Computer Architecture Project 3 : Simulating Pipelined Execution

201911106 유제이

1. Files

소스파일은 다음과 같이 2가지로 구성되어 있다.

1) pipeline.cpp

: pipeline.cpp는 main function이 있는 소스파일로, 프로그램 작동을 위해서는 해당 파일을 컴파일 및 실행해야 한다.

2) StageRegister.h

: StageRegister.h는 pipeline.cpp가 include하는 헤더파일로, Parent인 StageRegister class와 그 children인 IF_ID, ID_EX, EX_MEM, MEM_WB class들로 구성되어있다. Pipeline에서의 각 StageRegister와 그 레지스터가 담고있는 Control bit 외 여러 상태 레지스터들을 구현한 것이다.

2. Compile and Execution

해당 코드는 Window hyper-V를 이용한 가상 컴퓨터 내 Ubuntu 20.04 환경에서 g++ 9.4.0 ver을 사용하여 컴파일 후 실행하였다. 컴파일 및 실행 방법은 아래와 같다.

1) Compile

pipeline.cpp, StageRegister.h, sample.o, sample2.o 등 실행 및 읽기에 관련된 파일이 있는 Directory에서 Open in terminal -> `g++ -o pline pipeline.cpp`을 입력한다.

2) Execution

Complie 후 다음과 같은 옵션을 조합하여 실행할 수 있다. -atp/antp, -m, -d, -p, -n flag 이외의 flag가 입력되면 "unknown flag inserted"를 출력하고 프로그램을 종료한다.

`./pline <-atp or -antp> [-m addr1:addr2] [-d] [-p] [-n num_instruction] <input file>`

1) `-atp` : Always Taken으로 분기 예측기를 실행한다.

2) `-antp` : Always Not Taken으로 분기 예측기를 실행한다.

-atp와 -antp는 각 필수 옵션으로, 둘 중 하나라도 입력되지 않을 경우 "The prediction mode must be given" message를 출력하고 프로그램을 종료한다.

3) **`-m`** : 메모리 주소 범위를 받아 프로그램이 종료될 때 범위 내 내용들을 출력한다. 4 Byte 단위로 address를 읽어들이며, 따라서 addr1과 addr2가 0x00400000, 0x10000000 + 4*m (m은 양의 정수) 꼴로 나타내질 수 있어야지만 해당 옵션에 따라 프로그램이 동작할 수 있으며, 유효한 범위가 아닐 때 message를 띄우고 프로그램을 종료한다.

예외 상황과 각각의 error message는 다음과 같다.

- addr1, addr2 is null or ":"로 범위가 나눠지지 않은 경우 : "The range of address is required with flag -m."

- addr의 입력 형태가 '"0x" hexadecimal이 아닌 경우 : "The range of address should be the form of hexadecimal like '0xffffffff'."

- addr1 > addr2 : "first address should be lower than second one."

- addr1 < 0x00400000 : "The address is out of the range. The minimum is 0x00400000."

- addr1과 addr2가 0x00400000, 0x10000000 + 4m의 꼴이 아닐 때 : "The Range of address should be the unit of word."

4) **-d** : -d 옵션이 주어질 경우 각 instruction의 실행이 끝나는 매 순간에 R0~R31 register 내 저장된 데이터를 출력한다. -d 옵션이 없을 경우는 끝날 때 한 번만 출력한다.

5) -p : -p 옵션이 주어질 경우 각 instruction이 끝나는 매 순간에 현재 파이프라인의 각 단계에 있는 PC를 출력한다.

3) **-n** : 실행할 instruction의 개수를 지정하고 이에 따라 실행횟수를 제어한다. 예외상황과 error message는 다음과 같다. 이때, n을 float값으로 입력하게 된다면 정수로 변환하여 실행횟수를 설정한다.

- n flag 이후 값이 주어지지 않는 경우 : "The number of instruction is required with flag -n."

- n이 음수인 경우 : "The number of instruction must be positive integer number."

- n이 0인 경우 : 파이프라인을 실행하지 않는 것으로 처리한다.

4) **Input file**: input file은 필수 입력 사항으로 파일 이름이 주어지지 않았을 때 "There's no file provided"를 출력하고 프로그램을 종료한다.

3. Functions and Flow

전체적인 플로우는 다음과 같다.

① 입력된 값들을 통해 출력 형식 결정 및 참조 파일 불러들이기

② 파일 내 정보를 읽어들이고 data 공간 할당 및 Instruction, word를 가상 data공간에 저장

③ IF, ID, EX, MEM, WB 단계 pipeline 실행 (구현에 있어서는 WB – MEM – EX – ID – IF 순으로 각 단계의 함수를 실행하였다. 그러나 한 사이클 내에서 이루어지는 활동들은 모두 동시에 실행되는 것으로 가정한다.)

④ PC 값을 받아 가리키는 Instruction Fetch (더불어 PC값에 4를 더해준다.), 이후 Decode 된 instruction을 실행, 상황에 따라 PC값을 업데이트 시키며 instruction이 없을 때까지 (혹은 지정한 개수만큼 instruction의 실행이 완료될 때까지) 실행 이후 내용 출력 및 종료

StageRegister.h

StageRegister : Control signal인 RegDst, ALUOp, ALUSrc, Brch, Jump, MemRead, MemWrite, RegWrite, MemtoReg, noop 와 decode된 instruction의 정보인 op, rs, rt, rd, sh, fn, imm를 protected member variable

로 가지고 있어 이를 상속하는 각기 다른 class들이 참조할 수 있게 한다.

IF_ID : IF stage와 ID stage 사이의 register로 앞서 StageRegister class의 member variable과 더불어 32 bit instruction, 다음 PC값인 NPC, ALU 계산에 사용되는 operand 값 (data1, data2) 그리고 operand 중 어떤 값이 forwarding이 되었는지 표기하는 forward를 private member variable로 갖는다.

