(1) [3] What is one of the simplest (and oldest) techniques for exploiting parallelism among instructions?

(2) [3] When dealing with control hazards, it is not enough to predict the branch direction - what else must we know?

(3) [3] Which is on average more effective, dynamic or static branch prediction?

(4) [3] Compilers have trouble scheduling code that involves reads and writes to memory. Why?

(5) [4] Once the trap handler has begun, what two things does it need to know in order to deal with an interrupt?

(6) [4] What are the 3 pipeline hazards? Which one can be eliminated by providing more resources?

(7) [4] What are the 3 types of data hazards? Which one can be dealt with by using forwarding?

(8) [4] Why is there a desire to create larger basic blocks? Give one example of a way to create a bigger basic block.

(9) [4] What is the definition of a precise interrupt?

(10) [4] Artificial intelligence techniques can provide more accurate branch prediction than existing approaches - why are these techniques not currently used?

(11) [4] Branch prediction requires the ability to squash (or undo) instructions. How do you "squash" an instruction in a pipelined machine? (What must be prevented from happening?)

(12) [4] Why do we care about techniques like forwarding, since hazards can be avoided by introducing stalls in the pipeline? Why do we want to avoid stalls?

(13) [4] Intel uses a Tournament predictor in some of their processors. Describe what a tournament predictor is, and why it is used. Your description can (and probably should) include sketches/drawings.

(14) [8] In your first design of a 6-stage pipeline (F,D,E1,E2,M,W), F takes 29 time units, D takes 32, E1 takes 16, E2 takes 14, M takes 31, and W takes 28.

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?

(15) [14] The standard MIPS implementation has a 5-stage pipeline, writes to the register file during the first half of the D cycle and reads during the second half. 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 D 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 stalls would this machine need (assuming it has forwarding logic and you are forwarding to E1) if the memory returns the value by the end of M2? M1?

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 the pipeline were modified to support out of order issue, what new data hazard would be introduced?

(16) [12] Here is a code sequence.

```
label:  lw      R4, 4(R4)

        sub     R3, R2, R4

        add     R9, R3, R8

        lw      R9, 8(R11)

        lw      R8, 4(R10)

        bne     R6, label       ; branch if R6 != 0
```

Assuming a 7-stage pipeline (F D E1 E2 M1 M2 WB) that does not have a branch delay slot, 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. Also, memory values are returned at the end of M2).

c) Circle the NOPs that can be removed if forwarding and hazard detection logic is implemented. (Values are needed at the beginning of E1 and are not available until the end of E2).

(17) [8] Here is a code sequence. This is (obviously) generic code.

```
        INSTRUCTION 1

        NOP

        INSTRUCTION 2

        INSTRUCTION 3

        INSTRUCTION 4

Label:  INSTRUCTION 5

        NOP

        INSTRUCTION 6

        INSTRUCTION 7

        NOP

        INSTRUCTION 8

        INSTRUCTION 9

        breq    CONDITION,Label         ; branch to Label if CONDITION
```

Assume there are the following dependencies in this code:

RAW between "INSTRUCTION 1" and "INSTRUCTION 2"
RAW between "INSTRUCTION 5" and "INSTRUCTION 6"
RAW between "INSTRUCTION 7" and "INSTRUCTION 8"

WAW between "INSTRUCTION 6" and "INSTRUCTION 7"
WAR between "INSTRUCTION 3" and "INSTRUCTION 4"

After you have drawn the dependency arrows, indicate what instructions you would move and where you would move them in order to schedule the code to remove the maximum number of stalls. How many NOPS are left when you are done?

(18) [10] 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 7 stage pipeline and supports imprecise interrupts.):

	1	2	3	4	5	6	7	8	9	10	11	12	13
i	IF	ID	E1	E2	M1	M2	WB	<- Interrupt detected					
i+1		IF	ID	E1	E2	M1	M2	WB	<- Instruction Squashed				
i+2			IF	ID	E1	E2	M1	M2	WB	<- Trap Handler fetched			
i+3				IF	ID	E1	E2	M1	M2	WB			

Fill out the following table if instruction i+1 experiences a fault in the E2 stage:

	1	2	3	4	5	6	7	8	9	10	11	12	13
i	IF	ID	E1	E2	M1	M2	WB						
i+1		IF	ID	E1	E2	M1	M2	WB					
i+2			IF	ID	E1	E2	M1	M2	WB				
i+3				IF	ID	E1	E2	M1	M2	WB			
i+4					IF	ID	E1	E2	M1	M2	WB		
i+5						IF	ID	E1	E2	M1	M2	WB	
i+6							IF	ID	E1	E2	M1	M2	WB

Assuming precise interrupts are being supported, what happens in this case?

	1	2	3	4	5	6	7	8	9	10	11	12	13
i	IF	ID	E1	E2	M1	M2	WB	<- M1 stage has fault					
i+1		IF	ID	E1	E2	M1	M2	WB	<- Inst Decode has Illegal Instruction				
i+2			IF	ID	E1	E2	M1	M2	WB				
i+3				IF	ID	E1	E2	M1	M2	WB			
i+4					IF	ID	E1	E2	M1	M2	WB		
i+5						IF	ID	E1	E2	M1	M2	WB	
i+6							IF	ID	E1	E2	M1	M2	WB
i+7								IF	ID	E1	E2	M1	M2

What is the maximum number of exceptions that could happen simultaneously in the above machine? Why?

Potentially useful information

F	D	E1	E2	M1	M2	W						
	F	D	E1	E2	M1	M2	W					
		F	D	E1	E2	M1	M2	W				
			F	D	E1	E2	M1	M2	W			
				F	D	E1	E2	M1	M2	W		
					F	D	E1	E2	M1	M2	W	
						F	D	E1	E2	M1	M2	W

F	D	E1	E2	M1	M2	W						
	F	D	E1	E2	M1	M2	W					
		F	D	E1	E2	M1	M2	W				
			F	D	E1	E2	M1	M2	W			
				F	D	E1	E2	M1	M2	W		
					F	D	E1	E2	M1	M2	W	
						F	D	E1	E2	M1	M2	W