Branch Prediction, Exceptions, and

CSCI 237: Computer Organization
25th Lecture, Wednesday, November 5, 2025

Kelly Shaw

1

Administrative Details

- Lab #5 checkpoint due Tuesday at 11pm
- Read CSAPP Ch. 4.6 and 6.1-6.2
- Apply to be a TA by Nov. 7
- Faculty candidate engagement

2

Last Time: Overcoming Hazards

- Data Forwarding
- Branch Prediction

3

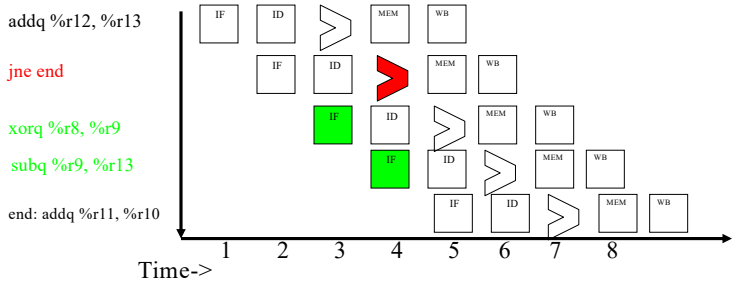
Today: Branch Prediction, Exceptions, and Storage

- Branch Prediction
- Exceptions
- Storage technologies and trends (Ch 6.1)
  - Memory technologies

4

Solution: Branch Prediction

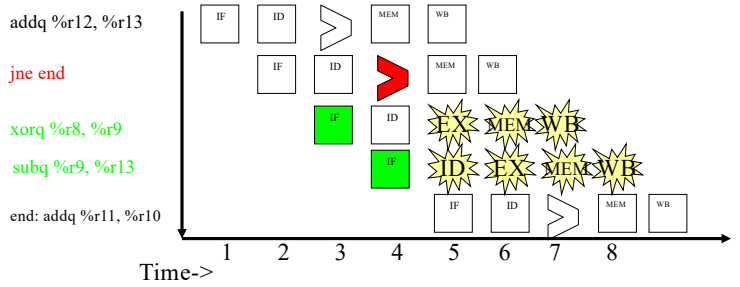
First: Always predict not taken
If we are right, how many cycles do we stall? 0



5

Solution: Branch Prediction

First: Always predict not taken
If we are wrong, then flush incorrect instruction(s)
How many cycles do we stall? 2



6

Branch Prediction

```
        rrmovq %r8, %r10     #i
        subq %r9, %r10       #i-n
        jge end
loop:   #do some work if i-n < 0

        irmovq $1, %r11
        addq %r11, %r8       #i++
        rrmovq %r8, %r10
        subq %r9, %r10       #i-n
        jl loop              #i-n<0
end:
```

```
for(i; i < n; i++)
        //do some work
```

Is jge often taken or not taken? **Not Taken**
Is jl often taken or not taken? **Taken**

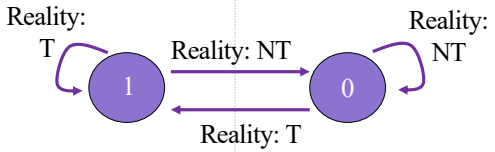
Conclusion: We want a prediction that is unique to each branch. Look up prediction by PC

7

7

Simplest Branch Predictor

Strategy: Predict whatever happened last time, then update the predictor for next time



If in this state: **Predict Taken**

If in this state: **Predict Not Taken**

8

8

Branch Prediction

```
       rrmovq %r8, %r10     #i
       subq %r9, %r10       #i-n
       jge end
loop:  #do some work if i-n < 0

       irmovq $1, %r11
       addq %r11, %r8       #i++
       rrmovq %r8, %r10
       subq %r9, %r10       #i-n
       jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
       //do some work
```

Consider two loop instances for a single static loop. The state of the predictor persists across iterations.

