INSTRUCTION LEVEL PARALLELISM

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

CS/ECE 6810: Computer Architecture

Overview

- Announcement
  - Tonight: release HW2 (due 11:59PM, Sept. 18)
    - Note: late submission = no submission
    - One of your lowest assignment scores will be dropped ☺

- This lecture
  - Recap multicycle
  - Impacts of data dependence
  - Pipeline performance
  - Instruction level parallelism

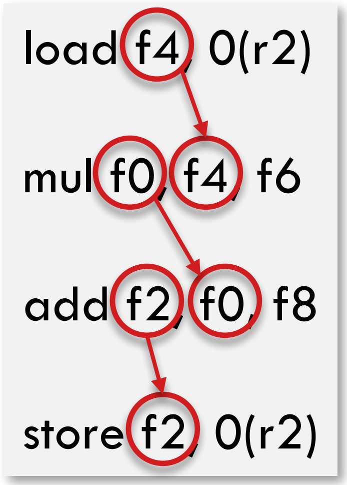
Multicycle Instructions

- Data hazards
  - more read-after-write hazards

```
load f4, 0(r2)

mul f0, f4, f6

add f2, f0, f8

store f2, 0(r2)
```

Multicycle Instructions

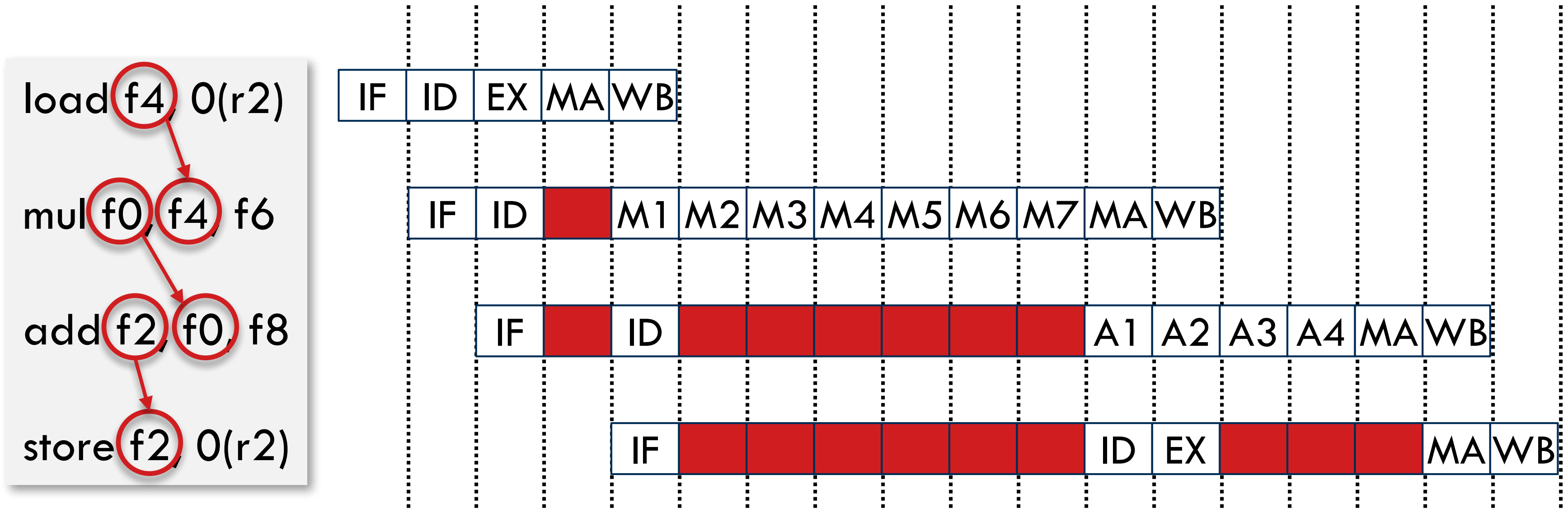
- Data hazards
  - more read-after-write hazards



```
load f4, 0(r2)

mul f0, f4, f6

add f2, f0, f8

store f2, 0(r2)
```

Multicycle Instructions

- Data hazards
  - more read-after-write hazards



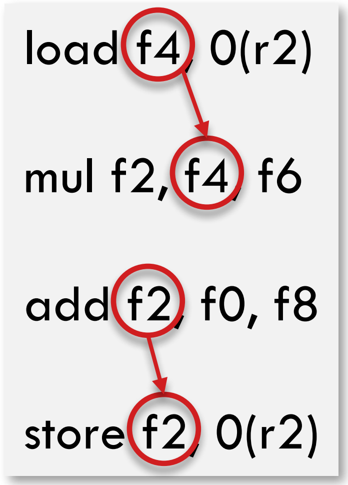
Multicycle Instructions

- Data hazards
  - potential write-after-write hazards

```
load f4, 0(r2)

mul f2, f4, f6

add f2, f0, f8

store f2, 0(r2)
```

Multicycle Instructions

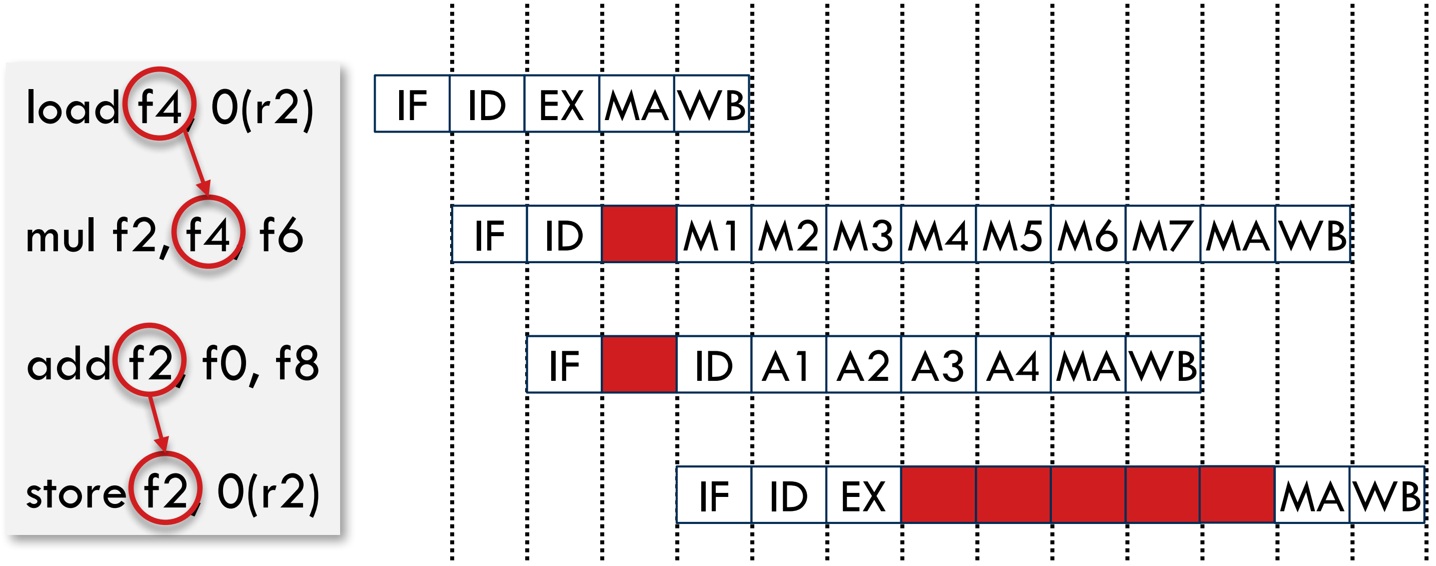
- Data hazards
  - potential write-after-write hazards



```
load f4, 0(r2)

mul f2, f4, f6

add f2, f0, f8

store f2, 0(r2)
```

