6.004 Worksheet Questions
L19 – 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

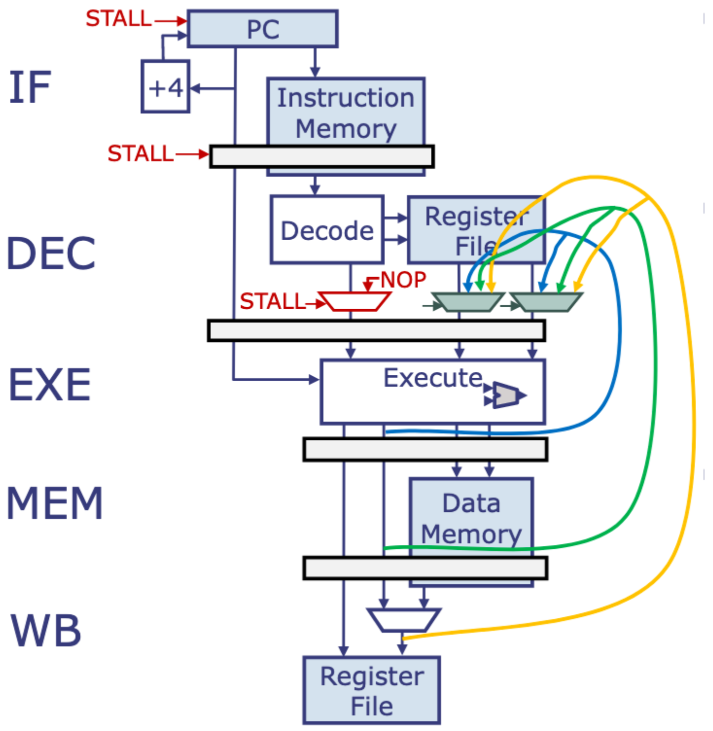
Load-To-Use Stalls

- 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	

Problem 3

For the following questions, assume that you are running code on the 4-stage pipelined processor whose stages are shown below. Also assume that:

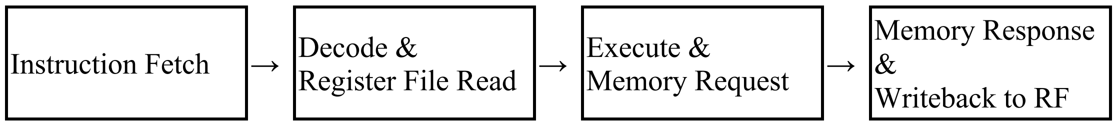
⇨ there is no bypassing,

⇨ the register file (RF) is *not* a bypass register file,

⇨ there are no early writebacks (meaning that every instruction must go through the last stage),

⇨ there is a register between pipeline stages, and

⇨ data memory responds immediately (in the next clock cycle) to any load request.


Instruction Fetch
Decode & Register File Read
Execute & Memory Request
Memory Response & Writeback to RF

The problems will ask about pipeline hazards; we will say a hazard is ***observable*** if it causes the pipeline to stall.

Note: For some problems it might be helpful to draw a pipeline diagram. There are several blank pipeline diagrams on the next page that you can use.

```
0x100       addi sp, sp, -4
0x104       srli a1, a0,  1
0x108       addi a4, a4, +1
0x10C       beqz a0, L0
0x110       andi a2, a1, 0x001
0x114   L0: sub  a3, a0, a2
0x118       lw   a5, +4(sp)
0x11C       sw   a3,  0(sp)
```

(A) Identify all the potential read-after-write (RAW) data hazards in the code above (including ones that are not observable). The number at the beginning of each line corresponds to the address of that line of code. For each hazard, write **Lines *a* and *b*** if a register is written in the line at address ***a*** and read in the line at address ***b***. If there are more spaces than you need, leave the extra spaces blank.

(1) Lines _________ and _________

(2) Lines _________ and _________

(3) Lines _________ and _________

(4) Lines _________ and _________

(5) Lines _________ and _________

(6) Lines _________ and _________

For each of the scenarios below, whenever you are asked to list the observable hazards, specify the number to the left of the hazard you are referring to from part A. So, if the hazards labelled (1) and (3) in part A of your answer are observable, the enter 1, 3 as your observable hazards response. Enter None if there are no observable hazards.

(B) Which of the hazards from part A are observable when the branch (address 0x10C***) is taken*** and is correctly predicted as being taken?

