CPSC 313: Computer Hardware and Operating Systems

Unit 2: Pipelining

Pipelining: Performance (Cycle Counting)

Administration

- Quiz 2: Don’t miss your time! You should already have registered.
- Quiz 0, Quiz 1, and Quiz 1 retake results available on PrairieLearn
- Lab 4: Due Sunday
- Tutorial 3: This week

- **> As always: See the Syllabus on Canvas for deadlines.**

Today

- Learning outcomes
  - Conduct accurate cycle counts of programs
  - Identify mis-predicts and stalls in code
  - Analyze the interaction of branch prediction and loop structuring
  - Determine when different predictors would improve performance
- How we'll get there
  - Go over a simple program under several different scenarios
  - Tips for today's exercise.

Let's practice computing cycle counts! (1a)

```
        xorq   %rax, %rax   # rax = 0 (count)
        xorq   %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx      # to increment
Loop:   irmovq 5, %rdi      # rdi = 5
        subq %rsi, %rdi     # check end cond
        jle Done            # if 5-j <= 0, jmp
        addq %rbx, %rsi     # j++
        addq %rbx, %rax     # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Assume no forwarding and no branch prediction

1. How many bubbles are produced?
2. How many CPI did this program achieve?

Let's practice computing cycle counts! (1b)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard 3 stalls

Control Hazard 2 stalls

Assume no forwarding and no branch prediction

1. How many bubbles are produced?
2. How many CPI did this program achieve?

Let's practice computing cycle counts! (1c)

```
        xorq   %rax, %rax   # rax = 0 (count)
        xorq   %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx      # to increment
Loop:   irmovq 5, %rdi      # rdi = 5
        subq %rsi, %rdi     # check end cond
        jle Done            # if 5-j <= 0, jmp
        addq %rbx, %rsi     # j++
        addq %rbx, %rax     # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard 3 stalls

Control Hazard 2 stalls

Assume no forwarding and no branch prediction

1. How many bubbles are produced?
   5 per iteration * 6 iterations = 30
2. How many CPI did this program achieve?

Let's practice computing cycle counts! (1d)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard 3 stalls

Control Hazard 2 stalls

Assume no forwarding and no branch prediction

1. How many bubbles are produced?
   5 per iteration * 6 iterations = 30
2. How many CPI did this program achieve?
   Instructions: 37
   - **3** before the loop
   - 6 * 3 (top of loop)
   - **5 * 3 (bottom of loop)**
   - **1** halt

Let's practice computing cycle counts! (1e)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard 3 stalls

Control Hazard 2 stalls

Assume no forwarding and no branch prediction

1. How many bubbles are produced?
   5 per iteration * 6 iterations = 30
2. How many CPI did this program achieve?
   Cycles:
   1 per instruction (37)
   4 to load the pipeline (4)
   5 * 6 for loop stalls (30)

CPI = **71**/37

Same program: With forwarding! (2a)

```
        xorq   %rax, %rax   # rax = 0 (count)
        xorq   %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx      # to increment
Loop:   irmovq 5, %rdi      # rdi = 5
        subq %rsi, %rdi     # check end cond
        jle Done            # if 5-j <= 0, jmp
        addq %rbx, %rsi     # j++
        addq %rbx, %rax     # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Assume **forwarding** and no branch prediction

1. How many bubbles are produced?
2. How many CPI did this program achieve?

Same program: With forwarding! (2b)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard? No! 0 stalls.

Control Hazard 2 stalls

Assume **forwarding** and no branch prediction

1. How many bubbles are produced?
2. How many CPI did this program achieve?

Same program: With forwarding! (2c)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard? No! 0 stalls.

Control Hazard 2 stalls

Assume **forwarding** and no branch prediction

1. How many bubbles are produced?
   2 per iteration = 2 * 6 = 12
2. How many CPI did this program achieve?

Same program: With forwarding! (2d)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Data Hazard? No! 0 stalls.

Control Hazard 2 stalls

Assume **forwarding** and no branch prediction

1. How many bubbles are produced?
   2 per iteration = 2 * 6 = 12
2. How many CPI did this program achieve?
   Still 37 instructions
   + 4 to load the pipeline
   + 12 stalls due to the conditional jump

CPI = 53/37

Now add branch prediction! (3a)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Assume **forwarding** and **always taken** branch prediction

1. How many bubbles are produced?
2. How many CPI did this program achieve?

Now add branch prediction! (3b)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Control Hazard

5 times we predict wrong
1 time we predict right

Assume **forwarding** and **always taken** branch prediction

1. How many instructions are squashed?
   5 * 2 = 10
2. How many CPI did this program achieve?
   37 + 4 (instructions + pipeline load)
   +10 for mis-predict.

CPI = 51 / 37

NOT MUCH BETTER!
We are predicting everything but the last branch wrong!

Better Predictor! (4a)

