Chapter 3 Exercises  
CDA3101  
Spring 2002

Solve the following problems. You must show your work in order to receive full credit. Your solutions for this assignment are due by 2:30PM Monday, 1/28/02. Turn in the solutions via e-mail to thomas@cs.fsu.edu. The message of the e-mail should contain the solutions for problem 1 and problem 2. Use pine or some other mailer and attach two files (one containing your code for problem 3 and the other containing your code for problem 4).

(1) Consider the MIPS assembly function below. Assume that $5$ is initialized with a positive integer $n$. Assume that $4$ is used for the output. What will be assigned to $4$ when funct is called with $5$ assigned a value of 3? Describe what the code does as a function of $n$. (10 points)

```
funct:       addi $2,$0,0
            addi $3,$0,1
again:      slt $7,$5,$3
            bne $7,$0,endit
            add $2,$2,$3
            addi $3,$3,1
            ja gain
endit:      add $4,$2,$0
```

(2) Encode the following MIPS instructions. For each of the instructions, you should identify the format type (R, I, or J) for the instruction and the decimal values of each of the fields for that format. The opcode or funct value for each of these instructions are given in the text (see Figure 3.18 or the back inside text cover for a summary of many of the encodings.) (20 points)

(a) andi $2,$3,5

(b) jal goto                # where goto is at address 500

(c) bne $4,$7,label         # where label precedes the 9th instruction after this branch

(e) sub $4,$3,$2
NOTE ON NEXT TWO PROBLEMS

As described in the slides for Chapter 3, you should use the general form of a MIPS assembly language program for testing the functions you write as solutions for the next two problems. However, you should only turn in the code for the function. The code you turn in will comprise the function label and the instructions in the function, which includes the last instruction (jr $31) in the function. You should not turn in any data declarations outside the function or the instructions to load arguments and call the function. We will provide our own test programs to call the functions you write and to test it. The general form of an assembly program is shown below.

```
.data # do not turn in
<declarations of variables passed to the function> # do not turn in
.text # do not turn in
.align 2 # do not turn in
.globl __start # do not turn in
__start: # do not turn in
<instructions to load values into the argument registers> # do not turn in
jal <function label> # do not turn in
ori $2,$0,10 # do not turn in
syscall # do not turn in

<function label>: # turn in
<instructions in function> # turn in
jr $31 # turn in
```

Of course, you will have to test your program and that will require initializing locations in memory with particular data in the section after the .data directive. Appendix A in the text is worthwhile reading on the MIPS instruction set. On page A51 a discussion begins on the assembler directives needed to correctly layout initialized and uninitialized data. In particular, note the discussions of the .byte, .word, and .align directives.
To load the address of data that is labelled in the data section you can use the `la` instruction. For example

```
data
label:
    .word 12
.text
main:
`la $2,label
```

(3) Below is a C (or C++) function.

```c
cvoid foo(int a[], int z[], int n, int c)
{
    int i;

    for (i = 0; i < n; i++) {
        a[i] = c + z[i+1];
        z[i] = c ;
    }
}
```

Translate this function to an equivalent function called `foo` in MIPS assembly. You may assume that the address of the array `a` is passed to the function in register `$2`, the address of the array `z` is passed to the function in register `$3`, the value of `n` is passed in register `$4`, and the value of `c` is passed in register `$5`. You may assume that any other register not reserved for the assembler is available for use in your code. Put meaningful comments beside each instruction. Note that the array `z` must have `n+1` values. Test your code using the `xspim` simulator on `linprog`. (30 points)

(4) Suppose you have an array `a[0], a[1], ... , a[n]` of integers in memory. Also suppose that `a[0], a[1], ... , a[n-1]` are positive and that `a[n] = -1`.

Write a program in MIPS assembly language that scans the array and determines the value of `n` and returns it in register `$3`; determines the value of the maximum integer in the array and returns it in register `$4`; and determines the index of the maximum integer in the array and returns it in register `$5`. If there is more than one element with the maximum value return the largest index value.

You may assume that the address of the array `a` is passed to the function in register `$2` and that any other register not reserved for the assembler is available for use in your code. You may also assume that `n` is greater than 0.