CS61c MIDTERM EXAM: 3/17/99

D. A. Patterson

Last name _____ **First name** _____
Student ID number _____ **Login: cs61c-**_____

Please circle the last two letters of your login name.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Discussion section meeting time_____ TA's name _____
(Sorry to ask this next question, but with almost 400 students, there may be a wider range of behavior.)
The student on my left is _____ login cs61c-_____
The student on my right is _____ login cs61c-_____

You are allowed to use one 8.5 x 11 double-sided handwritten page of notes. No calculators! This booklet contains 9 numbered pages including the cover page plus photocopied pages from COD. Put all answers on these pages, please; don't hand in stray pieces of paper. The exam contains 8 substantive questions, plus question 0, the box immediately below and one extra credit question. You have three hours, so relax this exam isn't worth having a heart failure over. Good luck!

I certify that my answers to this exam are all my own work. If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

Signature _____

Question	Name	Time (minutes)	Max Points	Your points
0	Signin	0	-1 to 0	
1	Overflow	5	6	
2	Register Size	5	2	
3	Pliable Data	10	7	
4	Interrupt	10	4	
5	Network Address	10	4	
6	Asm to HLL	15	7	
7	Shifty	10	8	
8	Compile by hand	20	12	
9	Extra Credit	0	(1)	
	Total	85	50	

Signin Question (-1 point if not followed): *Fill out the front page correctly and write your login at the top of each of the pages.* **Circle your login initials so we can read them.**

Overflow Question (6 points)[5 minutes]:

Here are two bit strings in hexadecimal:

`0x7f6cbcef` and `0xff70e1ab`

Will overflow occur when you add them up if they are:

a) unsigned integers?
b) signed integers in two's complement?
c) single-precision IEEE floating point?

For each case, give reasons why or why not.

a. (2 points) [unsigned] Yes / No Why?

b. (2 points) [signed] Yes / No Why?

c. (2 points) [float] Yes / No Why?

Register Size Question (2 points)[5 minutes]:

MIPS was invented in 1985 with 32 integer registers. According to Moore's Law, named after Intel founder Gordon Moore, the number of transistors per microprocessor doubles every 1.5 years. Thus, microprocessors could have 1000 times the number of transistors in 2000 as they could in 1985. It would seem that we could easily build microprocessors with, say, 256 registers.

Select **all the reasons** why MIPS has **not** increased the number of integer registers from 32 to 256.

a. There is no need for more than 32 registers, as compilers have difficulty using the 32 registers in the MIPS architecture now.

b. Due to the importance of binary compatibility, new MIPS processors must be able to execute old MIPS instructions, and it would be very hard to find room for 256 registers in the original MIPS instruction format.

c. 32 registers are much smaller than 256 registers, and since smaller is faster, the 32 registers makes it easier to build fast microprocessors than if there were 256 registers.

d. Moore's Law applies to Intel microprocessors, not MIPS microprocessors, hence the hypothesis is false. The MIPS chip would be too expensive if it had 256 registers.

Pliable Data Question (7 points)[10 minutes]:

Given the bit pattern:
10000000 00000000 00000000 00000000
Answer the following questions.

Note: here are some numbers that may prove useful:
2^{32} = 4,294,967,296
2^{31} = 2,147,483,648
2^{30} = 1,073,741,824

a. (2 points) Can this bit pattern represent a two's complement integer? Yes / No

If so, what does it represent? _____
If it cannot represent a two's complement integer, why not?

b. (2 points) Can this bit pattern represent an unsigned integer? Yes / No

If so, what does it represent? _____
If it cannot represent an unsigned integer, why not?

c. (3 points) Can this bit pattern represent a MIPS instruction? Yes / No

If so, what does it represent? _____
If so, which register (using register numbers $0, $1, $2, ..., $31) or memory location (using the notation Memory[register + number]) **has a new value** as a result of this instruction?_____

_____
If it cannot represent a MIPS instruction, why not?

Interrupt Routine Question (4 points)[10 minutes]:

