Project 2: Simple MIPS Emulator report

202011047 김승태

1. 컴파일/실행 방법, 환경

1-1. 환경 구성

환경은 Oracle VM VirtualBox에서 실행하였으며, Ubuntu를 이용하였다. 컴파일러는 g++을 이용하였으며, 각각의 version은 다음과 같다. Ubuntu 20.04.6 LTS, g++ 11.4.0

```
seungtae@seungtae-VirtualBox:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.6 LTS
Release:        20.04
Codename:       focal
seungtae@seungtae-VirtualBox:~$ g++ --version
g++ (Ubuntu 11.4.0-2ubuntu1~20.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

1-2. 컴파일과 실행 방법

실행 방법은 g++ 컴파일러를 이용하였으므로, 다음과 같다. .cpp 파일을 기본으로 열 경우에는, 컴파일을 실행시켜서 실행 프로그램을 제작 후 실행시키면 되므로, g++ -o runfile runfile.cpp 를 사용하여서 실행 파일을 만들고, ./runfile -m 0x400000:0x400010 -n 0 sample.o처럼 사용하면 된다. (sample.o 자리에 입력 파일을 넣는다.) 그렇게 하면, 출력이 나오게 된다. 사진에 자세히 나타내었으니, 확인하면 좋을 것이다.

1. Ubuntu 20.04 환경에서 g++ 11.4.0 버전을 이용하여 다음 명령어로 컴파일 하였다. g++ -o runfile runfile.cpp

```
seungtae@seungtae-VirtualBox:~/CAassi2$ gedit runfile.cpp
seungtae@seungtae-VirtualBox:~/CAassi2$ g++ -o runfile runfile.cpp
```

2. 과제 pdf에도 나왔던 것처럼, 기본 실행을 실시한다. 과제에서 명시되었던 방법과 완전 동일한 방법으로 실행 결과를 볼 수 있게끔 하였다.

./runfile -m 0x400000:0x400010 -n 0 sample.o 를 한 결과이다. sample.o 부분에 object file을 input으로 넣으면 된다.

sample.s를 넣은 결과로 sample.o가 생성되고, 결과값이 창에 보인 것을 알 수 있다.

```
seungtae@seungtae-VirtualBox:~/CAassi2$ ./runfile -m 0x400000:0x400010 -n 0 sample.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

