[2] When we talk about the number of operands in an instruction (a 1-operand or a 2-operand instruction, for example), what do we mean?

[2] What are the two main ways to define performance?

[2] Predicting the direction of a branch is not enough. What else is necessary?

[2] The power consumed by a chip has increased over time, but the clock rate has increased at a far greater rate. How was this possible? How did designers keep the chip from melting?

[2] What is a benchmark program?

[2] Do benchmark programs remain valid indefinitely? Why or why not?

[3] It is difficult for the internal processing elements on a chip to communicate with the outside world (things that are on other chips, for example). Explain why that is.

[3] You are in charge of designing a new embedded machine, and for a variety of reasons you **must** use a fixed 19 bit instruction size. You would like to support 256 different operations, use a 3-operand instruction format, and have 16 registers. If it is possible to do this, draw what an instruction would look like. If it is not possible, explain why, and give two different ways (excluding reducing the number of registers) to fix the problem.

[3] Processor A requires 400 instructions to execute a given program, uses 3 cycle per instruction, and has a cycle time of 2 ns. Processor B requires 5 cycles per instruction, but only requires 120 instructions to do the same program. What must the cycle time of Processor B be in order to give the same CPU time as Processor A? (Show your work)

[3] An important program spends 80% of its time doing Integer operations, and 20% of its time doing floating point arithmetic. By redesigning the hardware you can make the Integer unit 30% faster (take 70% as long), or you can make the Floating Point unit 99% faster (take 1% as long). Which should you do, and why?

[10] You are given a machine with an 8 stage pipeline, which writes to the register file during the D1 cycle and reads during D2. This machine uses neither a branch delay slot nor a load delay slot. These are the stages:

F   D1   D2   E1   E2   M1   M2   W

a. Assuming this machine has a branch predictor and the branch condition is calculated by the end of the D1 stage, how big is the branch penalty (measured in cycles) when the prediction is incorrect? What if the branch condition is calculated by the end of E2?

b. Assume on a load the value is available at the end of M2. How many stall cycles stalls would this machine need on a load if it has forwarding logic and you are forwarding to E1? What if you were forwarding to E2?

c) What type of data hazard does the above pipeline need to worry about?

d) If the above pipeline were modified to support out of order completion, what new data hazard would be introduced?

[16] Here is a code sequence.

      load   **R5, 0(R2)**

      load   **R2, 4(R6)**

      add   **R3, R2, R1**

      store   **R3, 20(R9)**

      sub   **R8,R7,R8**

      and   **R8,R9,R10**

Assuming a standard 5-stage pipeline that does not support hazard detection and does no forwarding,

a) Indicate all dependencies (draw lines/arrows between them, and write beside each line/arrow which hazard is involved).

b) Insert as many No Operations (NOPS) as required in order to ensure this code runs correctly. (Remember, writes to the register file occur on the first half of the cycle, and reads occur during the second half).

c) Circle the NOPs that can be removed if forwarding and hazard detection logic is implemented.

d) Assuming the pipeline has been modified so that it does support hazard detection and forwarding, schedule the code to remove as many stalls as possible. How many NOPS are left?

e) If R6 contains the value 0x24, what address in memory is read from when executing the second load instruction?

In this question, we are going to wire up a 12-bit processor. The machine is word-addressable, where a word is 12 bits. Immediates are sign extended, Offset is not. The machine has 3 different instruction formats: R, I, and J. Memory takes a single cycle to return a value.

R-type:	*(Arithmetic and logical:*			*rd = rs1 OP rs2)*
Opcode	rd	rs1	rs2	funct
11-8	7-6	5-4	3-2	1-0

I-type:	*(Arithmetic and logical:*		*rd = rs1 OP Immediate)*
	(Load:		*rd = mem[rs1+Immediate])*
	(Store:		*mem[rs1+Immediate]=rd)*
	(Branch:		*if (rd OP rs1)=COND, PC=PC+Immediate)*
Opcode	rd	rs1	Immediate
11-8	7-6	5-4	3-0

J-type:	*(Jump:*	*PC = Offset)*
Opcode	Offset	
11-8	7-0	

The ALU can perform 7 functions, written this way: OP [ALU2 ALU1 ALU0]
Increment Bus [111], Decrement Bus [110], Add [011], SUB [100], AND [101], OR [001], NOT [010]

