The Y86 Pipelined Datapath: Data and Control Hazards

CSCI 237: Computer Organization
25th Lecture, Wednesday, November 6, 2024

Kelly Shaw

Slides originally designed by Bryant and O'Hallaron @ CMU for use with Computer Systems: A Programmer's Perspective, Third Edition

1

1

Administrative Details

- Lab #4 due Thursday at 11pm
- Quiz on Glow open today at 2:35pm, due Friday at 2:35pm
- Read CSAPP 4.6 and 6.1
- Lab #5 partner signup form due today at noon
  - You and your partner must fill out the form

2

2

Last Time: The Y86 Pipelined Datapath

- Construction of a pipelined datapath for Y86
  - Adding pipeline registers
  - Data hazards
  - Ways to deal with data hazards
    - Stalling
    - Data forwarding
  - Control hazards
    - Branch prediction

3

3

Today: The Y86 Pipelined Datapath

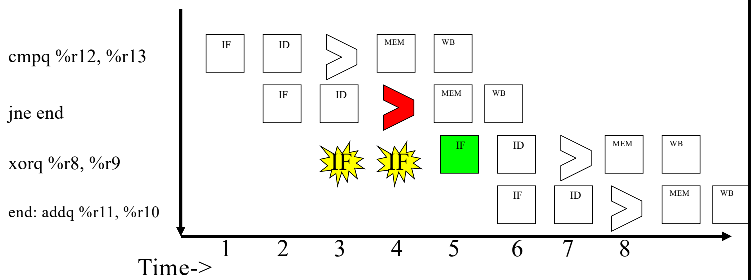
- Construction of a pipelined datapath for Y86
  - Control hazards
    - Branch prediction
  - Exceptions

4

4

Control Hazard: Stall until target known

In what cycle does the nextPC get calculated for the `jne`? End of 4
In what cycle does the `xorq` get fetched? Beginning of 3

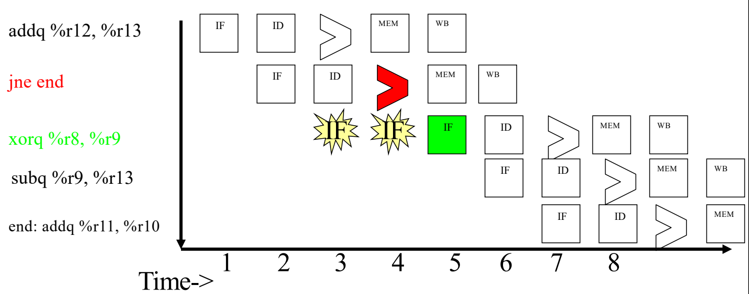


Barriers to Pipeline Performance

- Uneven stages
- Pipeline register delays
- Data Hazards
- Control Hazards
  - ***Whether*** an instruction will execute depends on the outcome of a control instruction still in the pipeline

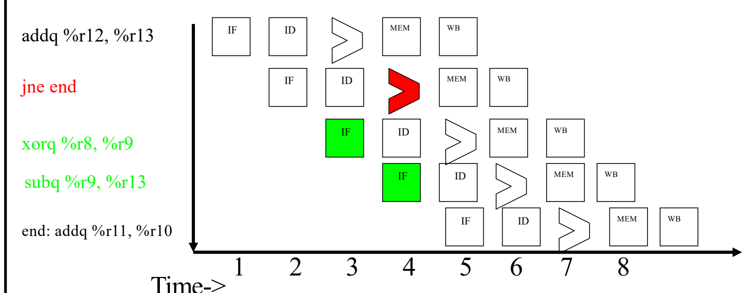
Solution: Branch Prediction

Guess which way the branch will go *before* calculation occurs.
Clean up if predictor is wrong.