Memory content [0x400000..0x400010]:
----------------------------------
0x400000: 0x24020400
0x400004: 0x421821
0x400008: 0x622025
0x40000c: 0x240504d2
0x400010: 0x53400
```

2. Code flow

2-1.전체적 flow

- 전체적으로, 개발은 ubuntu 내부가 아닌, visual studio에서 진행되었다. 즉각 피드백과 높은 가독성을 가져 선호하기 때문이다. 하지만, ubuntu 환경에서도 충분히 실행되었으며, gedit으로 따로 cpp를 제작하여 제출하였다. 하나의 .cpp 파일을 제출하고 싶었기 때문에, 따로 코드를 나누지 않고 진행하였다. 즉, 헤더 파일은 존재치 않고, make를 쓸 필요도 없다. 이 코드는 먼저, main에서 명령 인수들을 if문들로 처리하고, 예외를 모두 처리하게 된다. 메모리 출력 여부, 메모리 주소 범위, 실행 제한 등등의 옵션을 모두 설정하고, 위에서 정의한 함수들을 이용해

instruction을 읽어서 해석하고, 실행한다. 그리고 그것에 따라서 레지스터 값을 변화시키고, 메모리에 쓴다. R,I,J 다 따로 switch-case 구문으로 구현하였다. Memory는 Read하는 부분과 Write하는 부분으로 나누어 구현하였다.

2-2.세부적 flow와 설명

-각각 함수에 대해서 설명하려 한다.

MemoryRead, MemoryWrite : 각각 지정된 메모리 주소에서 isWord로 Word인지를 파악한 후, 데이터를 읽고 쓰게끔 하였다.

ExecuteR(I,J)TypeInstruction : 각각의 R,I,J 타입의 명령어들을 실행하는 것이다. 각각 다 실제와 동일하게 실행되도록 코딩하였다. MemoryRead, MemoryWrite와 Register라는 map을 이용하여서, 메모리에 저장할 것들과 레지스터에 저장할 것들이 구분되어 실행된다.

convertToBits : bitset 이용하여 string 형태로 bit를 만들어서 보낸다.

BitExtend : extend하는 함수이고, isSingExt에 따라서 0을 쭉 쓰거나, 가장 먼저 것을 앞에 쭉 써 준다.

DecodeAndExecuteInstruction() : 현재 PC가 가리키는 명령어를 디코딩하고, 명령어를 실행할 수 있도록 한다. 여기서, op로 R, J, I를 구분하여 실행시킨다.

PrintRegistersAndMemory : 현제 레지스터 값과 메모리를 출력하는 함수이다.

ChangeDecToHex : decimal 을 hex 형태로 나타내 주는 것이다.

Split, hasChar, isNumeric – 각각 구분자로 분할하여 벡터로 변환, 문자 포함되어 있는지 확인, 숫자로만 이루어져 있는지 확인한다.

MemoryAndInstructionExe – 메모리를 로드하고 명령어를 실행한다. Binaryfile을 받고 오픈하고, MemoryWrite를 돌리며, DecodeAndExecuteInstruction을 실행시킨다. 또한, printA도 받아서 -d를 실행시킬 수 있도록 한다.

Main : -m , -d, -n을 받고, 발생할 수 있는 오류들을 try catch문으로 한 번에 받아서 오류를 보여주는 것으로 하였다. 그리고, 위의 함수들을 이용하여, 우리가 원하는 실행을 하는 부분이라고 생각하면 된다.

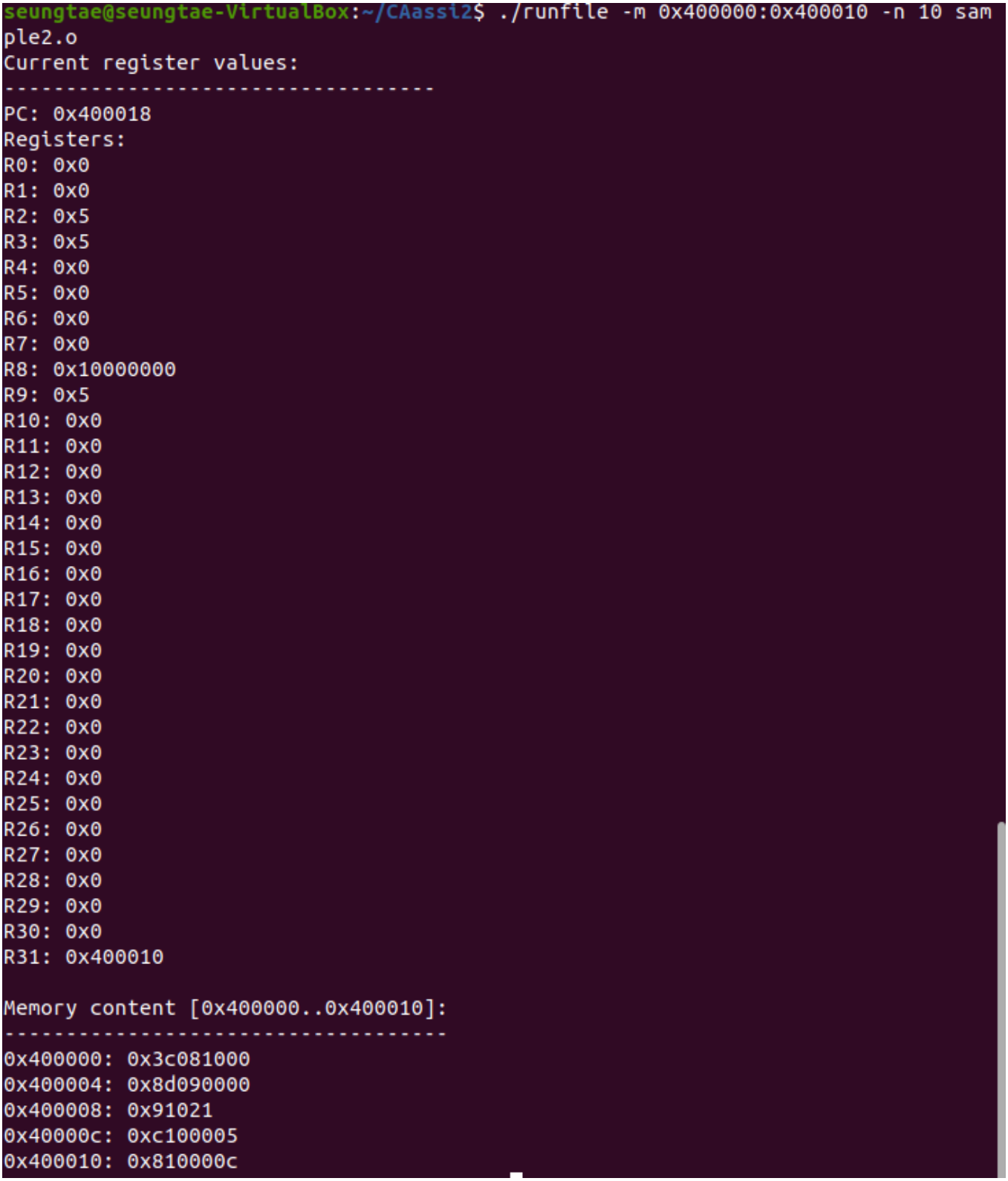
3. Sample1, sample2 결과

sample 1 result

```
seungtae@seungtae-VirtualBox:~/CAassi2$ ./runfile -m 0x400000:0x400010 -n 20 sam
ple.o
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 [0x400000..0x400010]:
----------------------------------
0x400000: 0x24020400
0x400004: 0x421821
0x400008: 0x622025
0x40000c: 0x240504d2
0x400010: 0x53400
```

sample 2 result



각각 -n을 다르게 하였으므로 참조하여 보시길 바랍니다.