Name:____

Problem 1: (25 points) Short Answers

a) **(10 points)** Executing a program

Following is a list of tasks that must be done to run a program written in C. Sort the tasks into the order in which they are executed and for each step, indicate which tool performs the step (where the tool is the compiler, assembler, linker or loader).

a. Load text and data segments in to memory.
b. Store resulting code in an executable file
c. Combine text and data segments from multiple input files.
d. Store resulting code in a.s file
e. Translate C code to equivalent assembly language code.
f. Store resulting code in a.o
g. Replace pseudoinstructions with one or more real instructions.
h. Compute offsets for all branch instructions.
i. Compute absolute address for all relocation table entries; rewrite instruction(s) to hold the new address.

Write your answers here:

1. Task: ______________________________Tool:______________________________

2. Task: ______________________________Tool:______________________________

3. Task: ______________________________Tool:______________________________

4. Task: ______________________________Tool:______________________________

5. Task: ______________________________Tool:______________________________

6. Task: ______________________________Tool:______________________________

7. Task: ______________________________Tool:______________________________

8. Task: ______________________________Tool:______________________________

9. Task: ______________________________Tool:______________________________

Name:____

b) **(5 points)**

```
char arr[] = "foo";
void foo(int arg){
   char *str = (char *) malloc(10*sizeof(char));
   char *ptr = arr;
}
```

In which memory sections (code, static, heap or stack) do the following variables reside?

arg:	arr:
*str:	ptr:

What is the value of the argument passed to malloc? When is this value determined (compiler, assembler, linker, loader, runtime)?

You look into the text portion of an executable file (a.out) and see the most significant six bits of an instruction are 0b000010. As a result of executing this instruction…

c) **(5 points)** What is the most that your program counter could change (be exact)?

d) **(5 points)** What is the least?

e) (2 points) Extra Credit: Who was the first woman programmer?

f) (2 points each) Extra Credit: Name one of the machines that the man who had a pathological hatred for organ grinders developed (though never completed).

g) (2 points each) Extra Credit: What does the acronym NP stand for?

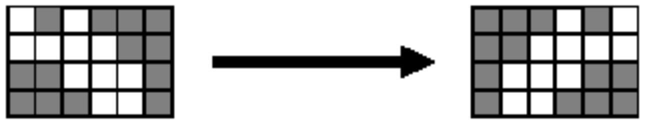
h) (PhD in Computer Science, $1,000,000 and Turing Award) Extra Credit: Prove or disprove that P = NP.

Name:____

Name:____

Problem 2: (30 points) Image Manipulation

The goal of this problem is to write a MIPS function *flipimage* that flips an image horizontally. For example, a simple image is shown on the left, and its flip is shown on the right.



A picture is composed of individual dots, or pixels, each of which will be represented by a single byte. The entire two-dimensional image is then stored in memory row by row. For example, we can store a 4 x 6 picture in memory starting at address 1000 as follows:

- The first row (consisting of 6 pixels) is stored at addresses 1000-1005.
- The second row is stored at addresses 1006-1011.
- The third row is stored at 1012-1017
- The last row is stored at addresses 1018-1023.

a. **(15 points)** Write a MIPS function fliprow to flip a single row of pixels. The function has two arguments, passed in $a0 and $a1: the address of the row and the number of pixels in that row. There is no return value. Be sure to follow all MIPS calling conventions.

Name:____

b. **(15 points)** Using the fliprow function, you should now be able to write flipimage. The arguments will be:

- The memory address where the image is stored ($a0)
- The number of columns in the image ($a1)
- The number of rows in the image ($a2)

Again, there is no return value, and you should follow normal MIPS calling conventions.

Name:____

Problem 3: (20 points) Swap

```
void swap(char **one, char **two) {
     char temp = **one;
     **one = **two;
     **two = temp;
}
```

(20 points) Write MIPS assembly code for the `swap` C function.

