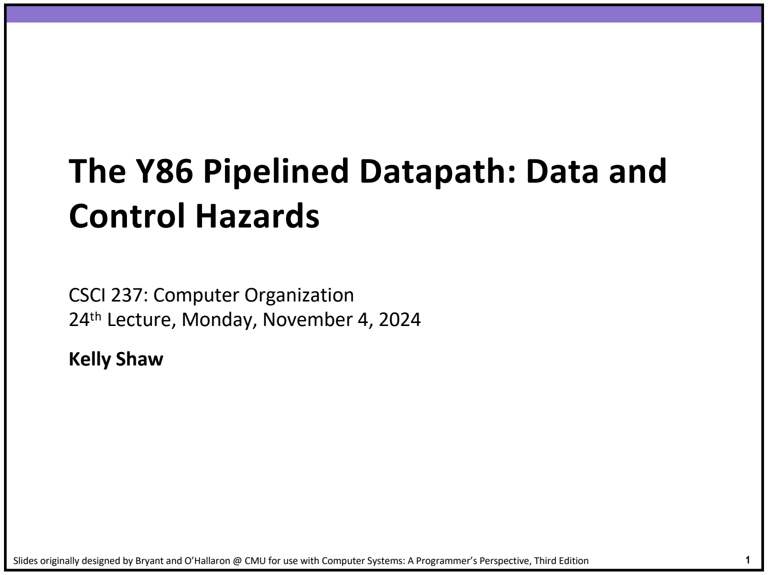


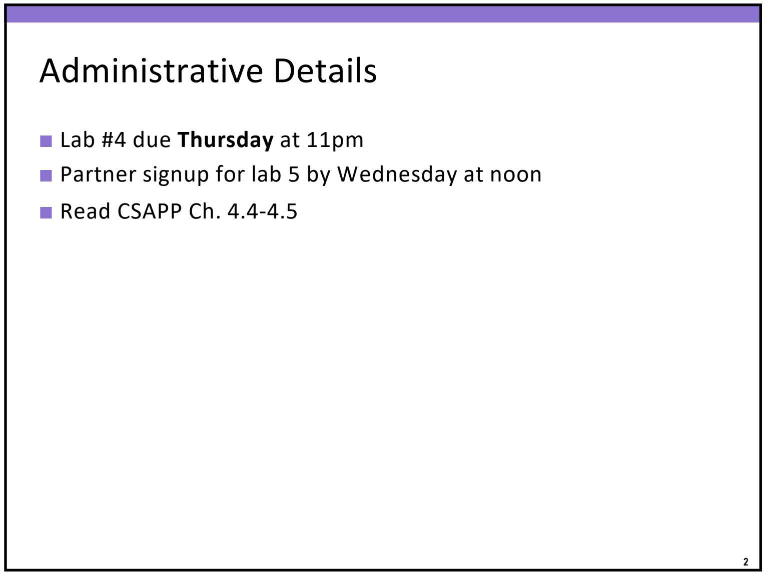
The Y86 Pipelined Datapath: Data and Control Hazards

CSCI 237: Computer Organization
24th Lecture, Monday, November 4, 2024

Kelly Shaw

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

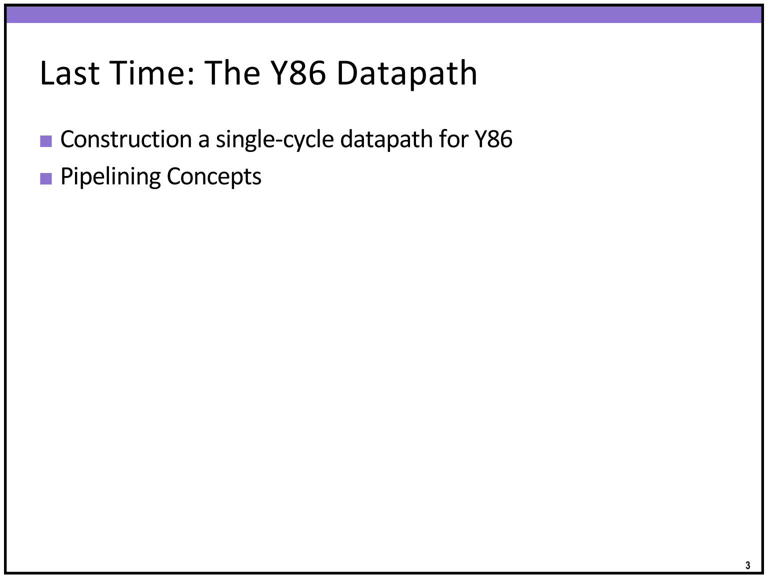
1



Administrative Details

- Lab #4 due **Thursday** at 11pm
- Partner signup for lab 5 by Wednesday at noon
- Read CSAPP Ch. 4.4-4.5

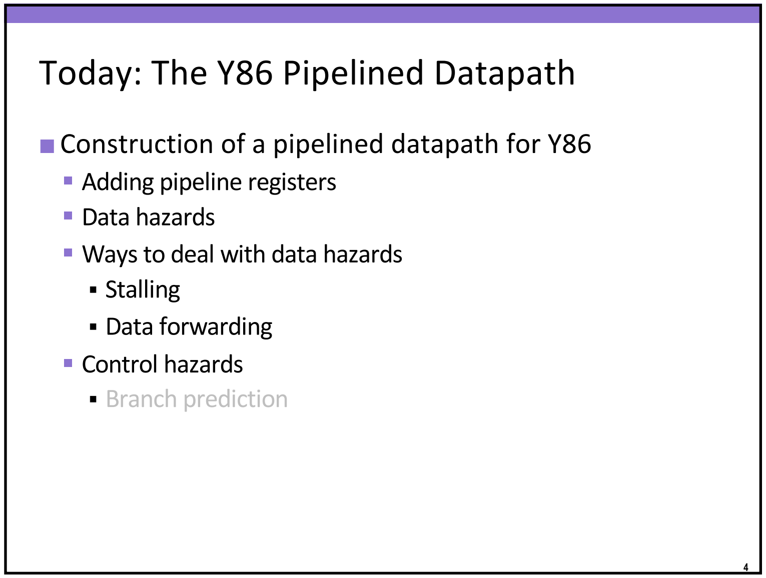
2



Last Time: The Y86 Datapath

- Construction a single-cycle datapath for Y86
- Pipelining Concepts

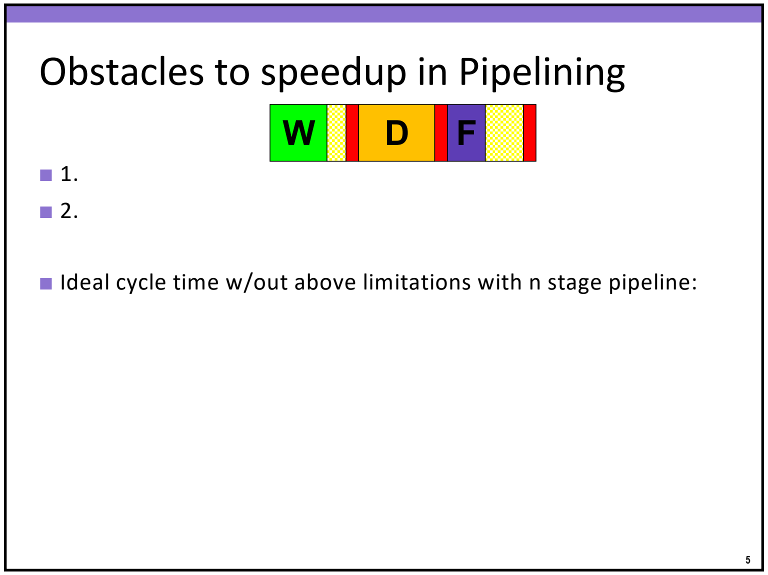
3



Today: 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

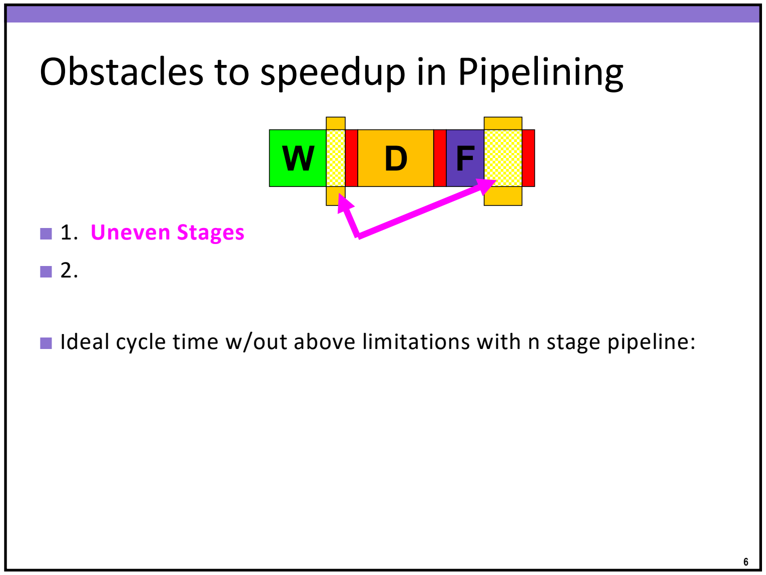
4



Obstacles to speedup in Pipelining

- 1.
- 2.

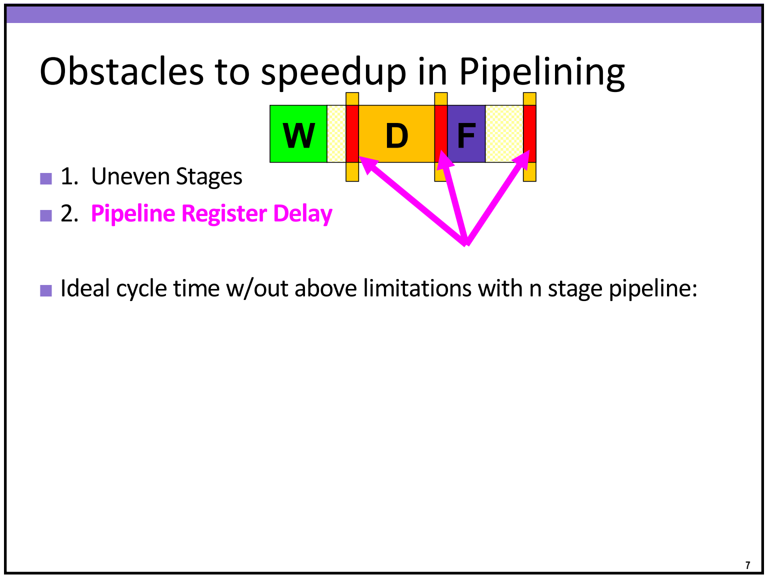
- Ideal cycle time w/out above limitations with n stage pipeline:



Obstacles to speedup in Pipelining

- 1. **Uneven Stages**
- 2.

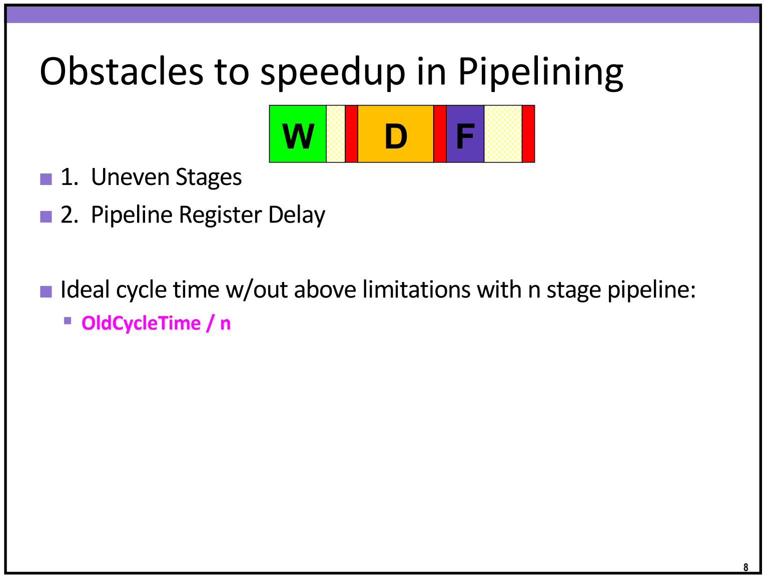
- Ideal cycle time w/out above limitations with n stage pipeline:



Obstacles to speedup in Pipelining

- 1. Uneven Stages
- 2. **Pipeline Register Delay**

- Ideal cycle time w/out above limitations with n stage pipeline:



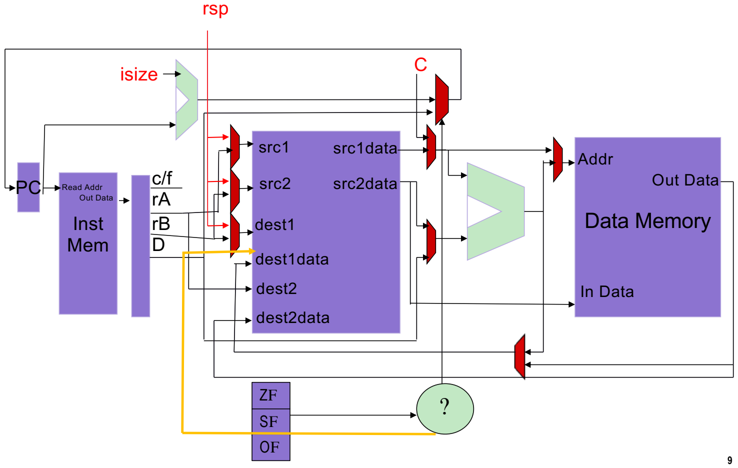
Obstacles to speedup in Pipelining

- 1. Uneven Stages
- 2. Pipeline Register Delay

- Ideal cycle time w/out above limitations with n stage pipeline:
  - **OldCycleTime / n**

