166 students took the exam. The average score was 32.2; the median was 33. Scores ranged from 9 to 45. There were 71 scores between 35 and 45, 78 between 23 and 34, 15 between 12 and 22, and 2 between 9 and 11. (Were you to receive scores of 75% on each in-class exam and 46 out of 60 on the final exam, plus good grades on homework and lab, you would receive an A–; similarly, a test grade of 23 out of 45 may be projected to a B– .)

There were four versions of the test. (The version indicator appears at the bottom of the first page.)

If you think we made a mistake in grading your exam, describe the mistake in writing and hand the description with the exam to your lab t.a. or to Mike Clancy. We will regrade the entire exam.

Problem 0 (2 points)

You lost 1 point on this problem if you did any one of the following: you earned some credit on a problem and did not put your login name on the page, you did not adequately identify your lab section, or you failed to put the names of your neighbors on the exam. The reason for this apparent harshness is that exams can get misplaced or come unstapled, and we want to make sure that every page is identifiable. We also need to know where you will expect to get your exam returned. Finally, we occasionally need to verify where students were sitting in the classroom while the exam was being administered.

Problem 1 (4 points)

"Representation X", a fixed-point representation, was given in the exam. You were to find a value representable in IEEE single-precision floating-point format but not in Representation X, and vice versa. (The versions differed in the other in which they asked for the values.) Anything with more than 24 bits of precision is not representable as an IEEE floating-point value; any value whose exponent in IEEE format is greater than or equal than 15 or less than –16 is not representable in Representation X.

Answers were worth 2 points each. An off-by-one error (e.g. forgetting the hidden bit or sign bit) received a 1-point deduction; anything worse received no credit.

Problem 2 (3 points)

You were to find IEEE single-precision floating-point values a and b such that (x + a) + b ≠ x + (a + b). x was a constant that was different on each version:

version	x
A	4.0
B	16.0
C	64.0
D	256.0

There were several ways to find the desired a and b, including the following.

- Pick small values so that adding x to a causes truncation error but adding x to a+b does not. When $x = 4.0$, $a = b = 2^{-22}$ works.
- Pick a = x and b so that adding a to b causes truncation error but adding x+a to b does not. When $x = 4.0$, $b = 2^{26}$ works.
- Pick for a any value whose exponent is at least 24 greater than x's, and pick b to be –a. When $x = 4.0$, $a = 2^{26}$ works. The sum (x+a)+b is then 0, while the sum x+(a+b) = x.

You lost 1 point for an off-by-one error, or for working in base 10 (i.e. $a = m \cdot 10^e$ rather than $m \cdot 2^e$). You lost all 3 points for providing only a *description* for a and b rather than actual values, or for misunderstanding what was required.

Problem 3 (6 points)

For this problem (the same on all versions), you were to add an overflow signal to a ripple subtractor. Part a, worth 4 points, was to provide values for overflow in a truth table. Here is the answer.

P_1	Q_1	Bin_1	D_1	$Bout_1$	Overflow
0	0	0	0	0	0
0	1	0	1	1	1
1	0	0	1	0	0
1	1	0	0	0	0
0	0	1	1	1	0
0	1	1	0	1	0
1	0	1	0	0	1
1	1	1	1	1	0

Recall that overflow occurs when the computed value is not representable in the given number of bits. Overflow cannot occur when the two subtraction arguments have the same sign. You lost 2 points for each missing case, 1 point for each extra case where P and Q have different signs, and 2 points for each extra case where P and Q have the same sign.

In part b, worth 2 points, you were to provide a minimum-gate Boolean expression for the overflow signal. Bin_1 XOR $Bout_1$ was the expected answer. You lost 1 point for a correct expression for overflow with more than one gate, and 2 points for an incorrect expression for overflow.

Problem 4 (8 points)

In this problem (the same on all versions), you were to implement a traffic light. Part a was to specify the new state (R_{new}, Y_{new}, G_{new}) in terms of the old (R_{old}, Y_{old}, G_{old}) using as few gates (other than inverters) as possible. Answers:

$R_{new} = Y_{old}$	$Y_{new} = G_{old} \bullet ! Y_{old}$ $Y_{new} = R_{old}$ NOR Y_{old}	$G_{new} = ! Y_{old}$

Each expression was worth 2 points. You lost 1 point for each answer if it used more than the minimum number of two-input gates (0 for R_{new} and G_{new}, 1 for Y_{new}).

Part b, worth 2 points, was to implement the traffic light circuit. Grading was based on your expressions in part a, regardless of whether or not they were correct. You were allowed to use three-input gates in this part. Each minor error lost 1 point. Each major error (e.g. using the clock, not having any inputs to flip-flops, trying to implement reset) lost 2.