1. You must follow all register conventions and procedure calling conventions
2. You must write comments. Code that is not adequately commented will be penalized.

Name:____

Problem 4: (40 points) MIPS Reverse Compilation

Consider the following MIPS assembly code:

```
venture:  lb $t0, 0($a0)
          beq $t0, $a1, brock
          beq $t0, $0, monarch
          addi $a0, $a0, 1
          j venture
brock:    addi $v0, $a0, 0
          jr $ra
monarch:  add $v0, $0, $0
          jr $ra
```

a) **(15 points)** Translate the above `venture` function into a high-level language like C or Java. Your function header should list the types of any arguments and return values. Also, your code should be as concise as possible, without any gotos. We will not deduct points for syntax errors unless they are significant enough to alter the meaning of your code.

b) **(5 points)** Describe briefly, in English, what this function does.

Name:____

c) **(20 points)** Convert the following instructions from the code above into 32 bit hexadecimal number. Assume that the address of the first instruction (`lb $t0, 0($a0))` is located at address `0x00400028.`

(5 points) `lb $t0, 0($a0)`

(5 points) `beq $t0, $0, monarch`

(5 points) `j venture`

(5 points) `addi $v0, $a0, 0`

Problem 5: (45 points) Data Structures

A list is a series of elements that are sequentially connected to each other. A list node is a structure that represents a single element of a list. A list node is formally defined as follows:

```
struct ListNode {
      int         data;  // this is the data contained in this element
      ListNode*   next;  // this is a pointer to the next element in the list
      ListNode*   prev;  // this is a pointer to the previous element in the list
};
```

A list is simply a series of these nodes connected using pointers.

The C function `reverse` that completely reverses the ordering of the elements in the list is shown below. The function takes one argument: a pointer to the first `ListNode` object of the list. It returns a pointer to the new first element of the list (which was previously the last element)

```
ListNode * reverse(ListNode *aNode)
{
      ListNode * tempNode;

      if(aNode == NULL)
            {return NULL;}        //check for incorrect input

      tempNode = aNode->next;     //save the next pointer
      aNode->next = aNode->prev;  //switch the prev and next ptrs
      aNode->prev = tempNode;

      if(aNode->prev == NULL)     //stop if end of list
            { return aNode; }     //    has been reached
      else
            { return Reverse(aNode->prev); } //otherwise continue
                                             //    recursively
}
```

a) **(5 points)** How many bytes is one instance of a ListNode struct?

Name:____

b) **(20 points)** Write a *recursive* MIPS assembly code for this C function. You must follow register conventions as well as standard procedure calling conventions for full credit on this question. In other words, make no assumptions about the calling procedure.

3. Solutions that are not recursive will not get any credit
4. You must follow all register conventions and procedure calling conventions
5. No pseudoinstructions. TAL (true assembly language) only. This means you can only use instructions from the green card.
6. You must write comments. Code that is not adequately commented will be penalized.

Name:____

c) **(20 points)** Translate that following memory layout to the pictorial description of a list. The head of the list is `0x472AF014`.

Address	Value
0xFFFFFFFF	. . .
	0x00000000
	0x472AF014
	0x472AF014
	0x123ABC00
0xE123014C	0x00000004
	. . .
	0x041ABC28
	0x00123AF0
	0x00000000
	0x00000000
0x80042AB0	0x00000003
	. . .
	0xE123014C
	0x041ABC28
	0x00123AF0
	0xE123014C
0x472AF014	0x00000003
	. . .

Address	Value
	. . .
	0xE123014C
	0x00123AF0
	0xE123014C
	0x00123AFO
0x123ABC00	0x00000001
	. . .
	0x472AF014
	0x80042AB0
	0x00000000
	0x00000000
0x041ABC28	0x00000004
	. . .
	0x80042AB0
	0x00000000
	0x123ABC00
	0x472AF014
0x00123AF0	0x00000002
	. . .