본디 decoding 이후 위의 정보들이 주어지지만, ID Forward Unit에서의 감지를 위해 미리 알고있는 instruction 32bit로 필요 정보를 해당 stage register에 저장하는 것이다.

Pipeline.cpp에서 fd instance가 IF_ID register에 해당한다.

ID_EX : ID stage와 EX stage 사이의 register로 앞서 StageRegister class의 member variable과 더불어 NPC, BR_TARGET (branch target 주소), shift, data1, data2를 private member variable로 갖고있다.

Pipeline.cpp에서 dx instance가 ID_EX register에 해당한다.

EX_MEM : EX stage와 MEM stage 사이의 register로 앞서 StageRegister class의 member variable과 더불어 NPC, op, dst (rd or rt 둘 중 무엇을 destination으로 삼는지 표기), BR_TARGET, ALU_OUT (ALU 연산 결과), shift를 private member variable로 갖는다.

Pipeline.cpp에서 em instance가 EX_MEM register에 해당한다.

MEM_WB : MEM stage와 WB stage 사이의 register로 앞서 StageRegister class의 member variable과 더불어 NPC, dst, BR_TARGET, ALU_OUT, Mem_Out (Memory에서 읽어들이는 값), shift을 private member variable로 갖는다.

Pipeline.cpp에서 mw instance가 MEM_WB register에 해당한다.

각 stage register는 noop이 들어왔을 경우 모든 값을 0으로 return한다.

pipeline.cpp

pipeline은 주어진 n (instruction number) 만큼의 instruction 실행이 완료될 때까지, 혹은 n option이 주어지지 않았을 경우 모든 instruction이 완료될 때까지 cycle을 돌며 instruction을 실행한다.

이때, 각각의 pipeline stage는 IF(pc), ID(pc), EX(pc), MEM(pc), WB(pc) 함수로 구현하였다.

IF(pc) : IF stage는 Instruction을 Fetch하는 단계로, 이때 ID stage에 있는 instruction이 jump라면 현 Fetch되고 있는 instruction을 Flush시킨다. Fetch 후 간단한 정보들을 IF_ID stage register로 보내 저장한다.

ID(pc) : ID stage는 Instruction을 Decode하는 단계로 Data Hazard Unit이 있어 noop insertion을 수행할 수 있다. 또한 앞의 instruction와 ID stage에서의 branch instruction이 dependency가 있는 경우의 Branch data forwarding도 이 단계에서 이루어진다. 이때 Jump target으로의 PC 주소 변경은 이 단계에서 이루어진다. Op code를 읽고 각 instruction에 맞는 control signal을 만들어 ID_EX stage register에 전달한다. 앞서 Decoding한 정보들도 ID_EX stage register에 전달한다.

Control Signal :

RegDst, ALUOp1, ALUOp2, ALUSrc (Used in EX stage), Brch, MemRead, MemWrite (Used in MEM stage), RegWrite, MemtoReg (Used in WB stage)

EX(pc) : EX stage는 ID_EX stage register에서 RegDst, ALUOp1, ALUOp2, ALUSrc의 control signal을 포함한 다른 정보들을 받아온다. 이때 ALUOp를 통해 ALU연산의 종류를 정하고 연산한 값을 EX_MEM stage register에 저장한다. ALU 연산을 사용하지 않는 sll, srl 등의 경우는 Shift Hardware를 이용해 별도로 필요한 연산을 한다. 이후 ID_EX stage register에서 전달하는 Brch, MemRead, MemWrite, RegWrite, MemtoReg 등의 control signal을 받아와 EX_MEM stage register에 저장하고 이외의 다른 필요 정보들도 해당 stage register에 보내준다.

MEM(pc) : MEM stage는 EX_MEM stage register에서 Brch, MemRead, MemWrite의 control signal을 포함한 다른 정보들을 받아온다. 이때 branch operation의 경우 branch target의 주소가 여기서 계산된다. Load instruction인 경우는 memory에서 값을 받아오고, store instruction인 경우는 memory에 값을 저장한다. 다음 WB stage에서 사용될 RegWrite, MemtoReg signal과 더불어 ALU_OUT, MEM_OUT,DstReg, NPC 등의 정보를 MEM_WB stage register에 전달한다.

WB(pc) : WB stage에서는 결과값을 destination register에 다시 쓴다. MemtoReg signal에 따라 alu result, memory data, shift result가 레지스터에 저장된다.

Data Forwarding

Dependency가 있는 경우 Data Forwarding type의 결정은 EXForwardUnit(), MEMForwardUnit(), IDForwardUnit()에서 이루어진다.

Branch Prediction

Atp : 항상 branch를 할 것이라고 예측한다. Branch 여부는 ID stage에서 알 수 있으므로 무조건 1cycle stall 한다. 이후 branch가 이루어지면 그대로 진행하고 이루어지지 않으면 3 cycle을 flush한다. 이 프로그램의 출력에서는 branch가 MEM 단계에 있을때는 Pipeline stage의 상태 및 PC를 보여주는 부분에서 Flush가 적용되어 IF, ID, EX stage가 비어있는 모습을 보여준다. 즉 flush가 일어나야할 경우는 바로 그 사이클에서 flush를 시켜버리고 flush 이후의 pipeline 상태를 보여준다고 가정하고 구현하였다.

Antp : 항상 branch를 하지 않을 것이라고 예측한다. Branch를 하게 되면 3 cycle을 flush한다. MEM 단계에 instruction의 Branch signal이 1인 경우 곧바로 IF, ID, EX를 flush 시킨다. 이 경우도 flush가 적용된 모습의 pipeline 상태를 보여준다고 가정하고 구현한 것이다.