CS 3410 Homework 4
Note: No slip days are allowed for this homework. This homework *must* be turned in by the deadline.

1. (a) List all the steps, in order, that a *caller* must go through when calling a subroutine.
   (b) List all the steps, in order, that a *callee* must go through when *finishing* a subroutine and returning to a caller.
   (c) In what situation(s) could $fp point below $sp?
   (d) What is dynamic linking? Describe one advantage and one disadvantage.

2. Consider the following MIPS subroutine:

```
BINS:
ADDI $sp, $sp, -48
SW $ra, 44($sp)
SW $fp, 40($sp)
SW $s0, 36($sp)
SW $s1, 32($sp)

ADD $s0, $a0, $a1
SRA $s0, $s0, 1
LW $s1, 0($s0)

SW $a0,28($sp)
SW $a1, 24($sp)
SW $a2, 20($sp)
SW $a3, 16($sp)
ADDI $a0, $s1, 0
ADDI $a1, $a2, 0
JALR $a3
LW $a0, 28($sp)
LW $a1, 24($sp)
LW $a2, 20($sp)
LW $a3, 16($sp)

ADDI $t0, $zero, 1
BNE $v0, $t0, ELSIF
NOP
ADDI $a1, $s0, $zero
JAL BINS
NOP
ADDI $s1, $v0, 0

ELSIF:
ADDI $t0, $zero, -1
BNE $v0, $t0, RET
NOP
ADDI $a0, $s0, $zero
JAL BINS
NOP
ADDI $s1, $v0, 0

RET:
ADDI $v0, $s1, 0
LW $ra, 44($sp)
LW $fp, 40($sp)
```

```
LW $s0, 36($sp)
LW $s1, 32($sp)
ADDI $sp, $sp, 48
JR $ra
```

(a) For each 4-byte entry in the acitvation record, label what that entry's purpose is in the table below.

| | |
|---|---|
| $sp+ 44 | |
| $sp+ 40 | |
| $sp+ 36 | |
| $sp+ 32 | |
| $sp+ 28 | |
| $sp+ 24 | |
| $sp+ 20 | |
| $sp+ 16 | |
| $sp+ 12 | |
| $sp+ 8 | |
| $sp+ 4 | |
| $sp | |

(b) Fill in the missing bits of pseudocode (anything with a ? should be replaced).

```
bins(a0, a1, a2, a3){
   s0 = ?
   s1 = Mem[?]
   v0 = ?
   if (?)
      s1 = ?
   elseif (?)
      s1 = ?
   return ?
}
```

(c) Fill in the blanks in the following sentences to describe what this function does.

This function performs a _____ in the sorted array whose endpoints are _____ and _____ to find the element associated with _____. Because the array elements are generic, possibly even pointers to other objects, we need argument _____ to act as a _____ to tell us "which direction to go" next and when to finish.

(d) This code is poorly written in that it could potentially recurse infinitely, resulting in a stack overflow. In what situation will this happen?

3. This problem will deal with caching.

(a) If, say, SRAM is so much faster, why don't we just build all memory out of SRAM?

(b) The practice of transferring contiguous blocks of memory into cache at once exploits what well-known behavior of memory accesses?

(c) Suppose we are using a direct mapped cache with two 32-byte cache lines, and consider the following pseudocode:

```
lw $1 <- M[4]
lw $2 <- M[72]
lw $3 <- M[40]
lw $1 <- M[68]
lw $2 <- M[80]
lw $3 <- M[0]
lw $1 <- M[44]
lw $2 <- M[72]
```

i. Which bits of a memory address are the cache tag? Which are the index? The offset of a word within the cache line? The offset of a byte within the word?

ii. How many cache hits and cache misses are there during the execution of this code, assuming a cold start with no prefetching?

iii. What which memory addresses are stored in each location in the cache at the end of the execution of this code?

4. In this problem you will implement Euclid's algorithm again. This is the same algorithm that you implemented in the last homework.

a) Implement in C or pseudocode a function `gcd(int num1, int num2)` that takes two arguments and returns their greatest common divisor. If you implemented this algorithm on the last homework in an iterative fashion, you *must* implement this algorithm in a recursive manner here. If you implemented this algorithm on the last homework in a recursive manner, you *must* implement this algorithm in an iterative manner here.

b) Translate your above function into MIPS code. You must account for delay slots in any branches or jumps that you take, but you do not need to optimize the code in any way (so NOPs are fine). Note that you *must* use proper calling conventions as described in lecture, along with proper epilogues and prologues.

Note 1: We are not looking for a piece of code that happens to produce the same result as the code above; we are looking for a *translation* of the code above. So each line of code above should be associated with one or more lines of MIPS code, and these should follow the same order on the page as the code above.