Seq Datapath

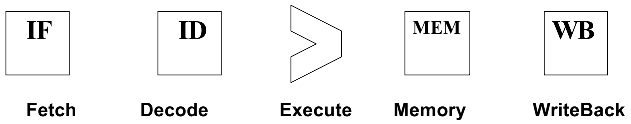
Entire instruction takes 1 cycle



Our goals for a better processor design:

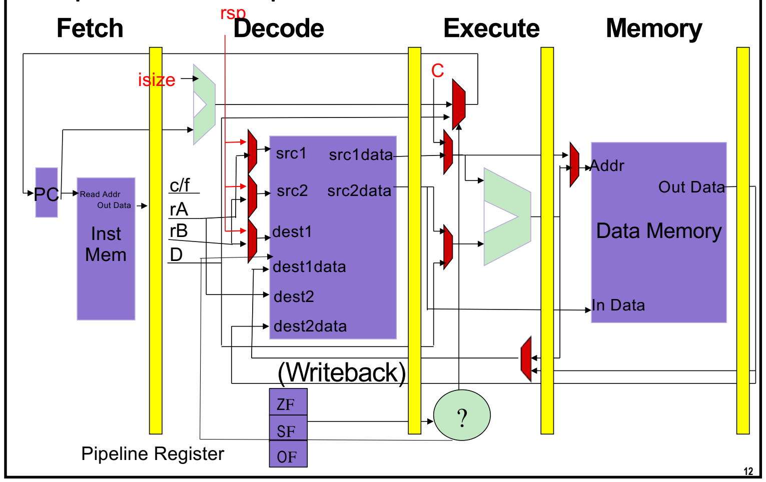
- Faster clock rate
- Use machine more efficiently
- No longer execute only one instruction at a time

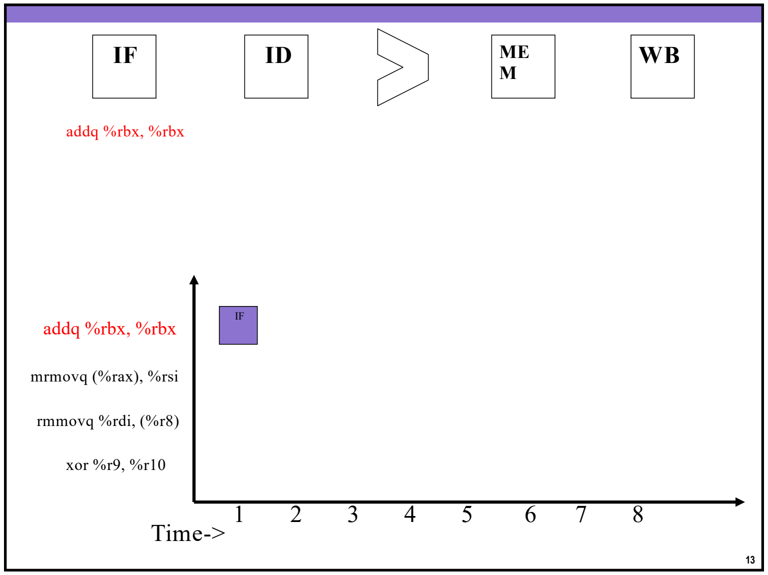
Creating Stages to transform Seq Design



- Fetch – get instruction
- Decode – read registers
- Execute – use ALU
- Memory – access memory
- WriteBack – write registers

Pipelined Datapath





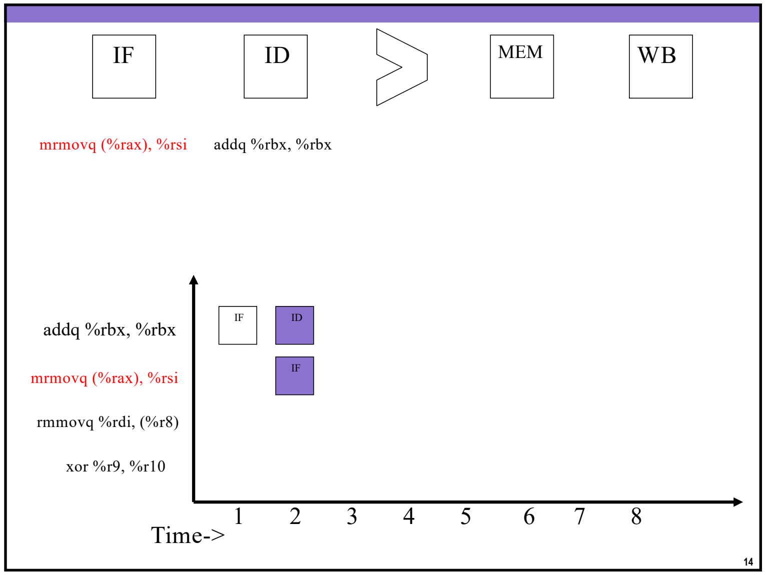
IF ID > ME M WB

addq %rbx, %rbx

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF							
mrmovq (%rax), %rsi								
rmmovq %rdi, (%r8)								
xor %r9, %r10								

Time->

13



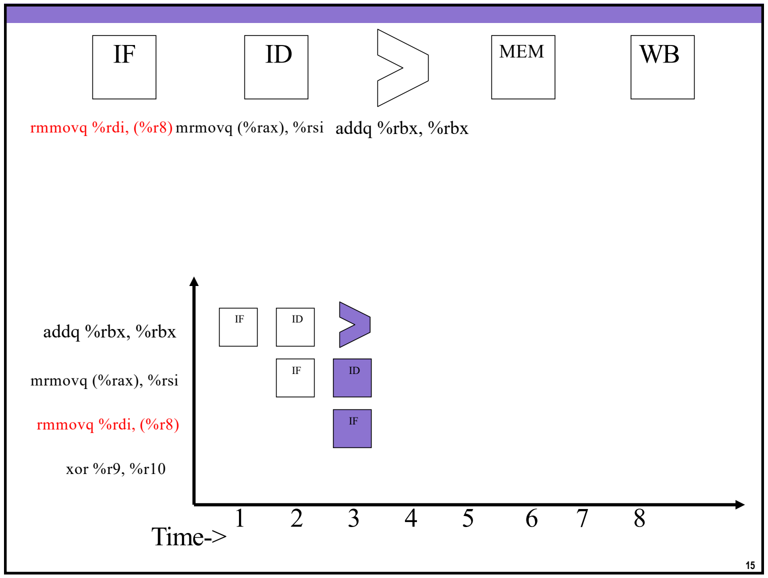
IF ID > MEM WB

mrmovq (%rax), %rsi addq %rbx, %rbx

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID						
mrmovq (%rax), %rsi		IF						
rmmovq %rdi, (%r8)								
xor %r9, %r10								

Time->

14



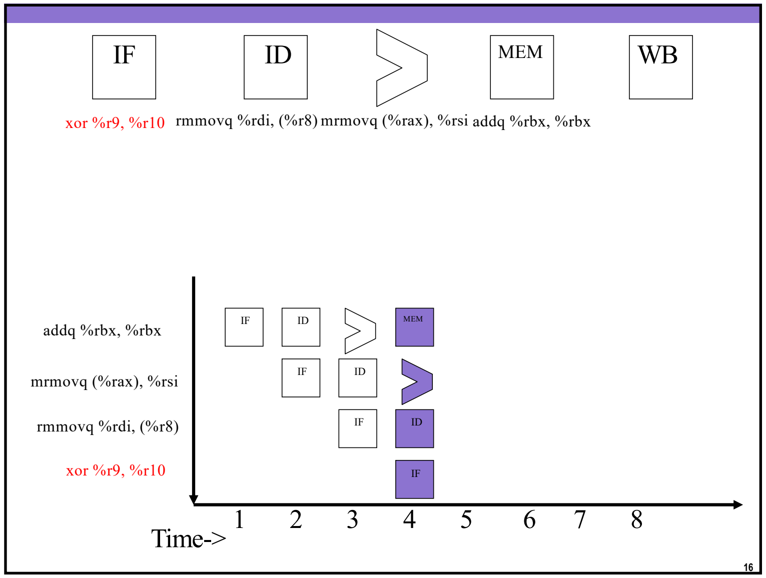
IF ID > MEM WB

rmmovq %rdi, (%r8) mrmovq (%rax), %rsi addq %rbx, %rbx

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>					
mrmovq (%rax), %rsi		IF	ID					
rmmovq %rdi, (%r8)			IF					
xor %r9, %r10								

Time->

15



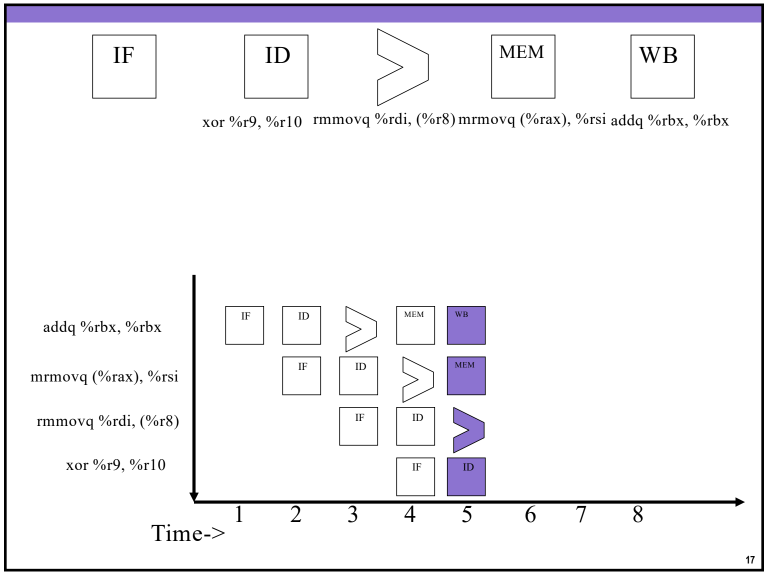
IF ID > MEM WB

xor %r9, %r10 rmmovq %rdi, (%r8) mrmovq (%rax), %rsi addq %rbx, %rbx

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>	MEM				
mrmovq (%rax), %rsi		IF	ID	>				
rmmovq %rdi, (%r8)			IF	ID				
xor %r9, %r10				IF				

Time->

16



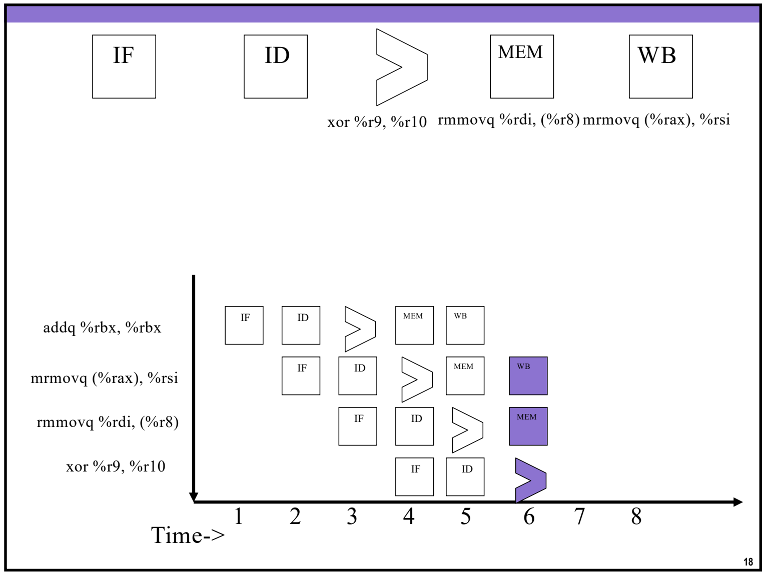
IF	ID	>	MEM	WB
	xor %r9, %r10	rmmovq %rdi, (%r8)	mrmovq (%rax), %rsi	addq %rbx, %rbx

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>	MEM	WB			
mrmovq (%rax), %rsi		IF	ID	>	MEM			
rmmovq %rdi, (%r8)			IF	ID	>			
xor %r9, %r10				IF	ID			

Time->

17

17



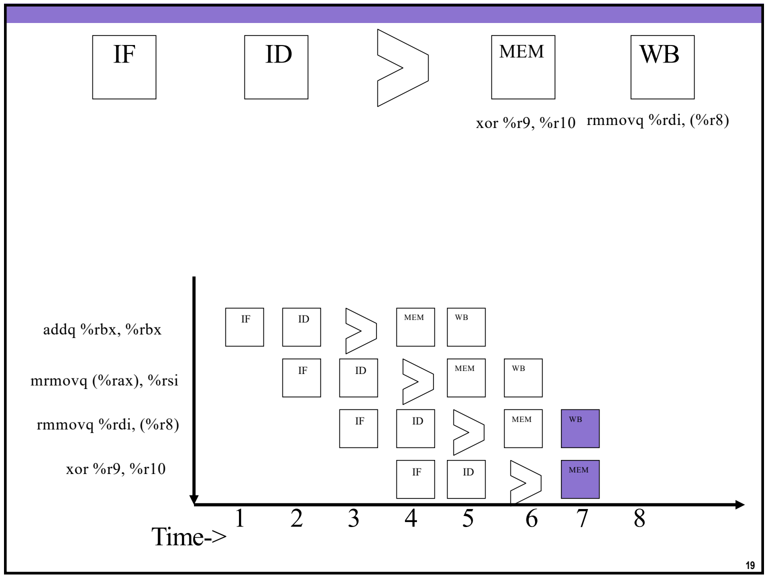
IF	ID	>	MEM	WB
		xor %r9, %r10	rmmovq %rdi, (%r8)	mrmovq (%rax), %rsi

