Lecture 19: Speculative execution

CMPE 120: Computer Organization & Architecture

Olivia Weng

Logistics

- Reminder: check-in with professor is 1% of your final grade
  - Sign up link

- Midterm grades released last Friday

- HW4 out - start now if you haven't!!

Reviewing mid-semester feedback survey

- 29 / 67 students responded (43% response rate)
  - Positives
    - Drawing on slides live
    - Interactive teaching with in-class participation using iClicker (even if iClicker finicky)
    - Being open to questions
    - Practice midterm exam
    - 1-on-1 meetings
  - Negatives
    - Grading scheme is stressful
    - Lecture pace is just right and too fast
    - More examples
    - Release practice final exam earlier
    - Hard to know how to tackle homework

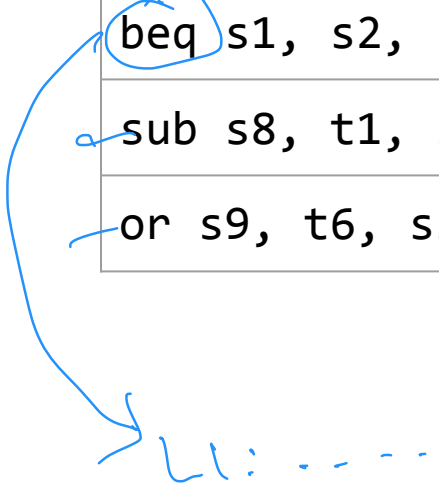
Addressing the negatives

- Grading scheme is stressful
  - Considering several options for grade remediation
- Lecture pace is just right and too fast
  - Will try to slow down a little
- More examples
  - Will try to include more in slides
- Release practice final exam earlier
  - Will do. Aiming for 12/1
- Hard to know how to tackle homework
  - TAs will host homework discussion and work sessions during their office hours

Is there any issue here?

branch if equal s1 == s2 goto L1

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	



Is there any issue here?

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	

- Depends! If branch is not taken, then no. If branch taken, then yes!

Control hazard

we'll only know if condition is true or Not in the Execute stage

- Branch condition not resolved until the Execute stage
  - Instructions will continue to be fetched !
  - These 2 instructions must be flushed if branch is taken

Control hazard

- Branch condition not resolved until the Execute stage
  - Instructions will continue to be fetched !
  - These 2 instructions must be flushed if branch is taken

- Control hazard: next instruction not decided yet (caused by branch)
  - ↳ pc ? = pc + 4

Control hazard

- Branch condition not resolved until the Execute stage
  - Instructions will continue to be fetched !
  - These 2 instructions must be flushed if branch is taken

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	

finally know if s1==s2

Control hazard

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	
…								
`L1: add s7, s3, s4`				F	D	E	M	W

How to handle control hazards?

- Idea: stall until branch is resolved

How to handle control hazards?

- Idea: stall until branch is resolved

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`						F	D	E
`or s9, t6, s5`							F	D

How to handle control hazards?

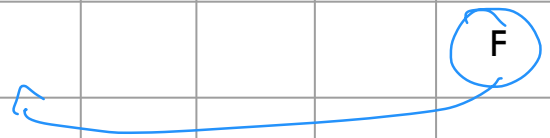
- Idea: stall until branch is resolved
  - Stalling for every branch is expensive…
    - 15-25% of all instructions are branches

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`						F	D	E
`or s9, t6, s5`							F	D

How to handle control hazards?

- Idea: stall until branch is resolved
  - Stalling for every branch is expensive…
    - 15-25% of all instructions are branches
  - Especially if branch is not taken--could have kept pipeline full!

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`						F	D	E
`or s9, t6, s5`							F	D



How to handle control hazards?

- Idea: Resolve the branch earlier in Decode stage
  - → Stall less!
  - Not as big penalty when branch not taken

How to handle control hazards?

