Project 3: Simulating Pipelined Execution

202011047 김승태

1. 컴파일/실행 방법, 환경

1-1. 환경 구성

환경은 Oracle VM VirtualBox에서 실행하였으며, Ubuntu를 이용하였다. 컴파일러는 gcc를 이용하였으며, 각각의 version은 다음과 같다. Ubuntu 20.04.6 LTS, gcc 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:~$ gcc --version
gcc (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 gcc 컴파일러를 이용하였으므로, 다음과 같다. .c 파일을 기본으로 열 경우에는, 컴파일을 실행시켜서 실행 프로그램을 제작 후 실행시키면 되므로, gcc -o runfile runfile.c 를 사용하여서 실행 파일을 만들고, $ ./runfile <-atp 또는 -antp> [-m addr1:addr2] [-d] [-p] [-n num_instr] 처럼 사용하면 된다. 그렇게 하면, 출력이 나오게 된다. 사진에 자세히 나타내었으니, 확인하면 좋을 것이다.

1. Ubuntu 20.04 환경에서 gcc 11.4.0 버전을 이용하여 다음 명령어로 컴파일 하였다. gcc -o runfile runfile.c

```
seungtae@seungtae-VirtualBox:~/project3$ gedit runfile.c
seungtae@seungtae-VirtualBox:~/project3$ gcc -o runfile runfile.c
```

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

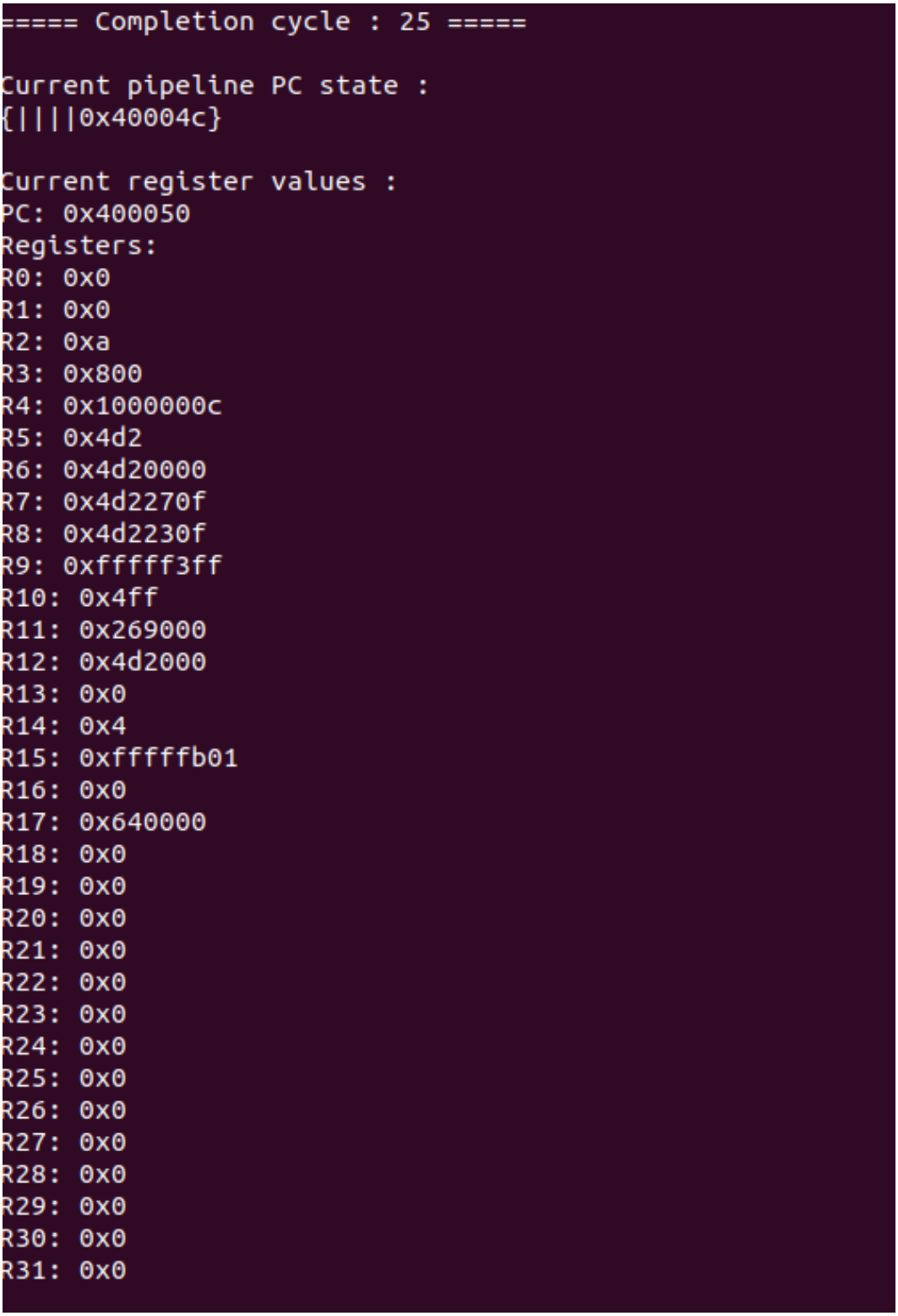
./runfile -atp -d -p sample.o를 한 결과이다. sample.o 부분에 object file을 input으로 넣으면 된다.