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>	MEM	WB			
mrmovq (%rax), %rsi		IF	ID	>	MEM	WB		
rmmovq %rdi, (%r8)			IF	ID	>	MEM		
xor %r9, %r10				IF	ID	>		

Time->

18

18



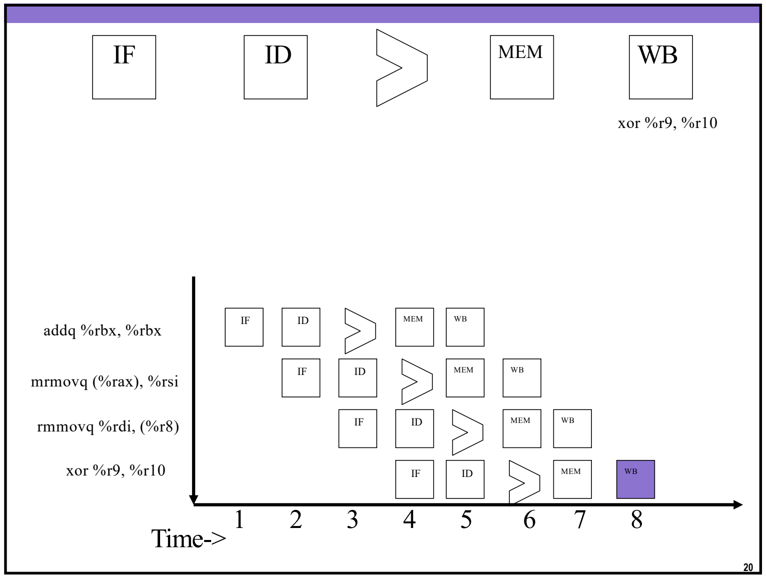
IF	ID	>	MEM	WB
			xor %r9, %r10	rmmovq %rdi, (%r8)

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>	MEM	WB			
mrmovq (%rax), %rsi		IF	ID	>	MEM	WB		
rmmovq %rdi, (%r8)			IF	ID	>	MEM	WB	
xor %r9, %r10				IF	ID	>	MEM	

Time->

19

19



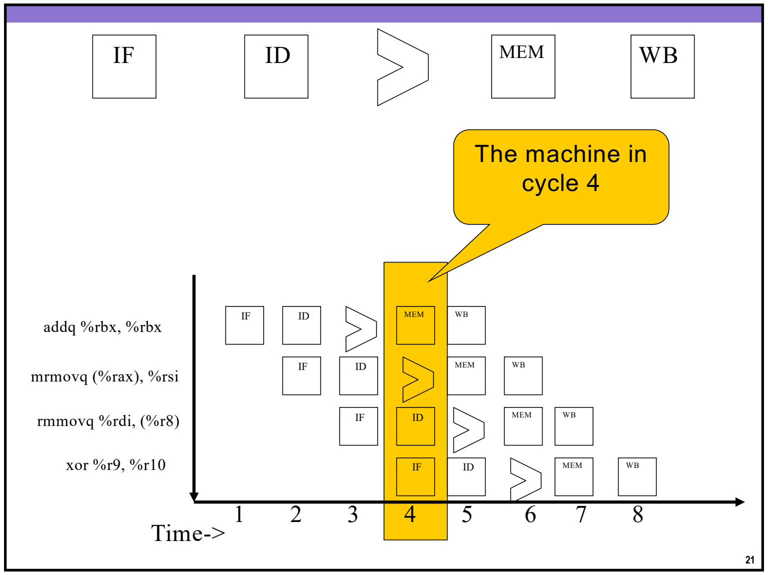
IF	ID	>	MEM	WB
				xor %r9, %r10

	1	2	3	4	5	6	7	8
addq %rbx, %rbx	IF	ID	>	MEM	WB			
mrmovq (%rax), %rsi		IF	ID	>	MEM	WB		
rmmovq %rdi, (%r8)			IF	ID	>	MEM	WB	
xor %r9, %r10				IF	ID	>	MEM	WB

Time->

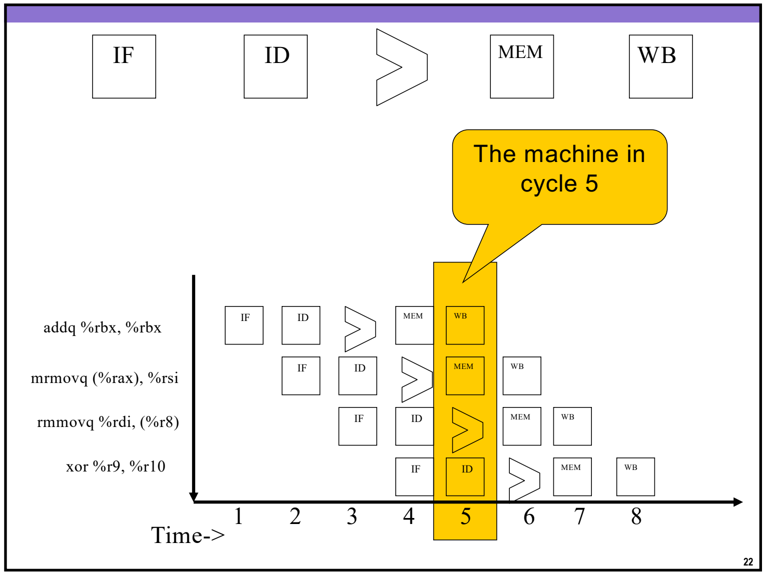
20

20



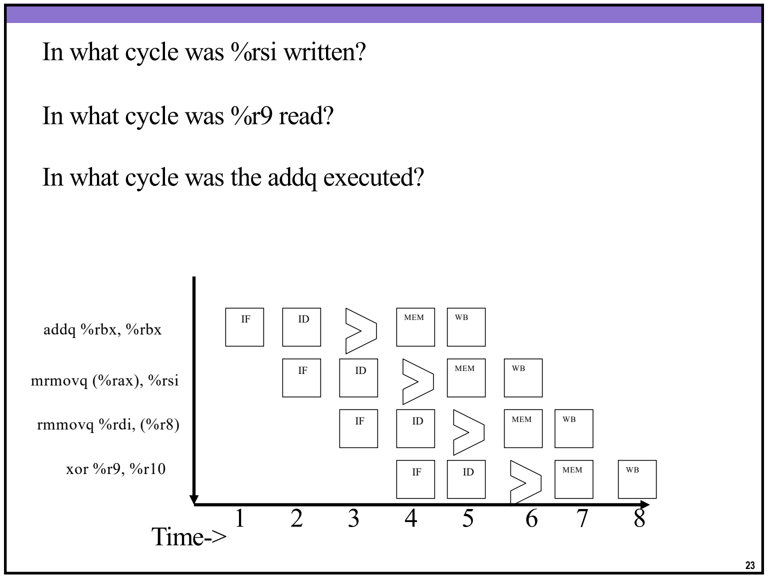
The machine in cycle 4

21



The machine in cycle 5

22

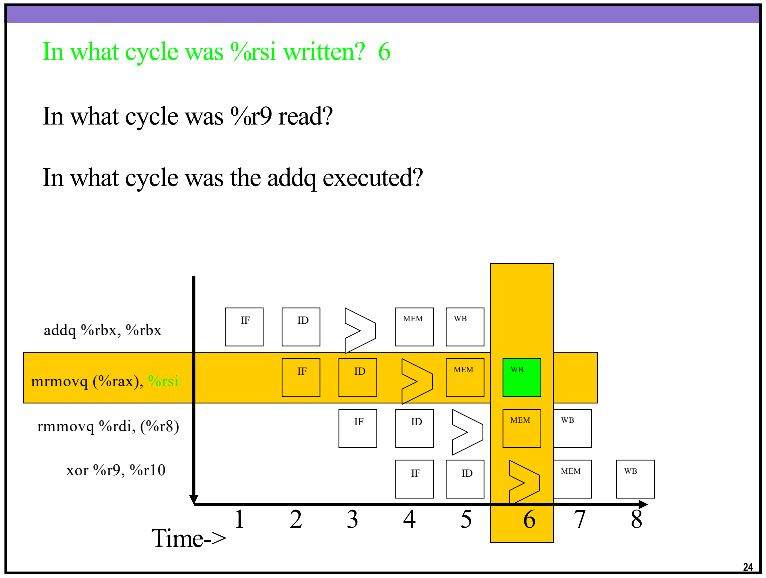


In what cycle was %rsi written?

In what cycle was %r9 read?

In what cycle was the addq executed?

23

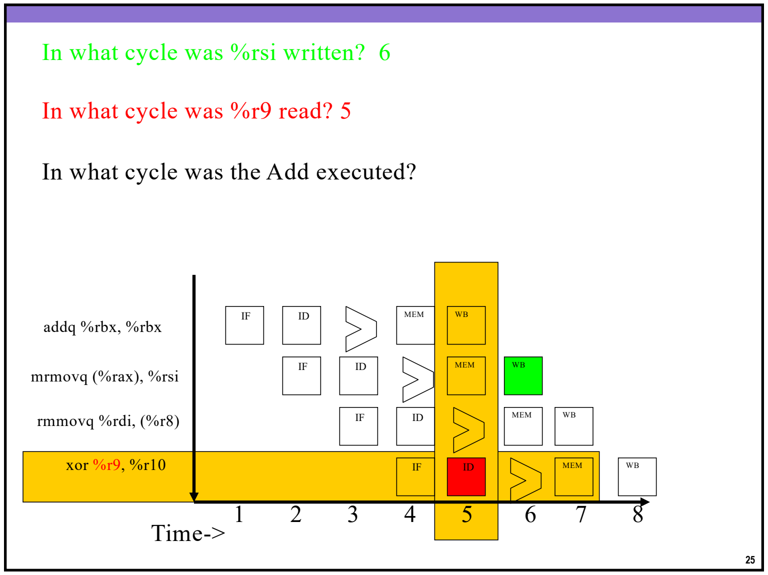


In what cycle was %rsi written? 6

In what cycle was %r9 read?

In what cycle was the addq executed?

24

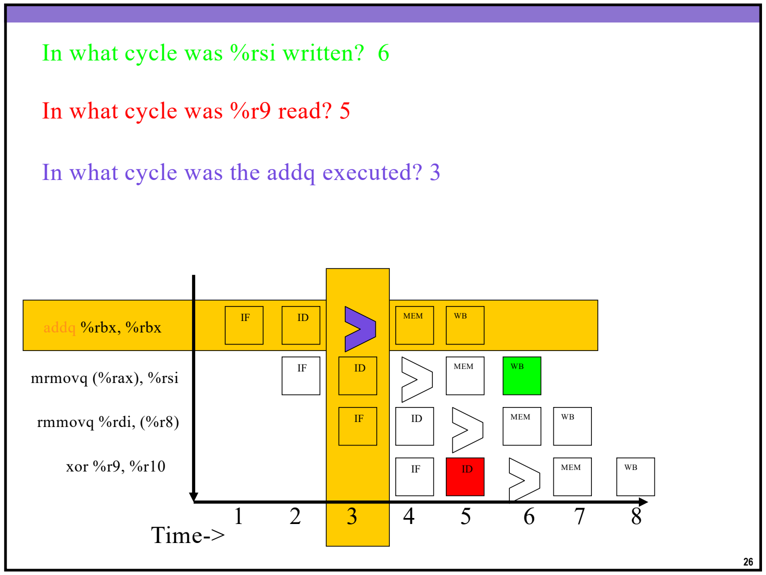


In what cycle was %rsi written? 6

In what cycle was %r9 read? 5

In what cycle was the Add executed?