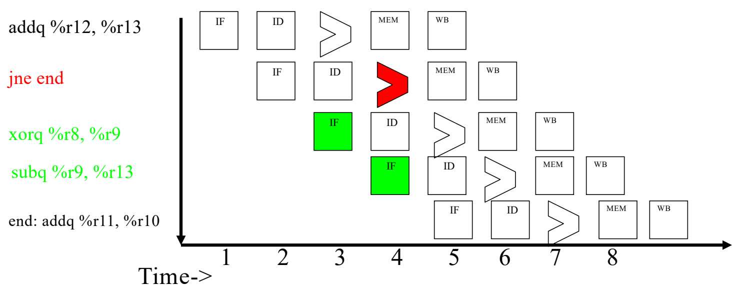
Solution: Branch Prediction

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



Solution: Branch Prediction

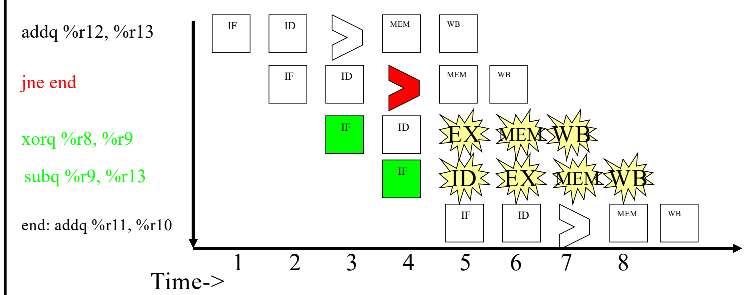
First: Always predict not taken
If we are right, how many cycles do we stall? 0



9

Solution: Branch Prediction

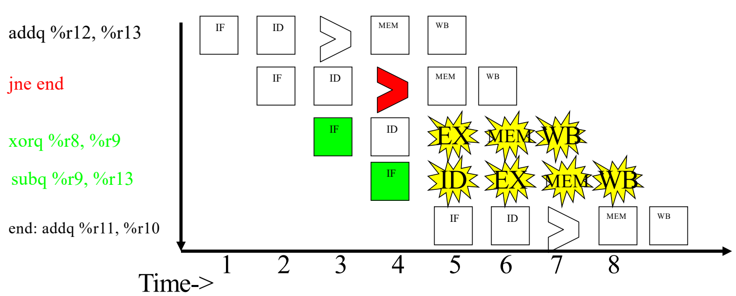
First: Always predict not taken
If we are wrong, then flush incorrect instruction(s)



10

Solution: Branch Prediction

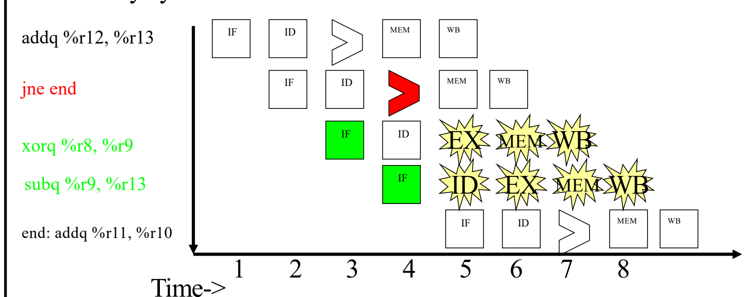
First: Always predict not taken
If we are wrong, then flush incorrect instruction(s)
How many cycles do we stall?



11

Solution: Branch Prediction

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



12

Branch Prediction

```
        rrmovq %r8, %r10     #i              for(i; i < n; i++)
        subq %r9, %r10       #i-n                   //do some work
        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:
```

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

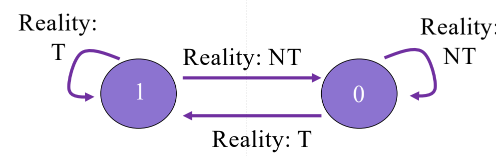
Conclusion: We want a prediction that is unique to each branch. Look up prediction by PC

13

13

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**

14

14

Branch Prediction

```
        rrmovq %r8, %r10     #i              for(i; i<n;i++)
        subq %r9, %r10       #i-n                   //do some work
        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:
```

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							

15

15

Branch Prediction

```
        rrmovq %r8, %r10     #i              for(i; i<n;i++)
        subq %r9, %r10       #i-n                   //do some work
        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:
```

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

16

16

Branch Prediction

```
        rrmovq %r8, %r10    #i              for(i; i<n;i++)
        subq %r9, %r10      #i-n                   //do some work
        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:
```

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

17

17

Branch Prediction

```
        rrmovq %r8, %r10    #i              for(i; i<n;i++)
        subq %r9, %r10      #i-n                   //do some work
        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:
```

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

18

18

Branch Prediction

```
        rrmovq %r8, %r10    #i              for(i; i<n;i++)
        subq %r9, %r10      #i-n                   //do some work
        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:
```

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			

19

19

Branch Prediction

```
        rrmovq %r8, %r10    #i              for(i; i<n;i++)
        subq %r9, %r10      #i-n                   //do some work
        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:
```

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		

20

20

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?????