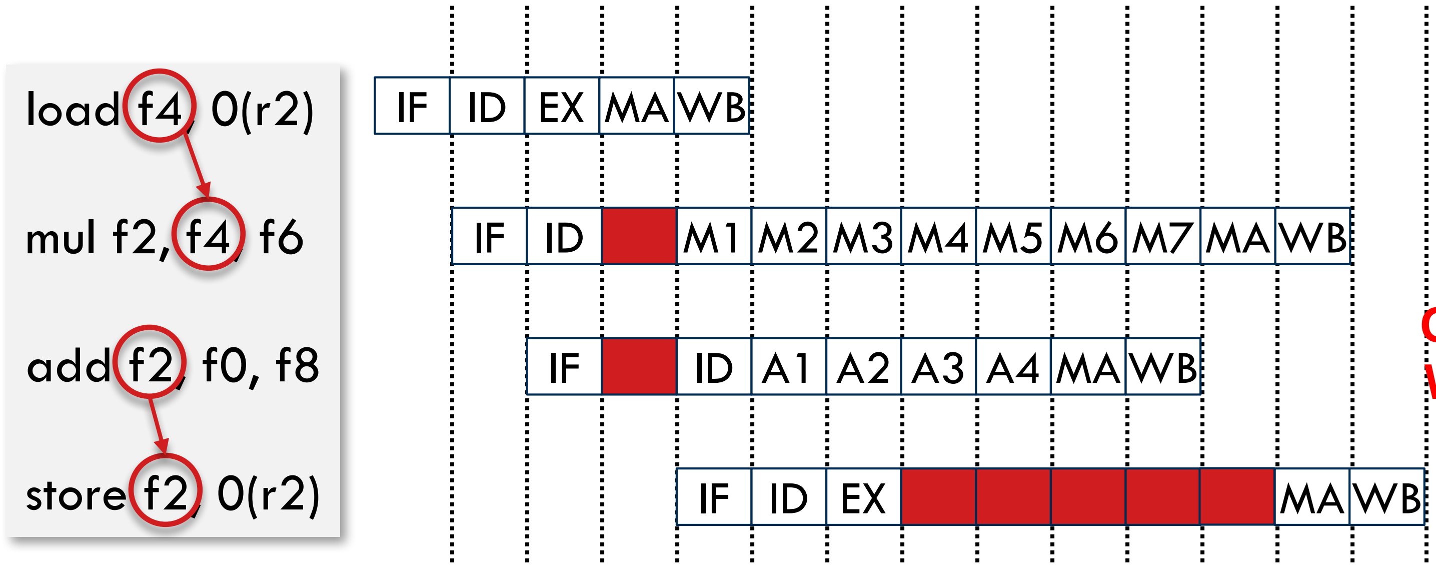
Multicycle Instructions

- Data hazards
  - potential write-after-write hazards



Multicycle Instructions

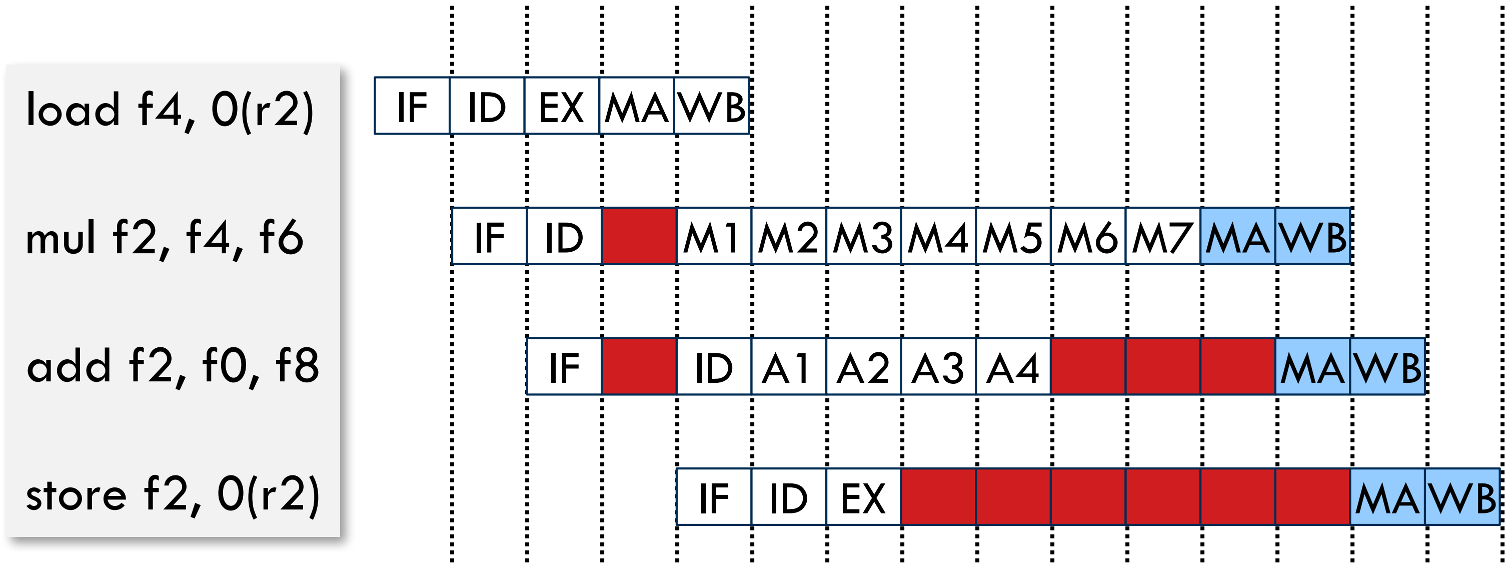
- Data hazards
  - potential write-after-write hazards



Out of Order Write-back!!