25

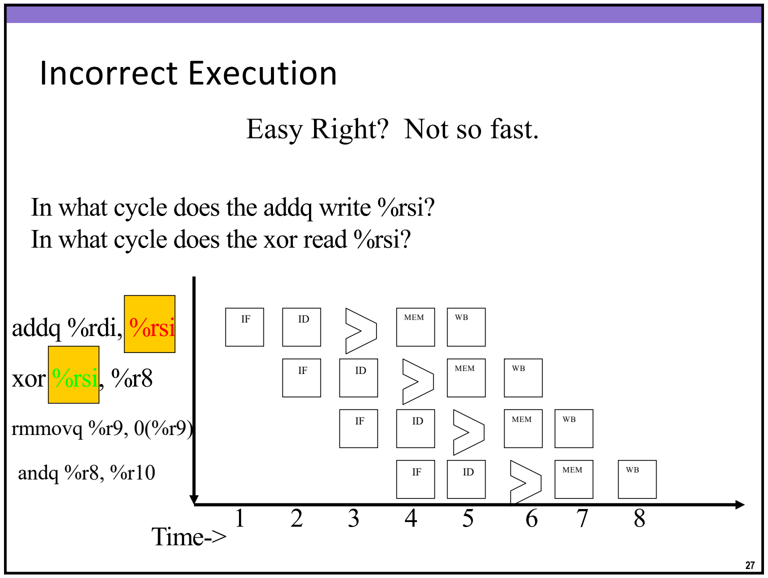


In what cycle was %rsi written? 6

In what cycle was %r9 read? 5

In what cycle was the addq executed? 3

26

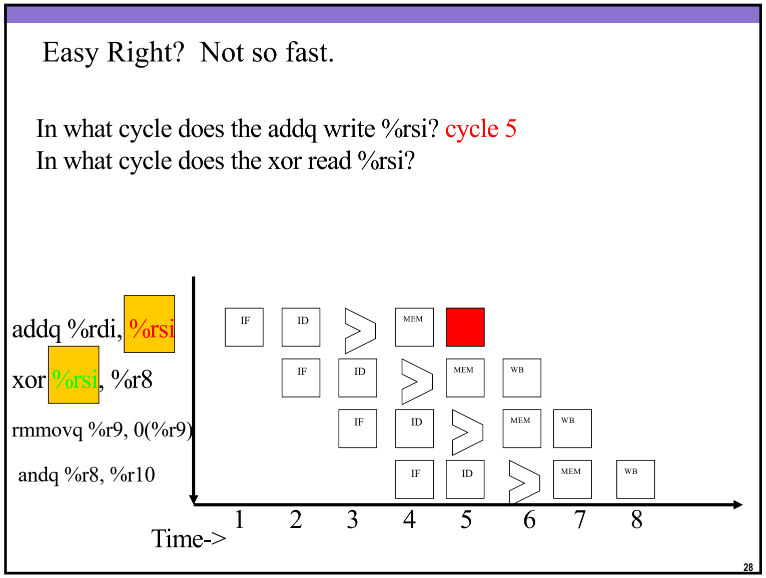


Incorrect Execution

Easy Right? Not so fast.

In what cycle does the addq write %rsi?
In what cycle does the xor read %rsi?

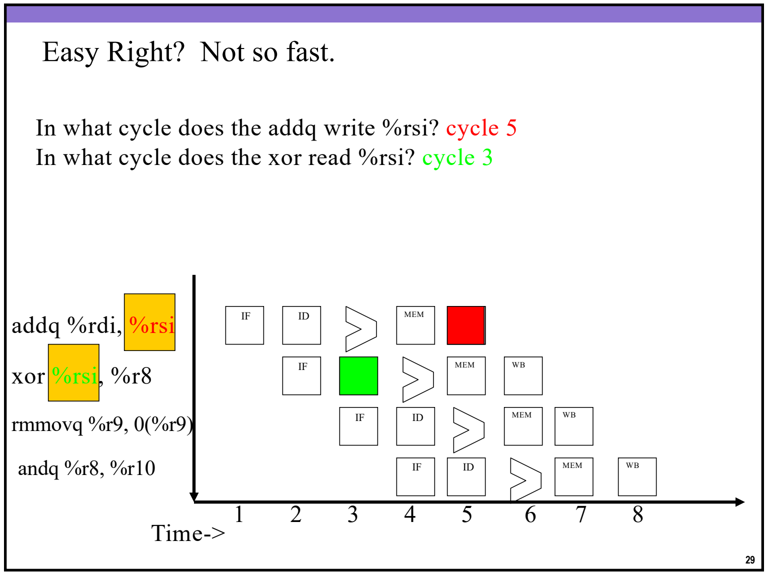
27



Easy Right? Not so fast.

In what cycle does the addq write %rsi? cycle 5
In what cycle does the xor read %rsi?

28



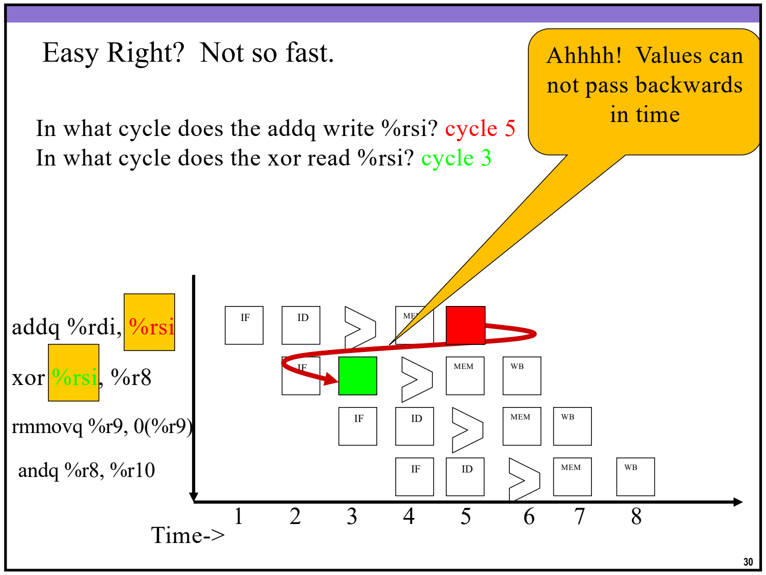
Easy Right? Not so fast.

In what cycle does the addq write %rsi? cycle 5
In what cycle does the xor read %rsi? cycle 3

	1	2	3	4	5	6	7	8
addq %rdi, %rsi	IF	ID	>	MEM	🟥			
xor %rsi, %r8		IF	🟩	>	MEM	WB		
rmmovq %r9, 0(%r9)			IF	ID	>	MEM	WB	
andq %r8, %r10				IF	ID	>	MEM	WB

Time->

29



Easy Right? Not so fast.

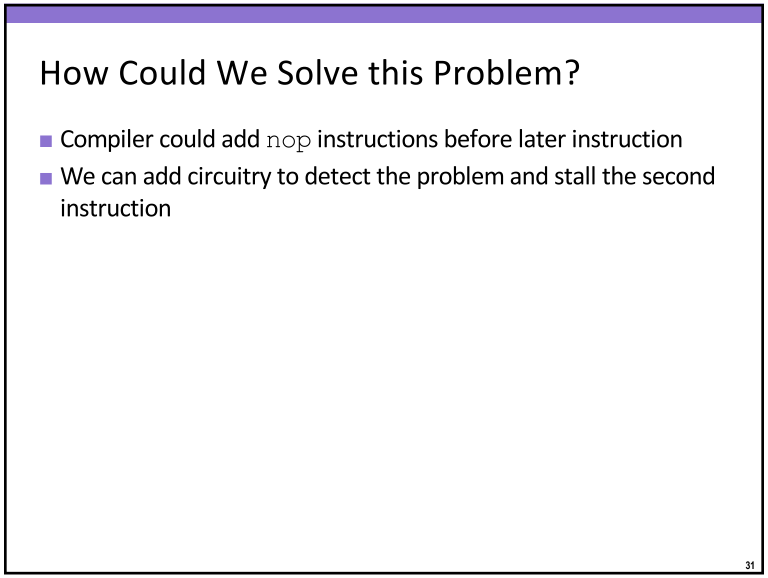
Ahhhh! Values can not pass backwards in time

In what cycle does the addq write %rsi? cycle 5
In what cycle does the xor read %rsi? cycle 3

	1	2	3	4	5	6	7	8
addq %rdi, %rsi	IF	ID	>	MEM	🟥			
xor %rsi, %r8		IF	🟩	>	MEM	WB		
rmmovq %r9, 0(%r9)			IF	ID	>	MEM	WB	
andq %r8, %r10				IF	ID	>	MEM	WB

Time->

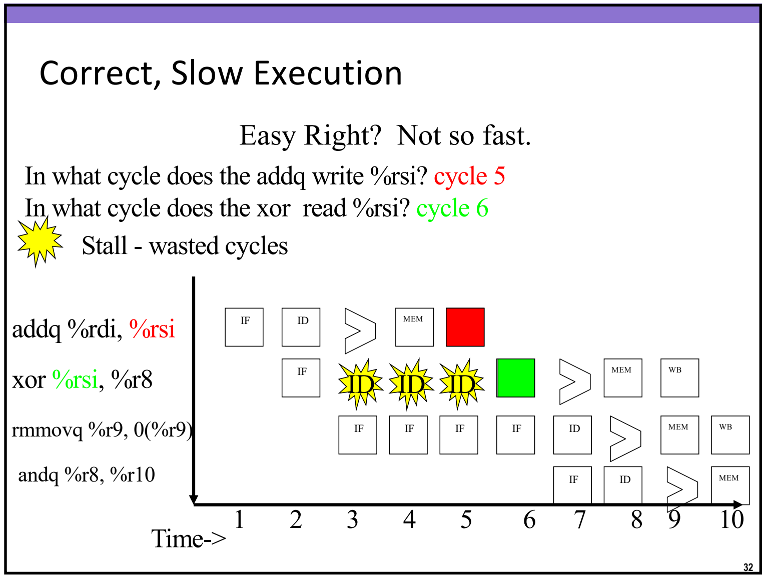
30



How Could We Solve this Problem?

- Compiler could add `nop` instructions before later instruction
- We can add circuitry to detect the problem and stall the second instruction

31



Correct, Slow Execution

Easy Right? Not so fast.

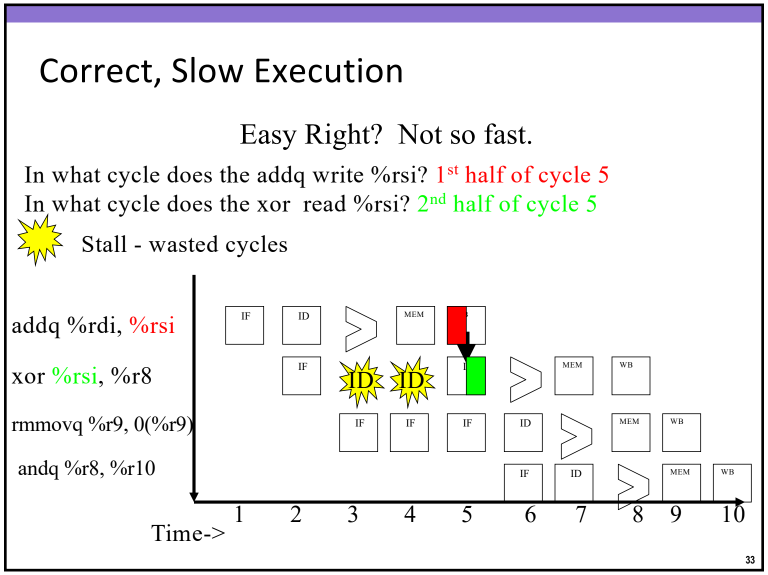
In what cycle does the addq write %rsi? cycle 5
In what cycle does the xor read %rsi? cycle 6

Stall - wasted cycles

	1	2	3	4	5	6	7	8	9	10
addq %rdi, %rsi	IF	ID	>	MEM	🟥					
xor %rsi, %r8		IF	ID	ID	ID	🟩	>	MEM	WB	
rmmovq %r9, 0(%r9)			IF	IF	IF	IF	ID	>	MEM	WB
andq %r8, %r10							IF	ID	>	MEM

Time->

32



Correct, Slow Execution

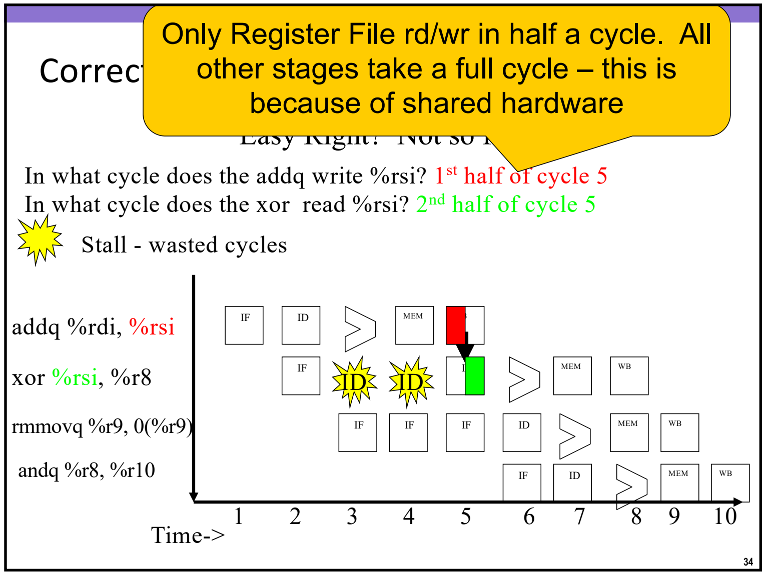
Easy Right? Not so fast.

