PROGRAMMING PROJECT 1
(Due: 21 / 04 / 2017 – 23:00)

In this project, you are required to implement some procedures in MIPS assembly language. You will use SPIM simulator to develop and test your code.

1. *(8 points)* Transform the following C code into a MIPS assembly code.

```
int f (int v[], int k)
   {
          int v1[100];
          int v2[100];
          int v3[100];
          int i;

          if ( k > 100)
               return -1;

          for (i = 0; i < k; i++)
          {
                 v1[i] = i + 1:
                 v2[i] = i + 2;
                 v3[i] = i + 3;
          }

          for (i = 0; i < k; i++)
                v[i] = v1[i] + v2[i] + v3[i];

          return (v[0]);
   }
```

2. *(8 points)* Write a MIPS program that takes an input string and an input char and count the number of input char in the input string.

Example:

```
Enter an input string (max 20 chars): Hello
Enter an input char: e
The number of e in Hello is 1.
```

3. *(8 points)* Write a MIPS program to compute the sum of N numbers such that each sum skips over certain numbers. Ask the user for how many numbers to skip (give option of 1-4) and then proceed to compute the sum.

Sum(x1) = N1 + N2 + N3 + …
Sum(x2) = N1 + N3 + N5 + ….
Sum(x3) = N1 + N4 + N7 + ….
Sum(x4) = N1 + N5 + N9 + ….

Sum(x1) determines the sum of all numbers,
Sum(x2) determines the sum of numbers skipped by 2,
Sum(x3) determines the sum of numbers skipped by 3,
Sum(x4) determines the sum of numbers skipped by 4.

Assume the numbers as given below where N1 = 100, N2 = -7, N3 = 11, N4 = 25 and so on.

```
.data
.strA: .asciiz "Please enter your choice to skip numbers (1-4)\n"
Numbers: .byte 100, -7, 11, 25, -66, 99, -1, 34, 12, 22, -2, -7, 100,
11, 4, 67, 2, -90, 22, 2, 56, 3, -89, 12, -10, 21, 10, -25, -6, 9, 111,
34, 12, 22, -2, -17, 100, 111, -4, 7, 14, -19, -2, 29, 36, 31, -79, 2

.globl main
.text
main: # Your fully commented program starts here
```

4. *(8 points)* Write a program in MIPS that reads the elements of an array and prints the uppercase characters represented by the integers in the array.

- Register *$t0* holds the address of the beginning of an array of 32-bit integers, each representing a character.
- The integer zero represents a space, and each integer *i* (1 <= *i* <= 26) represents the *i*'th letter of the uppercase alphabet.
- Register *$t1* holds the number of elements in the array.

Example:

```
Enter length of array: 11
Enter array element 0: 8
Enter array element 1: 5
Enter array element 2: 12
Enter array element 3: 12
Enter array element 4: 15
Enter array element 5: 0
Enter array element 6: 23
Enter array element 7: 15
Enter array element 8: 18
Enter array element 9: 12
Enter array element 10: 4
HELLO WORLD
$08 = 0x0000006c   $09 = 0x0000000b
```

5. *(12 points)* Write a program in MIPS to print the sum of two rational numbers (fractions) as a rational number too. The following restrictions apply:

- The numbers must be entered by the user as **pairs of integers** separated by a proper delimiter (e.g. 5/8).
- The program must use **integer arithmetic only**.

- The result of the program must be a rational number in its **simplified** form. For example, if you add (1/5 + 3/10) the result must be printed as 1/2 (not 25/50, which you can get by the following transformation: 1/5+3/10 = (1*10+3*5)/(5*10) = 25/50).

Example:

```
Enter the first fraction:  1/5
Enter the second fraction: 3/10
The sum is: 1/2
```

6. *(16 points)* Write a MIPS program to determine the height of a binary tree. Assume that the tree will contain at least one node and hence have height at least one.
The tree will be encoded in an array of 32-bit integers.

- Each node of the tree is encoded in three consecutive elements (words) of the array: a signed integer stored at the node, the node's left child, and the node's right child.
- Each child is specified as the array index of the child node.
- The integer -1 indicates that a node does not have a left or right child.

For example, the following tree:

```
   77
  /  \
22    -8
     /  \
   -36   999
```

could be encoded by following array:

```
A[0] = 77
A[1] = 3
A[2] = 6
A[3] = 22
A[4] = -1
A[5] = -1
A[6] = -8
A[7] = 9
A[8] = 12
A[9] = -36
A[10] = -1
A[11] = -1
A[12] = 999
A[13] = -1
A[14] = -1
```

in which the root is encoded by the elements A[0], A[1] and A[2];

- where A[0] represents the value of the root,
- A[1] represents the index of the left child (3 means look at A[3] to find left child),
- A[2] represents the indext of the rigth child (6 means look at A[6] to find rigth child).

the root's left child is encoded by the elements A[3], A[4] and A[5],

the root's right child is encoded by the elements A[6], A[7] and A[8], and so on.
This tree has height 3.

Register *$t0* holds the address of the beginning of the array, and Register $t1 holds the number of elements in the array. Determine the height of the tree, store it in Register $t2. Your solution should only read, but NOT MODIFY, the tree in memory.

Example:

```
Enter length of array: 15
Enter array element 0: 77
Enter array element 1: 3
Enter array element 2: 6
Enter array element 3: 22
Enter array element 4: -1
Enter array element 5: -1
Enter array element 6: -8
Enter array element 7: 9
Enter array element 8: 12
Enter array element 9: -36
Enter array element 10: -1
Enter array element 11: -1
Enter array element 12: 999
Enter array element 13: -1
Enter array element 14: -1
MIPS program completed normally.
$08 = 0x000000c4   $09 = 0x0000000f   $10 = 0x00000003
```

Notes:

- A good way to go about writing a MIPS program is to first write an equivalent C program. It is much easier and quicker to get all the control flow and data manipulations correct in C than in MIPS. Once the C code for a program is correct, one can translate it to an equivalent MIPS program statement-by-statement.

- **Bonus:** You will get 10% extra credit if your program support a *Menu* including all questions above. Example:

  Please select an option:
  1. C-to-MIPS convert
  2. Char Finder
  3. Sum of Numbers
  4. Sum of Rational Numbers
  5. Message Printer
  6. Find the Tree Height
  7. Exit

These options must be printed inside a loop until “Exit” option is selected.

Requirements

- The arguments to the procedures are stored in $a registers; i.e., the first one is in $a0, the second one is in $a1, and so on.
- The values in all $a registers should be as they were at the time of call at the end of the procedure.
- You have to use **QtSpim** simulator in your implementation. Any other simulator is not allowed.
- You are required to submit a minimum 2-pages report explaining implementation details and screenshot of your example runs for your Project and commented code (via e-mail: `cse338.projects@gmail.com`). Your implementation detail should be provided in the source code comment.

General Policies

- *You are allowed to work in groups of 3 members. You will select your partners and they will not be changed throughout the semester. It is not acceptable of a partner team to work with other teams.*
- *Some part of your project will be graded with the* ***Project Quiz****.*
- *The ones who do not submit a project will not attend to the* ***Project Quiz****.*
- *It is NOT acceptable to copy (or start your) solutions from Web.*
- *In case of any forms of cheating or copying among the groups, the penalties will be severe.* ***Both Giver and Receiver are equally culpable and suffer equal penalties!!!***
- ***No late submission will be accepted!***

References

[1] http://spimsimulator.sourceforge.net/