결과값이 창에 보인 것을 알 수 있다.

```
===== Cycle 24 =====
Current pipeline PC state :
{|||0x40004c|0x400048}

Current register values :
PC: 0x400050
Registers:
R0: 0x0
R1: 0x0
R2: 0x34
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

===== Completion cycle : 25 =====
```



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 c를 제작하여 제출하였다. 하나의 .c 파일을 제출하고 싶었기 때문에, 따로 코드를 나누지 않고 진행하였다. 즉, 헤더 파일은 존재치 않고, make를 쓸 필요도 없다.

1.초기화

명령줄 인수를 처리하여 설정값을 초기화한다.
입력 파일을 읽어 텍스트와 데이터 세그먼트를 메모리에 저장한다.
레지스터와 프로그램 카운터(PC)를 초기화한다.

2.사이클 반복

각 사이클마다 파이프라인의 각 단계(IF, ID, EX, MEM, WB)를 순차적으로 처리한다. NOW_IFID, NOW_IDEX, NOW_EXMEM, NOW_MEMWB는 현재 사이클의 파이프라인 레지스터를 나타내고, NEXT_IFID, NEXT_IDEX, NEXT_EXMEM, NEXT_MEMWB는 다음 사이클의 값을 저장한다. 각 단계에서 명령어를 디코딩하고 실행하고 레지스터와 메모리의 값을 업데이트한다.

3.파이프라인 제어 및 종료 조건

명령어의 종류에 따라 파이프라인 제어 신호를 설정한다.
브랜치 및 점프 명령어의 경우, 조건에 따라 프로그램 카운터(PC)를 업데이트하여 분기하고 지정된 사이클 수(isN)에 도달하거나 모든 명령어가 실행될 때까지 사이클을 반복한다.

4.출력 및 종료

각 사이클마다 현재 파이프라인 상태와 레지스터 값을 출력한다.
프로그램 종료 시 최종 파이프라인 상태와 레지스터 값을 출력한다.

2-2.세부적 flow와 설명
-각각 함수에 대해 설명하려 한다.

DATASEG_ADDR와 TEXTSEG_ADDR는 데이터와 텍스트 세그먼트의 시작 주소를 정의한다. 이 두 상수를 통해 프로그램에서 데이터와 명령어가 저장되는 메모리 위치를 지정한다.

함수 설명
changeStoI

문자열을 정수로 변환한다. 입력 문자열이 16진수인지 10진수인지 확인한 후, 적절히 변환한다.

setArguments

명령줄 인수를 분석하여 프로그램 설정값을 초기화한다. 입력된 인수에 따라 다양한 설정을 적용한다.

printRegs

현재 레지스터 값을 출력한다. 프로그램의 상태를 확인할 수 있도록 각 레지스터의 값을 깔끔하게 보여준다.

writeBTM 및 loadBFM

메모리에 바이트 단위로 데이터를 저장하고 읽어오는 기능을 한다. 특정 주소에서 바이트 단위로 데이터를 처리할 때 사용한다.

printMemoryContent

지정된 메모리 범위의 내용을 출력한다. 데이터 세그먼트와 텍스트 세그먼트의 내용을 확인할 수 있다.

printNullMemoryContent

지정된 메모리 범위의 내용을 0으로 출력한다. 초기화된 상태의 메모리 내용을 확인할 때 유용하다.

printPipelineState

파이프라인의 현재 상태를 출력한다. 각 사이클마다 파이프라인의 변화를 추적할 수 있다.

loadFile

파일을 읽어와 메모리에 저장한다. 프로그램이 실행될 때 필요한 데이터를 파일에서 불러온다.

handleEmptyFile

파일이 비어있을 경우 초기 상태를 출력하고 프로그램을 종료한다. 파일에 명령어가 없을 때 적절한 메시지를 출력하고 종료한다.

main 함수

프로그램의 진입점이다. 여기서부터 프로그램이 시작된다.

명령줄 인수를 처리하고, 입력된 파일을 읽어와 초기화한다.

각 사이클마다 파이프라인의 상태를 업데이트하고, 각 단계별로 명령어를 실행한다.

특정 조건에 따라 프로그램을 종료하고, 최종 상태를 출력한다.

이렇게 구성된 프로그램은 MIPS 시뮬레이터로서, 각 명령어를 사이클 단위로 처리하며 파이프라인의 상태를 관리한다. 프로그램이 종료될 때는 최종 레지스터 값과 메모리 상태를 출력하여 전체 실행 과정을 확인할 수 있다.

3. StateRegister

IFID, IDEX, EXMEM, MEMWB 상태 레지스터를 구성하였는데, 각각의 레지스터를 설명하겠다. 먼저, 전반적으로 val을 사용하여, 값이 있는지 확인하였다. 이것은 레지스터는 아니다. IFID에는 , pc, NPC, instr를 넣어 각각, 주소, 다음 주소, 32비트 instruction이 저장되어 있다.

IDEX에는 pc, NPC, op, rs, rt, rd, simm , uimm, control을 사용하여서 주소, 다음 주소, op정보, rs, rt, rd에, 16비트 를 sign, unsigned extend한 값과, control bit가 담겨 있도록 한다.

EXMEM에는 pc와 ALUOUT을 사용하여서 ALU 결과를 저장하고, 이것은 MEM, WB에서도 쓰이게 된다. Rt를 사용하였고, BRtarget으로 브랜치 넘어가는 target을 지정하였으며, REGDst으로 레지스터 번호를 나타낼 수 있도록 하였다. 이도, control을 담고 있다.

MEMWB에는 pc,ALUOUT, MEMOUT, REGDst가 쓰였고, MEMOUT은 MEM단계에서 읽은 메모리를 출력한다.