- Idea: Resolve the branch earlier in Decode stage
  - → Stall less!
  - Not as big penalty when branch not taken

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`bubble/nop`			D	E	M	W		
`sub s8, t1, s3`		F	F	D	E	M	W	
`or s9, t6, s5`				F	D	E	M	W

How to handle control hazards?

- Idea: Resolve the branch earlier in Decode stage
  - → Stall less! Not as big penalty if branch taken either

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F						
…								
`bubble/nop`			D	E	M	W		
`L1: add s7, s3, s4`			F	D	E	M	W	

actually fetch correct instruction 1 cycle after beg's Decode stage

prevents writing incorrect data back.

How to handle control hazards?

- Early resolution in Decode is nice, but still need to stall at least 1 cycle…

How to handle control hazards?

- Early resolution in Decode is nice, but still need to stall at least 1 cycle…

- Observation: Pipeline remains full when branch not taken

PC = PC+4 !

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	

How to handle control hazards?

- Observation: Pipeline remains full when branch not taken

- Is there way to keep the pipeline full when branch is taken?

A: Good

B: Bad

C: Depends.

How to handle control hazards?

- Observation: Pipeline remains full when branch not taken

- Is there way to keep the pipeline full when branch is taken?

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`...`								
`L1: add s7, s3, s4`		F	D	E	M	W		

Speculative execution

- The problem is that we must wait for branches to resolve

- But program behavior is predictable!
  - Examples?

for (int i = 0; i < 1 000 000; i++) if

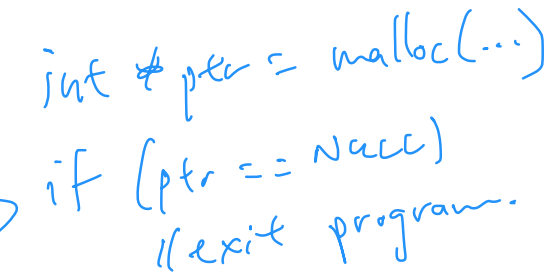
→ condition we check.

for, while } branches

Speculative execution

- The problem is that we must wait for branches to resolve

- But program behavior is predictable!
  - Loops
  - If statements checking for unlikely error conditions
  - Unconditional branches, i.e., jumps



Speculative execution

- Idea: Processor makes educated guess if branch is taken or not
  - Known as speculative execution or branch prediction
  - Guess in Fetch stage--no more hazard!

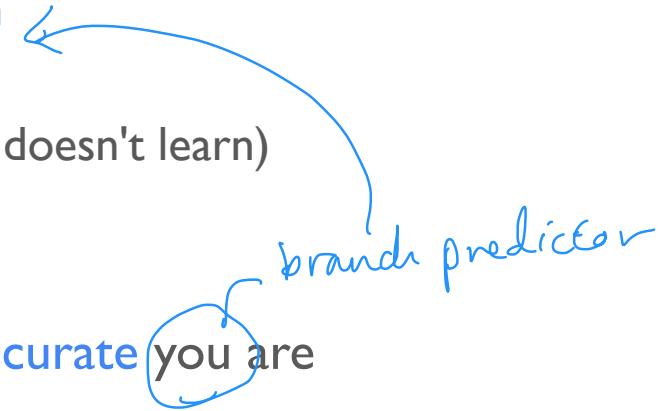
- Two key questions:
  - How to guess?
  - What to do when guess is wrong?

Static Branch Prediction

- Simple idea: Always predict branch not taken
  - NextPC = PC + 4
  - Known as static branch prediction (because doesn't learn)

Static Branch Prediction

- Simple idea: Always predict branch not taken
  - NextPC = PC + 4
  - Known as static branch prediction (because doesn't learn)



- Performance (from CPI) depends on how accurate you are

- What kind of program would this be bad for?

Static Branch Prediction

- Simple idea: Always predict branch not taken
  - NextPC = PC + 4
  - Known as static prediction (because doesn't learn)

- Performance (from CPI) depends on how accurate you are

- What kind of program would this be bad for?
  - Loop that's taken 95% of the time

How to predict branches?

