Project 3: Simulating Pipelined Execution

201811061 박동진

1. 소스 코드에 대한 간략한 설명

이 소스 코드는 C++로 작성되었습니다.
Project 2에서 만들었던 에뮬레이터를 IF/ID/EX/MEM/WB 단계로 쪼개어 구현하였습니다. State register를 표현하기 위해 class를 사용하였습니다.

-State Register Class
-Class IFID
멤버 변수
bool isRun, isExit;
unsigned int inst, NPC, rs, rt;
추가한 멤버 변수는 isRun, isExit, rs, rt입니다. isRun은 hazard를 해결하기 위한 flush를 진행하기 위해 state register의 shut down여부를 결정하기 위한 변수입니다. isExit는 마지막 인스트럭션이 fetch 되었을 때, while loop를 break하기 위한 변수입니다. isRun과 isExit는 모든 state register에 포함되어있습니다. rs, rt는 load-use hazard를 해결하기 위해 IF단계에서 rs와 rt를 알기 위한 변수입니다. 멤버함수는 getter와 setter, update로 이루어져있습니다. update 함수는 파이프라인 안의 인스트럭션들이 모두 동작을 완료하였을 때, state register의 정보를 업데이트함으로써 다음 while loop에서 state register를 통해 인스트럭션들이 동작할 수 있도록 하는 함수입니다.

-Class Control
멤버 변수
bool RW, MR, MW, isSame
IFID를 제외한 state register들이 상속하는 클래스입니다. ID stage에서 control signal을 만들어 WB Stage까지 사용하기 때문에 상속하는 클래스를 만들었습니다. RW는 RegisterWrite이고 MR은 MemoryRead, MW는 MemoryWrite 입니다. 이 값들은 data hazard를 해결하기 위해 사용됩니다. isSame은 beq, bne 에서 rs와 rt의 값이 같을 경우 branch를 진행하기 위해 만든 변수입니다. 멤버함수는 getter, setter 입니다.

-Class IDEX : public Control
멤버 변수
bool isRun, isExit
unsigned int NPC, IMM, rs, rt, rd, shamt, rsData, rtData
string inst, type
추가한 멤버 변수는 shamt, rsData, rtData, inst, type입니다. shamt는 sll, srl을 간편하게 실행하기 위한 변수입니다. rsData, rtData는 data hazard 조건에서 EX를 끝낸 인스트럭션의 정보를 IDEX 상태 레지스터에 바로 전달해주기 위해 만든 변수입니다. MEM forwarding도 마찬가지입니다. inst는 다음 단계에 인스트럭션 정보를 전달해주기 위해 만든 변수입니다. 하지만 inst만 알아서는 중복되는 코드가 너무 많아서 논리가 비슷한 인스트럭션을 type으로 정의해 반복되는 코드를 줄였습니다. 멤버함수는 getter, setter, update입니다.

-Class EXMEM : public Control
멤버 변수
bool isRun, isExit

unsigned int NPC, rs, rt, rd, target, ALU
string inst, type
추가한 멤버 변수는 rs, rt, rd입니다. addiu 같은 I type 인스트럭션은 결과를 rt에 저장하기 때문에 data hazard control 코드를 rd뿐만 아니라 rt에 대한 코드도 만들어야 합니다. 어차피 I type 인스트럭션에서 rd는 사용하지 않습니다. 반복되는 코드를 막기 위해 rd = rt를 EX, MEM 단계에서 I type 인스트럭션에 설정해주어 data hazard control 코드의 중복을 방지하였습니다. target은 분기 주소입니다. ALU는 ALU에서 출력된 값입니다. 멤버함수는 getter, setter, update입니다.

-Class MEMWB: public Control
멤버 변수
bool isRun, isExit
unsigned int NPC, rs, rt, rd, target, ALU
string inst, type
MEM/WB to EX forwarding을 control 하기 위해 EXMEM에 추가한 것처럼 rs, rt, rd를 추가하였습니다. 멤버 변수는 getter, setter, update입니다.

