컴퓨터구조 (CSED311)

Lab 4-2a Report

20200220 오상윤

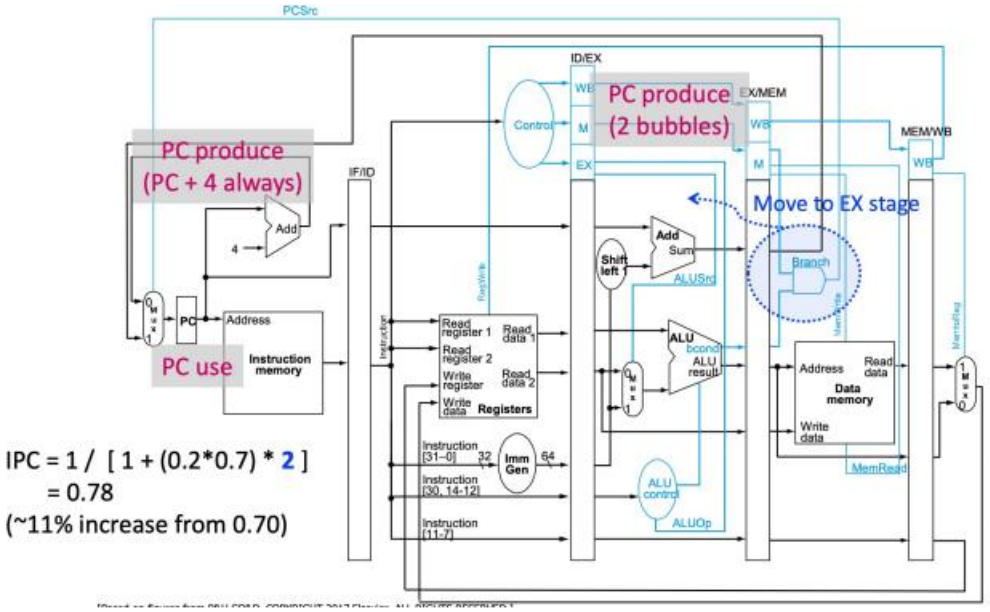
(1) Introduction

지난 Lab4-1에서 베릴로그를 이용하여 구현한 non-control flow instruction을 위한 5 stage pipelined cpu에 control flow instruction을 추가하여 Control hazard의 발생 또한 처리할 수 있는 cpu를 구현하였다. Branch prediction 방법 총 4가지를 순차적으로 구현해보았으며, Always not taken, Always taken, 2-bit global prediction, Gshare 솔루션을 각각 적용해보았을 때 달라지는 총 cycle 수를 비교해보았다.

파일 구성은 cpu module을 담고 있는 cpu.v, PC module, Adder module을 담고 있는 pc.v, ControlUnit module을 담고 있는 ControlUnit.v, ImmediateGenerator module을 담고 있는 ImmediateGenerator.v, ALUControlUnit module, ALU module을 담고 있는 ALU.v, Mux module, MuxForIsEcall module, Mux2bit module을 담고 있는 mux.v, HazardDetectionUnit module을 담고 있는 HazardDetectionUnit.v, ForwardingUnit module, ForwardingMuxControlUnit module을 담고 있는 ForwardingUnit.v, constant들의 define을 담고 있는 opcodes.v, BTB module, MissPredictionDetector module을 담고 있는 BTB.v로 구성되어 있다.

(2) Design

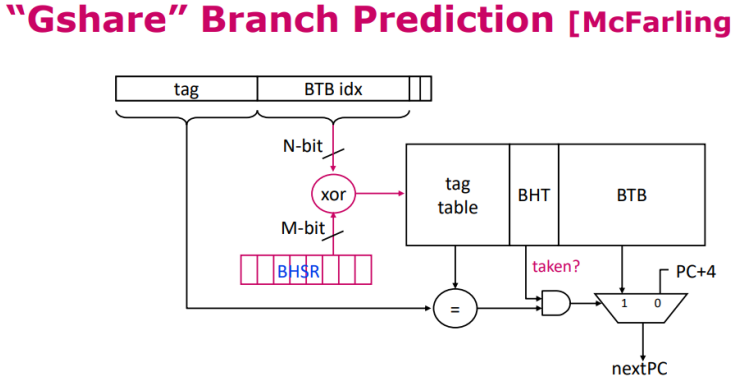
Lab4-2a pdf에 나와 있는 Datapath Design을 토대로 하여 전체적인 module을 Design 하였다.



지난 Lab4-1에서 구현한 cpu에서 jal, jalr, branch instruction을 처리할 수 있도록 내부 구조를 추가/수정하였다. rd 레지스터에 pc+4 값을 저장하는 jal, jalr instruction을 위해 pc+4 값을 각 IF/ID, ID/EX, EX/MEM, MEM/WB stage register를 통해 전달하여, mux를 통해 jal, jalr instruction의 경우는 RegisterFile module에 write data로 pc+4를 보내줄 수 있도록 하였으며, Data Forwarding도 적절히 일어날 수 있도록 각 관련 module의 input값을 수정하였다. 또한 EX stage에서 jal, jalr, branch instruction을 taken 했을 때의 pc 값 및 branch의 taken 여부를 계산하여 miss prediction이 발생할 경우 2 bubbles를 발생시켜 resolve할 수 있도록 하였다.