Some printings of Appendix A have this code for a simple (not re-entrant) interrupt routine which prints an exception error message if the exception code (bits 5 to 2 of the Cause register) is 8 or greater:

```
.ktext 0x80000080
      sw    $a0, save0
      sw    $a1, save1
      mfc0  $k0, $13    # Move Cause into $k0
      mfc0  $k1, $14    # Move EPC into $k1
      slti  $v0, $k0,0x44 # Ignore interrupts (exception in bits 5 to 2 is < 8)
      bne   $v0, $0, done
      move  $a0, $k0    # Move Cause into $a0
      move  $a1, $k1    # Move EPC into $a1
      jal   print_excp  # Print exception error message
done: lw    $a0, save0
      lw    $a1, save1
      addiu $k1, $k1, 4 # Do not re-execute faulting instruction
      rfe               # Restore interrupt state
      jr    $k1

.kdata
save0:      .word 0
save1:      .word 0
```

There is at least one bug in the code from the book. From the options below, select **all that are bugs**: (if none of these are bugs, don't pick any of a to e)

a. It uses $k0 and $k1 without saving and restoring them, as it did for $a0 and $a1.

b. The hex number 0x44 in the `slt` instruction (to test if exception code field is less than 8).

c. The interrupt routine uses `$ra ($31)` without saving and restoring it, as was done for $a0 and $a1.

d. `addiu $k1,$k1,4` (the address in EPC already points to the instruction following the faulting instruction, so this instruction is unnecessary)

e. `jr $k1 (jr` must use register `$31)`

Network Address Question (4 points)[10 minutes]:

The networking community apparently is afraid of hexadecimal notation. As a result, they invented a new notation to represent a 32-bit number. Their notation is:

a.b.c.d

where a, b, c, and d are all decimal numbers between 0 and 255. The unsigned 32-bit number represented by the notation is:

$$(a \times 2^{24}) + (b \times 2^{16}) + (c \times 2^{8}) + d$$

Thus

255.0.255.0

represents the bit pattern

$$11111111\ 00000000\ 11111111\ 00000000_{two}$$

This notation is the way they number "Internet Protocol" (IP) addresses.

a. (1 point) What is the 32-bit hexadecimal number equivalent to this IP address 128.32.39.124?

0x_____

b. (3 points) One question for Internet Protocols is whether an IP address is local (nearby) or not local (you must go across the Internet). To specify locality, users must specify a mask, also using this network notation. To answer the question of whether a new address is local or not, the test for locality is

((local IP address) AND (mask)) == ((new IP address) AND (mask))

If they are equal, then the new address is local.
(In case you are interested, the official name of the mask is *Subnet Mask*.)

Suppose the following network addresses are **local**:

128.32.39.120
128.32.39.121
128.32.39.122
128.32.39.123
128.32.39.124
128.32.39.125
128.32.39.126
128.32.39.127

All other network addresses are **not local**. For example, the IP address 128.32.39.33 is **not local**.

What mask is needed to make these eight IP addresses local and all others not local?

Use the IP address notation: _____ . _____ . _____ . _____

Assembly to HLL Question (7 points)[15 minutes]:

Below are several assembly language instructions or operands, including some from MIPS:

1. Add instruction (`add`, `addu` in MIPS)
2. Address
3. AND instruction (`and` in MIPS)
4. Byte data transfer (`lbu`, `sb` in MIPS)
5. Comparison instructions (`slt`, `sltu` in MIPS)
6. Conditional branches (`beq`, `bne` in MIPS)
7. Immediates
8. Load (`lw`, `lb`, `lbu` in MIPS)
9. Memory
10. MIPS convention for registers `$a0, $a1, $a2, $a3`
11. MIPS convention for registers `$v0, $v1`
12. MIPS instructions: `add.d`, `sub.d`, `mult.d`, `div.d`
13. MIPS instructions: `add.s`, `sub.s`, `mult.s`, `div.s`
14. MIPS jump and link instruction `jal`
15. MIPS jump register instruction `jr`
16. Multiply instruction (`mul`, `mulu` in MIPS)
17. OR instruction (`or` in MIPS)
18. Registers
19. Shift instructions (`sll`, `srl` in MIPS)
20. Subtract instruction (`sub`, `subu` in MIPS)
21. Store (`sw`, `sb` in MIPS)
22. Unconditional Branch (`j` in MIPS)

Associate the high-level language operators or concept below (listed in alphabetical order a to s) with the **most appropriate** assembly language instruction or operand (1 to 22 above) by filling in the blank with the number. **You can use the same assembly language instruction or operand more than once.** In 3 cases there is room for 2 associations. If there is **no such correlation**, write **NONE**.

