

컴퓨터 구조 과제 2 보고서

20201100 김지호

1. 간단한 Flow

- (1) 콘솔 입력 처리 : `std::cerr`를 사용해 명령어 실행횟수, 디버깅 모드 여부, 출력한 메모리 영역을 받습니다.
- (2) 메모리 할당: `loadMemory(filename)` 함수를 이용해 `std::vector memory_text, memory_data`에 주소와 code를 입력합니다. 여기에서 text section size, data section size를 이용해 instruction과 data를 구분했습니다.
- (3) 명령어 실행 : `execute_inst` 함수로 `memory_text[PC]`에 들어있는 instruction을 읽고 opcode, rs, rt, rd, shamt, func, imm, target으로 분해합니다. 명령어에 따라 `state(registers, memory_data)`를 변경하고 적절한 PC값을 업데이트해줍니다. 이를 위해 `read_memory_data, write_memory_data`와 같은 함수를 사용했습니다. 사용한 registers 객체는 `std::array`입니다.
- (4) 최종 출력 : `printRegisters(), printMemory(메모리명, 시작점, 끝점)`함수를 이용해 output을 정의해두었습니다. `main()`에서 while문으로 `-n` 옵션으로 받은 명령어 실행횟수만큼 `execute_inst`를 실행합니다. 디버깅 모드라면, while문 내의 출력함수를 이용해 명령어가 끝날 때마다 레지스터의 내용을 출력합니다.

2. 컴파일 방법 및 컴파일 환경

g++9.4.0으로 compile을 진행했습니다.

```
zeeho@zeeho-VirtualBox:~/project1$ g++ --version
g++ (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 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.
```

사용한 OS는 리눅스 배포판 Ubuntu 20.04 LTS입니다.

 ubuntu-20.04 [실행 중] - Oracle VM VirtualBox

리눅스 콘솔창에서 아래와 같이 `runfile.cpp`를 실행했습니다. 컴파일이 성공적으로 이뤄진 것을 확인할 수 있습니다.

```
zeeho@zeeho-VirtualBox:~/project2$ ls
runfile.cpp sample.o sample2.o
zeeho@zeeho-VirtualBox:~/project2$ g++ -o runfile runfile.cpp
zeeho@zeeho-VirtualBox:~/project2$ ls
runfile runfile.cpp sample.o sample2.o
```

3. 과제 실행 방법 및 실행환경

실행 명령어는 `./runfile [-m addr1:addr2]] [-d] [-n num_instruction] <input file>`입니다. 옵션을 순서 상관없이 입력할 수 있고 과제1의 샘플을 assembler로 변환한 바이너리코드 sample.o과 sample2.o으로 여러 test를 진행했을 때 모두 정상적으로 동작했습니다.

(1) -n option으로 아무 인스트럭션도 실행하지 않은 경우

명령어가 실행되지 않았고 레지스터의 초기상태를 잘 볼 수 있습니다.

```
zeeho@zeeho-VirtualBox:~/project2$ ./runfile -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
```

(2) 메모리 주소 정의한 경우

사용한 sample.o(명령어19개)을 실행하여 최종 PC 가 $0x400000 + 0xd20 * 4 = 0x400050$ 가 된 것을 보아 모든 명령어가 잘 수행되었음을 알 수 있습니다. 지정한 메모리 출력물이 입력파일과 일치하므로 메모리할당이 잘 이루어짐을 알 수 있습니다.

```
zeeho@zeeho-VirtualBox:~/project2$ ./runfile -m 400000:40004c sample.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: 0xffffffff3ff
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 [400000..40004c]:
```

```
-----
0x400000: 0x24020400
0x400004: 0x421821
0x400008: 0x622025
0x40000c: 0x240504d2
0x400010: 0x53400
0x400014: 0x24c7270f
0x400018: 0xe24023
0x40001c: 0x834827
0x400020: 0x344a00ff
0x400024: 0x65942
0x400028: 0x66102
0x40002c: 0x3c041000
0x400030: 0x3484000c
0x400034: 0x80820001
0x400038: 0xa0820006
0x40003c: 0x1656824
0x400040: 0x308e0064
0x400044: 0xa7823
0x400048: 0x3c110064
0x40004c: 0x2402000a
```

1	0x50
2	0x14
3	0x24020400
4	0x421821
5	0x622025
6	0x240504d2
7	0x53400
8	0x24c7270f
9	0xe24023
10	0x834827
11	0x344a00ff
12	0x65942
13	0x66102
14	0x3c041000
15	0x3484000c
16	0x80820001
17	0xa0820006
18	0x1656824
19	0x308e0064
20	0xa7823
21	0x3c110064
22	0x2402000a
23	0x3
24	0x123
25	0x4346
26	0x12345678
27	0xffffffff

사용한 sample.o