Multicycle Instructions

- Data hazards
  - potential write-after-write hazards



In-Order Writes

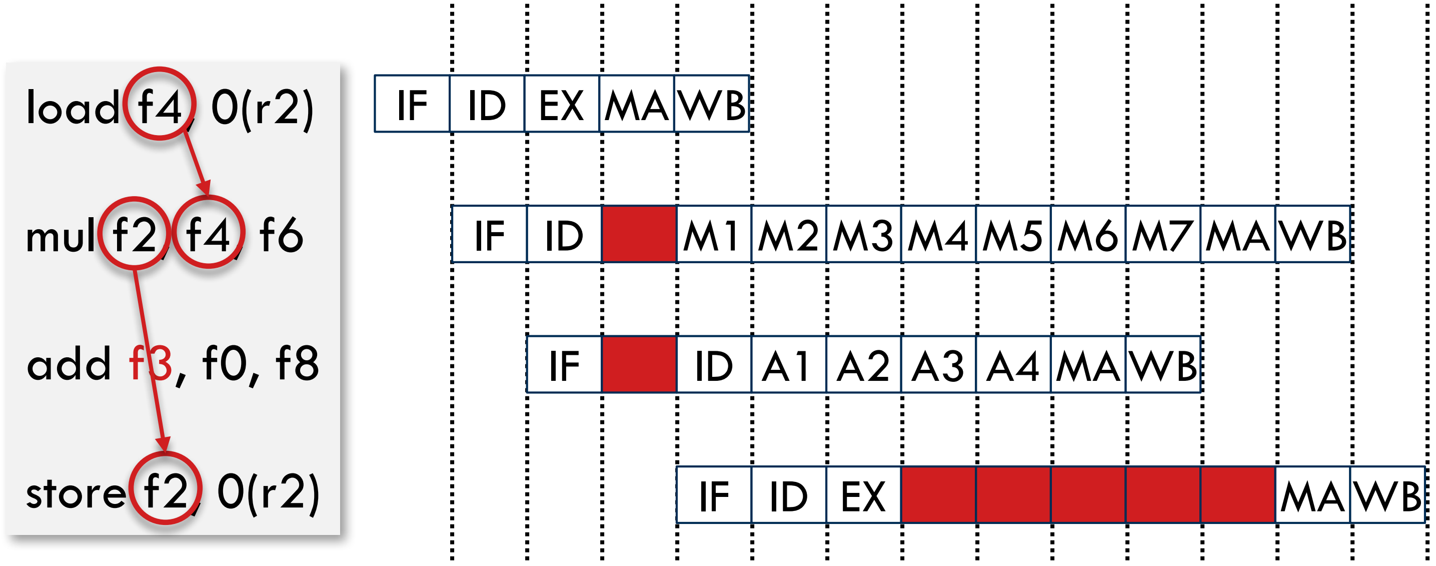
Multicycle Instructions

- 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)
```

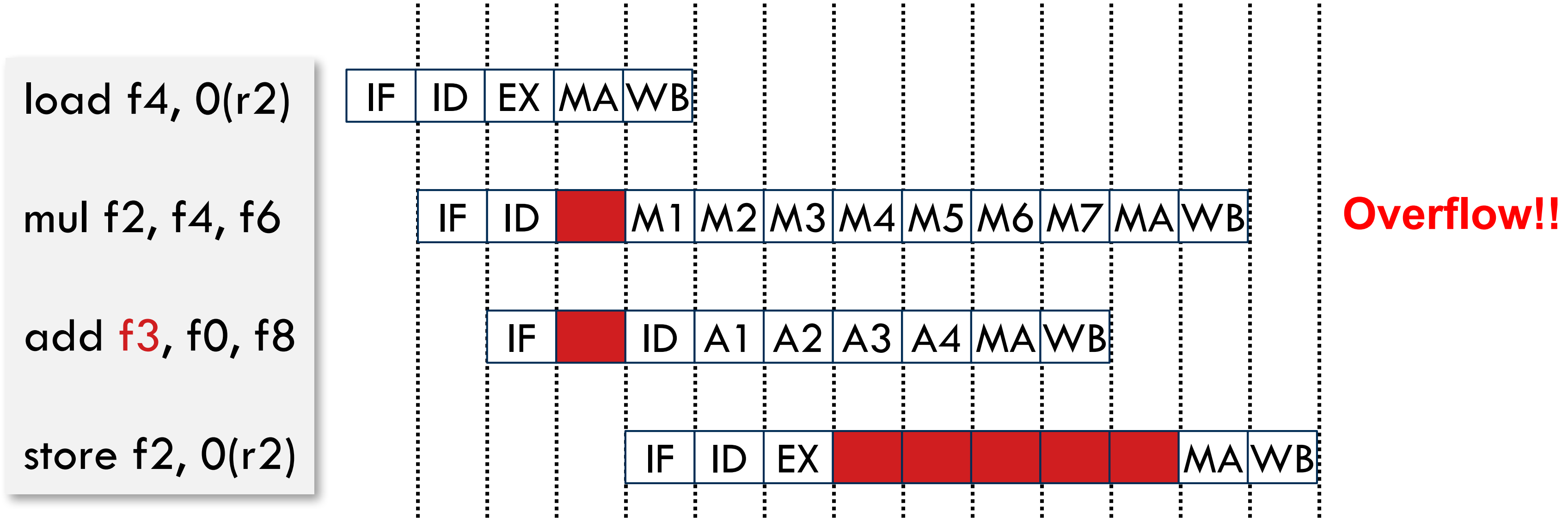
Multicycle Instructions

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



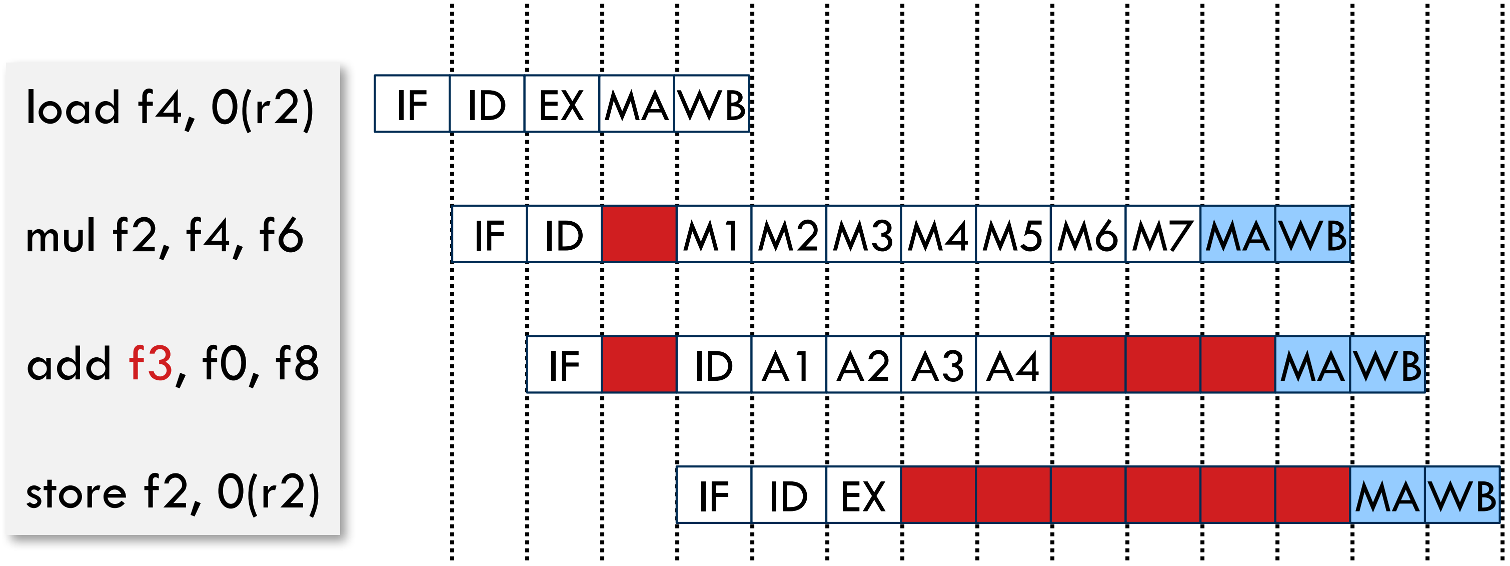
Multicycle Instructions

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



Multicycle Instructions

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

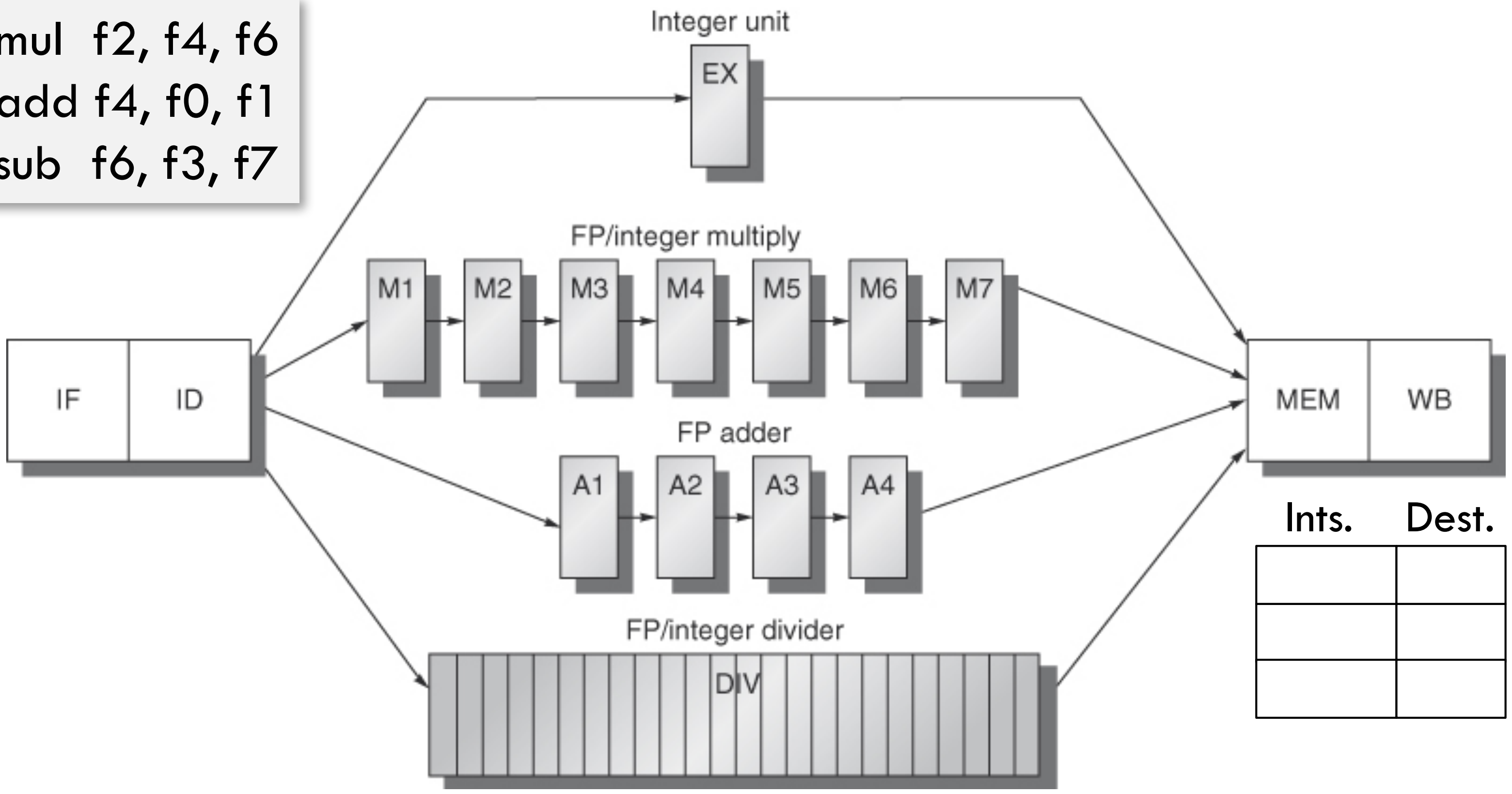


In-order register file updates