Iteration	1	2 … x	1	2 … y
CurState	0			
Prediction				
Reality				
NextState				

9

9

Branch Prediction

```
       rrmovq %r8, %r10     #i
       subq %r9, %r10       #i-n
       jge end
loop:  #do some work if i-n < 0

       irmovq $1, %r11
       addq %r11, %r8       #i++
       rrmovq %r8, %r10
       subq %r9, %r10       #i-n
       jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
       //do some work
```

Iteration	1	2 … x	1	2 … y
CurState	0	1		
Prediction	NT			
Reality	T			
NextState	1			

10

10

Branch Prediction

```
       rrmovq %r8, %r10     #i
       subq %r9, %r10       #i-n
       jge end
loop:  #do some work if i-n < 0

       irmovq $1, %r11
       addq %r11, %r8       #i++
       rrmovq %r8, %r10
       subq %r9, %r10       #i-n
       jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
       //do some work
```

Iteration	1	2 … x	1	2 … y
CurState	0	1 1		
Prediction	NT	T		
Reality	T	T		
NextState	1	1		

11

11

Branch Prediction

```
       rrmovq %r8, %r10     #i
       subq %r9, %r10       #i-n
       jge end
loop:  #do some work if i-n < 0

       irmovq $1, %r11
       addq %r11, %r8       #i++
       rrmovq %r8, %r10
       subq %r9, %r10       #i-n
       jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
       //do some work
```

Iteration	1	2 … x	1	2 … y
CurState	0	1 1	0	
Prediction	NT	T T		
Reality	T	T NT		
NextState	1	1 0		

12

12

Branch Prediction

```
        rrmovq %r8, %r10     #i
        subq %r9, %r10       #i-n
        jge end
loop:   #do some work if i-n < 0

        irmovq $1, %r11
        addq %r11, %r8       #i++
        rrmovq %r8, %r10
        subq %r9, %r10       #i-n
        jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
        //do some work
```

Iteration	1	2	…	x	1	2	…	y
CurState	0	1		1	0	1		
Prediction	NT	T		T	NT			
Reality	T	T		NT	T			
NextState	1	1		0	1			

13

13

Branch Prediction

```
        rrmovq %r8, %r10     #i
        subq %r9, %r10       #i-n
        jge end
loop:   #do some work if i-n < 0

        irmovq $1, %r11
        addq %r11, %r8       #i++
        rrmovq %r8, %r10
        subq %r9, %r10       #i-n
        jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
        //do some work
```

Iteration	1	2	…	x	1	2	…	y
CurState	0	1		1	0	1		1
Prediction	NT	T		T	NT	T		
Reality	T	T		NT	T	T		
NextState	1	1		0	1	1		

14

14

Branch Prediction

```
        rrmovq %r8, %r10     #i
        subq %r9, %r10       #i-n
        jge end
loop:   #do some work if i-n < 0

        irmovq $1, %r11
        addq %r11, %r8       #i++
        rrmovq %r8, %r10
        subq %r9, %r10       #i-n
        jl loop              #i-n<0
end:
```

```
for(i; i<n;i++)
        //do some work
```

When are we wrong?????

First and last iteration of each loop

Iteration	1	2	…	x	1	2	…	y
CurState	0	1		1	0	1		1
Prediction	NT	T		T	NT	T		T
Reality	T	T		NT	T	T		NT
NextState	1	1		0	1	1		0

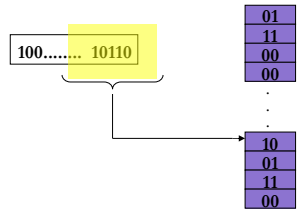
15

15

High-level Overview: Simplest Branch Predictors

Branch History Table (BHT)

- Memory indexed by lower portion of address PC
- Entry contains few bits specifying prediction
- Accessed in **IF** stage so fetching of target occurs in next cycle



16

16

High Level Overview: Real Branch Predictors

- Limited space, so different branches may map to the same predictor
  - errors?
- TargetPC saved with predictor

17

17

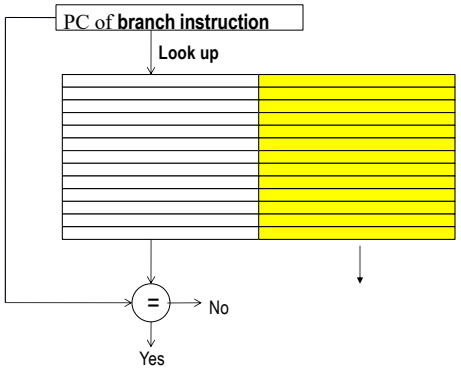
Branch Prediction

- If we're going to predict taken, we need to know where to branch to earlier than when we determine where the branch actually goes.
  - How?

