University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2016   Instructors: Vladimir Stojanovic, Nicholas Weaver   2016-02-25

☹ 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 (please circle)*	Alex | Chris | Howard | Jack | Jason | Rebecca | Stephan | William
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 9 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 110 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) of notes in addition to the provided green sheet.
- 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.

	Q1	Q2	Q3	Q4	Q5	Q6	EC	Total
Points Possible	10	10	20	25	20	20	??	105

Corrections

Q3:
Assume arr -> $a0
        n -> $a1
   min_val -> $a2

The loop should be:
```
loop:
beq _____
lw $t1, _____
slt $t2, _____, ____
_____
_____
_____
j loop
```

Q6.
a.== and != are considered conditionals

Q1: In*structor*s keep their students *aligned* (10 points)

Consider the C code below. Assume `ints` and pointers are 4 bytes in size. Remember that C structs are densely packed, meaning their elements are contiguous in memory, and that structs **may include padding** at the end for alignment. Each struct is located at a memory address that is a **multiple of the size of its largest element**.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    char *name;
    unsigned int id;
    char grade;
} Student;

int main(void) {
    Student *students = malloc(2 * sizeof(Student));
    students[0].name = "Oski Bear";
    students[0].id = 12345;
    students[0].grade = 'A';
    students[1].name = "Ken Thompson";
    students[1].id = 5678;
    students[1].grade = 'A';

    printf("students: %p\n", students);
    printf("Address of students[0]: %p\n", &(students[0]));
    printf("Address of students[0].id: %p\n", &(students[0].id));

    printf("students + 1: %p\n", students + 1);
    printf("Address of students[1].grade: %p\n",
            &(students[1].grade));

    return 0;
}
```

a) Fill in the blanks in the program's output below. Assume that the region of memory on the heap allocated by the call to `malloc` starts at address `0x1000`. Also, remember that C will print pointer values and memory addresses in **hexadecimal notation**.

```
students: 0x1000
Address of students[0]: _____
Address of students[0].id: _____
students + 1: _____
Address of students[1].grade: _____
```

b) **True | False** The address of the `students` pointer is less than its value,
i.e. `&students < students`

Q2: This *tree* question needs *acorny* pun (10 points)

Write a function to sum up the values and free a tree of arbitrary size constructed using the `tree_node struct` as defined. Each node can have an arbitrary number of children. Assume that there will always be a valid pointer in the location of children.

```c
struct tree_node {
    int value;
    struct tree_node ** children;
    int num_children;
}


int sum_and_free_tree( struct tree_node * root ) {
    int i, sum;
    if ( root == NULL ) { // This is equivalent to being stumped
        return 0;
    }
    sum = _____;
    for ( i = _____;
             _____;
             _____ ) {
        _____;
    }
    free( _____ );
    free( _____ );
    return sum;
}
```

Q3: A *filter* in the blank question (20 points)

Convert the `filter_array` function, which counts the number of elements greater than `min_val` in the array and returns an integer, to MIPS assembly. You may not need all of the lines, but you should try to use as few lines as possible.

```
int filter_array(int* arr, size_t n, int min_val) {

    int count = 0, i;
    for(i=0; i < n; i++){
        if(arr[i] > min_val)
            count++;
    }
    return count;
}
```

Example:
```
int* p = (int*) malloc(sizeof(int)*3)
p[0] = 1
p[1] = 2
p[2] = 3
printf("%d\n", filter_array(p,3,1))
```

Output: 2

```
filter_array:
  addiu $sp, $sp, ____

  _____

  _____

  _____

  addiu    $s0, $zero, 0   # We'll store the count in $s0
  addiu    $s1, $zero, 0   # We'll store i in $s1
  addiu    $t0, $a0, 0
loop:
  beq _____
  lw $t1, _____
  slt $t2, _____, ____

  _____

  _____

  _____

  j loop
done:

  _____

  _____

  _____

  _____

  addiu $sp, $sp, ____
  jr $ra
```

Q4: Have you seen this *MIPS*tery before? (25 points)

The following is a recursive function that saves its arguments and return address on the stack as it executes.

