### 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.

### Table

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>EC</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>??</td>
<td>105</td>
</tr>
</tbody>
</table>

| Points Possible | 10 | 10 | 20 | 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: Instructors 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]: 0x1000
   Address of students[0].id: 0x1004
   students + 1: 0x100C
   Address of students[1].grade: 0x1014

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 = root->value;
    for ( i = 0; i < root->num_children; i++ ) {
        sum += sum_and_free_tree((root->children)[i]);
    }
    free( root->children );
    free( root );
    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.

```c
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;
}
```

```mips
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
```

```mips
filter_array:
    addiu $sp, $sp, -8
    sw $s0, 0($sp)
    sw $s1, 4($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 $s1, $a1, done
    lw $t1, 0($t0)
    slt $t2, $a2, $t1
    addu $s0, $s0, $t2
    addiu $s1, $s1, 1
    addiu $t0, $t0, 4
    j loop
done:
    addiu $v0, $s0, 0
    lw $s0, 0($sp)
    lw $s1, 4($sp)
    addiu $sp, $sp, 8
    jr $ra
```
Q4: Have you seen this MIPStery before? (20 points)

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

<table>
<thead>
<tr>
<th>address</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4000</td>
<td>bne $a0, $0, recurse</td>
</tr>
<tr>
<td>0x4004</td>
<td>li $v0, 1</td>
</tr>
<tr>
<td>0x4008</td>
<td>jr $ra</td>
</tr>
<tr>
<td>0x400C</td>
<td>recurse:</td>
</tr>
<tr>
<td>0x4010</td>
<td>addiu $sp, $sp, -8</td>
</tr>
<tr>
<td>0x4014</td>
<td>sw $ra, 0($sp)</td>
</tr>
<tr>
<td>0x4018</td>
<td>sw $a0, 4($sp)</td>
</tr>
<tr>
<td>0x401C</td>
<td>addiu $a0, $a0, -1</td>
</tr>
<tr>
<td>0x4020</td>
<td>jal mystery</td>
</tr>
<tr>
<td>0x4024</td>
<td>lw $ra, 0($sp)</td>
</tr>
<tr>
<td>0x4028</td>
<td>lw $a0, 4($sp)</td>
</tr>
<tr>
<td>0x402C</td>
<td>addiu $sp, $sp, 8</td>
</tr>
<tr>
<td>0x4030</td>
<td>mult $a0, $v0</td>
</tr>
<tr>
<td>0x4034</td>
<td>mflo $v0</td>
</tr>
<tr>
<td></td>
<td>jr $ra</td>
</tr>
</tbody>
</table>

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?

   $a0 = 0
   $ra = 0x4020
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.

<table>
<thead>
<tr>
<th>0x5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
</tr>
<tr>
<td>0x4</td>
</tr>
<tr>
<td>0x4020</td>
</tr>
<tr>
<td>0x3</td>
</tr>
<tr>
<td>0x4020</td>
</tr>
<tr>
<td>0x2</td>
</tr>
<tr>
<td>0x4020</td>
</tr>
<tr>
<td>0x1</td>
</tr>
<tr>
<td>0x4020</td>
</tr>
</tbody>
</table>


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

Computes the factorial of $a0$
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.

| 0000 | 0000 | 1010 | 0000 | 0100 | 1000 | 0010 | 0000 |
| 0001 | 0001 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 |
| 0010 | 0001 | 0000 | 1000 | 1111 | 1111 | 1111 | 1111 |
| 1000 | 1101 | 0010 | 1011 | 0000 | 0000 | 0000 | 0000 |

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

6)  add $t2 $t2 $t3

8)  j LabelB

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

Adds together the first $a0$ elements of the array at $a1$ and returns it into $v0.
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 0xA1A but true for 0xA1B. No conditionals or loops are permitted.

```c
int f( uint16_t n ) {
    return ((n >> 8) ^ n) & 0xFF;
}
```

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

```c
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. 255

ii. 254

iii. 0

iv. 0

===break===

v. -1

vi. 255

vii. 255

===end===

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

<table>
<thead>
<tr>
<th>What phase(s) is it written to?</th>
<th>Symbol Table</th>
<th>Relocation Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fill in with one of the CALL stages.</td>
<td>Assembler</td>
<td>Assembler</td>
</tr>
<tr>
<td>What phase(s) is it read from?</td>
<td>Linker</td>
<td>Linker</td>
</tr>
<tr>
<td>Fill in with one of the CALL stages.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Why would you save a label into this table?</td>
<td>Labels that can be used by other files</td>
<td>Labels needed by this file</td>
</tr>
</tbody>
</table>

d) The following questions refer to the job of the loader. Circle true or false.
i. True | False Creates an address space for the program
ii. True | False Reads the object file to determine the size of text and data segments
iii. True | False Initializes all machine registers to 0
iv. True | False Copies the instructions from the executable file into the stack
v. True | False Is currently implemented as part of the OS

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

C++ + C++

Undefined in the specs.