ICS 233 – Computer Architecture & Assembly Language

Assignment 3: Procedures

For the following problems, the table holds C code functions. Assume that the first function listed in the table is called first. You will be asked to translate these C code routines into MIPS assembly.

a.	`int compare(int a, int b) {`
	`  if (sub(a, b) >= 0) return 1;`
	`  else return 0;`
	`}`
	`int sub(int a, int b) {`
	`  return a – b;`
	`}`

b.	`int fib_iter(int a, int b, int n) {`
	`  if (n == 0) return b;`
	`  else return fib_iter(a+b, a, n-1);`
	`}`

1. Implement the C code in the table in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

2. Functions can often be implemented by compilers "in-line". An in-line function is when the body of the function is copied into the program space, allowing the overhead of the function call to be eliminated. Implement an "in-line" version of the above C code in MIPS assembly. What is the reduction in the total number of MIPS assembly instructions needed to complete the function?

3. For each function call, show the contents of the stack after the function call is made. Assume that the stack pointer is originally at address 0x7ffffffc.

The following problems refer to a function f that calls another function func. The function declaration for func is "int func(int a, int b);". The code for function f is as follows:

a.	`int f(int a, int b, int c) {`
	`  return func(func(a, b), c);`
	`}`

b.	`int f(int a, int b, int c) {`
	`  return func(a, b) + func(b, c);`
	`}`

4. Translate function f into MIPS assembly code, using the MIPS calling convention. If you need to use register $t0 through $t7, use the lower-numbered registers first.

5. Right before your function f of Problem 4 returns, what do you know about contents of registers $t5, $s3, $ra, and $sp? Keep in mind that we know what the entire function f looks like, but for function func we only know its declaration.

For the following problems, the table has an assembly code fragment that computes a Fibonacci number. However, the entries in the table have errors, and you will be asked to fix these errors.

```
fib:   addi  $sp, $sp, -12
       sw    $ra, 8($sp)
       sw    $s1, 4($sp)
       sw    $a0, 0($sp)
       slti  $t0, $a0, 3
       beq   $t0, $0, L1
       addi  $v0, $0, 1
       j     exit

L1:    addi  $a0, $a0, -1
       jal   fib
       addi  $s1, $v0, $0
       addi  $a0, $a0, -1
       jal   fib
       add   $v0, $v0, $s1

exit:  lw    $a0, 8($sp)
       lw    $s1, 0($sp)
       lw    $ra, 4($sp)
       addi  $sp, $sp, 12
       jr    $ra
```

6. The MIPS assembly program above computes the Fibonacci of a given input. The integer input is passed through register $a0, and the result is returned in register $v0. In the assembly code, there are few errors. Correct the MIPS errors.

7. For the recursive Fibonacci MIPS program above, assume that the input is 4. Rewrite the Fibonacci program to operate in a non-recursive manner. Restrict your register use to registers $s0 - $s7. What is the total number of instructions used to execute your non-recursive solution versus the recursive version of the factorial program?

In this exercise, you will be asked to write a MIPS assembly program that converts strings into the number format as specified in the table.

a.	`Positive integer decimal string`
b.	`String of hexadecimal digits`

8. Write a program in MIPS assembly language to convert an ASCII number string with the conditions listed in the table above, to an integer. Your program should expect register $a0 to hold the address of a null-terminated string containing some combination of the digits 0 though 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register $v0. If a nondigit character appears anywhere in the string, your program should stop with the value -1 in register $v0.