Lab3 Report_2019-10737_CHOI

전기정보공학부
2019-10737
최동호

Computer Architecture

Lab 3: Branch Prediction Hardware Implementation

1. Baseline

Baseline	bst_array	fibo	matmul	prime_fact	quicksort	spmv	spconv
CORE_CYCLE	10060	871	22431	18884	34502	42430	249824

2. Gshare

Gshare	bst_array	fibo	matmul	prime_fact	quicksort	spmv	spconv	total
CORE_CYCLE	8266	856	15789	13886	29765	41944	198206	
NUM_COND_BRANCHES	1942	200	3272	4207	5488	8302	36569	
NUM_UNCOND_BRANCHES	103	6	1029	1035	668	94	11424	
BP_CORRECT	1229	142	2609	3348	3755	4136	26436	
BP_INCORRECT	713	58	663	859	1733	4166	10133	
Accuracy	63.3%	71.0%	79.7%	79.6%	68.4%	49.8%	72.3%	69.2%
Speedup over Baseline	1.22	1.02	1.42	1.36	1.16	1.01	1.26	1.21

3. Perceptron

Gshare	bst_array	fibo	matmul	prime_fact	quicksort	spmv	spconv	total
CORE_CYCLE	7933	832	15291	13250	29366	35896	187064	
NUM_COND_BRANCHES	1942	200	3272	4207	5488	8302	36569	
NUM_UNCOND_BRANCHES	103	6	1029	1035	668	94	11424	
BP_CORRECT	1340	150	2775	3560	3888	6152	30150	
BP_INCORRECT	602	50	497	647	1600	2150	6419	
Accuracy	69.0%	75.0%	84.8%	84.6%	70.8%	74.1%	82.4%	77.3%
Speedup over Baseline	1.27	1.05	1.47	1.43	1.17	1.18	1.34	1.27

4. Implementation Detail (Gshare, Perceptron)

이는 GShare 및 Perceptron predictor에 공통적으로 적용되는 implementation이다. 전체 overview는 첨부한 CPU_Diagram에서 확인할 수 있다. 크게 IF stage에서의 branch hardware 및 MEM stage에서 resolve logic으로 나눌 수 있다.

Branch Hardware (IF stage)

1. access Branch HW

오직 jump, branch instruction에 대해서 접근해야하므로, opcode[6]으로 predecoding 하였다.

2. next_PC source

resolve	pred & hit	next_PC source
0	0	PC+4
0	1	pred_target_PC

resolve	pred & hit	next_PC source
1	x	resolved_target_PC

- If `resolve` is asserted, must resolve next PC

3. branch predictor

- using snapshot of BHR
  - access 와 update가 한 clk cycle에서 진행되는 경우, BHR이 update 되면 pred가 바뀌는 문제가 발생한다.
  - 따라서 BHR_snapshot에서 combinational manner로 access 및, negedge of clk에서 PHT or perceptron table을 update 하고 next clk posedge에 `BHR <= BHR_snapshot` 으로 업데이트하였다.
- jump instruction
  - jump instruction에 대해서는 무조건 taken prediction을 한다. 이를 위해 Branch HW module 밖에서 2x1 mux, opcode[2]&opcode[6]의 select signal을 사용하여 jump instruction일 경우 pred=1 을 보장하였다.

4. BTB

- update manner
  - 아직 BTB가 채워지지 않아 unvalid 한 경우를 포함하여, index에 따른 BTB update에는 기존의 값을 override 하였다.
- hit
  - hit = valid & (same tag between pc and TAG table)

Resolve Logic (MEM stage)