```
        mystery:
0x4000    bne $a0, $0, recurse
0x4004    li $v0, 1
0x4008    jr $ra
        recurse:
0x400C    addiu $sp, $sp, -8
0x4010    sw $ra, 0($sp)
0x4014    sw $a0, 4($sp)
0x4018    addiu $a0, $a0, -1
0x401C    jal mystery
0x4020    lw $ra, 0($sp)
0x4024    lw $a0, 4($sp)
0x4028    addiu $sp, $sp, 8
0x402C    mult $a0, $v0
0x4030    mflo $v0
0x4034    jr $ra
```

a) If the function is called with the argument $a0 set to 5, what values will be in registers $a0 and $ra before you return from the base case?

b) What does the stack look like at the beginning of the base case?
   Write your answers in the table below. Assume that, when the function is first called, $a0 is set to 5 and $ra is set to 0x1000. Remember that the stack starts at the top and expands downward. Each box is **one word**, and you only need to fill in the box with the hexadecimal value.

c) In a sentence, what does this function do? Assume that $a0 is unsigned.

Q5: *MIPS Instructions Per Second* (20 points)

Assume $a0 contains some positive integer, and $a1 contains the address to the start of an integer array. The numbers on the left are line numbers; they aren't related to the address of each line, and the address of the instruction on line 0 is 0x00000000. Consider the following MIPS code and its instruction format representation:

```
0                 add $t0 $a0 $0        <=>        0x00804020
1                 add $t1 $a1 $0        <=>        _____
2     LabelA:     add $t2 $0 $0         <=>        0x00005020
3     LabelB:     beq $t0 $0 END        <=>        _____
4     LabelC:     addi $t0 $t0 -1       <=>        _____
5                 lw $t3 0($t1)         <=>        _____
6                 _____          <=>        0x014b5020
7                 addi $t1 $t1 4        <=>        0x21290004
8                 _____          <=>        0x08000003
9     END:        add $v0 $t2 $0        <=>        0x01201020
```

a) Convert the following lines to their machine code representation. Write your representation in binary. Each of the boxes is divided into 8 sections of 4 bits each; please format your answer accordingly. Line 1 is given as an example.

1) | 0000 | 0000 | 1010 | 0000 | 0100 | 1000 | 0010 | 0000 |

3) | | | | | | | | |

4) | | | | | | | | |

5) | | | | | | | | |

b) Convert the following lines from machine code into their MIPS instruction.

6) _____

8) _____

c) In a sentence or two, describe what this function does.

Q6: Mishmash, Hodgepodge, Potpourri (20 points)

a) Implement a function that only uses bitwise operations to return true if the most significant byte of a 16-bit unsigned integer is different from its least significant byte. For example, this function returns false for `0x1A1A` but true for `0x1A1B`. No conditionals or loops are permitted.

```
int f( uint16_t n ) {

    return _____;

}
```

b) What is the output of the following snippet of code? %d prints a signed integer and %u prints an unsigned integer.

```
int main(){
    int8_t x = -1;
    uint8_t y = 255;

    printf("===begin===\n");
    printf("i. %u\n", (uint8_t) x--);
    printf("ii. %u\n", (uint8_t) x);
    printf("iii. %u\n", ++y);
    printf("iv. %u\n", y);

     printf("===break===\n");

    uint8_t z = 255;
     printf("v. %d\n", (int8_t) z);
     printf("vi. %u\n", (uint8_t) z);

    z -= 256;
    printf("vii. %d\n", z);
    printf("===end===\n");
}
```

Fill in your answer here:
===begin===

i.

ii.

iii.

iv.

===break===

v.

vi.

vii.

c) Fill in the blank cells with the characteristics of each table:

	Symbol Table	Relocation Table
What phase(s) is it written to? Fill in with one of the CALL stages.		
What phase(s) is it read from? Fill in with one of the CALL stages.		
Why would you save a label into this table?		

d) The following questions refer to the job of the loader. Circle true or false.

    i.   **True I False** Creates an address space for the program

    ii. **True I False** Reads the object file to determine the size of text and data segments

    iii. **True I False** Initializes all machine registers to 0

    iv. **True I False** Copies the instructions from the executable file into the stack

    v. **True I False** Is currently implemented as part of the OS

Extra Credit (?? Points): What does the following line of code do in C?

$$C++ + C++$$