In what cycle does the addq write %rsi? 1st half of cycle 5

In what cycle does the xor read %rsi? 2nd half of cycle 5

Stall - wasted cycles

33



Correct, Slow Execution

Only Register File rd/wr in half a cycle. All other stages take a full cycle – this is because of shared hardware

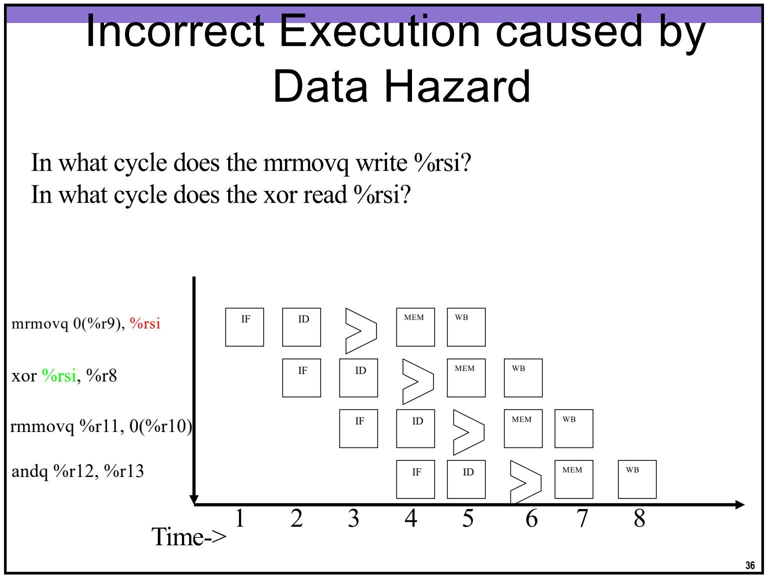
Easy Right? Not so fast.

In what cycle does the addq write %rsi? 1st half of cycle 5

In what cycle does the xor read %rsi? 2nd half of cycle 5

Stall - wasted cycles

34

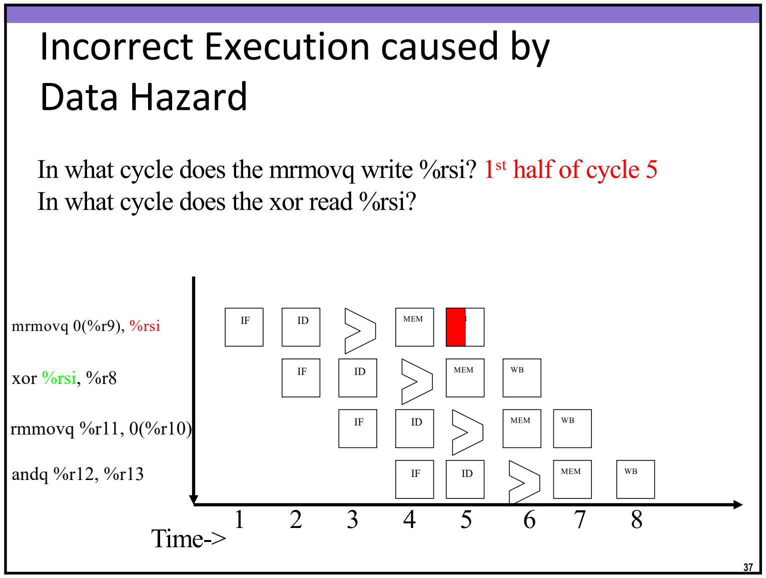


Incorrect Execution caused by Data Hazard

In what cycle does the mrmovq write %rsi?

In what cycle does the xor read %rsi?

36

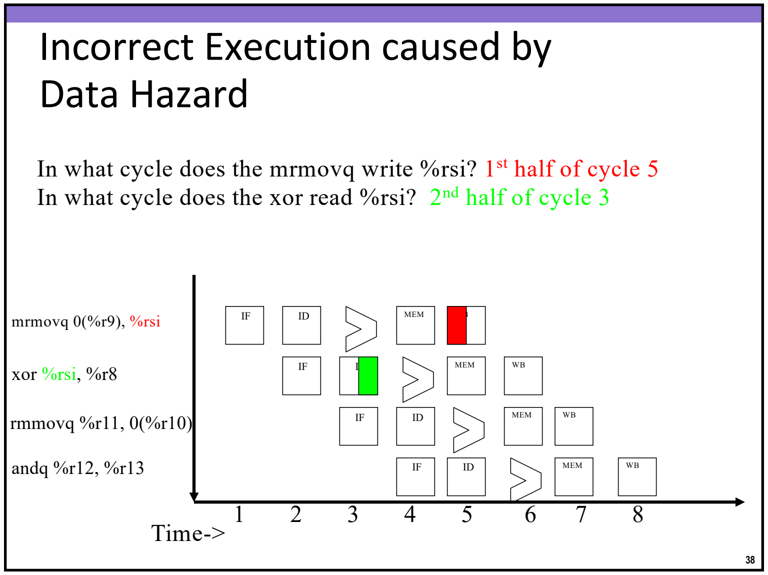


Incorrect Execution caused by Data Hazard

In what cycle does the mrmovq write %rsi? 1st half of cycle 5

In what cycle does the xor read %rsi?

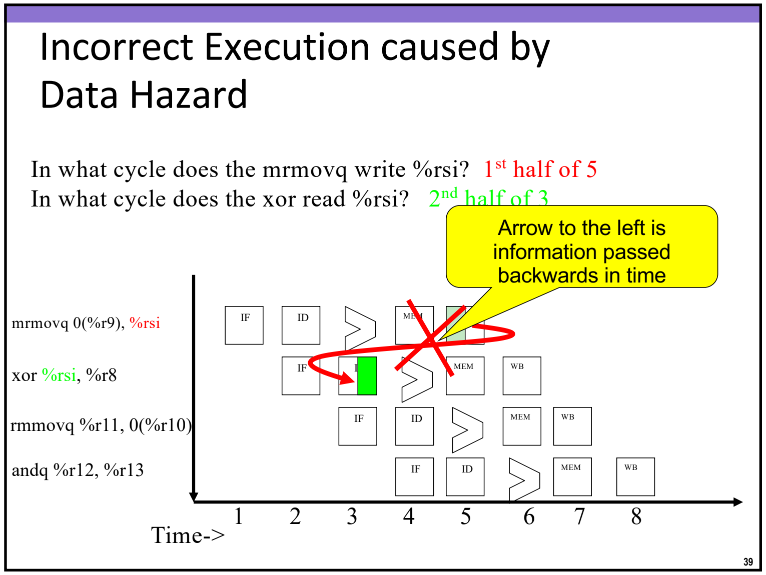
37



Incorrect Execution caused by Data Hazard

In what cycle does the mrmovq write %rsi? 1st half of cycle 5
In what cycle does the xor read %rsi? 2nd half of cycle 3

38

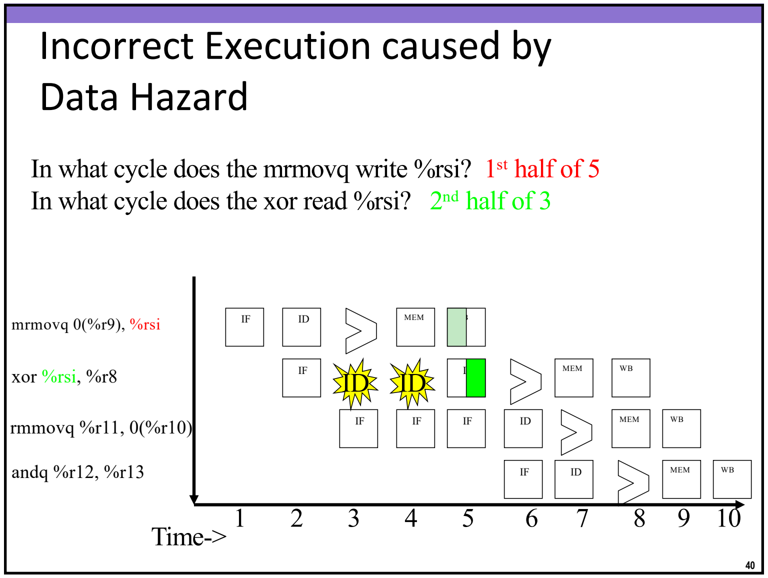


Incorrect Execution caused by Data Hazard

In what cycle does the mrmovq write %rsi? 1st half of 5
In what cycle does the xor read %rsi? 2nd half of 3

Arrow to the left is information passed backwards in time

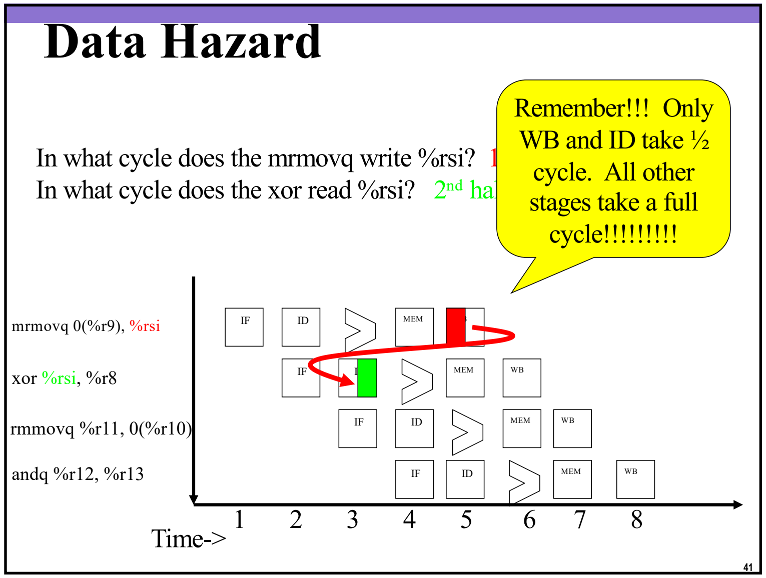
39



Incorrect Execution caused by Data Hazard

In what cycle does the mrmovq write %rsi? 1st half of 5
In what cycle does the xor read %rsi? 2nd half of 3

40

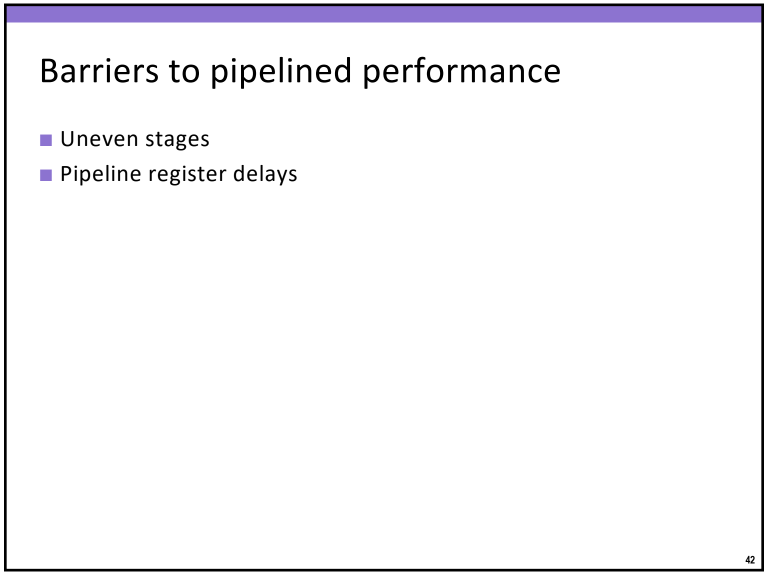


Data Hazard

In what cycle does the mrmovq write %rsi? 1
In what cycle does the xor read %rsi? 2nd ha

Remember!!! Only WB and ID take ½ cycle. All other stages take a full cycle!!!!!!!!!