Reorder Buffer

- Multicycle Instructions

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

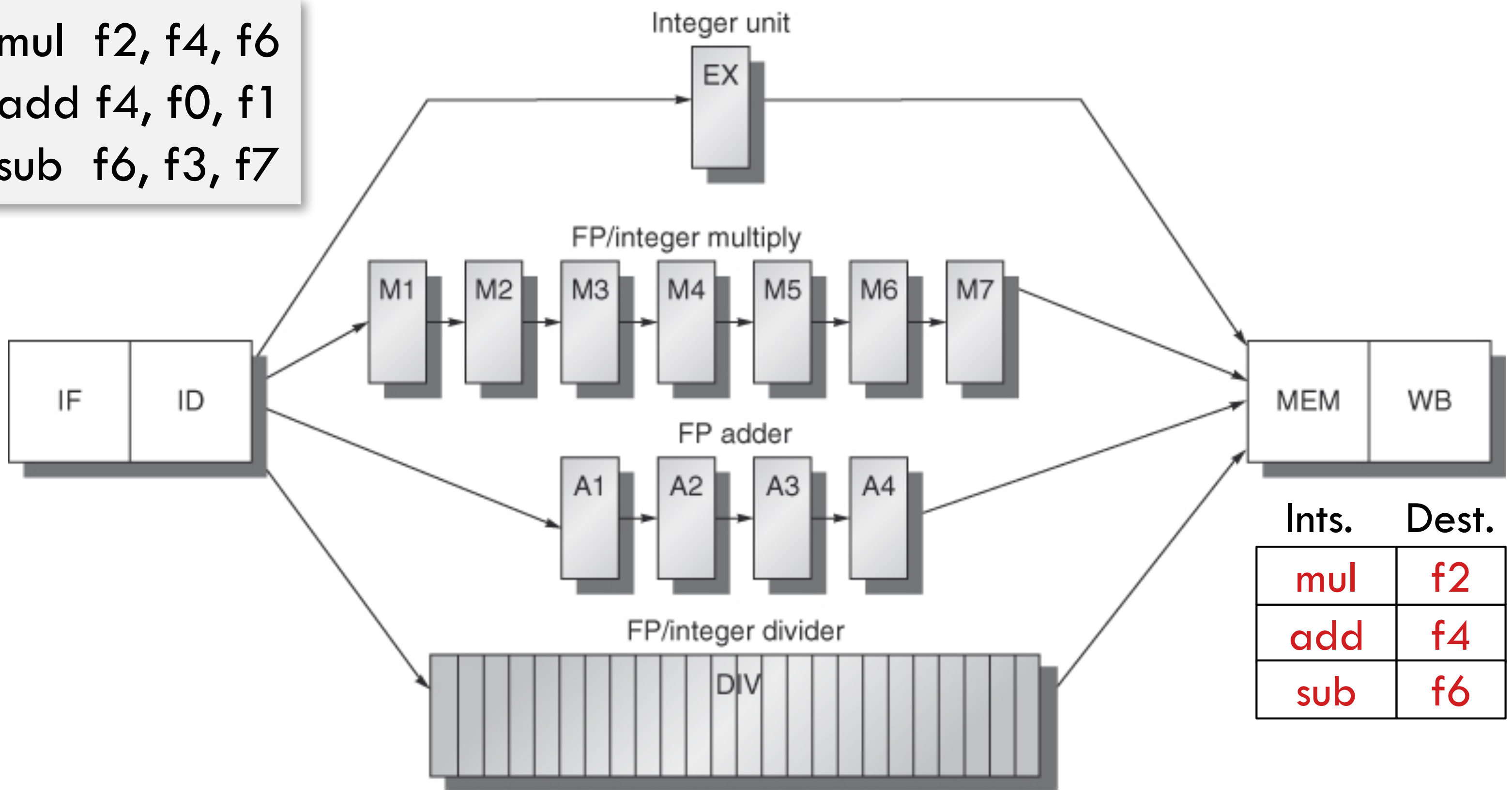


Ints.	Dest.

Reorder Buffer

- Multicycle Instructions

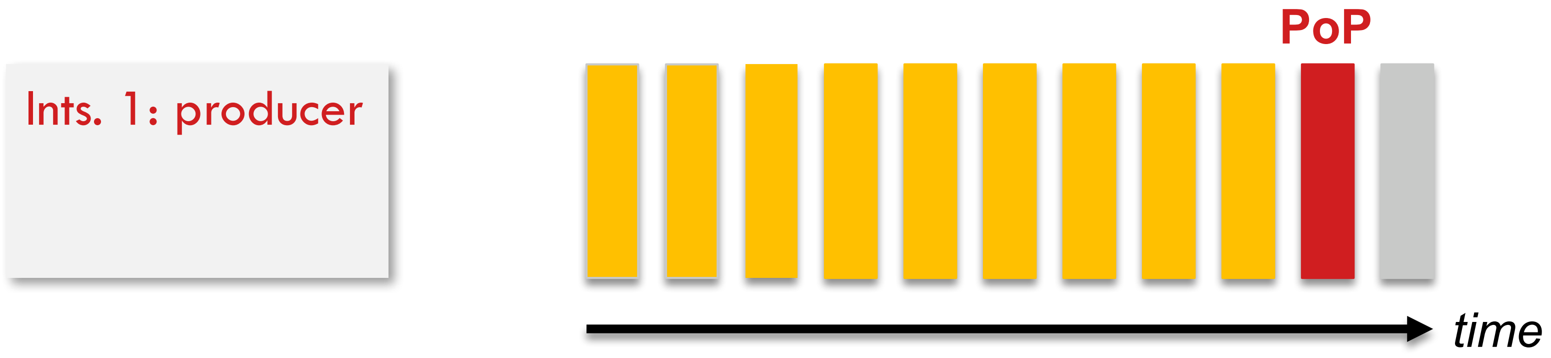
```
mul  f2, f4, f6
add  f4, f0, f1
sub  f6, f3, f7
```



Ints.	Dest.
mul	f2
add	f4
sub	f6

Data Dependence

- Point of production
  - The pipeline stage where an instruction produces a value that can be used by its following instructions

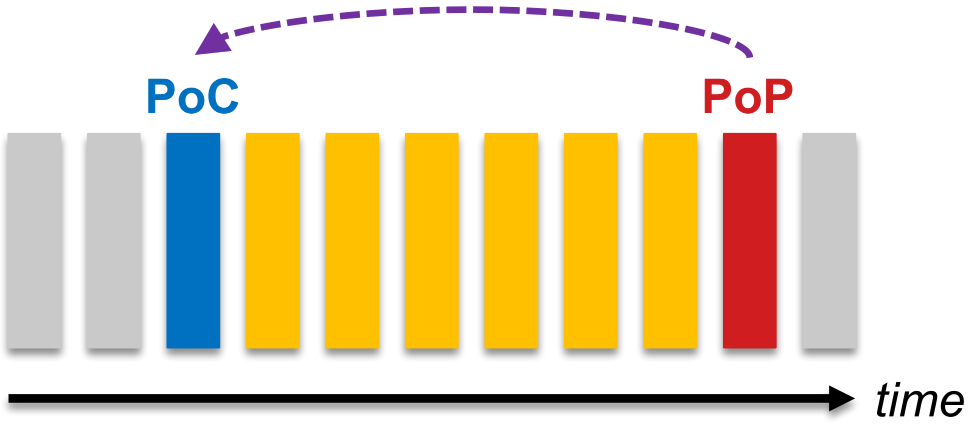


Data Dependence

- Point of production
  - The pipeline stage where an instruction produces a value that can be used by its following instructions
