3. [7 points] Pipelining

(a) Consider the classic 5-stage processor pipeline we've discussed in class. Suppose we found a way to get Fetch to run twice as fast, but all other stages run at the same speed as before. Would this help overall performance? Why or why not?

Will not help since Decode can process only one instruction at a time in a given clock cycle

(b) Fill in the pipeline diagram below, specifying when each insn enters each stage of the pipeline. You can assume that there are no branch mispredictions, and that each insn spends just 1 cycle in the Memory stage. You should model a pipeline *without any bypassing*. You should assume that if insn A writes the register file in the same cycle that insn B reads from it, B reads the updated value from A.

insn	1	2	3	4	5	6	7	8	9	10	11	12
`ld [r0+1]->r1`	F	D	X	M	W							
`add r1,r2->r3`		F	D	D	D	X	M	W				
`sub r1,r2->r4`			F	F	F	D	X	M	W			
`st r3->[r1+1]`						F	D	D	X	M	W	

3. **[9 points] Pipelining.**

The standard pipeline discussed in class has five stages: fetch (F), decode (D), execute (X), memory (M), and writeback (W). Consider the following four-stage pipelines, formed by combining two stages of the classic five-stage pipeline.

(a) Name a specific **positive** impact combining the D and X stages would have on CPI.

Reduce branch misprediction penalty from 2 to 1.

(b) Name a specific **positive** impact combining the X and M stages would have on CPI.

Eliminate LOAD-to-USE stall cycle!

Name a specific **positive** impact combining the M and W stages would have on clock frequency.

Reduces a bypass path, so less hardware overhead.
If this was the slowest stage, will result in lower overall latency.

(d) Name a specific impact combining the "F" and "D" stages would have on the implementation of branch prediction.

Name a **negative** impact moving from a 5-stage to a 4-stage pipeline would have.

Lower clock frequency because of higher cycle time.

4.) Pipelines (12 points total)
*Each short answer must be **30 words or less**, but you do not have to write complete sentences.*

4.1) (3 points) A common mistake on Lab 3 (Pipelined Datapath) is putting a decoder module in every stage of the pipeline rather than decoding once and passing the control signals from stage to stage. Why is this bad design?

(i) Every stage is now a bit slower
(ii) Why even have a decoder stage anymore?

4.2) (2 points) Why does pipelining improve instruction throughput but not instruction latency?

(i) more instructions per unit time
(ii) each individual instruction is still the same in terms of latency (practically may even be slightly worse).

4.3) (3 points) Shrek wants to execute a program with a store instruction that overwrites the instruction immediately following it. This is an insidious case of dynamic code generation, but Shrek is an ogre after all. To keep the pipeline fast, he wants to add a bypass from **X** to **D** for the fetched instruction itself. Assume each stage takes the entire cycle to compute its results. Explain why this will or will not work.

Stored updated address is only available in the Memory stage (after X is computed) so this will not work.

4.4) (4 points) List four instances where you need to bypass information from one stage to another in Lab 3 (Pipelined Datapath). For each one, list the stage in which the data is produced (**From**), the stage in which it is consumed (**To**), the type of data being forwarded (or a reasonable name for the wire that carries it), and an LC4 (pseudo-)assembly example illustrating when it is necessary. We will grade the example based on whether the reason for bypassing is clear, correct, and concise, not on whether your LC4 assembly has perfect syntax.

From	To	Data/Wire Name	Code Example
M	X	R2	ADD R0, R1 → R2 ADD R2, R3 → R4
W	X	R2	ADD R0, R1 → R2 ADD R2, R3 → R4 ADD R2, R5 → R6
W	M	memory	LDR [R1, 2] → R3 load-to-store STR R3 → [R6, 7]
M	X	R2	ADD R5, R4 → R2 SUB R6, R2 → R3

3. [9 points] Pipelining

(a) Consider the classic 5-stage processor pipeline with MX, WX and WM bypassing as we've discussed in class. Dr. Arch E. Teck is considering a new **MF** bypass to bypass results from the M stage to the F stage. He claims this can help reduce stalls, thereby improving performance. Is this true? Why or why not?

[3 points] It would not help performance because the Fetch stage can't use any information from the Memory stage. Fetch is responsible for branch prediction and reading from the Insn Memory. Both of these steps depend on the insn in Decode, not in Memory. An insn i1 in Fetch that has a register dependence on i2 in Memory will be able to get its result either via the register file or a WX bypass.