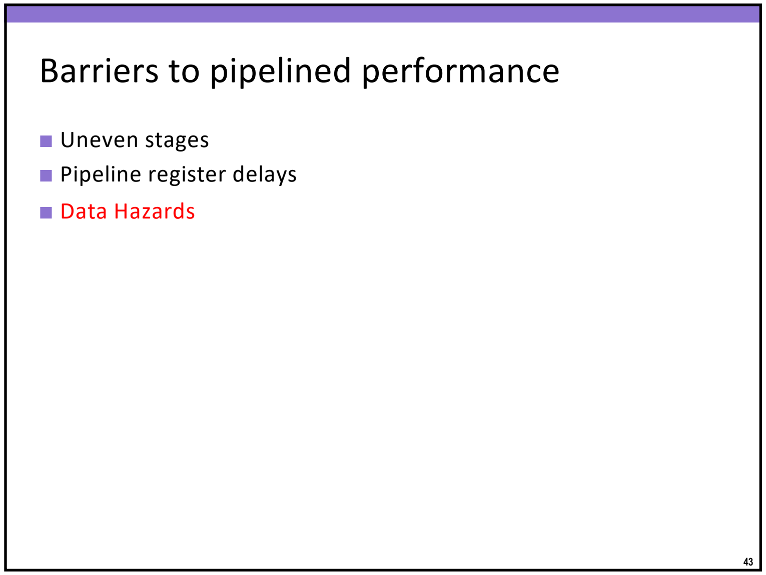
41



Barriers to pipelined performance

- Uneven stages
- Pipeline register delays

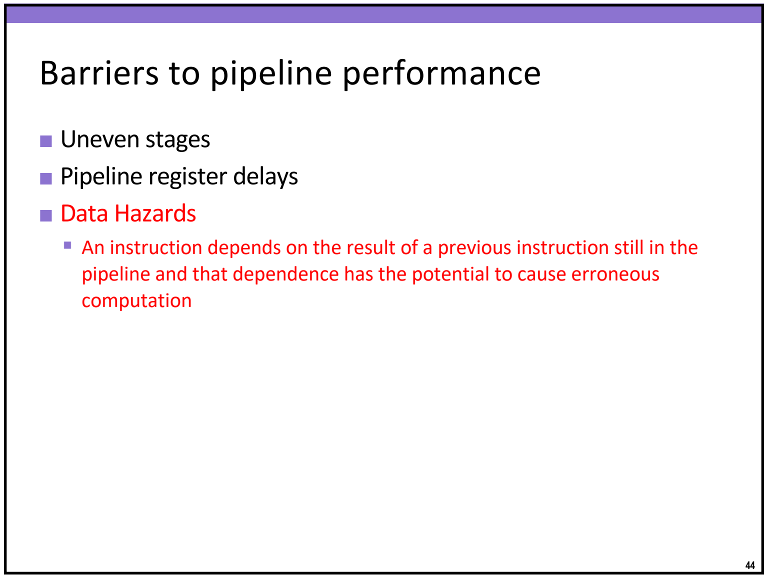
42



Barriers to pipelined performance

- Uneven stages
- Pipeline register delays
- Data Hazards

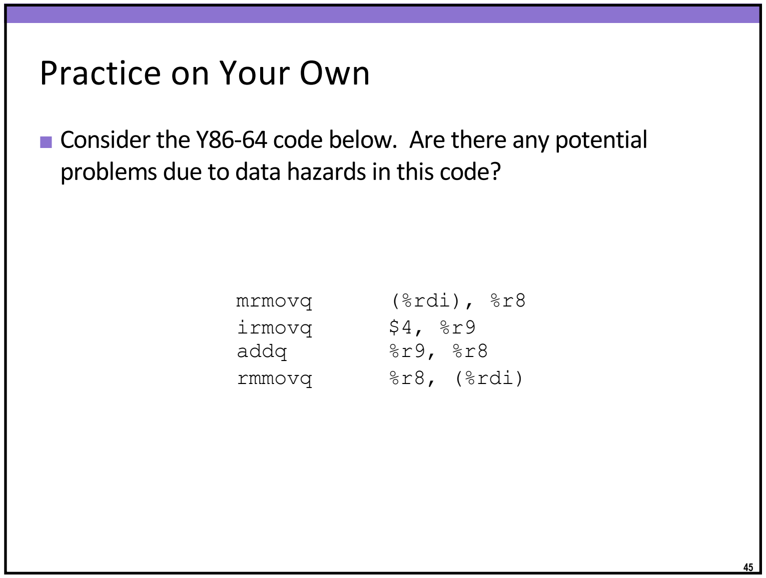
43



Barriers to pipeline performance

- Uneven stages
- Pipeline register delays
- Data Hazards
  - An instruction depends on the result of a previous instruction still in the pipeline and that dependence has the potential to cause erroneous computation

44

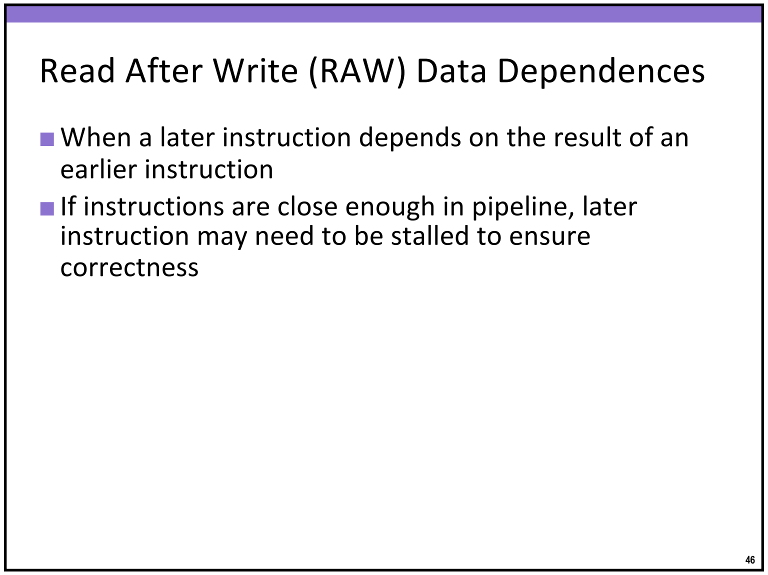


Practice on Your Own

- Consider the Y86-64 code below. Are there any potential problems due to data hazards in this code?

```
mrmovq      (%rdi), %r8
irmovq      $4, %r9
addq        %r9, %r8
rmmovq      %r8, (%rdi)
```

45

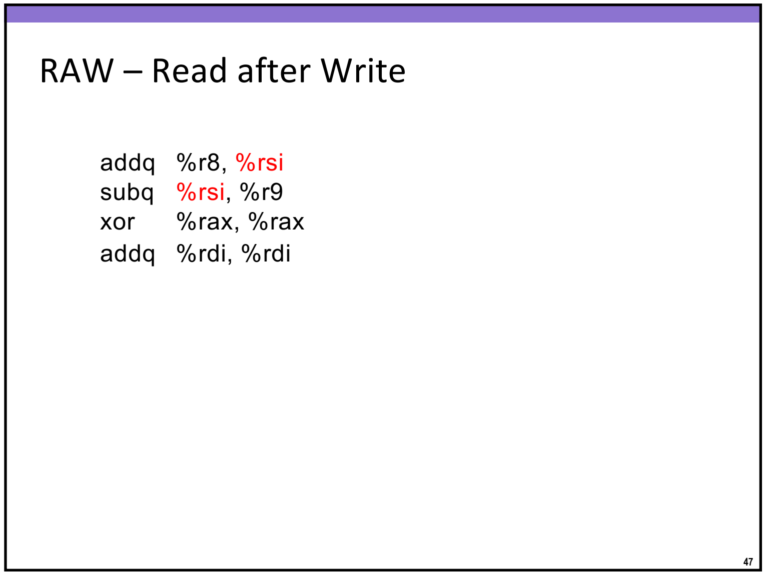


Read After Write (RAW) Data Dependences

- When a later instruction depends on the result of an earlier instruction
- If instructions are close enough in pipeline, later instruction may need to be stalled to ensure correctness

46

46

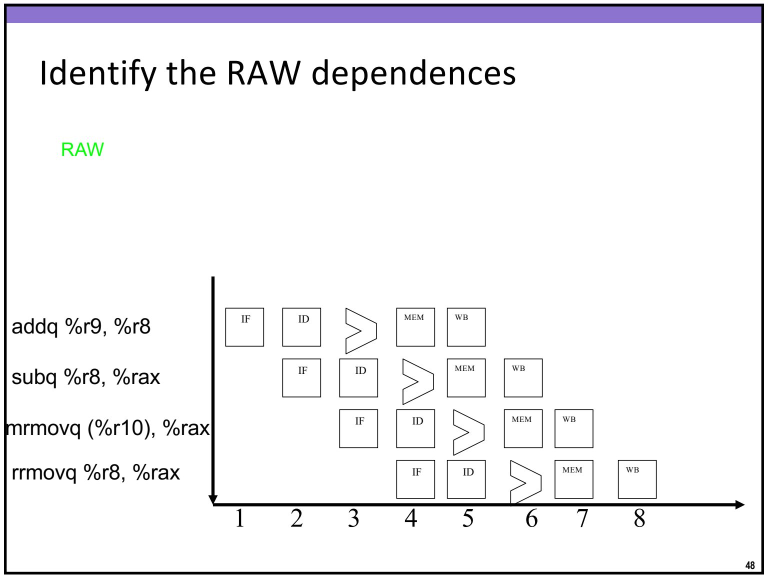


RAW – Read after Write

```
addq   %r8, %rsi
subq   %rsi, %r9
xor    %rax, %rax
addq   %rdi, %rdi
```

47

47



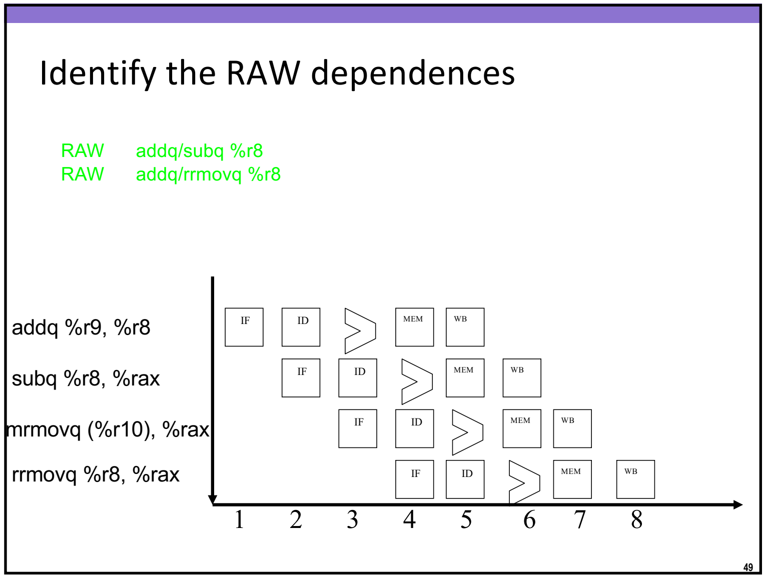
Identify the RAW dependences

RAW

addq %r9, %r8
subq %r8, %rax
mrmovq (%r10), %rax
rrmovq %r8, %rax

48

48



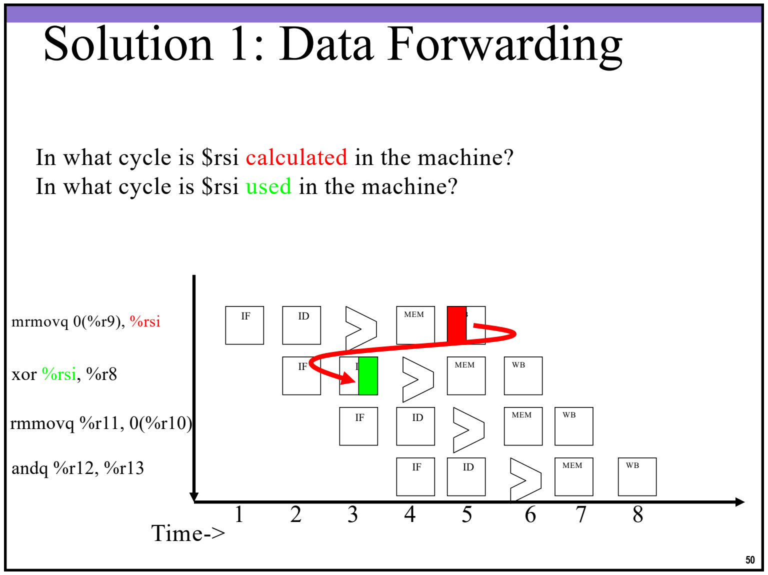
Identify the RAW dependences

RAW addq/subq %r8
RAW addq/rrmovq %r8

addq %r9, %r8
subq %r8, %rax
mrmovq (%r10), %rax
rrmovq %r8, %rax

49

49

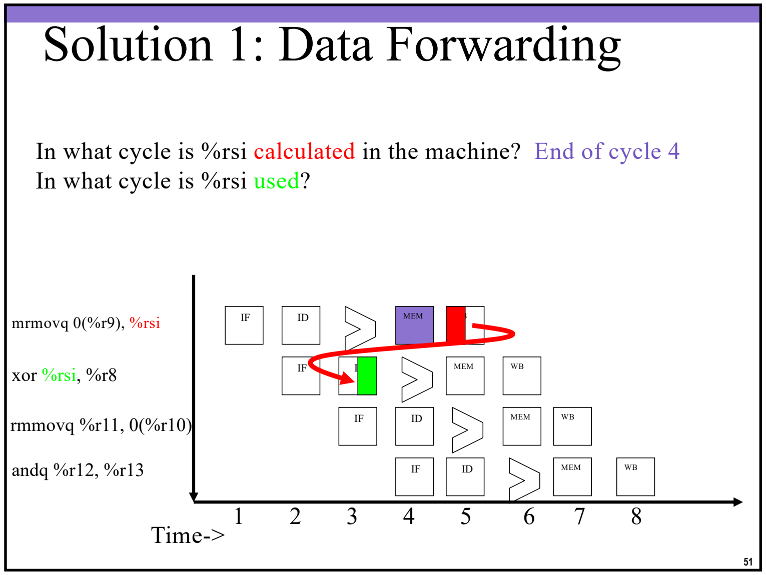


Solution 1: Data Forwarding

In what cycle is $rsi calculated in the machine?
In what cycle is $rsi used in the machine?

mrmovq 0(%r9), %rsi
xor %rsi, %r8
rmmovq %r11, 0(%r10)
andq %r12, %r13

50

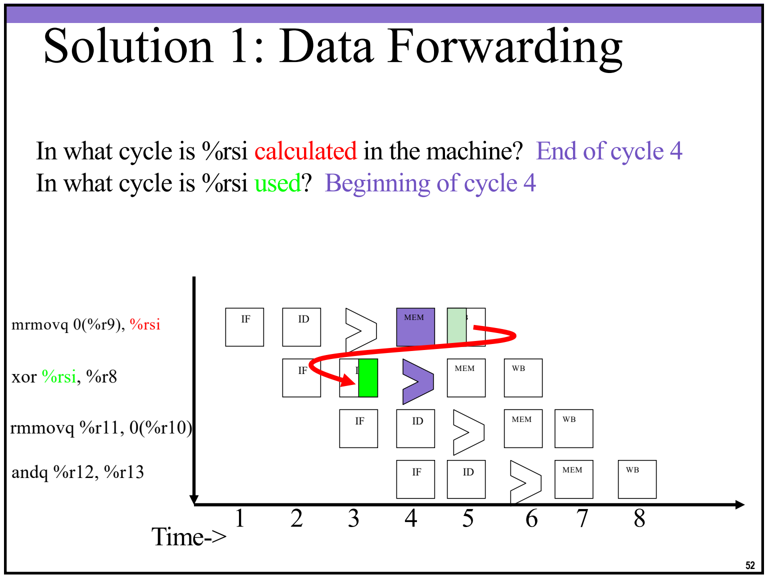


Solution 1: Data Forwarding

In what cycle is %rsi calculated in the machine? End of cycle 4
In what cycle is %rsi used?