- Point of consumption
  - The pipeline stage where an instruction consumes a produced data

Ints. 1: producer

Inst. 2: consumer

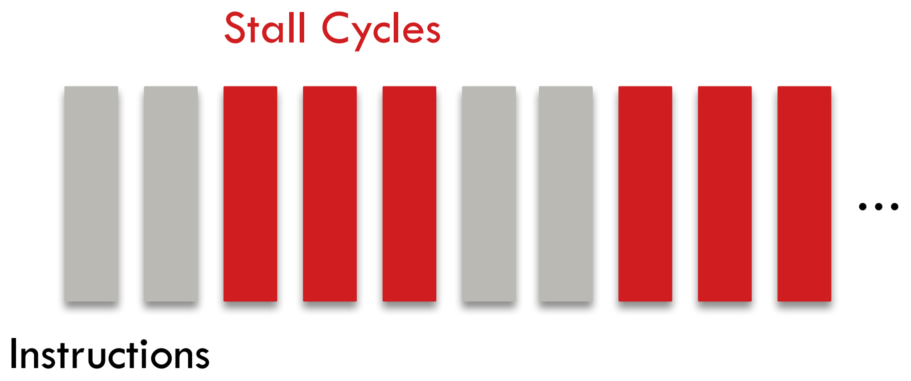


Problem

- Consider a 10-stage pipeline processor, where point of production and point of consumption are separated by 4 cycles. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction. What is the maximum attainable IPC?

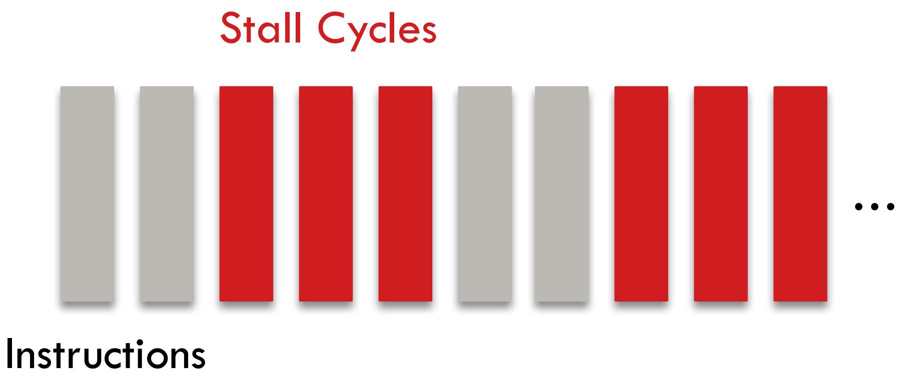
Problem

- Consider a 10-stage pipeline processor, where point of production and point of consumption are separated by 4 cycles. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction. What is the maximum attainable IPC?