Here are some of the instructions that have been defined:

Name	Opcode(Funct)	Name	Opcode(Funct)	Name	Opcode(Funct)
NOP	0000(00)	lw	0001(xx)	sw	0011(xx)
NOT	1000(00)	BEQZ	0100(xx)	j	0101(xx)
AND	1000(01)	AND Imm	1001(xx)	ADD	1100(01)
OR	1000(10)	OR Imm	1010(xx)	ADD Imm	1101(xx)
XOR	1000(11)	XOR Imm	1011(xx)	SUB	1100(01)

Here are the 21 control signals.

PCin	PCout	IRin	IRout	MDRin	MDRout	MARin
Zin	Zout	RDin	RDout	MemRead	MmWrite	Imm
RS1out	RS2out	#0out	ALU0	ALU1	ALU2	Xin

Here is a diagram of the machine.
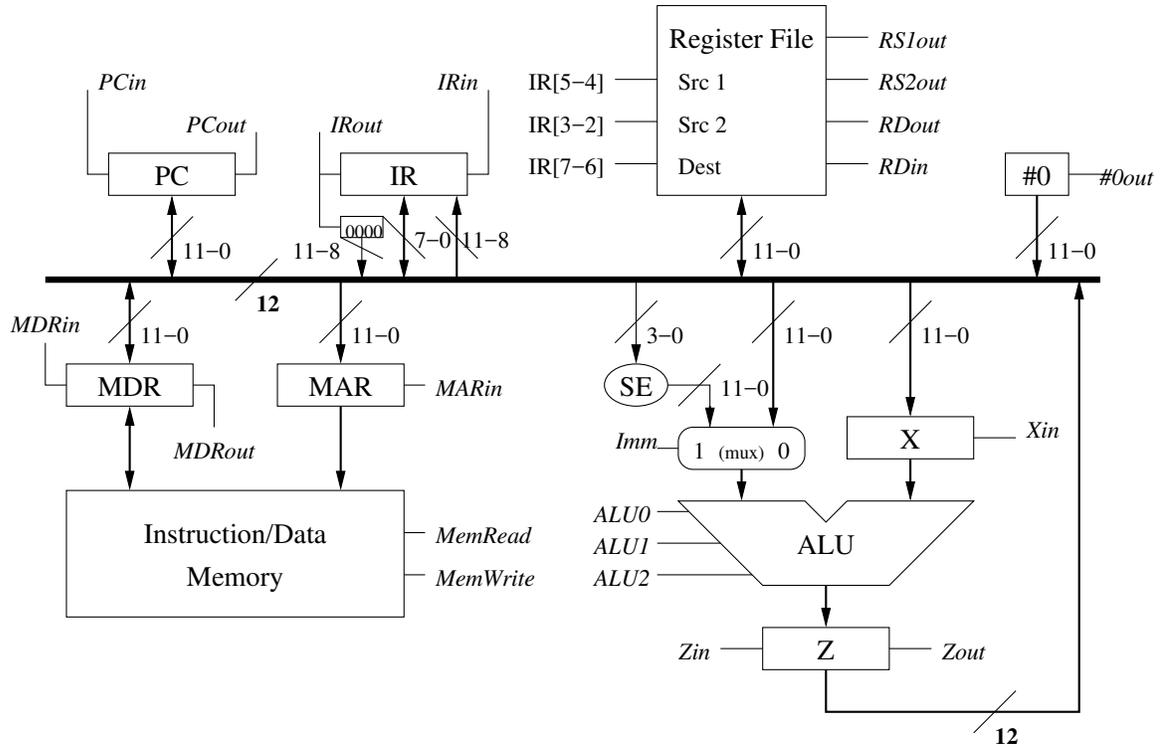
[9] Fill in the microcode steps necessary to perform an instruction fetch (incrementing the PC is considered part of the instruction fetch process).

Step	PCin	IRin	MARin	MDRin	RDin	Xin	Zin	IRout	PCout	MDRout	Zout	RDout	RS1out	RS2out	#0out	ALU2	ALU1	ALU0	Mread	Mwrite	Imm
0																					
1																					
2																					
3																					
4																					
5																					

[4] Write down the binary bit pattern (the bit pattern that will be in the IR after you have done the instruction fetch) for the instruction: **AND R3,R1,R2**

[12] Now that you have done the instruction fetch, fill in the microcode steps necessary to perform the following instruction: **LW R1, 2[R3]** Assume memory can respond in a single cycle.

