Project 2: Building a Simple MIPS Emulator

Due 23:59, 15th April, 2023

TA: Minhyeok An(mh.an@dgist.ac.kr), Jungbo Kim(jbkim@dgist.ac.kr)

Sungju Kim(sungju_kim@dgist.ac.kr), YunHyeong Jeon(yhjeon@dgist.ac.kr)

Introduction

두 번째 과제의 목적은 MIPS 인스트럭션을 실행시킬 수 있는 간단한 MIPS 에뮬레이터를 구현하는 것이다. 앞서 진행된 첫 번째 과제가 어셈블리 코드를 바이너리로 바꾸는 작업이었다면, 이번 과제의 목표는 생성된 바이너리를 입력 받아 메모리에 로드하고, 인스트럭션을 실행시키는 에뮬레이터를 구현하는 것이다. 인스트럭션이 수행됨에 따라 레지스터와 메모리의 상태가 변하게 된다.

1. Emulation Details

이번 과제에서 만들 에뮬레이터는 과제 1 의 출력인 바이너리 파일을 사용하게 된다. 에뮬레이터는 이 바이너리 파일을 통해 MIPS ISA 가 실행되는 것을 모사해야 한다.

A. States

에뮬레이터는 MIPS 인스트럭션들을 실행시킴과 동시에 시스템(레지스터 및 메모리)의 상태를 올바르게 유지해야 한다. 사용되는 레지스터(R0-R31, PC)와 메모리는 에뮬레이터가 실행되는 순간에 생성 및 초기화 되어야 한다. 각 장소에는 음수가 올 수 있으며, 음수 값의 경우 two's complement 로 처리하여 sign extended 상태로 저장된다. 단, 실행에 있어서는 각 instruction 에 따라 signed/unsigned 로 나뉘어 지며 이는 이전 과제의 명세서에 따로 언급되어 있다.

B. Loading an input binary

이전 과제에서 MIPS 바이너리 파일의 첫 번째 줄과 두 번째 줄이 각각 text 와 data 영역의 크기를 표현하도록 작성했다. 에뮬레이터는 바이너리 파일을 받아 text 와 data 영역의 크기를 읽어서 에뮬레이트된 메모리를 생성해야 한다. 이전 과제의 메모리 레이아웃을 참조하여, text 영역은 0x400000, data 영역은 0x10000000 의 주소로부터 시작하도록 해야 한다.

이번 과제는 학생들의 이해를 돕기 위해 작성되었으므로, 단순한 구조의 loader 만 구현하면 된다. 이번 과제의 loader 는 stack 영역을 생성하지 않는다.

C. Initial states

- PC (Program Counter) : PC 의 초기값은 text 영역의 시작 주소인 0x400000 이다.

- Registers : 모든 레지스터(R0-R31)는 초기값으로 0 을 갖는다. jal 등의 인스트럭션에서 사용하는 $ra 레지스터는 R31 에 해당한다.
- Memory : 불러온 data 영역과 text 영역을 제외하고 모든 메모리는 초기값으로 0 을 갖는다.

D. Instruction execution

에뮬레이터는 현재 PC 값(instruction address)을 참조하여 해당 주소의 인스트럭션인 4 바이트 바이너리를 읽어온다. 읽어온 4 바이트 바이너리를 해석하여 어떤 명령어인지, 어떤 인자를 가지고 있는지 파악한 후 해석된 명령어를 실행한다. MIPS ISA 에 기반하여, 각 명령어가 실행될 때마다 PC, 레지스터, 메모리의 값이 정확하게 변경되어야 한다.

E. Completion

에뮬레이터는 주어진 인스트럭션을 모두 수행한 후 종료된다. 단, 에뮬레이터의 매개변수에 옵션(`-n`)으로 수행할 인스트럭션의 개수가 주어진다면, 주어진 개수까지의 인스트럭션만을 실행한다.

F. Support Instruction Set

지원하는 인스트럭션은 과제 1 에서 구현되었던 인스트럭션과 동일하다. 인스트럭션의 동작 방법 등은 과제 1 의 설명에서 확인할 수 있다.

`j` 와 `jal` 인스트럭션의 경우, 분기(jump) 대상의 주소를 4 로 나누어 하위 2 비트를 제거하고, PC 의 상위 4 비트로 쓰이는 부분을 제거한 총 26 비트를 분기의 target 으로 지정해야 한다.

`lb` 인스트럭션은 8 비트를 load 한 뒤 32 비트로 sign-extension 후 지정된 레지스터에 값을 저장한다.

`sb` 인스트럭션은 least significant 8 비트를 지정된 메모리 주소에 저장한다.

