 입출력을 지원한다.

fstream: 파일 생성 및 read/write를 지원한다.

string: 문자열 형식을 지원한다.

vector: vector 자료형을 지원한다.

bitset: 이진수 변환을 지원한다.

<class>

class는 Memory(instruction, data), Register, Program Counter를 관리하기 위해서 사용하였다.

1) PROGRAMCOUNTER class definition

```
private:
```

int program_counter: PC값을 담고있다.

vector<DATA> data_list*: 해당 라벨에 속해 있는 data들이 담겨있는 리스트이다.

string line: 해당 라벨이 속해있는 '줄'이 담겨있다.

public:

PROGRAMCOUNTER(): PRGRAMCOUNTER 클래스의 constructor이다. constructor에서는 initializing 진행한다. PC를 0x400000로 초기화 하고, PC_pointer 값을 해당 instance로 초기화 한다.

static PROGRAMCOUNTER PCPointer*: 모든 함수에서 PC값에 접근할 수 있게 하기 위한 static pointer 변수이다.

void UpdateProgramCounter(int update_address): PC값을 업데이트 해주는 함수이다.

int GetPC(): PC값을 가져오는 함수이다.

2) REGISTER class definition

private:

int register_num: 레지스터 번호가 담겨있다.

int data: 레지스터에 담겨있는 데이터가 담겨있다.

public:

REGISTER(int register_num): REGISTER 클래스의 constructor이다. constructor에서는 레지스터 번호를 초기화 하고, register_list에 해당 instance를 추가해 준다.

static vector<REGISTER>register_list*: RGISTER class의 instance들이 담겨있는 리스트이다.

int ReadRegNum(): 레지스터 번호를 읽기 위한 함수이다.

void UpdateData(int update_data): 레지스터 값을 업데이트하기 위한 함수이다.

int ReadData(): 레지스터 데이터를 읽기 위한 함수이다.

3) TEXT_MEMORY class definition

private:

string line: Text가 담겨있다.

int address: Text의 주소가 담겨있다.

public:

TEXT_MEMORY(string line, int address): TEXT_MEMORY 클래스의 constructor이다. constructor에서는 line과 address를 초기화 해주고, text_memory_list에 해당 instance를 추가해 준다.

static vector<TEXT_MEMORY>text_memory_list*: TEXT_MEMORY class의 instance들이 담겨있는 리스트이다.

string ReadData(): line(Text)을 읽기 위한 함수이다.

int ReadAddress(): line(Text)가 담겨있는 주소를 읽기 위한 함수이다.

void UpdateData(string update_data): line(Text)을 업데이트 하기 위한 함수이다.

3) DATA_MEMORY class definition

private:

string line: Data가 담겨있다.

int address: Data의 주소가 담겨있다.

public:

DATA_MEMORY(string line, int address): DATA_MEMORY 클래스의 constructor이다. constructor에서는 line과 address를 초기화 해주고, data_memory_list에 해당 instance를 추가해 준다.

static vector<DATA_MEMORY>data_memory_list*: DATA_MEMORY class의 instance들이 담겨있는 리스트이다.

string ReadData(): line(Data)을 읽기 위한 함수이다.

int ReadAddress(): line(Data)가 담겨있는 주소를 읽기 위한 함수이다.

void UpdateData(string update_data): line(Data)을 업데이트 하기 위한 함수이다.

<other functions>

main 함수를 제외한 함수의 종류는 크게 일곱 가지로 구분할 수 있다.

- 진수 변환을 하는 함수들: *Hexadecimaler(string num), Binaryer(int num, int size)*
- 출력을 위한 함수들: *PrintRgisterData(), PrintMemoryContents(int begin_address, int end_address)*
- 산술 연산을 함수: *ALU()*
- 레지스터와 메모리에 접근해 데이터를 읽고, 쓰는 함수들: *int ReadReg(int RegNum), void WriteReg(int RegNum, int Data), string ReadMem(int Address), void WriteMem(int Address, string Data)*
- 각 instruction의 detail한 부분을 다루는 함수들: addiu(string line), addu(string line), … etc …
- 각 instruction을 관리하는 함수들: int R_format(int int_line), int Instruction(string line)
- Emulator 함수: void Emulator(ifstream& input, int d, int n, int num_of_instruction, int begin_address_int, int end_address_int, int m)

이를 조금 더 자세히 알아보자.

1) 진수 변환을 하는 함수들

Hexadecimaler(string num): 2진수를 16진수로 변환하는 함수이다. 이 때 16진수의 자리를 고정하지 않는다.

Binaryer(int num, int size): 10진수를 2진수로 변환하는 함수이다. bitset 라이브러리를 사용하였다.

2) 출력을 위한 함수들

PrintRgisterData(): 레지스터를 읽어와 레지스터의 정보를 assignment가 요구하는 출력 형식으로 출력하는 함수이다.

PrintMemoryContents(int begin_address, int end_address): 메모리를 읽어와 메모리의 정보를 assignment가 요구하는 출력 형식으로 출력하는 함수이다.