_____a. Arguments for Functions or Procedures
_____b. Arrays
_____c. C & operator (binary version: e.g. `a & b`)
_____d. C & operator (unary version: e.g. `&a`)
_____e. C * operator (binary version: e.g. `a * b`)
_____ _____f. C * operator (unary version: e.g. `*a`)
_____g. C <<, >> operators
_____h. C goto statement
_____i. C if-then statement
_____j. C pointers
_____k. C relations <, >=
_____l. C relations ==, !=
_____ _____m. C return statement
_____n. C while statement
_____o. Character variable (`char` in C)
_____p. Constants
_____q. Declaration of variable type
_____ _____r. Function Calls (e.g. `f(a,b)`)
_____s. Operations on variables declared as `float` in C

Shifty Question (8 points)[10 minutes]:

Imagine that there is a MIPS instruction:

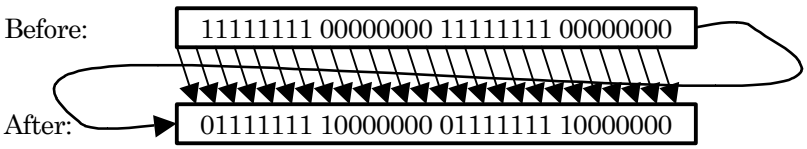
```
sllv    rd, rt, rs
```

that causes the bits in register rt to be left-shifted by the amount indicated in register rs, and the result is to be put into register rd. Likewise, imagine that there are similar instructions called "srlv" and "srav". Each corresponds to one of the shift instructions you are used to, except that the shift amount is specified in a register rather than as an immediate.

Using these new variable shift instructions, write a MIPS function called "*RightRotate*." It accepts two arguments:

$a0 contains the bit string to be rotated.
$a1 contains the number of bits by which to rotate the string. (Assume that $a1 contains a positive integer between 0 and 31: 0 <= $a1 <= 31.)

For example, rotating to the right **by a single bit** is shown in the diagram below:



Your function should adhere to all function conventions and return properly with the rotated result in the proper register. **Use comments to document the code.**

(Hint: The optimal answer takes just six instructions, but answers with more instructions can still get full credit.)

Label	Instruction	Comment

```
RightRotate:
```

Compile-by-Hand Question (12 points)[20 minutes]:

One way of doing error detection on the Internet is to use a checksum. The idea is to add up all the words that are transmitted and then transmit the result of that sum, and then check the data and the sum when it arrives. We actually return the negative of the sum minus one (which inverts every bit and is called the *one's complement*, in case you are interested.) The checksum is inverted to make sure that sending all 0s is not considered a valid message, which is a common error condition. Below is a C program which accomplishes such a task.

Your job is to translate the C program into a MIPS program. Since it's an Internet program, we want it to be as short and fast as possible. So **minimize the amount of stack usage and amount of instructions** as much as you are able to; that is, only save the minimum necessary registers.

Write the MIPS program following the proper MIPS register conventions, and **use comments to document the code.** (You can use the back or this page or part of the next page for space if necessary)

```
int checksum (int start, int frame[]) {
      /* frame[] is the array that contains pointers to data we want to send.
      start is guaranteed to be less than 512.*/

  int sum, result, x;
  int max_length = 512;
  result = 0;
  sum = 0;

  for (x = start; x < max_length; x++) {
    result = getdata(frame[x]); /* don't write getdata, just call it */
    sum = sum + result;
  }

  return ((-sum) - 1);
}
```

Extra Credit Question (1 bonus point)[0 minutes]:

Given: a semicircle of radius 1; take two points: one with coordinates (0,t)-- t up the Y axis, and the other one t up along the semicircle (starting at (0,0)). Connect these two points with a line.

Question: Find x(t) - the intersect of this line with the x-axis as t goes to zero.