Problem

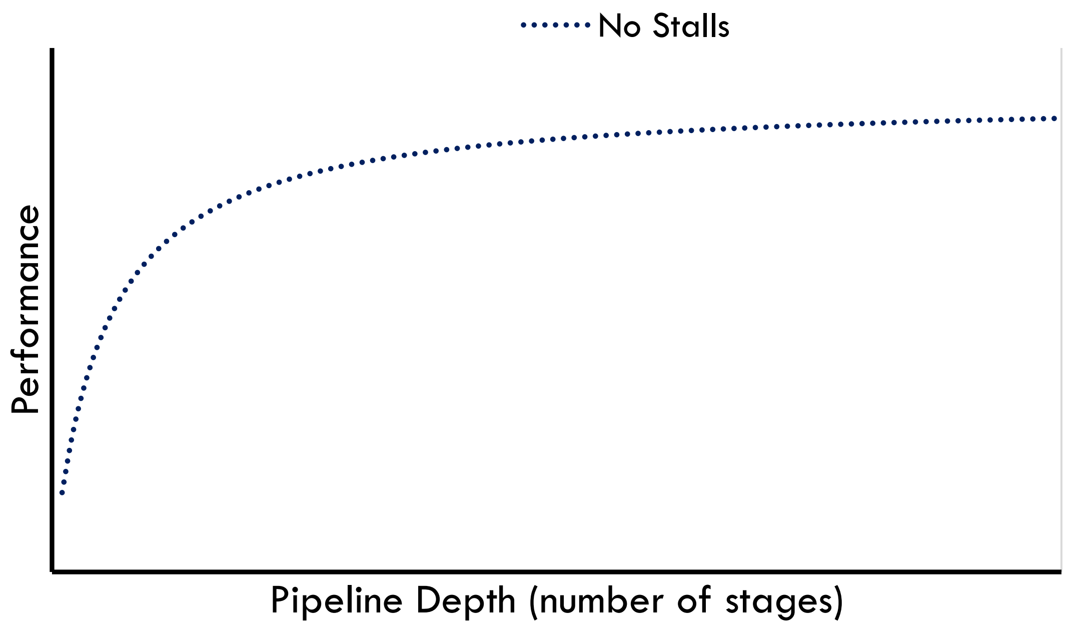
- Consider a 10-stage pipeline processor, where point of production and point of consumption are separated by 4 cycles. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction. What is the maximum attainable IPC?



$$\text{IPC} = \frac{2}{5} = 0.4$$

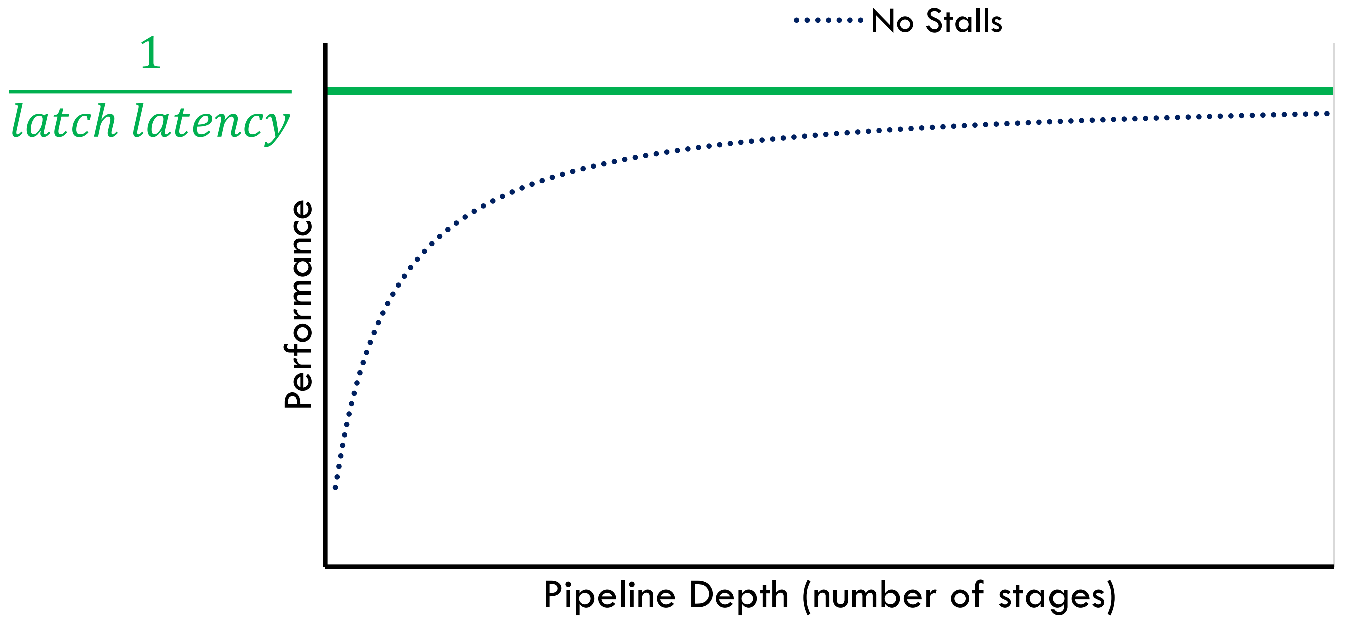
Performance vs. Pipeline Depth

- Impact of stall cycles on performance
  - Independent instructions
  - Dependent instructions



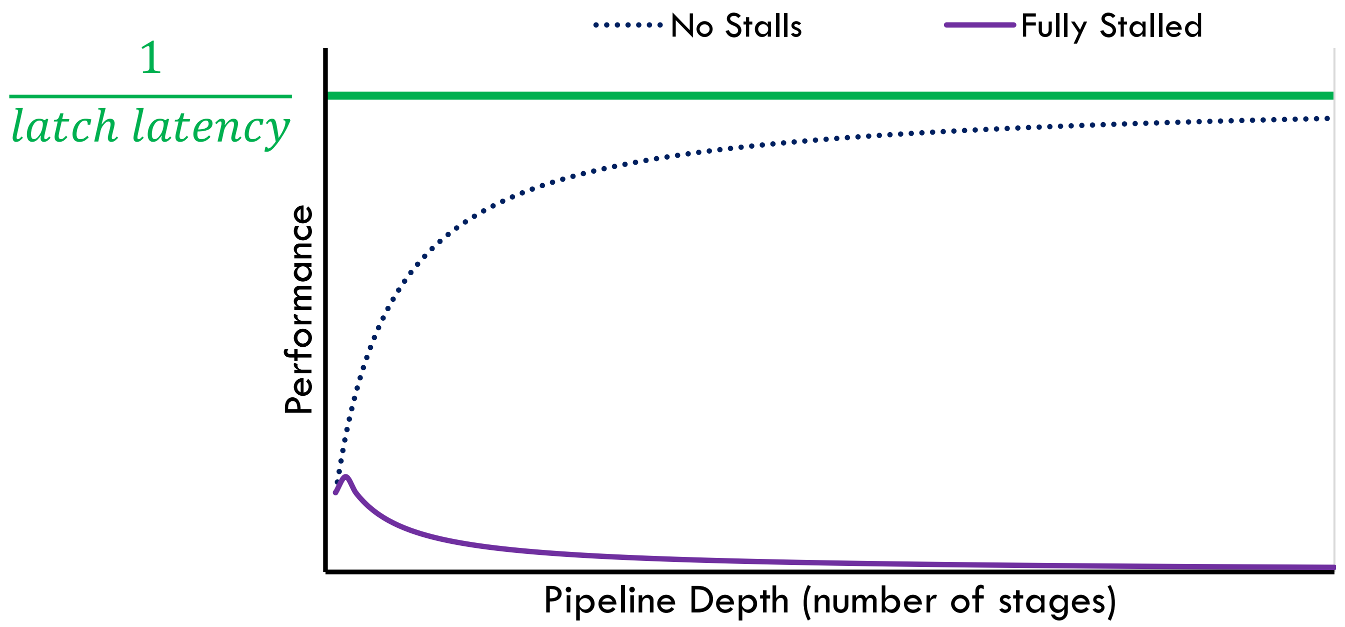
Performance vs. Pipeline Depth