3) 산술 연산을 함수

ALU(): ALU control(ex. 0000: AND, 0001: OR … etc …)과 인자 두 개를 받아서 ALU 연산을 하는 함수이다.

4) 레지스터와 메모리에 접근해 데이터를 읽고, 쓰는 함수들

int ReadReg(int RegNum): 레지스터의 값을 읽는 함수이다.

void WriteReg(int RegNum, int Data): 레지스터의 값을 업데이트하는 함수이다.

string ReadMem(int Address): 메모리의 값을 읽는 함수이다.

void WriteMem(int Address, string Data): 메모리의 값을 업데이트하는 함수이다.

5) 각 instruction의 detail한 부분을 다루는 함수들

이 함수들은 assignment 1에서 명시된 detail을 바탕으로 작성하였다. 각 instruction에서는 해당 instruction에 맞는 동작을 하고, 동작을 한 이후에는 다음에 올 PC값을 리턴 해준다.

6) 각 instruction을 관리하는 함수들

int R_format(int int_line): function field를 바탕으로 instruction들을 처리하는 함수이다.

int Instruction(string line): opcode를 바탕으로 instruction들을 처리하는 함수이다.

6) Emulator 함수

void Emulator(ifstream& input, int d, int n, int num_of_instruction, int begin_address_int, int end_address_int, int m): assignment 2의 요구에 맞게 instruction을 실행시키고, 시스템(reg, mem)을 관리하는 함수이다.

다음은 Emulator 함수의 일부이다.

```
        for (int i=0; i<TEXT_MEMORY::text_memory_list.size(); i++) {
                    if (new_program_counter->GetPC() ==
TEXT_MEMORY::text_memory_list[i]->ReadAddress()) {
                                // PC 주소랑 같은 인스트럭션을 찾았을 때
                  인스트럭션 실행
                                        Update_PC =
Instruction(TEXT_MEMORY::text_memory_list[i]->ReadData());
```

위와 같이 TEXT_MEMORY를 돌며, PC값과 일치하는 주소의 instruction을 실행시킨다. 이후에는 밑과 같이 PC를 업데이트한다.

```
PROGRAMCOUNTER::PCPointer->UpdateProgramCounter(Update_PC);
```

n option이 없을 때는 밑과 같은 조건문을 통해서 종료 시점(PC+4의 메모리에 아무것도 없을 때)을 알려준다.

```
if (new_program_counter->GetPC() !=
TEXT_MEMORY::text_memory_list.back()->ReadAddress()+4
```

<main function>

main 함수는 input 값을 해석하여, 해석 결과(n, m, d option의 여부, 및 각 option에서의 정보)를 Emulator 함수에 전달하는 역할을 한다.

3. 코드 아이디어

assignment에 사용한 몇 가지 코딩 아이디어이다.

1) shift: 32bit에서 opcode, rs, rt, rd, imm, offset, shamt, function field를 읽기 위해서 shift 연산을 사용하였다.

   예시로 rs 값을 읽어보자. 이 때는 32bit를 right shift 21 해주고, 이를 00000000000000000000000000011111와 & 비트 오퍼레이션을 사용하여, bit 연산을 취해준다.

```
(int_line>>21)&0x1F)
```

2) static cast: imm이나 offset을 sign extension 하기 위해서 사용한 코드이다. imm이나 offset을 16bit 기준으로 sign extension 한다.

```
static_cast<signed int>(static_cast<int16_t>(int_line & 0xFFFF))
```

4. 코드 흐름

코드는

1) main 함수를 통한 파일 read 및 input 해석

2) Emulator 함수를 통한 input 요구에 맞는 실행 (ex. -n 옵션이 있다면 -n 옵션 뒤에 오는 숫자 만큼 실행)

3) Emulator 함수를 통해 PC 값으로 이동

4) PC가 가르키는 instruction에 접근한 후 ALU 연산을 통한 instruction 실행 및 레지스터, 메모리 업데이트

5) PC 값 업데이트

6) Emulator 함수를 통한 input 요구에 맞는 출력 제공 (-d 옵션이 있다면 매 instruction마다 출력, 없다면 대기)

7) 다시 3),4),5),6) 반복

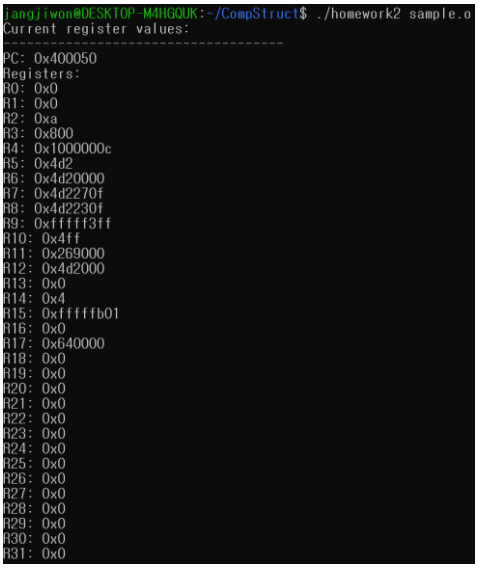
8) PC 값이 non을 가르키면 Emulator 종료 후 input 요구에 맞는 출력 제공

과정으로 실행된다.