- Static branch prediction
  - Predict always not taken — ~30 - 40% accurate
  - Predict always taken — 60 - 70% accurate.
  - Predict backward branches are taken, forward are not taken

How to predict branches?

- Static branch prediction
  - Predict always not taken
  - Predict always taken
  - Predict backward branches are taken, forward are not taken

- Pros:
  - Simple to implement

- Cons:
  - Poor accuracy

CPI tied to accuracy for branches
to potentially > 1

How to predict branches?

- Dynamic branch prediction
  - Can we learn from the past and use it to predict the future?

How to predict branches?

- Dynamic branch prediction
  - Can we learn from the past and use it to predict the future?

- Pros:
  - Prediction based on history can adapt to changes in branching behavior

- Cons:
  - More complex to implement (need additional hardware)

↑ high branch prediction accuracy
↓ need more hardware.

Dynamic branch prediction

1 => taken

1-bit history

0 => not taken

- 1-bit branch predictor: guess branch will take same direction as last time
  - Also known as last time predictor

history predict

1 => taken => which wrong => update history 0

0 => not taken => which wrong => update history 1

1 => taken => right => stay same 1

0 => not taken => right => stay same 0

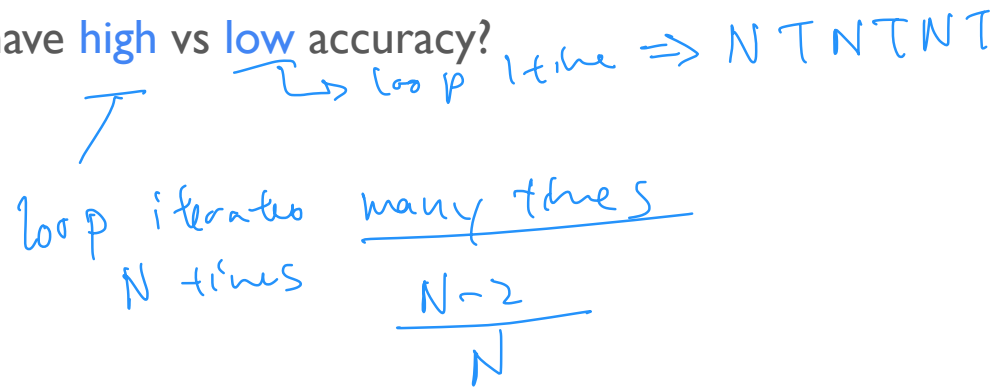
Dynamic branch prediction

- 1-bit branch predictor: guess branch will take same direction as last time
  - Also known as last time predictor
  - Ex: Last time branch was taken → Predict taken
  - Ex: Last time branch was not taken → Predict not taken

Dynamic branch prediction

- 1-bit branch predictor: guess branch will take same direction as last time
  - Also known as last time predictor

- Always mispredicts first and last iteration of loop branch
  - What kind of loops will have high vs low accuracy?



What is needed to implement 1-bit branch predictor?

- We know the direction (taken vs not taken).

- We also need to know the…

What is needed to implement 1-bit branch predictor?

- We know the direction (taken vs not taken).

- We also need to know the branch target (i.e, where to jump)
  - So that we can update the PC correctly

What is needed to implement 1-bit branch predictor?

- We know the direction (taken vs not taken).

- We also need to know the branch target (i.e, where to jump)
  - So that we can update the PC correctly

- Observation: Target address remains the same for conditional branches
  - Across dynamic instances

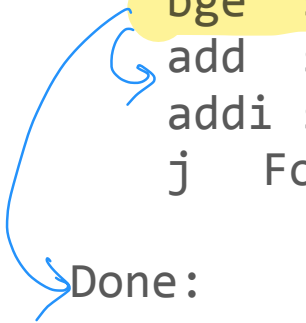
How to know where to jump to?