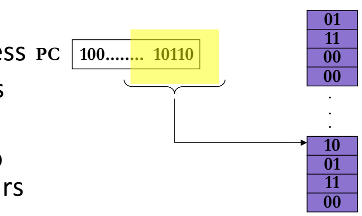
First and last iteration of each loop

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

High-level Overview: Simplest Branch Predictors

Branch History Table (BHT)

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



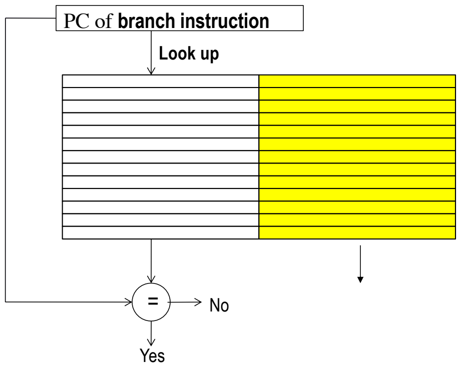
High Level Overview: Real Branch Predictors

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

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?

High Level Overview: Branch Target Buffer (BTB)

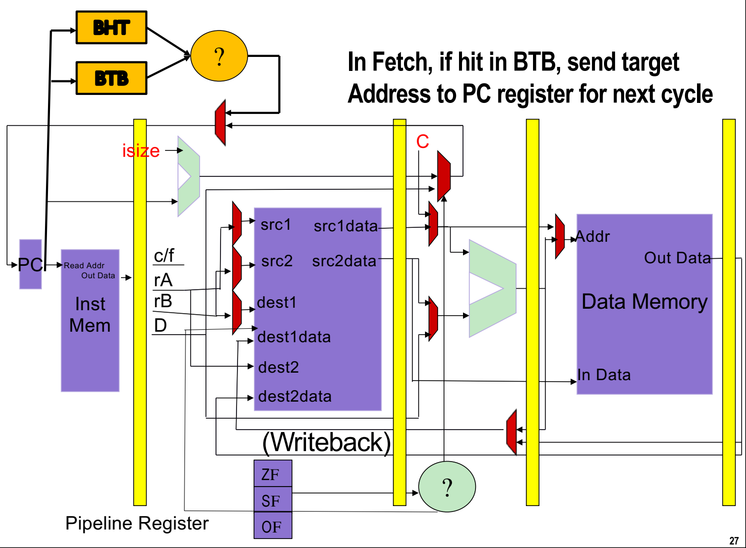


If there is a match on PC of branch, corresponding predicted target instruction address returned

Real Branch Predictors

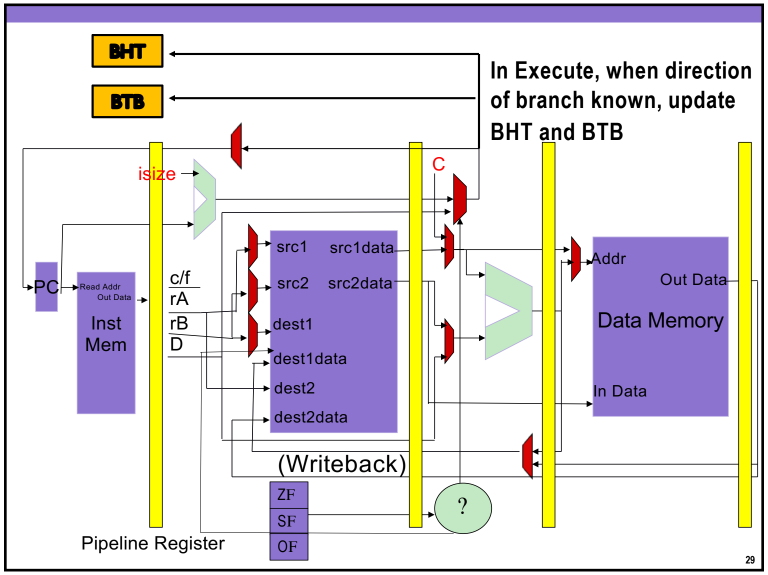
- Branch History Table (BHT)
  - Stores predictions for individual branch instructions
  - Store more than 1 bit to increase prediction accuracy
- Branch Target Buffer (BTB)
  - For branches that are predicted taken, stores target address
- Both accessed in FETCH stage on jump instructions

In Fetch, if hit in BTB, send target Address to PC register for next cycle



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



29

Advantages of Branch Prediction

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

30

30

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

31

31

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

32

32

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

33

33

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

34

34

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:
```

35

35

Today: The Y86 Pipelined Datapath

- Construction of a pipelined datapath for Y86
  - Control hazards
    - Branch prediction
  - Exceptions

36

36