mrmovq 0(%r9), %rsi
xor %rsi, %r8
rmmovq %r11, 0(%r10)
andq %r12, %r13

51

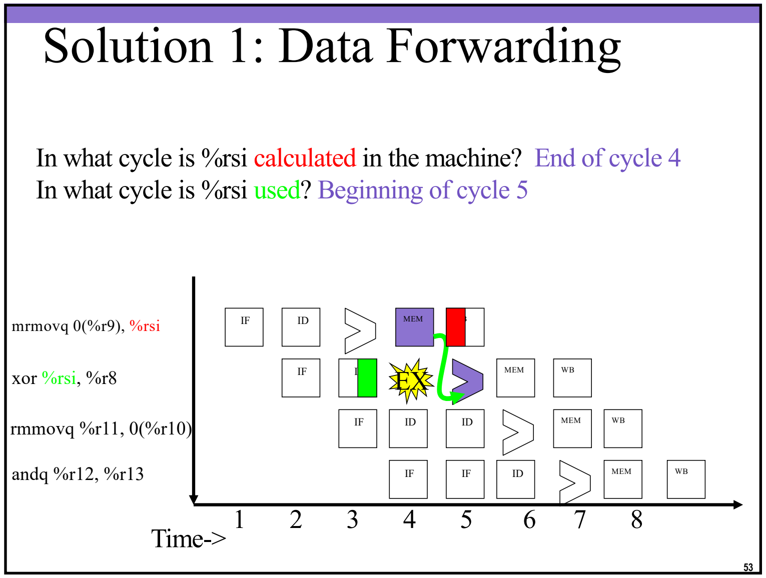


Solution 1: Data Forwarding

In what cycle is %rsi calculated in the machine? End of cycle 4
In what cycle is %rsi used? Beginning of cycle 4

mrmovq 0(%r9), %rsi
xor %rsi, %r8
rmmovq %r11, 0(%r10)
andq %r12, %r13

52

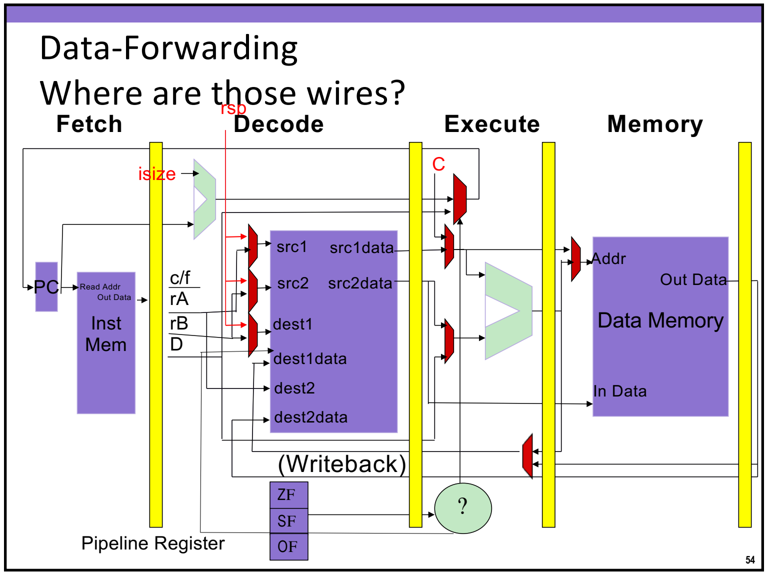


Solution 1: Data Forwarding

In what cycle is %rsi calculated in the machine? End of cycle 4
In what cycle is %rsi used? Beginning of cycle 5

mrmovq 0(%r9), %rsi
xor %rsi, %r8
rmmovq %r11, 0(%r10)
andq %r12, %r13

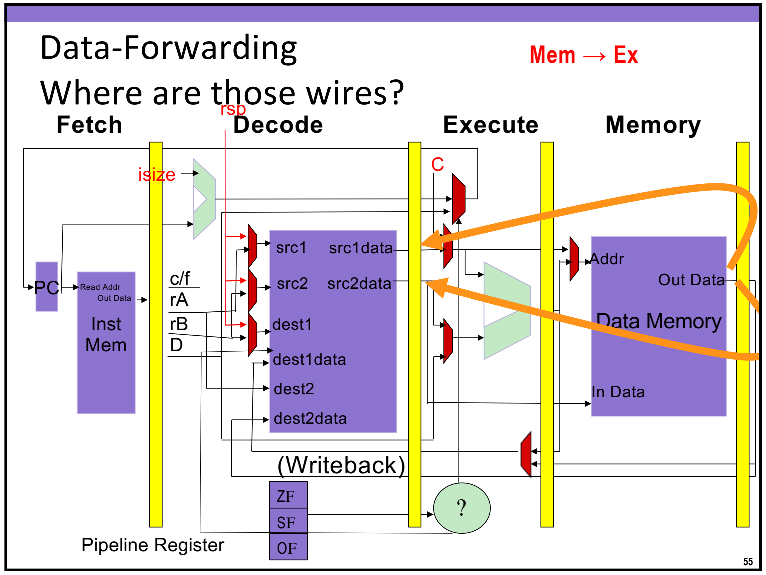
53



Data-Forwarding

Where are those wires?

54

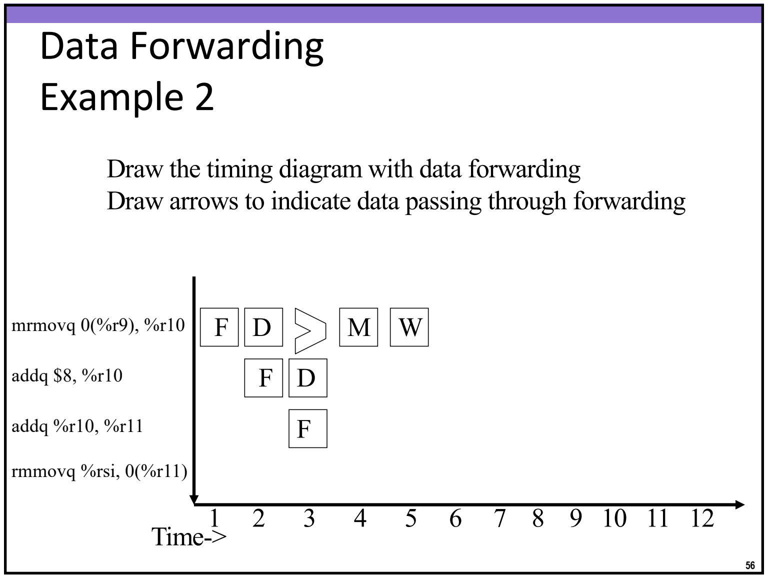


Data-Forwarding

Where are those wires?

Mem → Ex

55



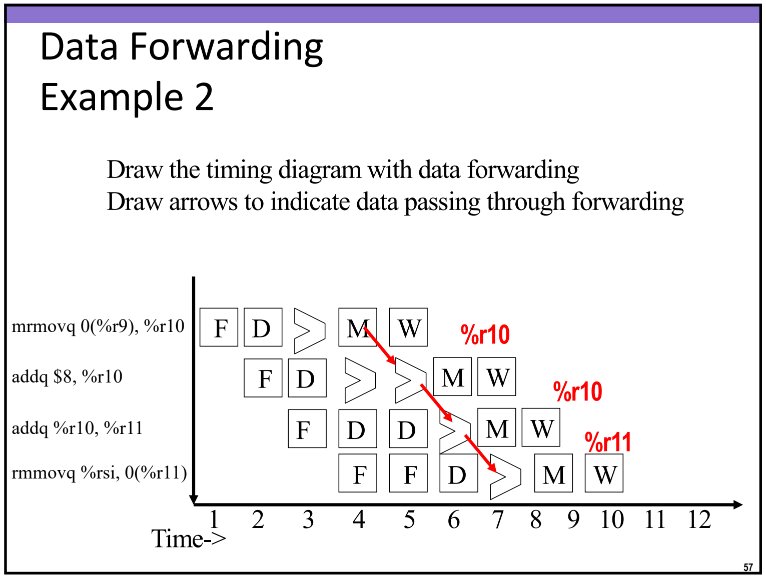
Data Forwarding Example 2

Draw the timing diagram with data forwarding
Draw arrows to indicate data passing through forwarding

	1	2	3	4	5	6	7	8	9	10	11	12
mrmovq 0(%r9), %r10	F	D	>	M	W							
addq $8, %r10		F	D									
addq %r10, %r11			F									
rmmovq %rsi, 0(%r11)												

Time->

56



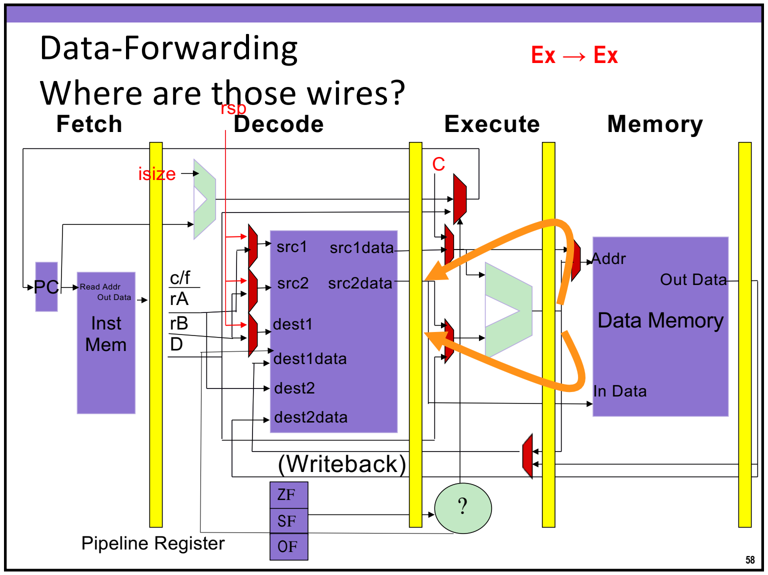
Data Forwarding Example 2

Draw the timing diagram with data forwarding
Draw arrows to indicate data passing through forwarding

	1	2	3	4	5	6	7	8	9	10	11	12
mrmovq 0(%r9), %r10	F	D	>	M	W							
addq $8, %r10		F	D	>	>	M	W					
addq %r10, %r11			F	D	D	>	M	W				
rmmovq %rsi, 0(%r11)				F	F	D	>	M	W			

Time->

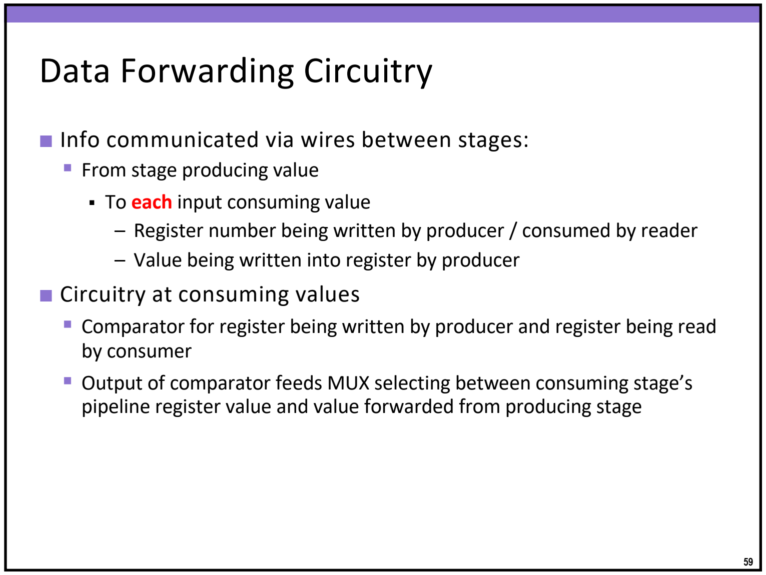
57



Data-Forwarding

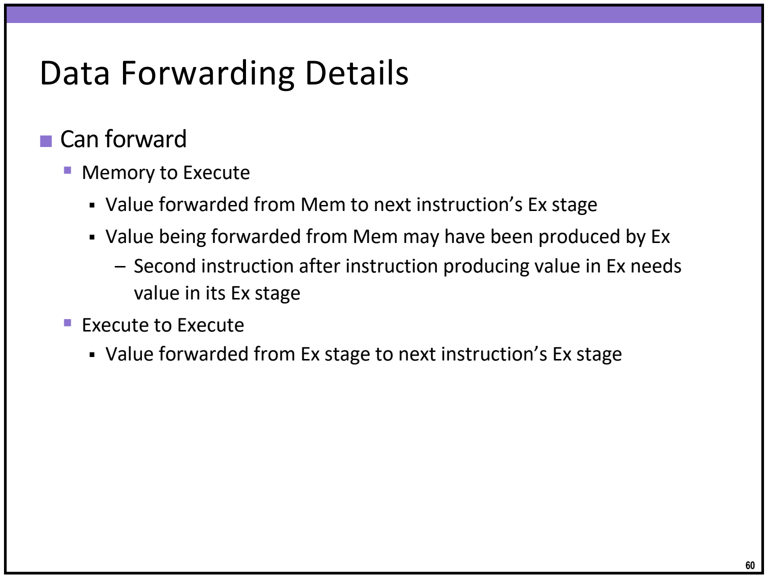
Where are those wires?