List of observable hazards or None: ____________________

(C) Which of the hazards from part A are observable when the branch is ***not taken*** and is correctly predicted as being not taken? How many nops are inserted (i.e., for how many cycles is the pipeline stalled) in this case?

List of observable hazards or None: ____________________

Number of nops/stalled cycles:____________________

(D) Now reconsider part (C) where the branch is ***not taken*** and is correctly predicted as being not taken. However, this time assume that there is a cache miss fetching the `lw` instruction at address 0x118 causing a three cycle stall. Which hazards are now observable?

List of observable hazards or None: ____________________

(E) Once again reconsider part (C) where the branch is ***not taken*** and is correctly predicted as being not taken. However, now suppose we add bypassing from the writeback stage to the decode stage (or equivalently, we replace the register file with a bypass register file), and the instruction cache no longer misses. Now which hazards are observable? How many nops are inserted (i.e., for how many cycles is the pipeline stalled) in this case?

List of observable hazards or None: ____________________

Number of nops/stalled cycles:____________________

IF									
DEC									
EXE									
WB									

IF									
DEC									
EXE									
WB									

IF									
DEC									
EXE									
WB									

Problem 4 (From Past Quiz)

Val wants to have a Thanksgiving party, but there's a nasty virus going around. Instead, she will buy the most expensive vegan turkey, a tomato, and a celery. She is at Whole Foods. She already knows that she can buy a vegan turkey for $10 at Wal-Mart, so she will be looking for something more expensive. Val wants to figure out how much her vegan meal will cost so she writes some C code, before converting it into assembly (shown on the right). Assume the registers are initialized to the values specified in the assembly code comments.

C Code

```
int price[6] = {7, 5, 8, 10, 15, 7};
int maximum = 10;
int celery = 3;
int tomato = 5;
for (int i = 0; i < 6; i++) {
    if (price[i] > maximum)
        maximum = price[i];
}
int total_cost = maximum + celery
                 + tomato;
```

Assembly Code

```
// x4 = 0x24  - length of price in bytes
// x5 = 0x3   - celery
// x6 = 0x5   - tomato
// x7 = 0xA   - maximum
// x1 = 0x400 - address of price[0]

start:  addi x2, x0, 0
        slli x2, x2, 2
loop:   add  x8, x2, x1
        lw x3, 0(x8)
        bge x7, x3, skip
        mv x7, x3
        ori x7, x7, 0
skip:   addi x2, x2, 4
        blt x2, x4, loop
        add x7, x7, x5
        add x7, x7, x6
```

In the following five-stage pipelined RISC-V processor (IF, DEC, EXE, MEM, WB):

- All branches are predicted not-taken. (Always fetch from PC + 4).
- Branch decisions are made in the EXE stage.
- The pipeline has **full bypassing**.
- The processor annuls instructions following taken branches.
- Assume that in the first iteration of the loop **both branches are taken.**

(A) Her program just started running "start". **Fill in the pipeline diagram for the first 14 cycles. Show all bypass passed used in each cycle.**

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IF	addi													
DEC														
EXE														
MEM														
WB														

(B) How many cycles did it take to execute the first loop iteration on this processor? Make sure not to include the first two instructions at label `start` in your cycle count.

Cycles to execute first iteration of the loop on this processor: _________

(C) If you could modify your fetch stage to always fetch the correct next instruction instead of predicting all branches not taken, how many cycles will it now take to execute the first iteration of the loop on this modified processor? Explain your answer.

(D) Val spent so much money on vegan turkey that she couldn't afford a processor with bypassing. In the cheapo processor she bought, **all data hazards are resolved by stalling.** Also, once again, **all branches are predicted not taken.**

Her program just started running "start". **Fill in the pipeline diagram for the first 14 cycles:**

Cycle	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
IF	addi													
DEC														
EXE														
MEM														
WB														

(E) How many cycles did it take to execute the first loop on this processor? Hint: To answer this question use the bypass and stall information you determined in parts (A) and (D) to calculate how many cycles the first iteration of the loop would take on the cheapo processor. **Explain how you arrived at your solution.**

Cycles per iteration on this processor: _________

Explanation:

Extra pipeline diagrams (for parts A and D):

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IF	addi													
DEC														
EXE														
MEM														
WB														

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IF	addi													
DEC														
EXE														
MEM														
WB														