→ will not really help

(b) Fill in the pipeline diagram below, specifying when each insn enters each stage of the pipeline. You can assume that there are no branch mispredictions, and that each insn spends just 1 cycle in the Memory stage. You should model a pipeline *without any bypassing*. You should assume that if insn A writes the register file in the same cycle that insn B reads from it, B reads the updated value from A.

insn	1	2	3	4	5	6	7	8	9	10	11	12
add r1<-r2,r3	F	D	X	M	W							
xor r4<-r2,r1		F	D	.	.	X	M	W				
ld r5<-[r1,7]			F	.	.	D	X	M	W			
st r5->[r1,4]						F	D	.	.	X	M	W

[6 points] Basically, there is only a "WD" bypass for dependent insns.
-1 point for each illegal bypass used
-1 point for each pair of insns in a stage simultaneously (structural hazard)

3. [10 points] Pipelining

By analyzing an existing 5-stage pipelined processor design, Dr. Chip D. Signer realizes that the X stage is the slowest. He proposes splitting X into 3 stages: X1, X2 and X3.

(a) What additional bypassing, if any, would be needed for this new 7-stage pipeline beyond the standard M→X1, W→X1 and W→M bypasses?

None, you can't bypass values into the middle of an ALU instruction.

(b) Give **one** performance advantage the 7-stage design is likely to have over the 5-stage design.

Lower clock period ⇒ higher clock frequency

(c) Give **two** performance penalties the 7-stage design is likely to suffer compared to the 5-stage design.

(i) Back to back ALU instructions must stall
(ii) Load-to-use now has a 3-cycle penalty

(d) Give the stall logic for the X1 stage. Under what conditions do we need to prevent the insn currently in the X1 stage from advancing? Assume for simplicity that each insn has only a **single** source register, and also assume any bypassing you mentioned in your answer to 3a above. You can reference the insns in other stages via the name of their stage, and can access the following information about each insn:

- `.src` – the insn's source register
- `.dst` – the insn's destination register
- `.isLoad` – bit indicating whether the insn is a load or not
- `.isStore` – bit indicating whether the insn is a store or not

For example, you can write `M.src` to get the source register of the insn in the M stage; you can use `X2.isLoad` to tell whether the insn in the X2 stage is a load or not; etc.

```
stall = (X1.src == X2.dst) ||
        (X1.src == X3.dst) ||
        (M.isLoad && !X1.isStore && X1.src = M.dst)
```

The last condition tests for the following:
(i) Previous instruction was a LDR
(ii) It is not a LOAD & STORE condition
(iii) There is a data dependence

3.) Stalling for Time (12 points total)

Congratulations! You've scored a coveted interview with Sunoco Logistics for a job designing a custom processor to control critical functions of the new Mariner East 2 pipeline. This processor must be designed on time and perform to expectations, or the pipeline may end up flooding Philadelphia in propane. You know you'll be asked plenty of hard questions about pipelines and the lobby to bypass Philadelphia completely, so you've decided to prepare some answers relative to your pipeline design from Lab 3, as well as some intelligent things to say while you stall for time.

Prepare your answers to each potential interview question below. Each answer must be **30 words or less**, but you do not have to write complete sentences.

3.1) (4 points) Suppose that bypass logic is too difficult to implement, so you can only stall. What issues could this cause, if any?

The system would function correctly. Performance will degrade (throughput will reduce) because of the stall cycles.

3.2) (4 points) Suppose the stall logic is too difficult to implement, and it is only possible to bypass. What issues could this cause, if any?

The system will NOT function correctly because there is no way to handle a LOAD-to-USE hazard without a stall.

3.3) (4 points) Suppose bypass logic is too difficult to implement, but all instructions except load/store instructions skip from execute straight to write-back. What issues could this cause, if any?

A sequence of LOAD followed by an instruction that requires the register that LOAD just updated will result in a structural hazard

LOR [R1, 7] → R3

ADD R3, R4 → R5

4.) Pipe Dreams (30 points total)

Last night, after a relaxing evening of studying, you found yourself dreaming about pipeline diagrams. Before the details of the dream fade completely, you decide to commit the diagrams you still remember to paper. Fill in the cycle diagram for the code sequence below, starting at memory address **x0000** and ending when the program reaches a memory address that is not included in the listing below. Mark all bypasses with arrows. Mark stalls with empty circles. There may be empty rows and columns in the completed table. The first instruction is filled in for you (except any bypassing that may be required to the second instruction).

```
.CODE
.ADDR x0000
I1 CONST R0, x00
I2 HICONST R0, x40
I3 CONST R1, x1
I4 ADD R1, R1, x1
I5 TRAP DOOR
```

```
.OS
.CODE
.ADDR x8200
DOOR
I6 ADD R1, R1, R1
I7 STR R1, R0, x0
I8 LDR R7, R0, x0
I9 RTI
```