18

18

High Level Overview: Branch Target Buffer (BTB)



If there is a match on PC of branch, corresponding predicted target instruction address returned

19

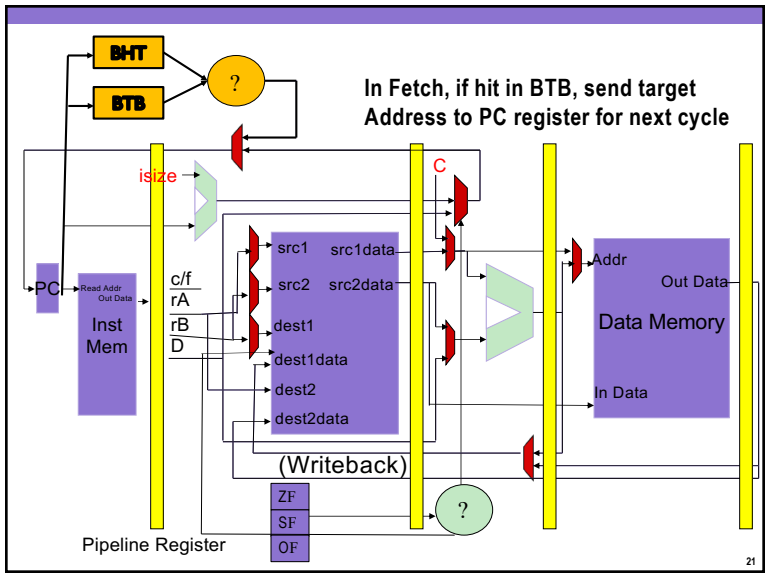
19

Real Branch Predictors

- Branch History Table (BHT)
  - Stores predictions for individual branch instructions
  - Store more than 1 bit to increase prediction accuracy
- Branch Target Buffer (BTB)
  - For branches that are predicted taken, stores target address
- Both accessed in FETCH stage on jump instructions

20

20



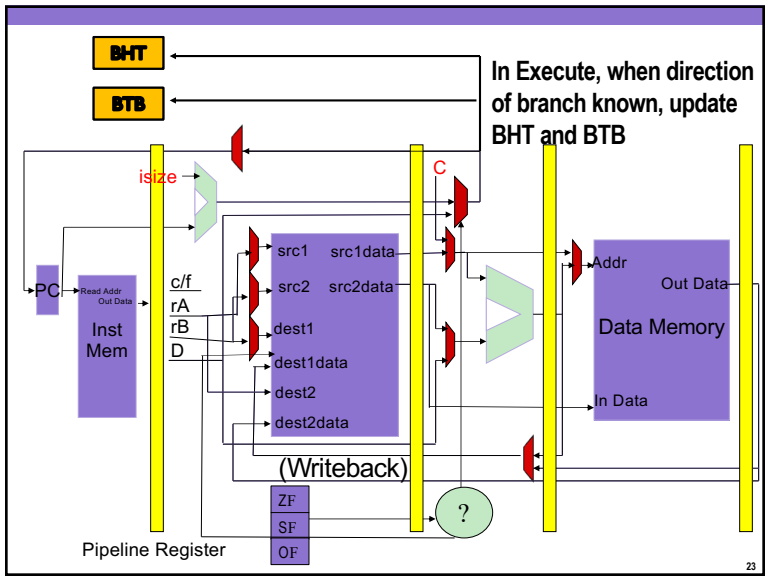
21

Real Branch Predictors

- Branch History Table (BHT)
  - Stores predictions for individual branch instructions
  - Store more than 1 bit to increase prediction accuracy
- Branch Target Buffer (BTB)
  - For branches that are predicted taken, stores target address
- Both accessed in FETCH stage on jump instructions
- Both updated after EXECUTE stage

22

22



23

Advantages of Branch Prediction

- Highly predictable branches have no stalls
- Works well with loops.
- All hardware - no compiler necessary

24

24

Disadvantages/Limits of Branch Prediction

- Large penalty when wrong
  - Badly behaved branches kill performance
- Large amount of chip area used for BTB and BHT
- Non-productive instructions waste energy and dissipate heat

25

25

What about unconditional control instructions?

