INTRODUCTION TO COMPUTER ENGINEERING

WINTER 2014

ASSIGNMENT 4: ASSEMBLY LANGUAGE
DUE DATE: MONDAY MARCH 30TH, 2014 AT 5:00PM

Please submit this assignment by 5:00 pm on the due date via *MyCourses*. Your assignment must be packaged as follows: one compressed file (.zip) containing the following files:

- Answers to all questions as well as discussions (including schematic circuit diagrams and timing printouts) in **one single** Microsoft Word (doc/docx) or portable document format (pdf) file. Note that scanned handwritten solutions are accepted.
- All files related to your SPIM programs. All files should be named according to the following convention:

  `LAST_FIRST_QX_DESC.xxx`

  where "`LAST`" and "`FIRST`" are you last and first name respectively (no spaces), "`QX`" should be replaced by Q1, Q2 or Q3 depending on the question in the assignment to which your file belongs, "`DESC`" is an optional short description and "`xxx`" is the file extension.

A penalty of 10% will be applied if you do not follow these guidelines.

This assignment will be marked out of 100 points. **Late submissions will NOT be accepted.**

QUESTION 1 (10 POINTS) – FUCNTION CALLS

MIPS' native assembly code only has two branch instructions, `beq` and `bne`, and only one comparison instruction, `slt`. Using these instructions (along with, e.g., the `ori` instruction to set a register to 0 or 1), write the MIPS assembly language equivalents for each of the following "C" code snippets, and then also translate the resulting assembly instructions into machine code explaining for each line how you worked out the machine instruction.

(Assume that `x` is stored in register $6, `y` is stored in register `$7`, and `z` is stored in register `$8`. Remember that in "C", if an expression is true, it evaluates to 1 and if false, it evaluates to 0. Refer to Appendix A of the textbook for descriptions of MIPS assembly language)

(a) `z=0;z+=(x>y);`

(b) `x=((++z)<=y);`

(c) `z=(x>y)||(x<z);`

QUESTION 2 (50 POINTS) – FUNCTION CALLS

Translate the following recursive factorial program from C to MIPS assembly language. Make sure to pay attention to context preservation between function calls, as it is essential for such a recursive implementation to work. Write an implementation for the `main` function as well as the `fact` function. The main function should read one unsigned integer from the console and output its factorial (see the SPIM syscall instruction for reading from/writing to the console). Comment your assembly code in detail (comment sections of the code as well as

each individual line). Implement your code on SPIM and check that it work correctly for a few values of the argument.

```
include <stdio.h>

unsigned long int fact(unsigned long int x)
{
  if (x==0)
    return(1);
  else
    return(fact(x-1)*x);
}

int main(int argc, char* argv[])
{
  unsigned long int n;
  scanf("%d", &n);
  printf("%lu",fact(n));
}
```

QUESTION 3 (50 POINTS) – 32-BIT MULTIPLY AND ACCUMULATE

The Multiply-and-accumulate operation is a very common operation in several areas of electrical engineering. It consists in repeatedly multiplying values from two different arrays and summing up the result. For instance, if integers are stored in arrays `a` and `b` of length `n`, the multiply and accumulate operation on those two arrays can be written mathematically as

$$MAC(a,b) = \sum_{i=0}^{n-1} a[i] * b[i].$$

You are asked to design and test an assembly language program to carry out this operation, without using the `mult` MIPS instruction (i.e., you will have to implement your own multiplication in assembly).

The input to your program will be two arrays `a` and `b` of 32-bit integers stored in memory as well as their length `n`. Your program should return the value MAC(a,b) as per the expression above.

Follow the steps below:

(i) first, write a C implementation as a reference; This C program should be used as the basis for translation into MIPS assembly; it should also be thoroughly tested for correctness so that it can be considered as a reliable reference for the subsequent MIPS assembly implementation. It should adhere to the following template (this assumes a machine where `long int` corresponds to 32 bits):

```
#include <stdio.h>
#include <stdlib.h>

long int mult(long int x, long int y)
```

```
/* implementation of a 32-bit multiplication of 2 integers;
   x and y are the two integers to be multiplied;
   this function should not use the built in "*"
   operator, but should carry out long binary multiplication
*/
{
[INSERT YOUR CODE HERE]
}

long int MAC(long int *a, long int *b, int n)
/* implementation of the MAC function
   a is a pointer to the first array
   b is a pointer to the second array
   n is their common length
*/
{
[INSERT YOUR CODE HERE – should call mult function above]
}

int main(int agrc, char argv)
{
   long int a[10]; /* Change size of arrays as needed */
   long int b[10];
   [INITIALIZE VARIABLES HERE]
   printf("%d", MAC(a,b), 10); /* Change size of arrays as needed */
}
```

(ii) Then, translate your implementation into MIPS Assembly language. Test this implementation in SPIM, and compare with the C implementation for a few examples of a, b and n.