```
        xorq   %rax, %rax   # rax = 0 (count)
        xorq   %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx      # to increment
Loop:   irmovq 5, %rdi      # rdi = 5
        subq %rsi, %rdi     # check end cond
        jle Done            # if 5-j <= 0, jmp
        addq %rbx, %rsi     # j++
        addq %rbx, %rax     # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Control Hazard

5 times we predict right
1 time we predict wrong

Assume **forwarding** and **never taken** branch prediction

1. How many instructions are squashed?
   Just 2 on the last iteration
2. How many CPI did this program achieve?

Better Predictor! (4b)

```
        xorq    %rax, %rax   # rax = 0 (count)
        xorq    %rsi, %rsi   # rsi = 0 (j)
        irmovq 1, %rbx       # to increment
Loop:   irmovq 5, %rdi       # rdi = 5
        subq %rsi, %rdi      # check end cond
        jle Done             # if 5-j <= 0, jmp
        addq %rbx, %rsi      # j++
        addq %rbx, %rax      # count++
        jmp Loop
Done:   halt
```

```
count = 0;
for (j = 0; j < 5; j++)
   count++;
```

Control Hazard

5 times we predict right
1 time we predict wrong

Assume **forwarding** and **never taken** branch prediction

1. How many instructions are squashed?
   Just 2 on the last iteration
2. How many CPI did this program achieve?
   37 + 4 (instructions + pipeline load)
   +2 for mis-predict

CPI = 43/37

What is the best branch predictor?

- It depends on the program
- A (relatively) simple predictor that tends to work quite well:
  - Predict not taken for forward branches
  - Predict taken for backward branches
- Rationale:
  - Loops are backward branches – loops generally actually loop
  - Ifs are forward branches – they are taken less often

A Note on PL's explanations (on a new little program)

```
# With no forwarding and "always taken" branch prediction:

        irmovq 1, %r11      # No stalls
        irmovq 3, %rax      # No stalls
loop:   subq %r11, %rax     # 3 stalls due to data hazard on %rax (sometimes also %r11)
                            # 2 stalls due to data hazard on %rax (2 times)
        jg loop             # 2 squashes due to branch prediction failure
        halt                # No stalls
```

A Note on PL's explanations

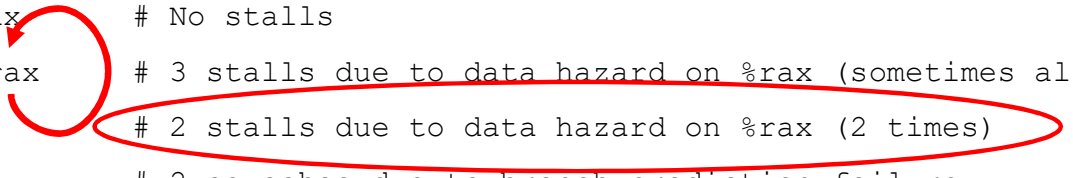
```
# With no forwarding and "always taken" branch prediction:

        irmovq 1, %r11      # No stalls
        irmovq 3, %rax      # No stalls
loop:   subq %r11, %rax     # 3 stalls due to data hazard on %rax (sometimes also %r11)
                            # 2 stalls due to data hazard on %rax (2 times)
        jg loop             # 2 squashes due to branch prediction failure
        halt                # No stalls
```

A Note on PL's explanations

```
# With no forwarding and "always taken" branch prediction:

        irmovq 1, %r11      # No stalls
        irmovq 3, %rax      # No stalls
loop:   subq %r11, %rax     # 3 stalls due to data hazard on %rax (sometimes also %r11)
                            # 2 stalls due to data hazard on %rax (2 times)
        jg loop             # 2 squashes due to branch prediction failure
        halt                # No stalls
```



In-class exercise: Tips for Peer Problems

- Walk through and comment the code so you understand how registers are being used and what the program is doing.
- Break the program into chunks where each instruction in a chunk will be executed the same number of times (compilers call these “basic blocks”).
- Identify any data/control hazards given the implementation assigned to you.
- Now, try to answer the questions!