- `call` and unconditional `jmp` *always* use target in instruction
  - Have to decode the instruction to get those values
  - But you could use branch prediction for those as well to prevent stalling
- `ret` may go back to multiple locations if called from multiple locations
  - Can stall until return address obtained from memory
  - Return address prediction can be done via a stack in HW

26

26

Pipeline Summary

- Concept
  - Break instruction execution into 5 stages
  - Run instructions through in *pipelined* mode
- Limitations
  - Can't handle dependencies between instructions when instructions follow too closely
  - Data dependencies
    - One instruction writes register, later one reads it
  - Control dependencies
    - Instruction sets PC in way that pipeline did not predict correctly
- Solutions to hazards other than stalling
  - Data hazards
    - Data forwarding
  - Control hazards
    - Branch prediction

27

27

Why Should Programmers Care

- Performance matters
  - Lots of branches that aren't predictable will slow down your code
    - Why conditional moves are good
  - Lots of data dependences slow down your code too
- In general, compiler and hardware optimizations do a good job
  - But, the compiler can't always determine if there is a true data dependence when pointers are being used
  - Sometimes hardware will mispredict branches and result in wasted cycles
- Sometimes we can restructure our code to make things easier for the compiler
  - Remove unnecessary branches or move them out of loops, link different cases together (if/else if/else instead of sequence of if statements)
  - Loop unrolling
    - Make loop bodies longer so more instructions to choose from

28

28

Practice on Your Own

- Draw the pipeline diagram for the following code, assuming predict not taken but with the reality of the branch specified in comments

```
            irmovq $4, %r8
            subq %rdi, %r8
            jl end                  # not taken
            mrmovq (%rsi), %r9
            irmovq $0, %r10
            subq %r10, %r9
            je end                  # taken
            mrmovq 8(%rsi), %r11
            addq %r11, %r9
end:
```

30

30

Today: Branch Prediction, Exceptions, and Storage

- Branch Prediction
- Exceptions
- Storage technologies and trends (Ch 6.1)
  - Memory technologies

31

31

Dealing w/ Exceptions

- Conditions under which processor cannot continue normal op
- Causes
  - Halt instruction
  - Bad address for instruction or data
  - Invalid instruction
- Typical Desired Action
  - Complete some instructions
    - Either current or previous (depends on exception type)
  - Discard others
  - Call exception handler
    - Like an unexpected procedure call
- Our Implementation
  - Halt when instruction causes exception

32

32

Exception Examples

- Detect in Decode Stage

```
jmp $-1                 # Invalid jump target

.byte 0xFF              # Invalid instruction code

halt                    # Halt instruction
```

- Detect in Memory Stage

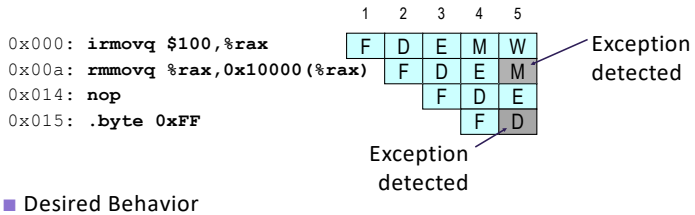
```
irmovq $100, %rax
rmmovq %rax, 0x10000(%rax) # invalid address
```

33

33

Exceptions in Pipeline Processor #1

```
irmovq $100,%rax
rmmovq %rax,0x10000(%rax) # Invalid address
nop
.byte 0xFF                # Invalid instruction code
```



- Desired Behavior
  - `rmmovq` should cause exception
  - Following instructions should have no effect on processor state

34

34