MIPS 는 big endian 으로 실행되며, 이 부분을 유의하여 구현하도록 한다.

2. Emulator Options

```
$./runfile [-m addr1:addr2] [-d] [-n num_instruction] input file
```

- `-m` : 프로그램이 종료될 때 메모리 주소 범위 (addr1 ~ addr2)에 있는 내용들을 출력한다. 해당 주소가 data section 의 주소(0x10000000 에서 시작) 혹은 text section 의 주소(0x400000 에서 시작)를 벗어난 범위를 가리키고 있을 경우, 해당 주소에는 값이 할당되지 않았으므로 0x0 을 출력한다. 디폴트 값은 없기 때문에 본 플래그가 있을 때는 항상 주소 값을 입력해주어야 한다.

- `-d` : 한 인스트럭션의 실행이 끝날 때마다 모든 레지스터(R0-R31, PC)의 내용을 출력한다. –m 옵션으로 출력할 메모리 주소 범위가 지정되어 있다면 지정된 메모리 범위의 내용도 출력한다.
- `-n` : 수행될 명령어의 개수를 지정한다. 디폴트 값은 없기 때문에 본 플래그가 있을 때는 항상 개수를 지정해주어야 한다.

`-d` 옵션이 지정되지 않았을 경우에는 프로그램이 종료되는 시점에 모든 레지스터(R0-R31, PC)의 내용을 출력한다. 만약 `-m` 옵션을 통해 출력할 메모리가 지정되었다면 프로그램이 종료되는 시점에 지정된 메모리의 내용 또한 출력해준다.

3. Coding and Execution

이번 과제에서 사용되는 프로그래밍 언어 및 컴파일 환경은 과제 1 과 동일하다.

A. Environment

제출된 코드는 Ubuntu 20.04, Windows Subsystem for Linux(Ubuntu 20.04) 환경에서 테스트될 것이다. 위 환경 중 하나에서 코드가 실행 가능하면 채점 가능하다. 이외의 OS 에서 코딩을 진행하는 경우 OS API 등의 코드로 인해 코드 컴파일이 불가능한 경우가 있을 수 있으니, 위 환경 중 최소 하나 이상에서 테스트한 후 제출하여야 한다.

B. Program Language

프로그래밍 언어는 C, C++만이 허용된다. 이 언어들 중에 어떤 것을 사용하여도 좋다.

- C 혹은 C++의 경우, gcc 4.8, g++ 4.8 버전 이상의 컴파일러를 이용하여 채점을 진행하게 된다.
- C 혹은 C++의 경우 Microsoft Visual Studio를 사용해야만 컴파일이 되는 경우, **컴파일이 불가능한 것으로 간주**한다. 또한, Microsoft Visual Studio 프로젝트 파일을 제출하는 것은 인정되지 않는다.

C. Execution command

```
$./runfile [-m addr1:addr2] [-d] [-n num_instruction] <input file>
```

완성된 프로그램은 위 명령어를 통해 실행되며, object file (*.o) 을 입력으로 받는다. 추가적인 옵션에 따라 PC, 레지스터, 메모리를 **콘솔에 출력**해야 한다.

D. Input format

입력 파일은 이전 과제에서 결과물로 생성된 object 파일(*.o)이다. 한 줄에 최대 4 바이트씩 16 진수로 적힌 숫자로 이루어져 있으며 첫 번째 줄과 두 번째 줄은 각각 text 영역과 data 영역의 크기를 의미한다. 그 다음 줄부터는 PC 순으로 text 영역 내의

인스트럭션을 나열하고 있다. Text 영역을 모두 나열한 후에는 data 영역 내의 word 를 나열하고 있다.

E. Output format

프로그램 종료 시에 **표준 출력(printf, std::cout 등)으로 PC 와 레지스터의 내용을 콘솔에 출력**해야 한다. 추가적으로 `-m` 옵션을 통해 출력할 메모리가 지정되었다면 해당 메모리 범위의 내용도 출력해야 한다. `-d` 옵션이 주어졌을 경우 매 인스트럭션 마다 레지스터의 내용을 출력해야 한다.