- Observation: Target address remains the same for conditional branches
  - Across dynamic instances

```
    addi s1, zero, 0    # s1 = sum
    addi s0, zero, 0    # s0 = i
    addi t0, zero, 10   # t0 = 10

For:                    # for (i = 0; i < 10; i += 1)
    bge  s0, t0, Done
    add  s1, s1, s0     # sum = sum + i
    addi s0, s0, 1      # i = i + 1
    j   For

Done:
```



How to know where to jump to?

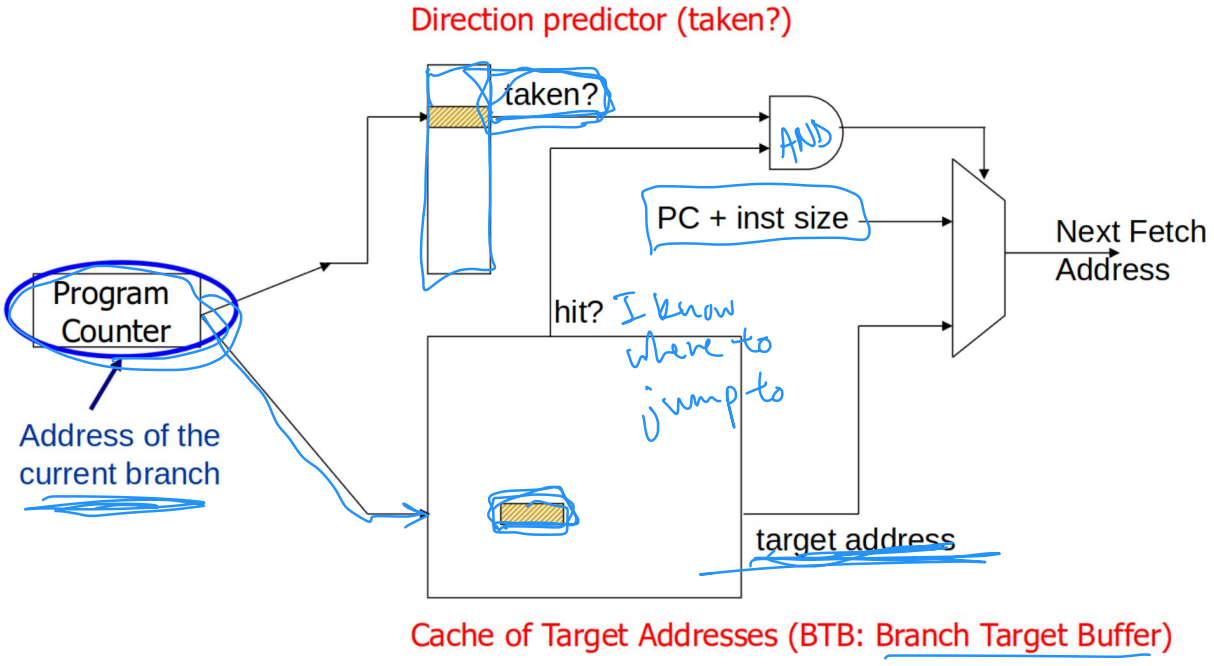
- Observation: Target address remains the same for conditional branch

- Idea: store target address from last time and access it with the PC
  - Called the Branch Target Buffer (BTB)

points to branch instr.

How to know where to jump to?

- Idea: store target address from last time and access it with the PC
  - Called the Branch Target Buffer (BTB)



Dynamic branch prediction

- 1-bit branch predictor
- 2-bit branch predictor
- Cache of target addresses (Branch target buffer)

What if I guess wrong?

- Branch condition not resolved until the Execute stage
  - These 2 instructions must be flushed if branch is taken
  - Known as flushing the pipeline

	Cycle							
	1	2	3	4	5	6	7	8
`beq s1, s2, L1`	F	D	E	M	W			
`sub s8, t1, s3`		F	D	E	M	W		
`or s9, t6, s5`			F	D	E	M	W	

- Branch misprediction penalty: # of instructions that must be flushed