컴퓨터 구조 Project 2, Building a Simple MIPS Emulator

201911039 김태연

1. 수행 과제에 대한 간략한 설명

A. 과제 내용

이 과제는 MIPS 에뮬레이터를 간단히 모방해 보는 과제이다. sample.o, 즉 바이너리를 입력 받아 메모리에 로드하고, 인스트럭션을 실행한다. 이 기능을 구현하기 위해 memory class와 register class를 singleton design pattern으로 하나씩 구현한 뒤, 각 클래스에 인스트럭션 실행 결과를 저장해 두는 방식을 사용했다.

B. 과제 파일(.cpp, .h)

main.cpp, memory.cpp, memory.h, decoder.cpp, decoder.h 파일을 작성하여 과제를 해결하였다. main.cpp는 main함수를 포함하고 있는 파일이며, 모든 헤더 파일을 include한다. decoder.cpp는 메모리 및 레지스터를 이용하기 위해, memory.cpp는 decoder에 구현된 함수를 이용하기 위해 각자의 헤더파일 외에 서로의 헤더파일을 include 한다.

파일이름	성격	내용
main.cpp	실행 파일	main함수 포함
memory.cpp	저장공간	Class Memory, Class Register가 구현되어 있다. 각 클래스에 대한 설명은 아래 표 참고.
decoder.cpp	인스트럭션 decoding 및 실행	PC에서 instruction을 불러와 실행하는 execute(PC), 이 함수에서 부르는 하위 함수 rExec, iExec, jExec(각 인스트럭션을 실행함), sign-extension을 위한 함수 sign_extend 가 포함되어 있다.
헤더파일들	각 cpp file의 함수 및 클래스 선언부	이름이 동일한 cpp file의 함수 및 사용 라이브러리 include부를 담고 있다.

C. Class overview

Memory
Instruction memory: (int*) iMem Data memory: (int*) dMem Start addr of instr: int instrStart Start addr of data: int dataStart End addr of instr: int instrEnd End addr of data: int dataEnd
Set memory space and initialize: setMem(instruction size, data size) Write: writeMem(value, address) Read: readMem(address) Show contents: show(addr1, addr2)

Register
Register: int Register[32]
Write: writeRegs(value, number) Read: readRegs(number) Show contents: show()

D. 수행 단계

프로그램이 실행되면 PC, 메모리, 레지스터가 생성된다. 레지스터는 생성 즉시 초기화된

다. 메모리 및 레지스터는 프로그램 전체에서 Memory.instance()와 Register.instance()로 접근 가능하다.

먼저 입력받은 옵션들을 정리해 출력 환경을 세팅한다. 그 후 파일을 열고, 첫째 줄과 둘째 줄을 참고하여 메모리 공간을 확보하고 instruction과 data를 읽어 초기화한다. 전부 읽었으면 파일을 닫는다.

loop를 돌며 execute(PC)를 불러 해당 PC의 인스트럭션을 수행한다. PC값의 변화는 인스트럭션 실행 함수 내부에서 일어난다. execute(PC)는 인스트럭션을 읽은 뒤, 종류를 분류하여 적절한 작업을 수행하게끔 한다.

-n 옵션이 주어지지 않으면 instruciton을 모두 수행하고 종료하고, -n 옵션이 있다면 주어진 개수만큼 수행한다. 만일 -n 옵션의 개수가 프로그램 끝까지 실행되는 명령어의 개수보다 많다면 프로그램이 끝났을 때 종료한다.

-d 옵션이 주어진 경우, 명령어 한 줄 수행마다 레지스터와 메모리의 값을 출력해야 한다. 이때 마지막 최종 상태 출력과 구분하기 쉽고, 각 출력이 몇 번째 명령어의 수행인지를 보기 용이하게끔 명령어 실행 직후 출력되는 현 상태의 레지스터와 메모리 출력 위에는 Execution[실행번호] 가 함께 출력된다. 출력 예시는 다음과 같이 나타난다. 왼쪽이 -d 옵션에서 출력되는 출력의 예시, 오른쪽이 최종 출력의 예시이다.

```
Execution[33]
Current register values:
--------------------------------------
PC: 0x400010
Registers:
R0: 0x0
R1: 0x1
R2: 0x0
R3: 0xf
R4: 0xf
R5: 0x0
R6: 0x0
R7: 0x0
R8: 0x10000000
R9: 0x5
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: 0x400010

Memory content [0x10000000..0x10000004]:
--------------------------------------
0x10000000: 0x5
0x10000004: 0x0
```

```
Current register values:
--------------------------------------
PC: 0x400050
Registers:
R0: 0x0
R1: 0x0
R2: 0xa
R3: 0x800
R4: 0x1000000c
R5: 0x4d2
R6: 0x4d20000
R7: 0x4d2270f
R8: 0x4d2230f
R9: 0xfffff3ff
R10: 0x4ff
R11: 0x269000
R12: 0x4d2000
R13: 0x0
R14: 0x4
R15: 0xfffffb01
R16: 0x0
R17: 0x640000
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

Memory content [0x10000000..0x1000001c]:
--------------------------------------
0x10000000: 0x3
0x10000004: 0x7b
0x10000008: 0x10fa
0x1000000c: 0x12345678
0x10000010: 0xffff34ff
0x10000014: 0x0
0x10000018: 0x0
0x1000001c: 0x0
```

2. 컴파일 및 실행 방법과 환경

Ubuntu 20.04 환경에서 **g++ 버전 9.3.0**을 이용하여 다음 명령어로 컴파일 하였다.

```
g++ main.cpp memory.cpp decoder.cpp -o runfile
```

위의 컴파일 명령으로 실행 파일인 runfile을 생성하여 다음과 같은 명령어로 실행하였다.

```
./runfile -m 0x10000000:0x1000001c sample.o
```

위의 예시에서는 -m 옵션만 포함했으나, 과제에서 제시된 바(아래)와 같이 여러 옵션을 주어 테스트했다.

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