resolve signal & resolved target PC

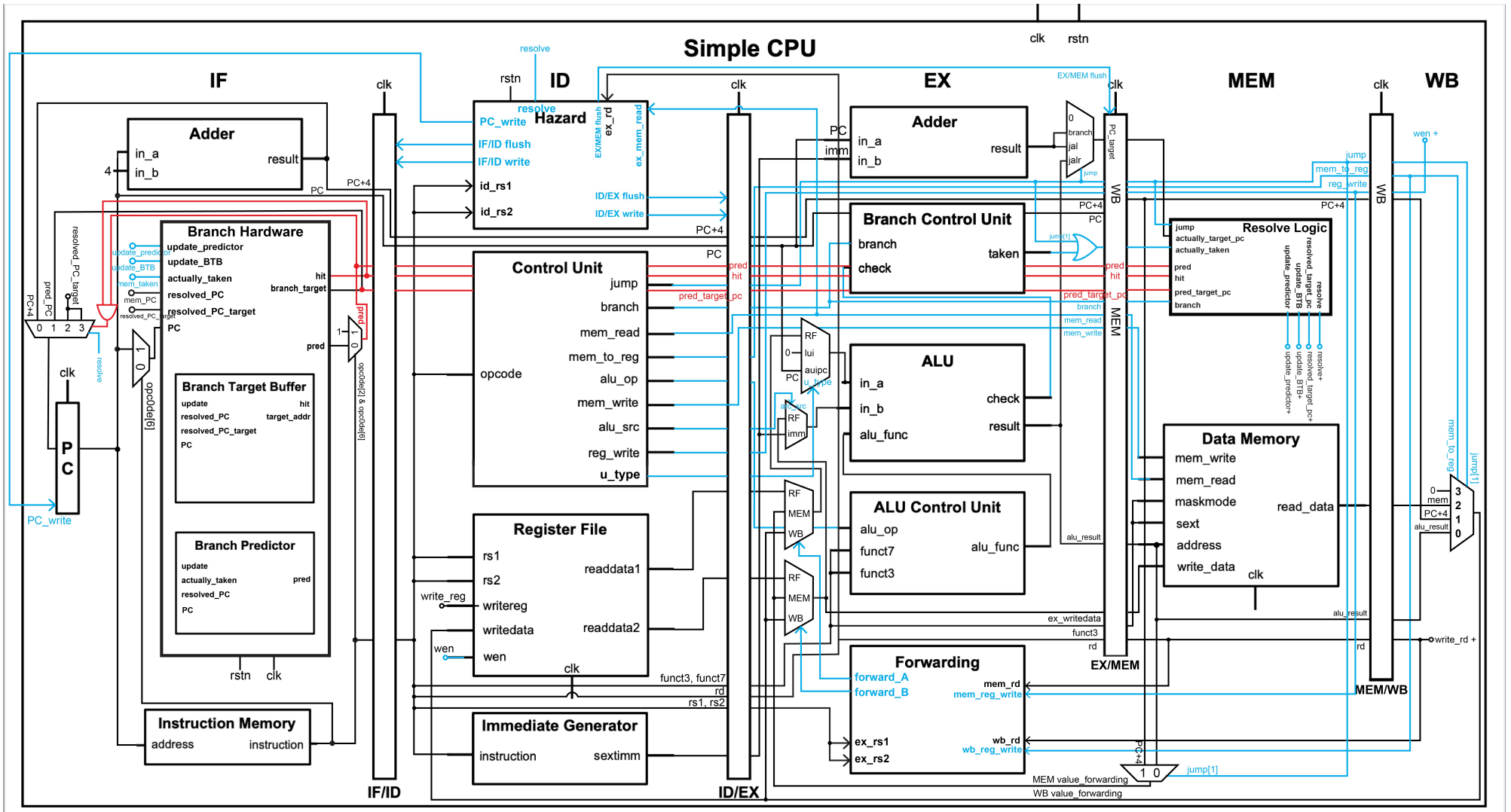
< branch >

pred & hit	actually_taken	pred_target == actual_target	resolve	resolved_target_PC
0	0	x	0	0
0	1	x	1	mem_target_pc
1	0	x	1	mem_pc+4
1	1	0	1	mem_target_pc
1	1	1	0	mem_target_pc

< jump >
Note that always (pred, actually_taken) = 1

hit	pred_target == actual_target	resolve	resolved_target_PC
0	0	1	mem_target_pc
1	0	1	mem_target_pc
0	1	1	mem_target_pc
1	1	0	mem_target_pc

5. Diagram


Simple CPU
IF
ID
EX
MEM
WB
clk
rstn
resolve
Adder
in_a
in_b
result
4
PC
PC+4
Branch Hardware
update_predictor
update_BTB
actually_taken
resolved_PC
resolved_PC_target
PC
hit
branch_target
pred
Branch Target Buffer
update
resolved_PC
resolved_PC_target
PC
hit
target_addr
Branch Predictor
update
actually_taken
resolved_PC
PC
pred
Instruction Memory
address
instruction
P C
PC_write
Hazard
PC_write
IF/ID flush
IF/ID write
id_rs1
id_rs2
ID/EX flush
ID/EX write
EX/MEM flush
ex_rd
ex_mem_read
Control Unit
opcode
jump
branch
mem_read
mem_to_reg
alu_op
mem_write
alu_src
reg_write
u_type
Register File
rs1
rs2
writereg
writedata
wen
readdata1
readdata2
write_reg
Immediate Generator
instruction
sextimm
Adder
Branch Control Unit
branch
check
taken
ALU
in_a
in_b
alu_func
check
result
ALU Control Unit
alu_op
funct7
funct3
alu_func
Forwarding
forward_A
forward_B
mem_rd
mem_reg_write
wb_rd
wb_reg_write
ex_rs1
ex_rs2
MEM value_forwarding
WB value_forwarding
Resolve Logic
actually_target_pc
actually_taken
pred
hit
pred_target_pc
branch
Data Memory
mem_write
mem_read
maskmode
sext
address
write_data
read_data
IF/ID
ID/EX
EX/MEM
MEM/WB
You should add ports, wires, and MUXs to complete the diagram