Step	PCin	IRin	MARin	MDRin	RDin	Xin	Zin	IRout	PCout	MDRout	Zout	RDout	RS1out	RS2out	#0out	ALU2	ALU1	ALU0	Mread	Mwrite	Imm
0																					
1																					
2																					
3																					
4																					

[3] Assuming you have done the instruction fetch, fill in the microcode steps necessary to perform the following instruction: **J 0x23**

Step	PCin	IRin	MARin	MDRin	RDin	Xin	Zin	IRout	PCout	MDRout	Zout	RDout	RS1out	RS2out	#0out	ALU2	ALU1	ALU0	Mread	Mwrite	Imm
0																					
1																					
2																					
3																					

[6] In your first design of a 5-stage pipeline (F,D,E,M,W), F takes 22 time units, D takes 26, E takes 30, M takes 12 and W takes 13.

a) What will the clock cycle time be for this pipeline?

b) Is it a balanced pipeline? If not, explain what you could do to fix it. What would the cycle time be now?

[4] We want to use microcode to provide the control signals for a given machine which has 21 control signals. Assuming there are 19 states in the State Transition Diagram and you are using the simplest microcode configuration,
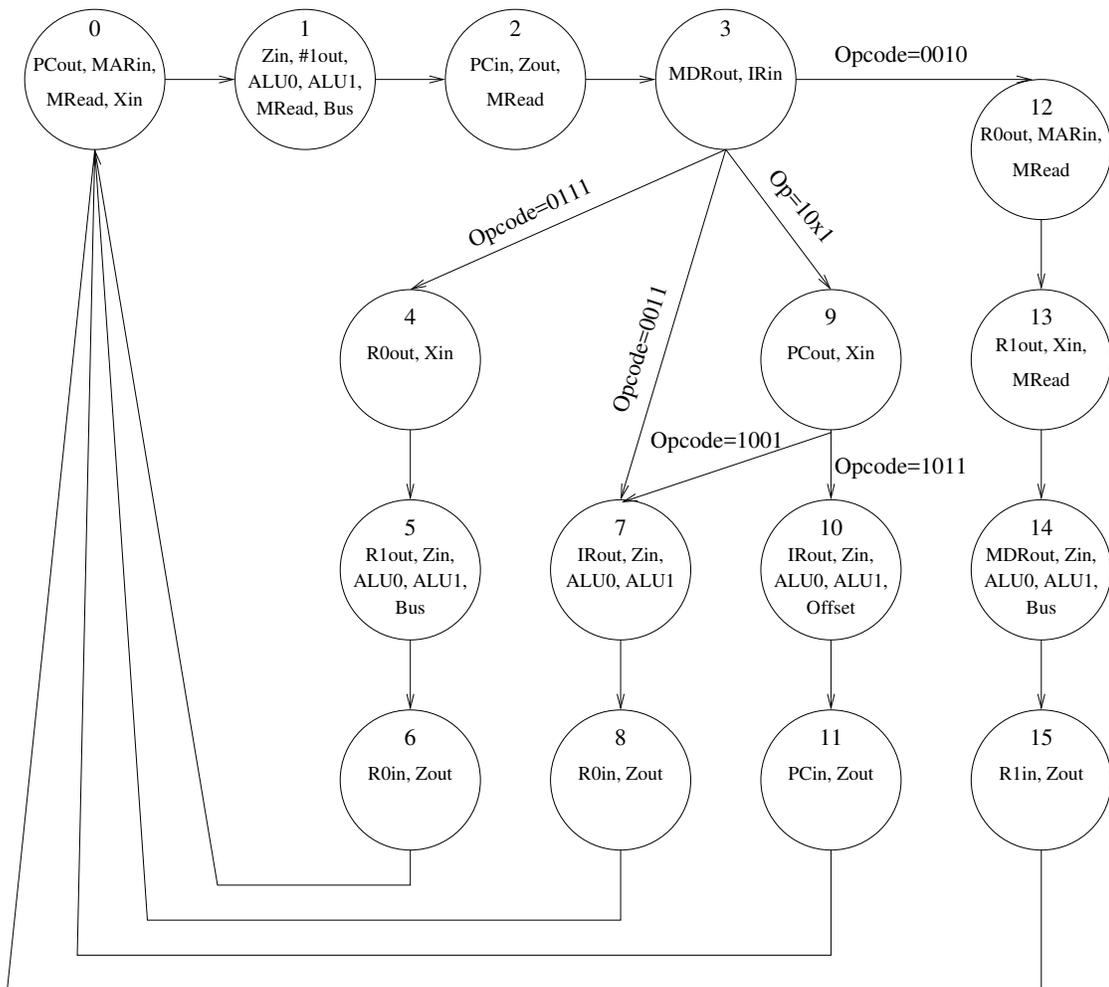
a. How many entries would the microcode memory have?

