University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2016 Instructors: Bernhard Boser, Randy Katz 2016-09-27

☹CS61C MIDTERM 1 ☺

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	
First Name	
Student ID Number	
CS61C Login	`cs61c-`
*The name of your **SECTION** TA and time*	
Name of the person to your LEFT	
Name of the person to your RIGHT	
*All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. **(please sign)***	

Instructions (Read Me!)

- This booklet contains 8 numbered pages including the cover page.
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 80 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use one handwritten 8.5"x11" page (front and back) crib sheet in addition to the MIPS Green Card, which we will provide.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided.
- Points are assigned by the approximate time to answer the question, 1 point = 1 minute. Pace yourself, and at least attempt every question for partial credit.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Total**
Points Possible	5	10	10	15	15	5	60

Q1: Number Representation (5 points)

1. Given the binary number `0b11111111`:

   If unsigned, then what is the number in decimal?

   If two's complement, then what is the number in decimal?

2. What is the range of integers represented by a n-bit binary number? Your answers should include expressions that use 2^n.

   If unsigned, smallest: largest:

   If two's complement, smallest: largest:

   How many unique integers can be represented in each case?

   Unsigned: Two's Complement:

3. What is the result of the two's complement negation applied to the most negative binary number that can be represented in an n-bit two's complement form?

   Answer:

4. Consider we have a base 32 number, with each number position represented by the numerals 0 through 9 plus the letters A (10), B (11), C(12), D(13), E (14), F (15), G (16), H (17), I (18), J (19), K (20), L (21), M (22), N (23), O (24), P (25), Q (26), R (27), S (28), T (29), U (30), and V (31).

   Convert FUN_{32} to:

   Binary:

   Hexadecimal:

   Decimal:

SID: ________________

Q2: Reverse Engineering (10 points)

The following MIPS code snippet implements a common coding design pattern in C. Your task is to reverse engineer the code to determine what this pattern is. Do not be concerned about the MIPS calling conventions, such as saving registers on the stack.

Assume the following:
$s0 holds the C integer variable b;
$s1 holds the C integer variable i;
$s2 holds the C integer constant 10;
$s3 holds the C pointer to an array of integers a;

The code is as follows, with space for comments following the # sign at the right:

```
     add  $s0, $zero, $zero   # b = 0;

     add  $s1, $zero, $zero   # i = 0;

     addi $s2, $zero, 10      # $s2 = const 10;

X:   slt  $t0, $s1, $s2       #

     bne  $t0, $zero, Y       #

     sll  $t1, $s1, 2         #

     add  $t2, $s3, $t1       #

     sw   $s1, 0($t2)         #

     add  $s0, $s0, $s1       #

     addi $s1, $s1, 1         #

     j X                      #

Y:                            #
```

Partial credit will be determined, in part, by the quality of your line-by-line comments. Please provide the comments in pseudocode format. The question continues on the next page.

What is the equivalent C code that is implemented by this MIPS code? Come up with the C equivalent with the fewest possible lines of code. You might not need all the lines.

SID: ________________

Q3: Number Pushing and Popping (10 points)

Your task is to implement a simple stack adding machine that uses a stack data structure (this is independent of the stack for calling/returning from subroutines). For example, Push 2, Push 3, PopAdd yields 5 in the top of the stack. Following this with Push 1, PopAdd would yield 6. Fill in the code for functions **push** and **popadd** so they meet the specifications stated in the comments. Do not make other code modifications. Calls to **malloc** always return a valid address. You may not need all the lines for your code solution, but do include comments for partial credit consideration.

```
/* Each item on the stack is represented
   by a pointer to the previous element
   (NULL if none) and its value. */
typedef struct stack_el {
    struct stack_el *prev;
    double val;
} stack_el;

/* PUSH: Push new value to top of stack. Return
   pointer to new top of stack. */
stack_el* push(stack_el *top_of_stack, double v) {




}

/* POPADD: Pop top stack element and add its value
   to the new top's value. Return new top of stack.
   Free no longer used memory. Do not change
   the stack if it has fewer than 2 elements. */
stack_el* popadd(stack_el *top_of_stack) {






}
```

