PIPELINING: BRANCH AND MULTICYCLE INSTRUCTIONS

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Control Hazards

Sample C++ code

```cpp
for (i=100; i > 0; i--) {
    sum = sum + i;
}

total = total + sum;
```

How many branches in this code?
Sample C++ code

```cpp
for (i=100; i > 0; i--) {
    sum = sum + i;
}
```

```cpp
total = total + sum;
```

What are possible target instructions?
Control Hazards

- Sample C++ code

```cpp
for (i=100; i > 0; i--) {
    sum = sum + i;
}
total = total + sum;
```

What happens inside the pipeline?
Handling Control Hazards

1. introducing stall cycles and delay slots
   - How many cycles/slots?
   - One branch per every six instructions on average!!

```
add r1, r0, #100
beq r0, r1, next
nothing
nothing
add r2, r2, r1
sub r1, r1, #1
```

2 additional delay slots per 6 cycles!
Handling Control Hazards

1. introducing stall cycles and delay slots
   - How many cycles/slots?
   - One branch per every six instructions on average!!

```
add r1, r0, #100
for: beq r0, r1, next
nothing
add r2, r2, r1
sub r1, r1, #1
J for
nothing
```

1 additional delay slot, but longer path
Handling Control Hazards

1. introducing stall cycles and delay slots
   - How many cycles/slots?
   - One branch per every six instructions on average!!

```
add r1, r0, #100
beq r0, r1, next
nothing
add r2, r2, r1
J for
sub r1, r1, #1
next: add r3, r3, r2
```

Reordering instructions may help
Handling Control Hazards

1. introducing stall cycles and delay slots

- How many cycles/slots?
- One branch per every six instructions on average!!

Jump and function calls can be resolved in the decode stage.
Handling Control Hazards

1. Introducing stall cycles and delay slots
2. Predict the branch outcome
   - Simply assume the branch is taken or not taken
   - Predict the next PC

Example:

```
for:  beq r0, r1, next

add r1, r0, #100
add r2, r2, r1
sub r1, r1, #1
J for
next: add r3, r3, r2
```

May need to cancel the wrong path
Multicycle Instructions

- Not all of the ALU operations complete in one cycle
  - Typically, FP operations need more time
Multicycle Instructions

- Not all of the ALU operations complete in one cycle
  - pipelined and un-pipelined multicycle functional units

Pipelined vs. un-pipelined?
Multicycle Instructions

- Structural hazards
  - potentially multiple RF writes

Possibly multiple writes to the Register File
Multicycle Instructions

- Data hazards
  - more read-after-write hazards

```plaintext
load f4, 0(r2)
mul f0, f4, f6
add f2, f0, f8
store f2, 0(r2)
```
Multicycle Instructions

- Data hazards
  - potential wire-after-write hazards

Load: f4, 0(r2)
mul f2, f4, f6
add f2, f0, f8
store f2, 0(r2)

Out of Order
Write-back!!
Multicycle Instructions

- Data hazards
  - potential wire-after-write hazards

```
load f4, 0(r2)
mul f2, f4, f6
add f2, f0, f8
store f2, 0(r2)
```
Multicycle Instructions

- Imprecise exception
  - Instructions do not necessarily complete in program order

```
load f4, 0(r2)
mul f2, f4, f6
add f3, f0, f8
store f2, 0(r2)
```

Overflow!!
Multicycle Instructions

- Imprecise exception
  - state of the processor must be kept updated with respect to the program order

In-order register file updates
Multicycle Instructions

mul f2, f4, f6
add f4, f0, f1
sub f6, f3, f7

Reorder Buffer
Reorder Buffer

- Multicycle Instructions

```
mul f2, f4, f6
add f4, f0, f1
sub f6, f3, f7
```