Label	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
I1	F	D	X	M	W																						
I2		F	D	X	M	W																					
I3			F	D	X	M	W																				
I4				F	D	X	M	W																			
I5					F	D	X	M	W																		
I6						-	-	F	D	X	M	W															
I7								*	F	D	X	M	W														
I8										F	D	X	M	W													
I9											F	D	*	X	*	W											
I4													-	-	F	D	X	M	W								
I5																F	D	X	M	W							
I6																	-	-	F	D	X	M	W				
I7																				F	D	X	M	W			
I8																					F	D	X	M	W		
I9																						F	D	*	X	M	W

1. Why the double stalls at I6, I4?

2. Why the stall at I9?
Framed to be a LOAD-to-USE since RTI depends on R7!

4. [22 points] Pipelining

(a) Dr. Chip D. Signer wants to improve processor performance by switching from a 5-stage pipeline to an 85-stage pipeline. His calculations show that the processor can indeed run at 100GHz. List two performance issues his design is likely to encounter.

(i) Large amounts of bypass logic (additional space and time)
(ii) More data hazards (inevitable stalls)
(iii) Large branch misprediction penalty
(iv) High energy consumption

(b) Fill in the pipeline diagram below, showing when each insn will reach each stage in a 5-stage pipelined LC-4 processor **without any bypassing**. You should indicate any stall cycles with a *. Within a given cycle, reading from a register occurs **before** writing.

	insn	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
(i)	ADD r0 ← r1, r2	F	D	X	M	W															
(ii)	SUB r2 ← r1, r2		F	D	X	M	W														
(iii)	XOR r3 ← r0, r1			F	D	*	*	X	M	W											
(iv)	STR r0 → [r2,3]				F	*	*	D	X	M	W										
(v)	CMP r0, r3							F	D	*	*	X	M	W							
(vi)	BRn #42								F	*	*	D	*	*	X	M	W				

1. Why are there 2 stalls in (iii) instead of just 1 stall?

2. Why are there 2 stalls in (v) instead of just 1 stall?

3. There was a stall in cycle 14 of (vi). Why?

(c) Given the pipeline diagram below, fill in a series of LC4 insns such that they flow through a **fully-bypassed** 5-stage pipeline with the given timing.

	insn	1	2	3	4	5	6	7	8	9	10
(i)	LOAD [R0,1] → R1	F	D	X	M	W					
(ii)	LOAD [R1,1] → R2		F	D	*	X	M	W			
(iii)	LOAD [R2,1] → R3			F	*	D	*	X	M	W	
(iv)	ADD R4,R5 → R6					F	*	D	X	M	W

load-to-use

load-to-use

1. Did (iii) have to be a LDR instruction?

5. [16 points] Pipeline Bypassing

(a) Give three LC4 insns that, when run back-to-back, exercise the MX, WX and WM bypasses to avoid *any* stalling.

(i) ADD R1, R2 → R3
(ii) LOAD R3, 8 → R4
(iii) STORE R4 → R3, 7

MX Bypass : R3 between (i) and (ii)
WM Bypass : R4 between (ii) and (iii)
WX Bypass : R3 between (i) and (iii)
↳ memory address calculation depends on the value of register R3.

(b) Fill in the pipeline diagram below for the given instructions, assuming a 5-stage LC4 pipeline with **only MX and WX** bypassing. The first insn has been completed for you. Use a * to indicate stall cycles.

	insn	1	2	3	4	5	6	7	8	9	10	11	12	13	14
(i)	LDR r1 <= [r0,0]	F	D	X	M	W									
(ii)	ADD r2 <= r1,r0		F	D	*	X	M	W							
(iii)	LDR r3 <= [r2,0]			F	*	D	X	M	W						
(iv)	STR r3 => [r0,3]					F	D	*	X	M	W				
(v)	LDR r4 <= [r3,2]						F	*	D	X	M	W			
(vi)	BRn target														

load to use

1. (i)-(ii) is a classic example of LOAD TO USE which requires a stall cycle and a WX Bypass
2. (ii)-(iii) requires a simple MX Bypass to calculate the address for the load insn.
3. (iii)-(iv) requires nothing special since there is already a one cycle stall
4. (iv)-(v) requires nothing
5. (v)-(vi)