- 명령어가 다음과 같이 실행되었을 경우, 아무 인스트럭션도 실행하지 않고, 다음 그림(Figure 2)과 같이 결과를 출력해야 한다.
  `$./runfile -n 0 input.o`

```
> ./runfile -n 0 input.o
Current register values:
------------------------------------
PC: 0x400000
Registers:
R0: 0x0
R1: 0x0
R2: 0x0
R3: 0x0
R4: 0x0
R5: 0x0
R6: 0x0
R7: 0x0
R8: 0x0
R9: 0x0
R10: 0x0
R11: 0x0
R12: 0x0
R13: 0x0
R14: 0x0
R15: 0x0
R16: 0x0
R17: 0x0
R18: 0x0
R19: 0x0
R20: 0x0
R21: 0x0
R22: 0x0
R23: 0x0
R24: 0x0
R25: 0x0
R26: 0x0
R27: 0x0
R28: 0x0
R29: 0x0
R30: 0x0
R31: 0x0
```

Figure 2. Output format at instruction 0

- 명령어가 다음과 같이 실행되었을 경우, 아무 인스트럭션도 실행하지 않고 다음 그림(Figure 3)과 같이 결과를 출력해야 한다. 유의할 점은, 다음 그림은 **아무 내용이 없는 바이너리**를 입력 파일로 지정한 것이다. 실제로 입력 바이너리 파일에 데이터 및 인스트럭션이 있을 경우, 해당 값이 먼저 메모리에 올라온 이후 인스트럭션이 실행된다. 그러므로, 다음 명령어에 의해 인스트럭션이 전혀 실행되지 않더라도 바이너리의 data 및 text 영역에 해당하는 내용이 지정한 메모리 범위에 포함될 경우에는 해당 내용을 출력해야 한다.
  `$./runfile –m 0x400000:0x400010 –n 0 input.o`
- each iteration 마다 요구된 출력 형식에 맞춰 상태를 출력해야 한다. 과제에서 요구하는 출력 형식은 다음 그림(Figure 3)와 같으며, 함께 올려진 파일을 참고해도 좋다. -m 옵션이 없는 경우는 위의 그림(Figure 2)과 같이 표현하면 된다. 출력 형식을 지키지 않는 경우엔 감점되니 주의해야 한다.