Problem 5 (8 points)

This problem (identical on all versions) asked you to implement the jal instruction in the single-cycle MIPS CPU. Part a, worth 6 points, was to add the circuitry:

- a multiplexor controlled by Jump that selects between the existing Write data signal and PC+4;
- a multiplexor controlled by Jump that selects between the existing Write register and a constant 31.

Deductions were made as follows:

> –1 for using PC+8 instead of PC+4 (Figure 4.24 depicts the *single-cycle* MIPS, not the pipelined MIPS);
>
> –1 for an incorrect or missing control signal for a multiplexor;
>
> –1 for storing the jump address in a flip-flop;
>
> –2 for inventing a new control signal, i.e. any signal derived from accessing the op code.

You needed some reference to new circuitry to get any points. Merely describing what the jal instruction does earned no credit for this part.

In part b, you were to choose the control signals that were required to be on to execute the jal instruction. These are Jump and RegWrite. (Others are 0 or "don't care".) These were worth 1 point each, for a potential total of 2. You lost 1 point for each extra signal. We did not look, but should have looked for situations where your correctly-working solution to part a required a different set of signals to be on. If your answer to part a correctly implements the jal and your answer to part b is correct based on your answer to part a, you are eligible to get some points back on a regrade.

Problem 6 (8 points)

Here, you were to explain which machine instructions in CAL-16 code had to have involved modification by the linker. Each version asked about four of the following instructions.

machine instruction	*assembly language instruction*
4444	addi
6666	ld
8888	lhi or llo
AAF2	bneg
CCCC	jr
F0FE	jmp

F0FE, the jump instruction, must have been modified since it contains an absolute address that needs fixing up. 8888, the load-immediate instruction, may have required modification; with a label argument, it produces an entry in the relocation table, but with a numeric argument, it doesn't. AAF2, the branch instruction, contains the *difference* in words between the branch's location and the target address; this difference remains unchanged when the module is moved around in memory, and thus requires no fixup by the linker. CCCC, the jump-register instruction, and 6666, the load instruction, similarly contain only register numbers and offsets that will remain unchanged during linking. 4444, the add-immediate instruction, contains only registers and a numeric constant that we wouldn't want the linker to change.

Each answer was worth 1 point, and each explanation was worth 1 point.

If you answered "may have" for a jump (F0FE) and qualified your answer with the fact that the linker may move the code to start at address 0 (so the relocated values are the same as the unrelocated values), you should have received both points. You should also have received full credit if you weren't sure whether lhi/llo (8888) can take numeric arguments, but noted that if they can, no relocation happens. If you assumed that lhi/llo can *only* take labels as arguments (i.e. you answered "must have" for lhi/llo), you got the explanation point but lost the answer point.

Explanations should have said something about the *content* of each instruction to decide about relocating, e.g. "addi doesn't get relocated because it only contains registers and an immediate constant." We also gave credit to answers that referred to output of the assembler, e.g. "the assembler did/didn't generate a relocation entry for instruction x, therefore the linker will/won't be able to modify it."

Problem 7 (6 points)

You were to rearrange the given code to remove all the no-ops. This was the same on all versions. Here was our attempt at a solution:

old version	*revision*

`lw $t1, 0($t0)`
`nop`
`addi $t2, $t1, 4`
`lw $t1, 4($t0)`
`nop`
`add $t1, $t2, $t1`
`beq $t0, $t2, foo`
`nop`
	`lw $t1, 0($t0)`
	`lw $t3, 4($t0)`
	`addi $t2, $t1, 4`
	`//` stall necessary here!
	`beq $t0, $t2, foo`
	`add $t1, $t2, $t3`

The problem with our purported solution is that beq requires $t2 to be available in stage 2 for timely determination of whether or not to branch. However, $t2's value isn't ready until the end of the ALU stage (page 378 in P&H fourth edition; page 419 in the third edition). It appears that it is not possible to remove all the no-ops. Thus, this problem will be regraded, giving full credit to a solution (like the one above) that removed *two* of the no-ops. You do not have to turn your exam in to get this problem regraded.

A common wrong answer, to be awarded 5 out of 6, was the following:

`lw $t1, 0($t0)`	`#` start first load
`lw $t1, 4($t0)`	`#` start second load
`addi $t2, $t1, 4`	`#` *invalid assumption*: first load finished so use its register `$t1`
`beq $t0, $t2, foo`	`#` second load finished; `$t1` now holds `4($t0)`
`add $t1, $t2, $t1`	`#` use `$t1` from second load

This doesn't work because the forwarding hardware inserts a stall between the second load into $t1 and the addi that uses $t1. The purpose here is to eliminate the ambiguity about which value of $t1 is actually used; an explanation appears on page 372 of P&H fourth edition (page 413 in the third edition).

A variation of this answer exchanges the beq and the add, thereby leaving one more no-op slot unfilled. This answer will receive 4 out of 6. Another common error, earning a 1-point deduction for each occurrence, was to change the destination register of one of the lw's, but not change where it is used, i.e., the addi or add.