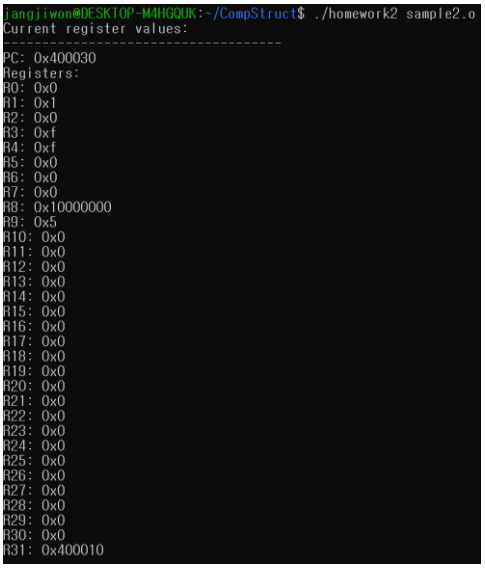
5. 결과

결과는 다음과 같다.

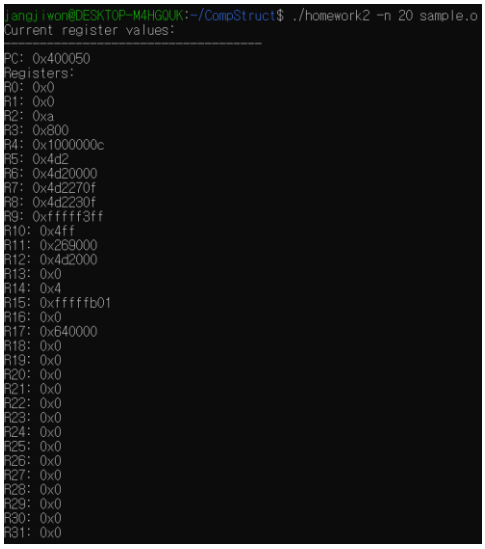
1) ./homework2 sample.o (without any option)



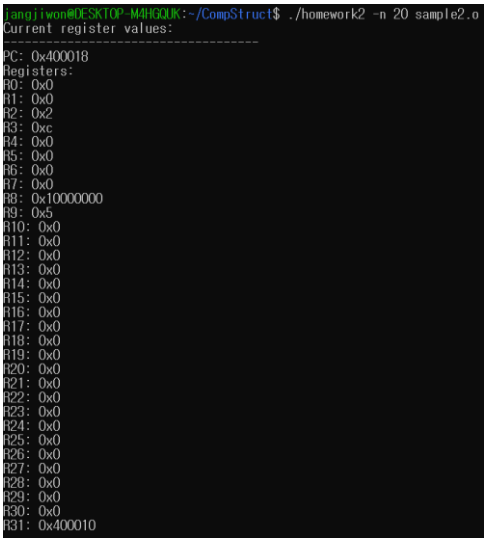
2) ./homework2 sample2.o (without any option)



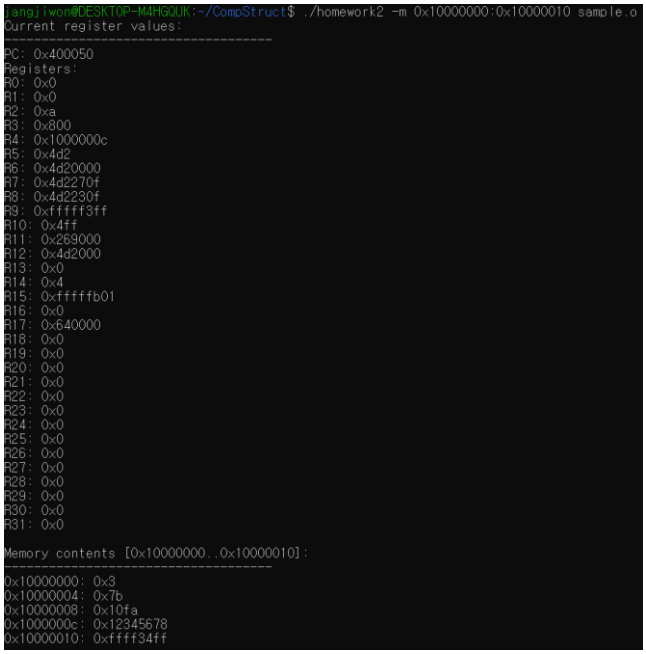
2) ./homework2 -n 20 sample.o (with -n option)



3) ./homework2 -n 20 sample2.o (with -n option)



4) ./homework2 -m 0x10000000:0x10000010 sample.o (with -m option)



5) ./homework2 -m 0x10000000:0x10000010 sample2.o (with -n option)