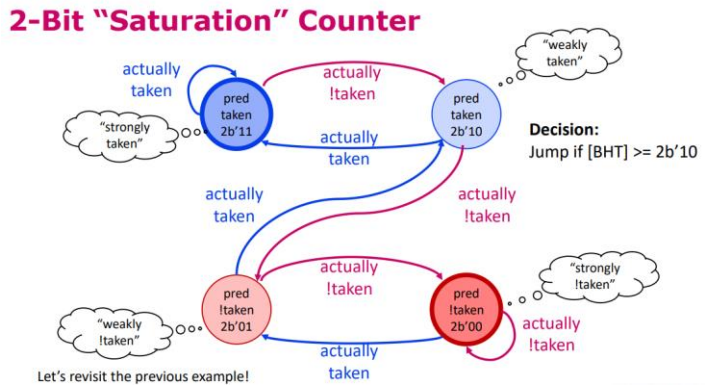
Control Hazard의 해결 및 **Gshare Branch prediction**을 위해 새롭게 추가한 module은 BTB module과 MissPredictionDetector module이며 각 module에 대한 설명은 아래와 같다.

1. BTB module



먼저 pc 값을 받아 pc 값으로부터 tag와 index를 추출한다. 이후 Tag table의 해당 index의 tag와 추출한 tag 값이 일치함과 동시에 해당 index의 BHT(Branch HistoryTable)가 taken을 나타내는 경우, BTB(Branch Target Buffer)의 해당 index에 저장되어 있던 pc 값을 다음 pc 값으로 내보내는 module이다. Tag table의 tag와 추출한 tag 값이 일치하지 않거나, BHT가 not taken을 나타내는 경우 pc+4의 값을 다음 pc 값으로 내보낸다. BTB, Tag table, BHT는 총 32개의 entries를 가지며, 따라서 index는 pc의 최하위 2bit를 제외한 하위 5bit와 5bit의 BHSR(Branch History Shift Register) 값을 XOR 연산한 결과값을 사용하였다. 또한 tag는 중복되는 경우를 방지하기 위해 pc 전체(32bit)를 사용하였다.

BHT의 경우 각 entry는 2-bit Saturation Counter로, 4가지 state를 가지며 각각의 state들이 나타내는 taken 여부는 다음과 같다.



BTB와 BHT, BHSR의 초기값은 0으로 설정하였으며 Tag table의 초기값은 -1로 설정해주었다. Tag table도 초기값을 0으로 설정할 경우 pc값이 0일 때 tag가 일치하여 의도하지 않은 결과를 나타낼 수 있어 -1로 설정하였다.

또한 BTB와 Tag table, BHT, BHSR은 EX stage의 pc를 이용하여 update하며, EX stage의 instruction이 jal, jalr 또는 branch instruction인 경우에만 update 한다. BTB와 Tag table은 tag가 일치하지 않거나 BTB의 값이 적절한 값이 저장되어 있지 않은 경우 update 해준다. BHT의 경우 해당 index의 값을 EX stage의 pc가 taken인 경우 1을 더해주며, !taken인 경우 1을 빼주도록 하였다. BHSR은 매번 전체를 왼쪽으로 shift 해주고 마지막 비트를 taken의 경우 1을, !taken의 경우 0을 지정해준다.

2. MissPredictionDetector module

EX stage의 pc값과 바로 이전 stage인 ID stage의 pc 값을 비교하여, prediction이 적절하게 이루어졌는지 확인하고 확인한 결과를 내보내는 module이다. EX stage의 pc가 taken이면서 ID stage의 pc 값이 taken 했을 때의 pc값이 아닌 경우 또는 EX stage의 pc가 not taken이면서 ID stage의 pc 값이 pc+4가 아닌 경우를 찾아내어 cpu에게 signal을 보낸다. cpu는 해당 signal을 받아 bubble을 생성하여 hazard를 처

리하고 올바른 pc 값을 Fetch하게 된다.

(3) Implementation

1. BTB module

input으로 pc, reset, clk, is_jal, is_jalr, branch, bcond, write_pc, pc_plus_imm, reg_plus_imm, write_bhsr을 가지며 output으로 n_pc와 bhsr을 갖는다.

pc값을 그대로 tag로 사용하고 pc[6:2]와 bhsr을 xor 연산하여 index를 얻는다. index와 tag 값을 바탕으로 tag가 tag_table과 일치하고 bht가 taken을 나타내는 경우 n_pc에 btb에 저장되어 있는 값을 asynchronous하게 넣어주고, 그렇지 않은 경우 n_pc에 pc+4 값을 asynchronous하게 넣어준다.

