# 컴퓨터 구조 과제 3 보고서

20201100 김지호

## 1. 간단한 Flow

(1) 콘솔 입력 처리 : std::cerr를 사용해 명령어 실행횟수, 디버깅 모드 여부, 출력한 메모리 영역, 파이프라인 상태, always taken/always not taken을 받습니다.

(2) 메모리 할당: loadMemory(filename) 함수를 이용해 std::vector memory_text, memory_data에 주소와 code를 입력합니다. 여기에서 text section size, data section size를 이용해 instruction과 data를 구분했습니다.

(3) 파이프라인 구성: 유용한 변수들(pc, instr, opcode, rs, rt, rd, shamt, func, imm, target, ALU_out, MEM_out, BR_target, rs_value, rt_value)을 갖는 struct를 이용해 5개의 stage(PC_IF, IF_ID, ID_EX, EX_MEM, MEM_WB, stall)을 구성했습니다. 사용한 registers 객체는 std::array입니다.

(4) forwarding: 파이프라인 1 cycle이 돌기 전,각 stage의 read addr, write addr가 같은 경우 ALU_out 또는 MEM_out을 rs_value 또는 rt_value에 forwarding해줍니다. load-use data hazard의 경우(flush가 필요할 때). 각 stage를 필요한 개수만큼 뒤로 밀고 flush되는 stage에 stall(모든 인자가 0)을 입력합니다.

(3) 파이프라인 실행 : 파이프라인 내에 남아있는 instr가 없을때까지 while문을 반복해 1 cycle에 IF, ID, EX, MEM, WB가 동시에 일어납니다 (함수로 구현). 이를 위해 read_memory_data, write_memory_data와 같은 함수를 사용했습니다.

(4) control hazard: 1 cycle 실행 후 branch가 fetch되었다면 ID stage를 flush하고 IF stage에 BR_target을 fetch합니다. conditional branch 예측이 틀린 경우(MEM_WB.taken!=always_taken) 3 stage를 flush하고 올바른 BR_target을 PC 값으로 업데이트합니다.

(4) 최종 출력 : printRegisters(), printMemory(메모리명, 시작점, 끝점), printPipelineStage()함수를 이용해 output을 정의해두었습니다. main()에서 while문으로 -n 옵션으로 받은 명령어 실행횟수만큼 명령어를 fetch합니다. -d, -p 옵션이 있다면, while문 내의 출력함수를 이용해 명령어가 끝날 때마다 레지스터의 내용, 파이프라인 상태를 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.


리눅스 콘솔창에서 아래와 같이 runfile.cpp를 실행했습니다. 컴파일이 성공적으로 이뤄진 것을 확인할 수 있습니다.

```
zeeho@zeeho-VirtualBox:~/project3$ ls
runfile.cpp  sample.o  sample2.o
zeeho@zeeho-VirtualBox:~/project3$ g++ -o runfile runfile.cpp
zeeho@zeeho-VirtualBox:~/project3$ ls
runfile  runfile.cpp  sample.o  sample2.o
```

## 3. 과제 실행 방법 및 실행환경

실행 명령어는 $./runfile <-atp or -antp> [-m addr1:addr2]] [-d] [-p] [-n num_instr] <binory file.o>입니다. 옵션을 순서 상관없이 입력할 수 있고 과제1의 샘플을 assembler로 변환한 바이너리코드 sample.o과 sample2.o으로 test를 진행했을 때 모두 정상적으로 동작했습니다.

### case (1) $./runfile -atp -p sample.o

파이프라인이 완전히 비워질때까지 프로그램이 실행되고, 프로그램 종료 후에도 적절한 PC값을 유지하고있습니다.

### case (2) $./runfile -atp -p sample2.o

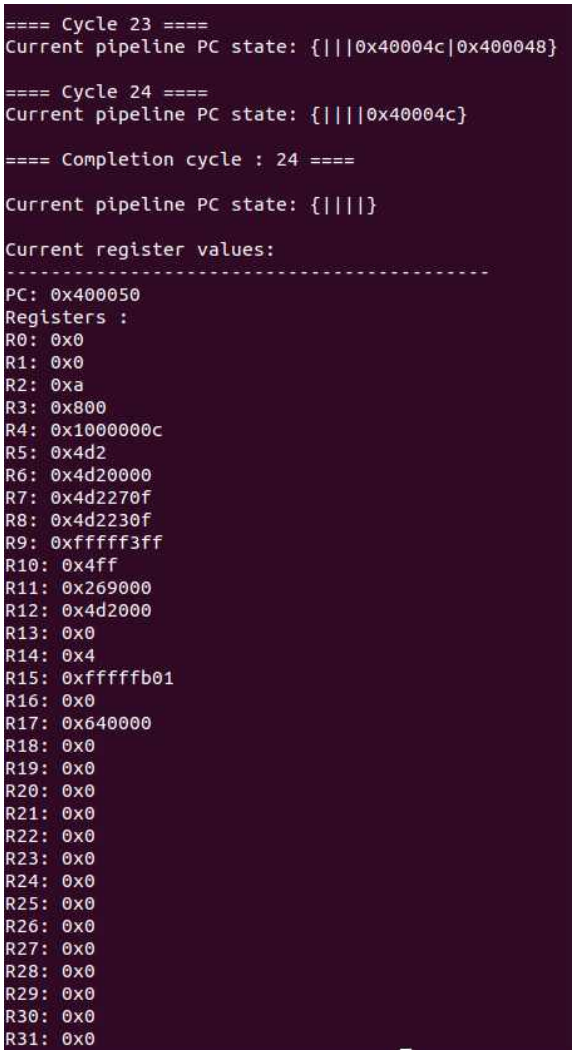
loop내에 bne구문을 가지고 있으므로 not taken이 taken보다 많이 일어납니다. always taken으로 예상한 경우 프로그램 완료까지 62 cycles가 소요됩니다.

### case (3) $./runfile -antp -p sample2.o

always not taken으로 예상한 경우 프로그램 완료까지 49 cycles가 소요됩니다.

### case (4) -n option

n=0인 경우 PC값이 0으로 유지되며 0 cycle이 소요되고, n이 1인 경우에는 5 cycle이 소요됩니다.


==== Cycle 23 ====
Current pipeline PC state: {|||0x40004c|0x400048}
==== Cycle 24 ====
Current pipeline PC state: {||||0x40004c}
==== Completion cycle : 24 ====
Current pipeline PC state: {||||}
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

case (1)

```
zeeho@zeeho-VirtualBox:~/project3$ ./runfile -atp sample2.o
==== Completion cycle : 62 ====

Current pipeline PC state: {|||}

Current register values:
-----------------------------------------
PC: 0x400030
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
```

case (2)

```
zeeho@zeeho-VirtualBox:~/project3$ ./runfile -antp sample2.o
==== Completion cycle : 49 ====

Current pipeline PC state: {|||}

Current register values:
-----------------------------------------
PC: 0x400030
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
```

case (3)

```
zeeho@zeeho-VirtualBox:~/project3$ ./runfile -n 1 sample.o
==== Completion cycle : 5 ====

Current pipeline PC state: {|||}

Current register values:
-----------------------------------------
PC: 0x400004
Registers :
R0: 0x0
R1: 0x0
R2: 0x400
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
```

case (4) n=1

```
zeeho@zeeho-VirtualBox:~/project3$ ./runfile -n 0 sample.o
==== Completion cycle : 0 ====

Current pipeline PC state: {|||}

Current register values:
-----------------------------------------
PC: 0x0
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
```

case (4) n=0