- Impact of stall cycles on performance
  - Independent instructions
  - Dependent instructions



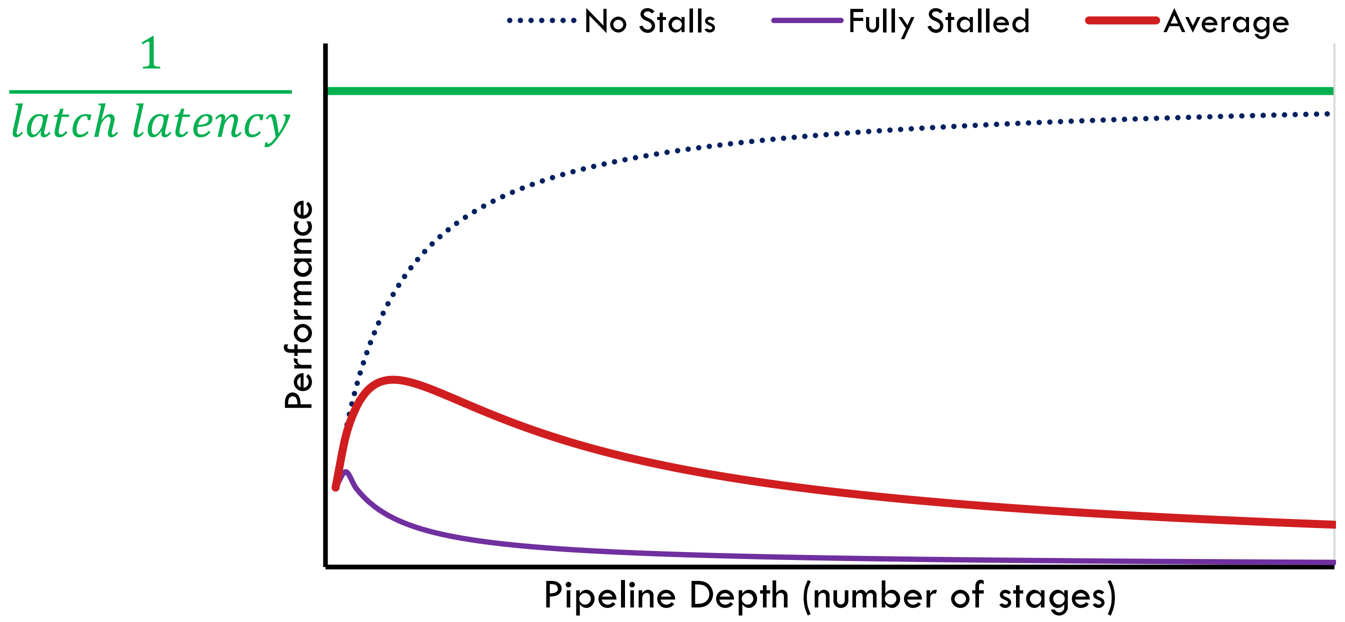
Performance vs. Pipeline Depth

- Impact of stall cycles on performance
  - Independent instructions
  - Dependent instructions



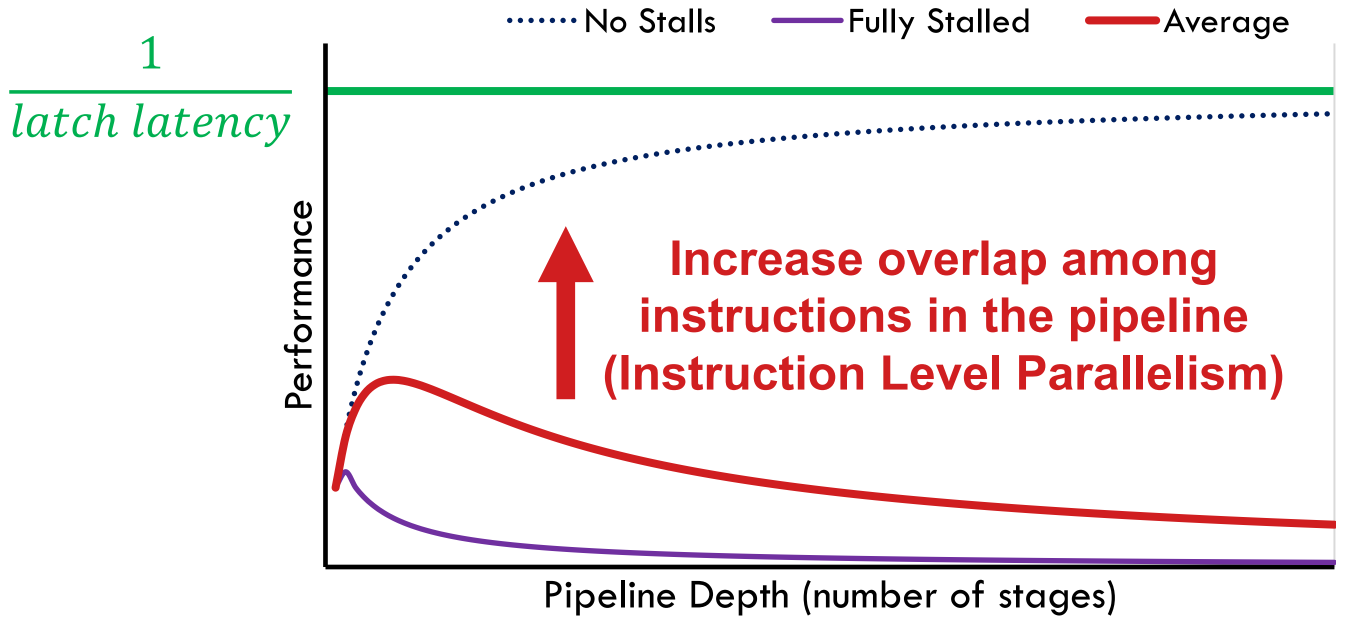
Performance vs. Pipeline Depth

- Impact of stall cycles on performance
  - Independent instructions
  - Dependent instructions



Performance vs. Pipeline Depth

- Impact of stall cycles on performance
  - Independent instructions
  - Dependent instructions



Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow

Code 1