b. How wide would each entry be?

(Drawing a picture might be useful).

[3] Briefly describe a Tournament Branch Predictor. Be sure to touch on how it works and what characteristic of branch predictors it is exploiting. Pictures are welcome.

Here is the state diagram for a random machine:

0
PCout, MARin, MRead, Xin

1
Zin, #1out, ALU0, ALU1, MRead, Bus

2
PCin, Zout, MRead

3
MDRout, IRin

Opcode=0010

12
R0out, MARin, MRead

13
R1out, Xin, MRead

Opcode=0111

Op=10x1

Opcode=0011

4
R0out, Xin

9
PCout, Xin

Opcode=1001

Opcode=1011

5
R1out, Zin, ALU0, ALU1, Bus

7
IRout, Zin, ALU0, ALU1

10
IRout, Zin, ALU0, ALU1, Offset

14
MDRout, Zin, ALU0, ALU1, Bus

6
R0in, Zout

8
R0in, Zout

11
PCin, Zout

15
R1in, Zout

[3] Assuming there are 4 state variables (Y3-Y0), that State0 = $\overline{Y3}*\overline{Y2}*\overline{Y1}*\overline{Y0}$ (0000) and State15 = Y3*Y2*Y1*Y0 (1111), write down the exact boolean equation for the **PCin** signal.

[4] Assuming the same situation as in the previous question, write down the exact boolean equation for **NextState7**.

[2] If you were using a minimized microcode configuration for this machine, circle on the diagram where the dispatch roms points are.

[7] In class, we talked about the cycle by cycle steps that occur on different interrupts. For example, here is what happens if there is an illegal operand interrupt generated by instruction i (note this machine has a 6 stage pipeline):

	1	2	3	4	5	6	7	8	9	
i	IF	ID	EX	M1	M2	WB				<- Interrupt detected
i+1		IF	ID	EX	M1	M2	WB			<- Instruction Squashed
i+2			IF	ID	EX	M1	M2	WB		<- Trap Handler fetched
i+3				IF	ID	EX	M1	M2	WB	
i+4					IF	ID	EX	M1	M2	WB

Fill out the following table if instruction i+1 experiences a fault in the EX stage (Overflow, for example) and we are assuming imprecise interrupts:

	1	2	3	4	5	6	7	8	9	10
i	IF	ID	EX	M1	M2	WB				
i+1		IF	ID	EX	M1	M2	WB			
i+2			IF	ID	EX	M1	M2	WB		
i+3				IF	ID	EX	M1	M2	WB	
i+4					IF	ID	EX	M1	M2	WB
i+5						IF	ID	EX	M1	M2	WB

Assuming we are supporting precise interrupts, what happens in this case?

	1	2	3	4	5	6	7	8	9	10	
i	IF	ID	EX	M1	M2	WB					<- M2 stage has Page Fault
i+1		IF	ID	EX	M1	M2	WB				<- EX unit has overflow error
i+2			IF	ID	EX	M1	M2	WB			<- Illegal Instruction detected
i+3				IF	ID	EX	M1	M2	WB		
i+4					IF	ID	EX	M1	M2	WB	
i+5						IF	ID	EX	M1	M2	WB
i+6							IF	ID	EX	M1	M2	WB
i+7								IF	ID	EX	M1	M2	WB

[3] What is the maximum number of exceptions that could happen at one time in the above machine? Why?