또한 write_pc 값을 그대로 write_tag로 사용하며 write_pc[6:2]와 write_brsh을 xor 연산하여 wirte_index를 얻는다. branch, is_jal, is_jalr 중 하나가 1일 경우 tag_table과 btb를 update해야하는지 확인하고 필요한 경우 asynchronous하게 update 해준다. write_index와 write_tag 값을 바탕으로 write_tag가 tag_table과 일치하지 않거나 btb 값이 적절한 pc값이 저장되어 있지 않은 경우(jal, branch instruction의 경우 pc_plus_imm, jalr instruction의 경우 reg_plus_imm이 적절한 pc값에 해당한다.) tag_table과 btb를 write_tag와 적절한 pc값으로 update 해준다.

또한 branch, is_jal, is_jalr 중 하나가 1일 경우 bht와 bhsr을 적절한 값으로 asynchronous하게 update 해준다. 먼저 bht는 is_taken이 1인 경우 bht 값을 1 증가시키고, 아닌 경우 1 감소시켜 state를 변화시킨다. 단 bht가 최대값(2'b11)인 경우 더 이상 증가시키지는 않으며, bht가 최소값(2'b00)인 경우도 더 이상 감소시키지는 않는다. 또한 bhsr은 1만큼 left shift 시킨 다음 is_taken이 1인 경우 하위 비트를 1로, is_taken이 0인 경우 하위 비트를 0으로 지정하여 update 한다.

2. MissPredictionDetector

input으로 IF_ID_pc, ID_EX_is_jal, ID_EX_is_jalr, ID_EX_branch, ID_EX_bcond, ID_EX_pc, pc_plus_imm, reg_plus_imm을 가지며, output으로 is_miss_pred를 갖는다.

IF_ID_pc 값을 확인하여 miss prediction인 것이 확인되었을 때 asynchronous하게 is_miss_pred 값을 1로 지정한다. ID_EX_is_jal이 1이거나 ID_EX_branch와 ID_EX_bcond가 동시에 1의 값을 갖는 경우(taken의 경우) IF_ID_pc가 pc_plus_imm과 다르다면 is_miss_pred를 1로 지정한다. ID_EX_is_jalr이 1인 경우(taken의 경우) IF_ID_pc가 reg_plus_imm과 다르면 is_miss_pred를 1로 지정한다. ID_EX_branch는 1이지만 ID_EX_bcond가 0인 경우(not taken의 경우) IF_ID_pc가 ID_EX_pc+4와 다르면

is_miss_pred를 1로 지정한다. 이외의 경우에는 is_miss_pred를 0으로 지정한다.

(4) Discussion

always taken 또는 2-bit global prediction 의 Branch prediction 을 위한 BTB 를 구현하는 과정에서는 index 를 오직 pc[6:2] 만을 사용하여 얻어내기 때문에 pc 들끼리 tag 와 index 값이 중복되는 경우가 발생하지 않았지만, Gshare Branch prediction 을 위해 BTB 를 구현하는 과정에서 pc[6:2]와 bhsr 을 xor 한 값을 index 로 사용하면서 tag 와 index 가 모두 같은 값을 가지는 pc 들이 종종 발생하는 것을 확인할 수 있었다. 이러한 문제를 해결하기 위해 Gshare Branch prediction 을 구현할 때는 pc 값 전체를 tag 로 사용하여 중복되는 경우가 발생하지 않도록 하였다.

주어진 reculsive_mem.txt file 을 Always not taken, Always taken, 2-bit global prediction, Gshare 의 Branch Prediction 방법을 적용한 4 가지 pipelined cpu 에 대해 각각 돌려보았다. Always not taken 의 경우 총 1187 cycles, Always taken 의 경우 총 1047 cycles, 2-bit global prediction 의 경우 총 1047 cycles, Gshare 의 경우 총 1091 cycles 가 소요되었다. Always not taken 과 Always taken 의 경우 그 결과에서 비교적 크게 개선된 결과가 나타난 반면, Always taken 과 2-bit global prediction 은 동일한 cycle 수가 나타났다. 이는 2-bit global prediction 에서 대부분의 jump, instruction 이 taken 됨에 따라 항상 taken 으로 예측하였기 때문인 것으로 생각된다. 또한 2-bit global prediction 과 Gshare 의 총 cycle 수를 비교해보았을 때 Gshare 의 경우 그 수가 오히려 증가하는 것을 확인할 수 있다. 이를 통해 Gshare 의 Branch Prediction 방법이 항상 더 좋은 것은 아니라는 것을 확인할 수 있었다.

(5) Conclusion

이번 Lab 을 통해 다양한 Branch prediction 방법을 각각 적용한 pipelined cpu 를 구현해보면서 control flow instruction 을 포함하여 동작하는 cpu 의 내부 구조를 이해하고, pc 값 예측을 위한 cpu 내부에서의 데이터 흐름을 살펴볼 수 있었다. 또한 Always not taken, Always taken, 2-bit global prediction, Gshare 를 각각 적용했을 때 나타나는 총 cycle 수를 비교해봄으로써 어떻게 개선이 이루어지는지를 확인해보았으며 2-bit global prediction 이나 Gshare prediction 방법이 항상 좋지는 않다는 것을 직접 확인할 수 있었다.