```
ADD R1, R2, R3
SUB R4, R1, R5
XOR R6, R4, R7
AND R8, R6, R9
```

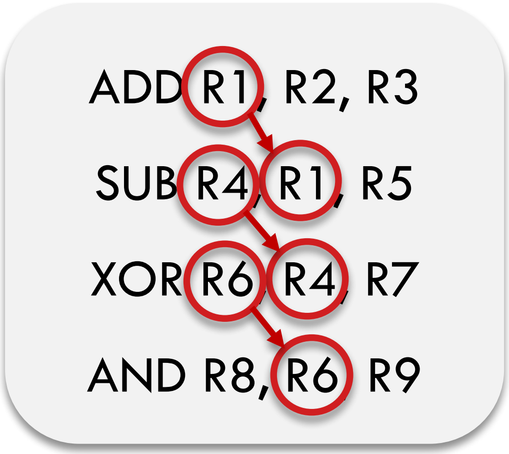
Code 2

```
ADD R1, R2, R3
SUB R4, R6, R5
XOR R8, R2, R7
AND R9, R6, R0
```

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow

Code 1



```
ADD R1, R2, R3
SUB R4, R1, R5
XOR R6, R4, R7
AND R8, R6, R9
```

ILP = 1
Fully serial

Code 2

```
ADD R1, R2, R3
SUB R4, R6, R5
XOR R8, R2, R7
AND R9, R6, R0
```

ILP = 4
Fully parallel

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler

$X \leftarrow A + B + C + D$

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler

$X \leftarrow A + B + C + D$

```
Code 1:
    ADD  R5, R1, R2
    ADD  R5, R5, R3
    ADD  R5, R5, R4
```

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler

$X \leftarrow A + B + C + D$

```
Code 1:
    ADD  R5, R1, R2
    ADD  R5, R5, R3
    ADD  R5, R5, R4
```

```
Code 2:
    ADD  R6, R1, R2
    ADD  R7, R3, R4
    ADD  R5, R6, R7
```

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler

$X \leftarrow A + B + C + D$

```
Code 1:
    ADD  R5, R1, R2
    ADD  R5, R5, R3
    ADD  R5, R5, R4
```

Average ILP = 3/3 = 1
Five registers

```
Code 2:
    ADD  R6, R1, R2
    ADD  R7, R3, R4
    ADD  R5, R6, R7
```

Average ILP = 3/2 = 1.5
Seven registers

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler
- An upper limit for attainable IPC for a given code
  - IPC represents exploited ILP

```
ADD  R5, R1, R2
ADD  R5, R5, R3
ADD  R5, R5, R4
```

Average ILP = 3/3 = 1
Five registers

```
ADD  R6, R1, R2
ADD  R7, R3, R4
ADD  R5, R6, R7
```

Average ILP = 3/2 = 1.5
Seven registers

Instruction Level Parallelism

- Potential overlap among instructions
  - A property of the program dataflow
  - Influenced by compiler
- An upper limit for attainable IPC for a given code
  - IPC represents exploited ILP
- Can be exploited by HW-/SW-intensive techniques
  - Dynamic scheduling in hardware
  - Static scheduling in software (compiler)