Ex → Ex



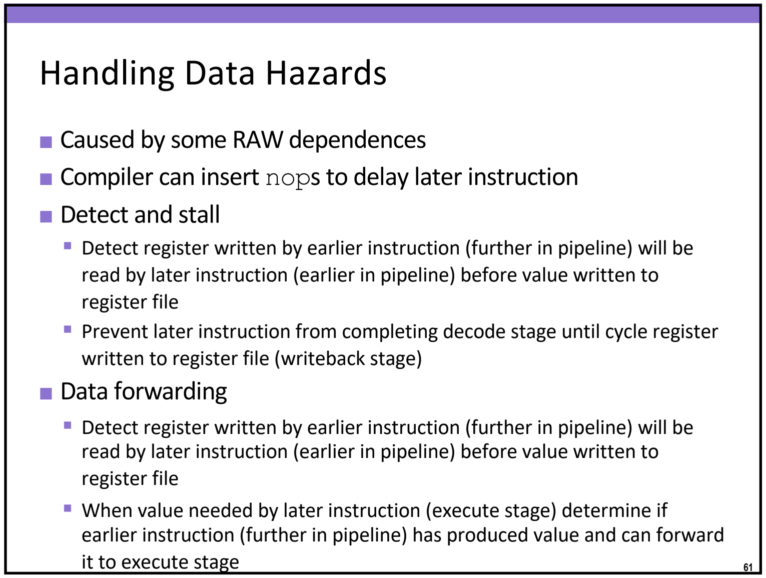
Data Forwarding Circuitry

- Info communicated via wires between stages:
  - From stage producing value
    - To **each** input consuming value
      - Register number being written by producer / consumed by reader
      - Value being written into register by producer
- Circuitry at consuming values
  - Comparator for register being written by producer and register being read by consumer
  - Output of comparator feeds MUX selecting between consuming stage's pipeline register value and value forwarded from producing stage



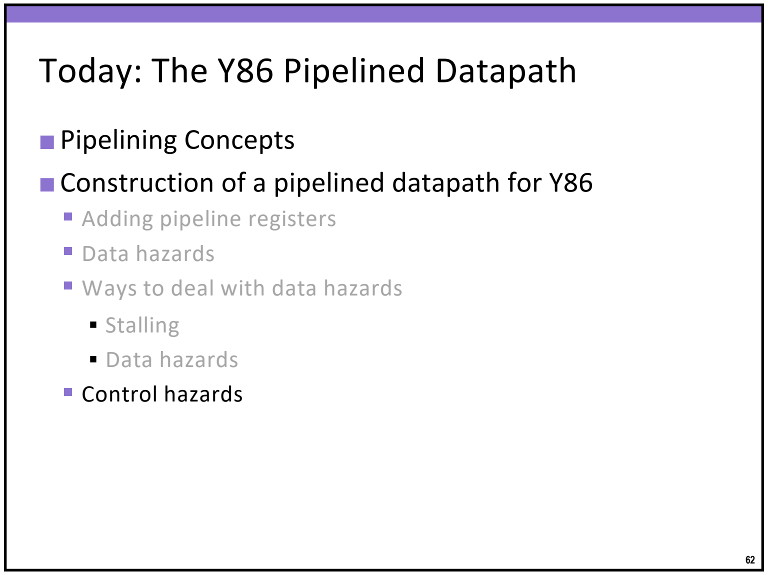
Data Forwarding Details

- Can forward
  - Memory to Execute
    - Value forwarded from Mem to next instruction's Ex stage
    - Value being forwarded from Mem may have been produced by Ex
      - Second instruction after instruction producing value in Ex needs value in its Ex stage
  - Execute to Execute
    - Value forwarded from Ex stage to next instruction's Ex stage



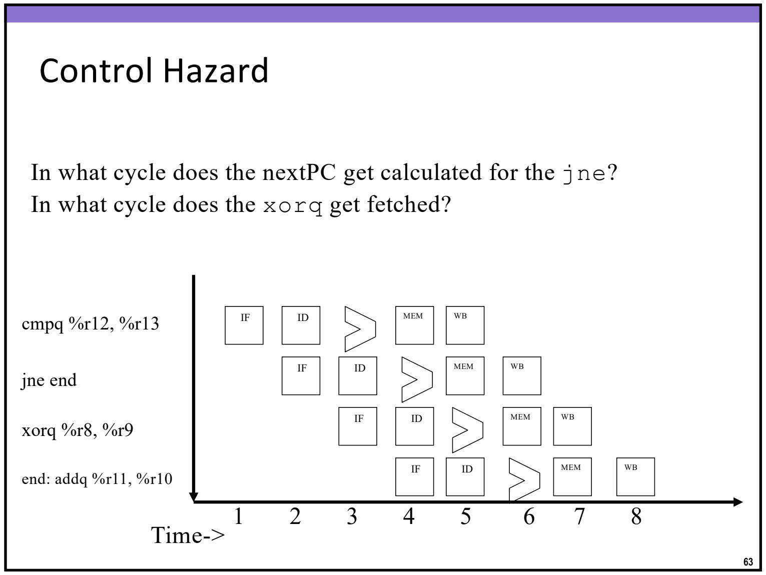
Handling Data Hazards

- Caused by some RAW dependences
- Compiler can insert `nops` to delay later instruction
- Detect and stall
  - Detect register written by earlier instruction (further in pipeline) will be read by later instruction (earlier in pipeline) before value written to register file
  - Prevent later instruction from completing decode stage until cycle register written to register file (writeback stage)
- Data forwarding
  - Detect register written by earlier instruction (further in pipeline) will be read by later instruction (earlier in pipeline) before value written to register file
  - When value needed by later instruction (execute stage) determine if earlier instruction (further in pipeline) has produced value and can forward it to execute stage



Today: The Y86 Pipelined Datapath

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

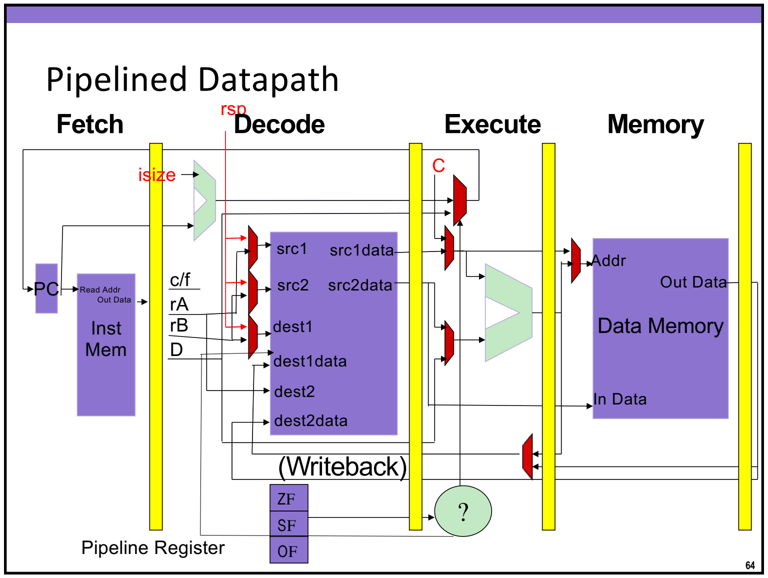


Control Hazard

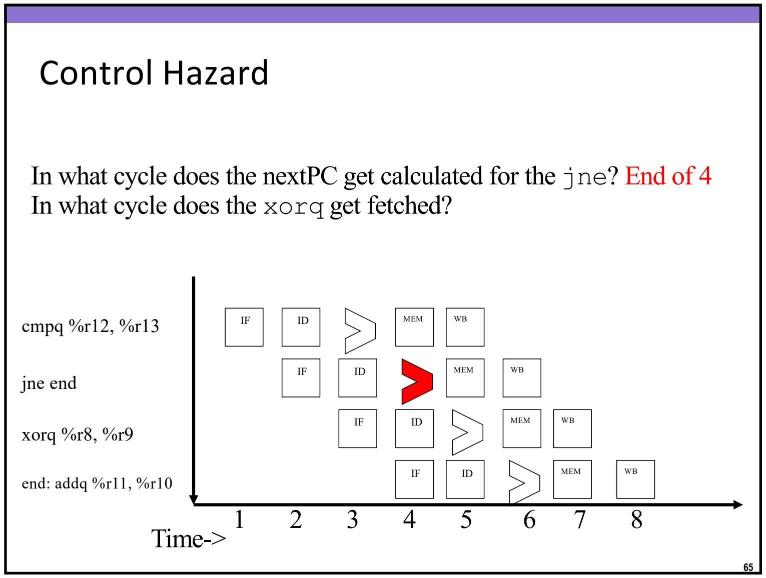
In what cycle does the nextPC get calculated for the `jne`?
In what cycle does the `xorq` get fetched?

	1	2	3	4	5	6	7	8
cmpq %r12, %r13	IF	ID	>	MEM	WB			
jne end		IF	ID	>	MEM	WB		
xorq %r8, %r9			IF	ID	>	MEM	WB	
end: addq %r11, %r10				IF	ID	>	MEM	WB

Time->



Pipelined Datapath

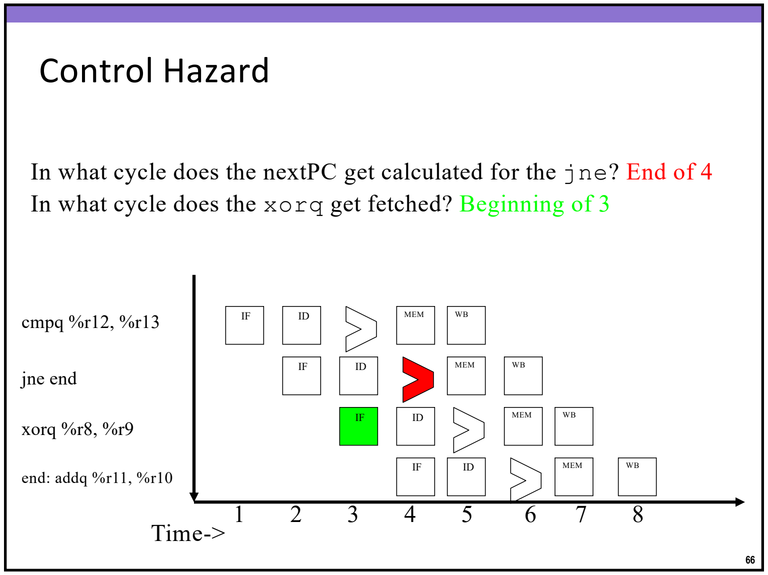


Control Hazard

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

	1	2	3	4	5	6	7	8
cmpq %r12, %r13	IF	ID	>	MEM	WB			
jne end		IF	ID	>	MEM	WB		
xorq %r8, %r9			IF	ID	>	MEM	WB	
end: addq %r11, %r10				IF	ID	>	MEM	WB

Time->



Control Hazard

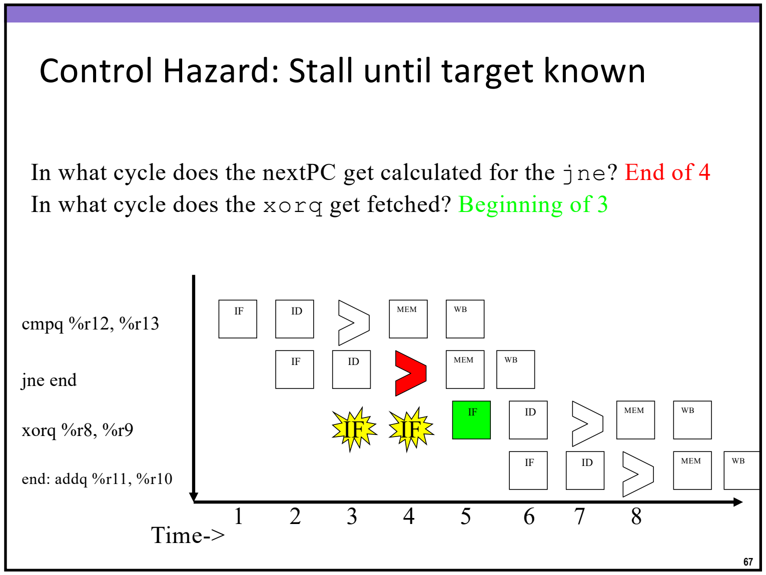
In what cycle does the nextPC get calculated for the `jne`? End of 4

In what cycle does the `xorq` get fetched? Beginning of 3

Time->	1	2	3	4	5	6	7	8
cmpq %r12, %r13	IF	ID	>	MEM	WB			
jne end		IF	ID	>	MEM	WB		
xorq %r8, %r9			IF	ID	>	MEM	WB	
end: addq %r11, %r10				IF	ID	>	MEM	WB

66

66



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

Time->	1	2	3	4	5	6	7	8		
cmpq %r12, %r13	IF	ID	>	MEM	WB					
jne end		IF	ID	>	MEM	WB				
xorq %r8, %r9			IF	IF	IF	ID	>	MEM	WB	
end: addq %r11, %r10						IF	ID	>	MEM	WB

67

67