6.004 Tutorial Problems
L18 – Data Hazards in Pipelined Processors

Resolving Data Hazards by Stalling

- Strategy 1: Stall. Wait for the result to be available by freezing earlier pipeline stages

```
addi x11, x10, 2
xor x13, x11, x12
sub x17, x15, x16
xori x19, x18, 0xF
```

Stall

	1	2	3	4	5	6	7	8
IF	addi	xor	sub	*sub*	*sub*	*sub*	xori	
DEC		addi	xor	*xor*	*xor*	*xor*	sub	xori
EXE			addi	**NOP**	**NOP**	**NOP**	xor	sub
MEM				addi	**NOP**	**NOP**	**NOP**	xor
WB					addi	**NOP**	**NOP**	**NOP**

↖ x11 updated

Stalls increase CPI!

Resolving Data Hazards by Bypassing

- Strategy 2: Bypass. Route data to the earlier pipeline stage as soon as it is calculated

```
addi x11, x10, 2
xor x13, x11, x12
sub x17, x15, x16
xori x19, x18, 0xF
```

- addi writes to x11 at the end of cycle 5… but the result is produced during cycle 3, at the EXE stage!

	1	2	3	4	5
IF	addi	xor	sub	xori	
DEC		addi	xor ↑	sub	xori
EXE			addi	xor	sub
MEM				addi	xor
WB					addi

addi result computed ↗ ↑x11 updated

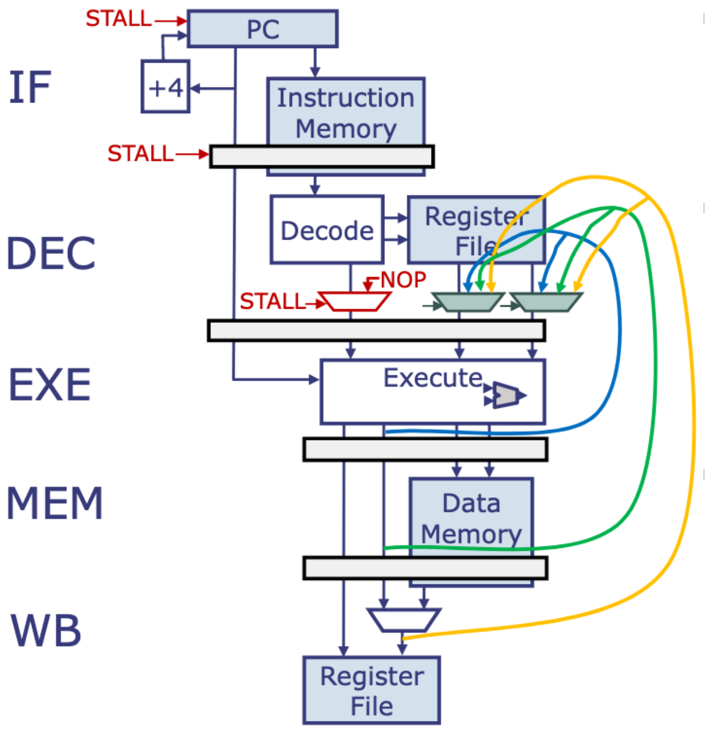
Load-To-Use Stalls

- Bypassing cannot eliminate load delays because their data is not available until the WB stage
- Bypassing from WB still saves a cycle:

```
lw x11, 0(x10)
xor x13, x11, x12
sub x17, x15, x16
xori x19, x18, 0xF
```

	1	2	3	4	5	6	7	8
IF	lw	xor	sub	*sub*	*sub*	xori		
DEC		lw	xor	*xor*	*xor*	sub	xori	
EXE			lw	**NOP**	**NOP**	xor	sub	xori
MEM				lw	**NOP**	**NOP**	xor	sub
WB					lw	**NOP**	**NOP**	xor

lw data available ↗ ↖ x11 updated



Problem 1. ★

The program shown on the right is executed on a 5-stage pipelined RISC-V processor with full bypassing.

The program has been running for a while and execution is halted at the end of cycle 105.

The pipeline diagram shown below shows the history of execution at the time the program was halted.

```
  . = 0
outer_loop:
  addi x11, x0, 16   // initialize loop index J
  addi x12, x0, 0

loop:                // add up elements in array
  addi x11, x11, -1  // decrement index
  slli x13, x11, 2   // convert to byte offset
  lw x14, 0x310(x13) // load value from A[J]
  add x12, x12, x14  // add to sum
  bnez x11, loop

  j outer_loop       // perform test again!
```

(A) Please indicate on which cycle(s), 100 through 105, each of the following actions occurred. If the action did not occur in any cycle, write "NONE".

cycle	*100*	*101*	*102*	*103*	*104*	*105*
IF	slli	lw	add	bnez	bnez	bnez
DEC	addi	slli	lw	add	add	add
EXE	NOP	addi	slli	lw	NOP	NOP
MEM	NOP	NOP	addi	slli	lw	NOP
WB	bnez	NOP	NOP	addi	slli	lw

Register value used from Register File: ________________

Register value bypassed from EXE stage to DEC stage: ________________

Register value bypassed from MEM stage to DEC stage: ________________

Register value bypassed from WB stage to DEC stage: ________________

(B) Why is the NOP instruction inserted in cycle 104?

Problem 2. ★

The following program fragments are being executed on the 5-stage pipelined RISC-V processor with full bypassing. For each fragment, the pipeline diagram shows the state of the pipeline for cycle 1000 of execution. Please fill in the diagram for cycle 1001; use "?" if you cannot tell what opcode to write into a stage. Then for **both** cycles use arrows to indicate any bypassing from the EXE/MEM/WB stages back to the DEC stage.

(A)

```
…
sw x1,  0(x0)
lw x17, 0xC(x1)
addi x2, x2, -4
slli x11, x17, 2
sw x11, 0(x2)
jal ra, fact
…
```

Cycle	*1000*	*1001*
IF	sw	
DEC	slli	
EXE	addi	
MEM	lw	
WB	sw	

(B)

```
…
xor x11, x11, x12
slli x12, x12, 3
sub x13, x12, x11
and x12, x13, x11
add x13, x12, x13
sw x13, 0x100(x0)
…
```

Cycle	*1000*	*1001*
IF	add	
DEC	and	
EXE	sub	
MEM	slli	
WB	xor	