[2] Which is on average more effective, dynamic or static branch prediction?

[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] 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 is a benchmark program?

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

[2] Does reducing the minimum feature size affect power density? If so, in what way?

[4] Occasionally the hardware detects a hazard and must insert a bubble into the pipeline. How is this done?

[4] You are in charge of designing a new embedded machine, and for a variety of reasons you **must** use a fixed 18 bit instruction size. You would like to support 256 different operations, use a 2-operand instruction format, and have 64 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 to fix the problem.

[3] Processor A requires 600 instructions to execute a given program, uses 1 cycle per instruction, and has a cycle time of 5 ns. Processor B requires 5 cycles per instruction, but only requires 60 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 70% of its time doing Integer operations, and 30% of its time doing floating point arithmetic. By redesigning the hardware you can make the Integer unit 40% faster (take 60% as long), or you can make the Floating Point unit two orders of magnitude faster (take 1% as long). Which should you do and why?

[11] The MIPS implementation we used in class has a 5-stage pipeline, writes to the register file during the first half of the cycle and reads during the second half, and uses both a branch delay slot and a load delay slot. If the machine is redesigned to be a 7-stage pipeline, with the following stages:

F D E1 E2 M1 M2 WB

a. Assuming this machine has a branch predictor and the branch condition is calculated by the end of the E1 stage, how big is the branch penalty (measured in cycles) when the prediction is incorrect? What if the branch condition is not calculated until the end of E2?

b. How many load delay slots would this machine need (assuming it has forwarding logic and you are forwarding to E1) assuming the memory returns the value by the end of M1? M2?

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?

e) If in addition to completing out of order, instructions were allowed to issue out of order, what new data hazard would be introduced?

[18] Here is a code sequence.

```
load   R1, 0(R10)

load   R2, 4(R1)

add    R3, R2, R1

store  R3, 20(R9)

sub    R3,R7,R8

load   R11, 4(R6)
```

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?

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 X [001], Decrement X [010], Add [011], SUB [100], AND [101], OR [110], NOT [111]

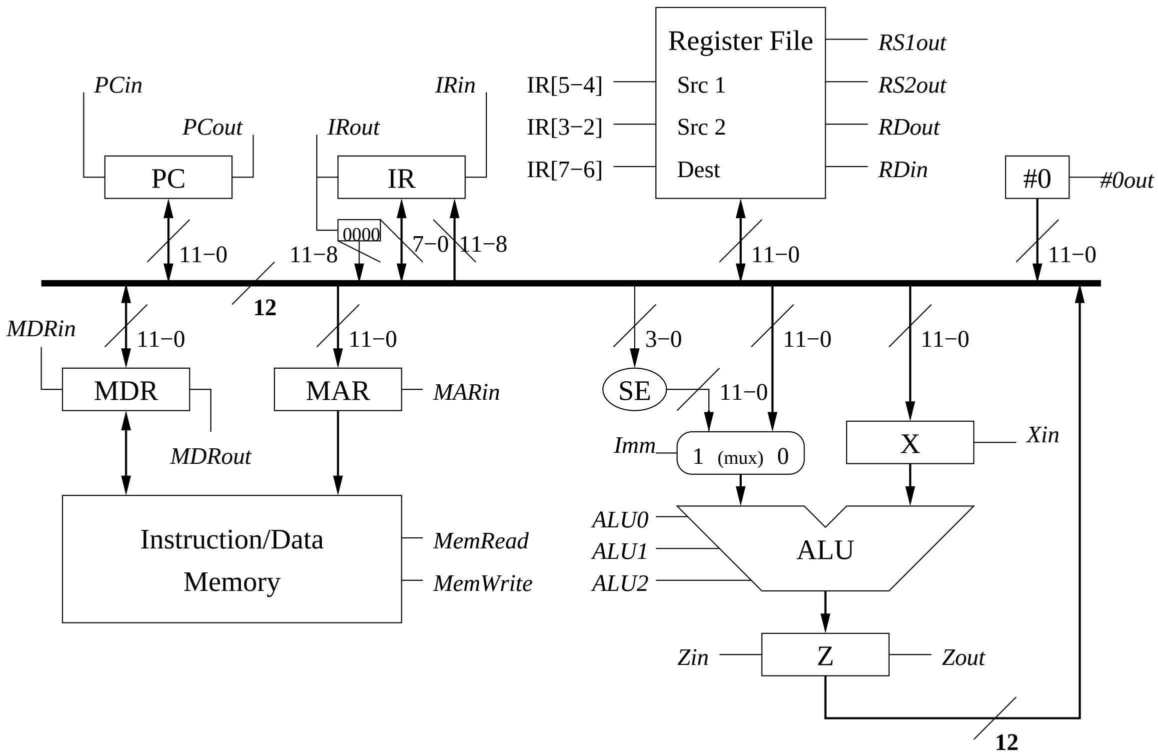
Here are some of the instructions that have been defined:

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.