SID: ________________

Q4: Branches are for N00Bs (15 points)

Consider the discrete-value function $f(x)$, defined on the integers $\{-3, -2, -1, 0, 1, 2, 3\}$:

$f(-3) = 6$ $f(-2) = 61$ $f(-1) = 17$ $f(0) = -38$
$f(1) = 19$ $f(2) = 42$ $f(3) = 5$

Your task is to implement this function in MIPS assembly language (including pseudo-instructions), but you may NOT use ANY conditional branch instructions (that is, no *beq, bne, blt, bgt, ble,* or *bge*). To assist in accomplishing this, we have stored the return values in contiguous words in the program's data segment.

You may assume that the input x is always one of the integers for which the function is defined (between -3 and 3). Assume x is stored in `$a0` and $f(x)$ is to be returned in `$v0`. Comment your code for possible partial credit. You may not need all the lines given.

```
.data
output: .word 6 61 17 -38 19 42 5

.text
f:
```

Alternate Solution

```
.text
f:
```

SID: ________________

Q5: *D*IPS *ISA* (15 points)

Archeologists uncover an ancient binary machine based on 4-bit entities (called nibbles rather than bytes) as its fundamental building block. Reverse engineering this machine, engineers discover that it has words, registers, and instructions consisting of **six nibbles *(a 24 bit word machine)***. The machine is **nibble addressed**, but instructions and data words always begin at a nibble address that is a **multiple of 6** (word 0 is at nibble 0, word 1 at nibble 6, word 2 at nibble 12, and so on). The machine happens to have **10 by 24-bit registers**. The discoverers name the machine "DIPS."

You are asked to reformat the MIPS 32-bit ISA for the smaller DIPS 24-bit instruction word, which looks remarkably similar (but not identical) to MIPS in terms of its instruction formats and fields. It is still nibble addressed with 24-bit words and instructions, but you are not otherwise constrained.

1) In the boxes below, specify the sizes of the fields for R- and I-type instructions to best utilize the constrained **24-bit** instructions in the DIPS ISA.

opcode	rs	rt	rd	shamt	funct
6					

opcode	rs	rt	imm
6			

Given your encoding for the R and I instructions above:

2) What is the maximum number of registers you can address in the width of your r fields?

3) What is the maximum number of distinct operations you can encode in the top instruction format?

4) If the PC points to the nibble address 1566_{10}, what is the highest (largest) nibble address **in decimal** you can branch to? (NOTE: 1566 is a multiple of 6 and thus a proper DIPS word address.)

5) Translate the following line of DIPS 24-bit machine code into MIPS assembly language, using your encoding from above. Use $register_number as your register names (e.g., $0, $1, …), and assume the same opcodes as in the MIPS ISA. **Remember instructions are only 24 bits (6 nibbles/hex digits)!**

   0x8C2408

SID: ________________

Q6: Mishmash, Hodgepodge, Potpourri (5 points)

The following are some multiple choice questions about CALL. Clearly circle the correct answer:

A system program that combines separately compiled modules of a program into a form suitable for execution is __________

A. Assembler
B. Loader
C. Linker
D. None of the above

Which flag would you put in a compilation command to include debugging information?

A. -o
B. -d
C. -g
D. --debug

At the end of the compiling stage, the symbol table contains the ___ of each symbol.

A. relative address
B. absolute address
C. the stack segment beginning address
D. the global segment beginning address

`beq` and `bne` instructions produce ____ and they _____.

A. PC-relative addressing, never relocate
B. PC-relative addressing, always relocate
C. Absolute addressing, never relocate
D. Absolute addressing, always relocate

`j` and `jal` instructions add symbols and ___ to ___.

A. instruction addresses, the symbol table
B. symbol addresses, the symbol table
C. instruction addresses, the relocation table
D. symbol addresses, the relocation table