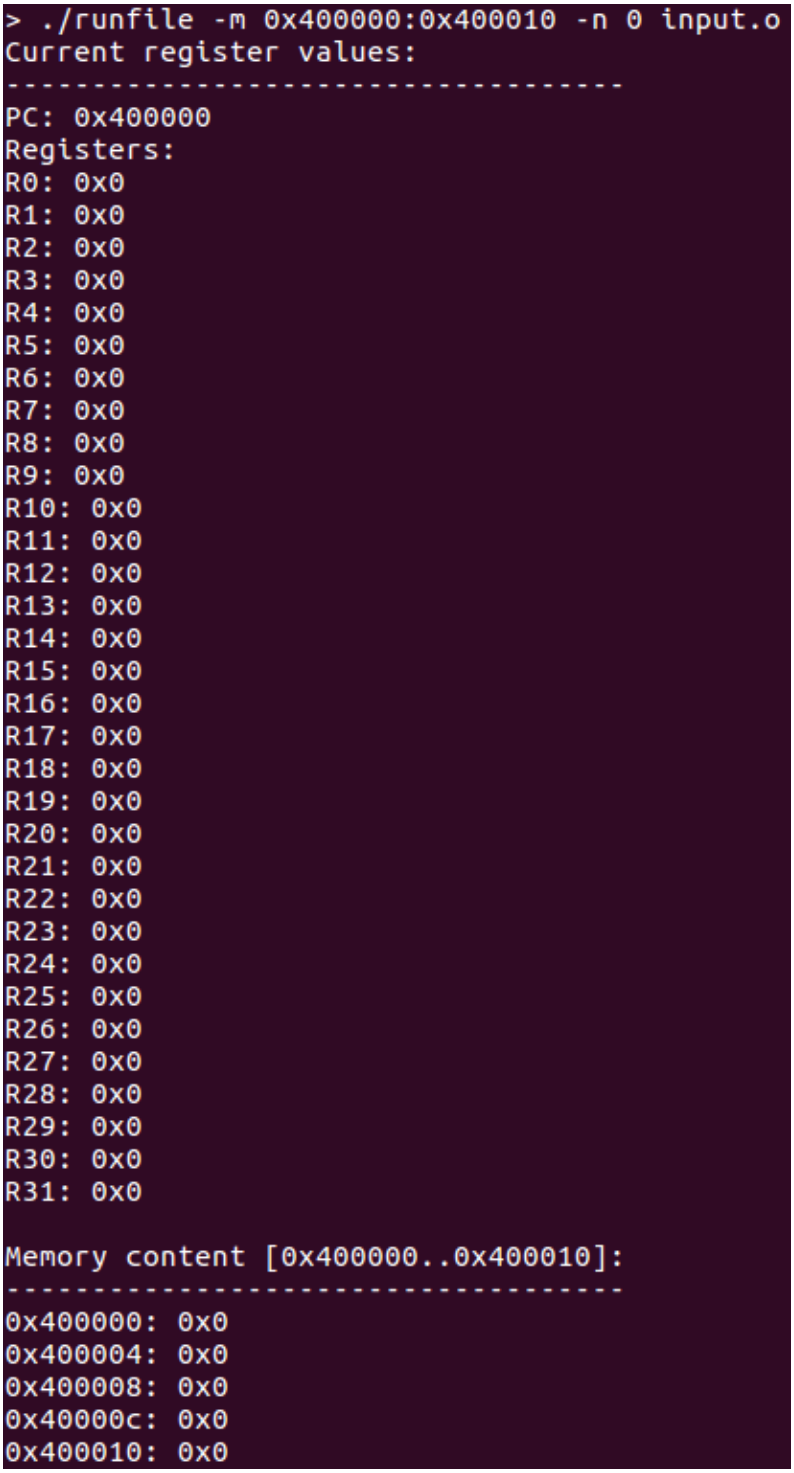

Figure 3. Output format at instruction 0 with memory

4. Report

과제를 수행한 학생들은 다음 내용이 들어있는 보고서를 작성하여 소스 파일과 함께 제출하여야 한다.

- 자신이 작성한 과제에 대한 간략한 설명
- 과제의 컴파일 방법 및 컴파일 환경(사용한 OS, 컴파일러 버전)
- 과제의 실행 방법 및 실행 환경

컴파일 방법 및 컴파일 환경에 대한 보고서 작성 예시는 다음과 같다.

- ✓ WSL Ubuntu 20.04 환경에서 gcc 8.4.0 버전을 이용하여 다음 명령어로 컴파일 하였다.
  `gcc -o main main.c -std=c++14`
- ✓ Ubuntu 20.04 환경에서 g++ 8.4.0 버전을 이용하여 컴파일 하였다. 첨부한 makefile 을 소스 코드와 같은 폴더에 넣고, 리눅스 커맨드라인에서 `make` 명령어로 컴파일 할 수 있다.

실행 방법 및 실행 환경도 위와 같은 형식으로 작성한다. 단, 4-C Execution Command 에 지정한 명령어와 다른 명령어로 프로그램이 실행되는 경우, 감점의 요인이 될 수 있다. **4-B Programming Language 에 명시 해놓은 gcc, g++ 이외의 컴파일러를 사용한 컴파일 방법을 보고서에 컴파일 방법으로 기재하였거나, 보고서에 컴파일 방법을 명시 해놓지 않은 경우 소스 코드를 컴파일 할 수 없는 것으로 간주한다.**

보고서는 ".pdf" 형식으로 변환하여 제출한다. 이를 지키지 않는 경우 감점된다. 더불어 보고서 내에 소스 코드를 기재할 필요는 없다.

5. Submission

프로젝트 결과물(소스 파일) 및 보고서를 zip 형식으로 압축하여 LMS 에 해당 과제 게시글로 제출하도록 한다. **압축 파일의 이름은, 학번_이름.zip** 으로 **꼭 양식을 지키도록** 한다. 양식에 맞지 않을 경우, 감점의 요인이 된다.

e.g. 202311999 김철수 학생의 경우, **202311999_김철수.zip** 형식으로 제출.

6. Notice

A. Due date: **23:59, 15th April, 2023**

B. Penalty

- Late due

Case	Penalty
1 일 지연	점수 10% 감점
2 일 지연	점수 30% 감점
3 일 지연	점수 50% 감점
4 일 이상 지연	0 점 처리

- Cheating 적발 시 D+ 이하의 학점 부여
- Compile 혹은 실행이 안 될 경우 조교가 학생에게 연락하여 코드 수정 요청. 마지막 Late due 까지 compile 이 안 될 경우 0 점 처리
- 파일 이름, 실행 방법 등이 규정한 양식과 다를 경우 감점 처리.
- 보고서에 컴파일 방법을 작성하지 않거나, 과제에서 규정한 컴파일러를 사용한 컴파일 방법이 적혀 있지 않을 경우, 컴파일이 불가능한 것으로 처리.

C. TA Contact

TA 에게 질문 사항 또는 기타 용건이 있다면, 네 명의 TA 모두에게 질문 메일로 문의할 것(e.g. 수신 1 명, 수신자 제외 참조 3 명). TA 를 직접 만나서 이야기하고 싶다면, 메일로 미리 약속을 잡을 것. 더불어 질문 메일의 제목 앞에는 항상 [컴퓨터구조 질문]을 붙인다.

- **Minhyeok An(mh.an@dgist.ac.kr)**
- **Jungbo Kim(jbkim@dgist.ac.kr)**
- **Sungju Kim(sungju_kim@dgist.ac.kr)**
- **YunHyeong Jeon(yhjeon@dgist.ac.kr)**