[12] 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																					

[3] 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: **LW R1, -1[R2]**

[12] Now that you have done the instruction fetch, fill in the microcode steps necessary to perform the following instruction: **LW R1, -1[R2]**

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																					

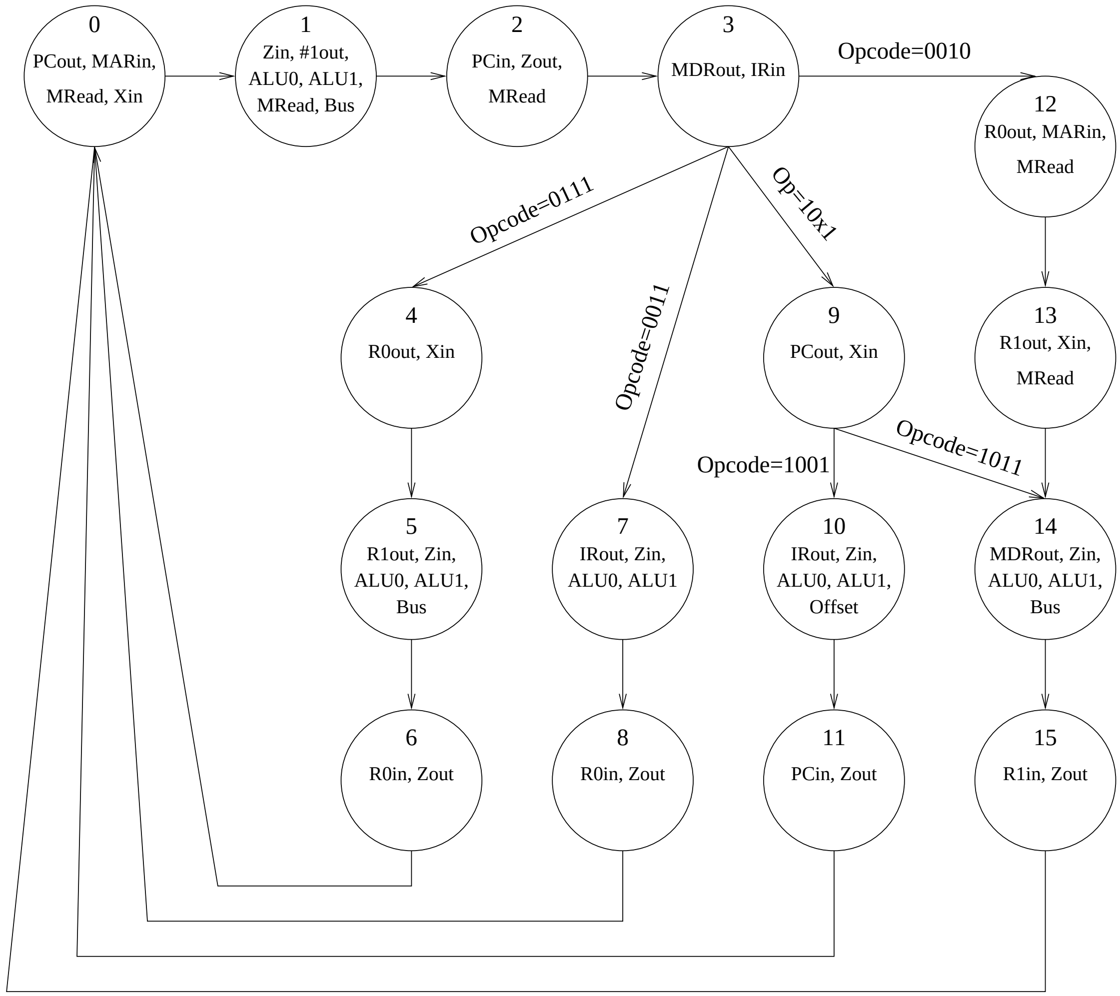
[4] We want to use microcode to provide the control signals for the above machine. Assuming there are 17 states in the State Transition Diagram and you are using the simplest microcode configuration,

a. How many entries would the microcode memory have?

b. How wide would each entry be?

(Drawing a picture might be useful).

Here is the state diagram for a random machine:



[4] 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 **R0in** signal.

[4] Assuming the same situation as in the previous question, write down the exact boolean equation for **NextState14**.

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