-동작 원리
while loop를 통해 구현하였습니다. while loop 마다 상태 레지스터 인스턴스를 생성하여 각각 단계를 진행하고 update 함수를 통해 상태 레지스터를 다음 while loop를 위해 업데이트 해줍니다. 업데이트 후에 data hazard control을 해줍니다. PC를 변경하며 control hazard는 파이프라인 내에서 구현하였습니다.

structural hazard를 해결하기 위해 WB단계를 ID단계 이전에 실행하였습니다. MEMWB 상태 레지스터가 실행중이어야 WB단계가 진행되고, WB단계의 정보를 전달해줄 상태 레지스터가 없기 때문에 while loop 안에서의 WB 단계의 순서는 상관이 없습니다.

각 단계는 상태 레지스터가 작동 중(isRun = true)이어야 진행됩니다.

while loop를 break하기 위해 bool isExitLoop, haveExit, whileExit를 사용하였습니다. haveExit는 jump나 branch 인스트럭션이 있을 경우 true가 됩니다. jump나 branch가 있을 경우 exit를 통해 프로그램을 종료하기 때문입니다. exit의 주소로 분기하게 될때, isExitLoop를 true로 만들어 while loop를 break하게 됩니다. jump나 branch가 없을 경우, PC주소를 endAddr과 비교하여 isExit를 true로 설정하였습니다.

WB 단계에 도달하는 인스트럭션의 숫자를 세어 n 옵션 만큼 인스트럭션을 실행하게 하였습니다.

각 단계에서 다음 단계에 정보를 전달해주기 위해 getter, setter 함수를 사용하였습니다.

control hazard 중 무조건 분기하는 jump 인스트럭션은 ID단계에서 진행하였고 currPc를 직접 수정하여 다음 while loop에서 target 주소를 fetch 할수 있도록 하였습니다. IFID의 인스턴스의 실행을 중지 시켜 stall을 구현하였습니다.

control hazard 중 branch 인스트럭션은 atp, antp 옵션에 따라 다르게 구현하였습니다. atp 옵션의 경우 무조건 분기하므로 jump와 마찬가지로 ID단계에서 currPc를 직접 수정하였습니다. 마찬가지로

IFID의 인스턴스의 실행을 중지시켜 stall 하였습니다. EX 단계에서 isSame을 사용하여 rs와 rt의 값이 같은지 확정하고 이를 MEM 단계에 넘겨줍니다. MEM에서 분기를 확정할지 말지 결정합니다. 분기를 확정할 경우 그대로 진행하고 분기를 취소할 경우 이미 stall을 한번 했으므로 2번의 stall을 발생하고 저장되어있는 NPC를 활용하여 branch의 다음 주소를 fetch 합니다. antp 옵션의 경우 무조건 분기 하지 않습니다. branch의 다음 인스트럭션들을 진행하다 MEM단계에서 분기를 결정합니다. 분기를 확정할 경우 3번의 stall을 하고 target주소를 Fetch 합니다. 분기를 안할 경우 그대로 진행합니다.

data hazard 중 EX/MEM to EX forward, MEM/WB to EX forward, load-use 코드는 강의자료를 참고하였습니다.

load 다음에 바로 store가 오는 경우 stall이 필요 없기 때문에 load-use 코드에 추가하였습니다.
load 다음 store를 진행하기 위해 MEM/WB to MEM forward가 필요하기 때문에 구현하였습니다.

printOutput 함수를 이용하여 출력을 구현하였습니다.

2. 컴파일/실행 환경, 방법

-컴파일/실행 환경

이 소스코드(main.cpp)는 Ubuntu 20.04.4 64bit 환경(VirtualBox)에서 작성되었습니다. 컴파일러는 g++ 9.4.0을 사용하였습니다.

-컴파일/실행

터미널에서 main.cpp와 input파일이 있는 디렉토리로 이동합니다. 터미널에 g++ -o runfile main.cpp를 입력합니다. 터미널에 $./runfile <-atp 또는 -antp> [-m addr1:addr2] [-d] [-p] [-n num_instruction] <binary file> 을 입력합니다. 예를 들어 input파일이 sample.o 일경우, 터미널에 ./runfile -atp -m 0x10000000:0x10000010 -d -p sample.o 를 입력합니다. 예를 들어, 인스트럭션을 5개 실행하고 싶을 경우 ./runfile -atp -m 0x10000000:0x10000010 -d -p -n 5 sample.o를 입력하면 됩니다.