Assignment – 1

Due: Fri. 16 August (at 11:00 AM)

Submit a printed copy of your answers, with a cover sheet which indicates your name, ID and the course number.

NOTE: Your assembly code must have comments if necessary.
Do not use pseudo instructions (except *la* if necessary.)

1. An array of integers is stored in memory in big-endian form. The base address of the array is given in register $t0, and it has 2048 elements. Write *the MIPS assembly instructions* for a little-endian machine, which divides each element of the array by 16 and stores the result in memory from starting from the address given in register $t1.

```
        # $t0: the base address of source array (which is in big-endian form)
        # $t1: the base address of destination array (which is in little-endian form)

        # as the data was initially stored in big-endian form in the input array, but the
        # processor used for the operations is little-endian, we cannot use lw instruction
        # to read each element of the array. We have to read byte by byte and arrange it
        # properly to be used for the little-endian processor.

        addi    $t2, $0, 2048       # no. of elements of the array
        add     $9, $0, $0
L1:     # read each element of the array to be processed in little-endian machine ….
        lbu     $t3, 0($t0)         # read the Most Significant Byte
        sll     $t3, $t3, 24
        ori     $t4, $t4, $t3
        lbu     $t3, 1($t0)
        sll     $t3, $t3, 16
        ori     $t4, $t4, $t3
        lbu     $t3, 2($t0)
        sll     $t3, $t3, 8
        ori     $t4, $t4, $t3
        lbu     $t3, 3($t0)         # read the Least Significant Byte
        ori     $t4, $t4, $t3       # now we have the 4-byte integer value in $t4

        sra     $t4, $t4, 4         # divide it by 16
        sw      $t4, 0($t1)         # store the result into the destination array
        addi    $t0, $10, 4         # change address for the next element of source array
        addi    $t1, $11, 4         # change address for the next element of destination array
        addi    $t2, $t2, -1
        bne     $t2, $0, L1
```

2. (a) Write a **function**, which takes a null-terminated string (an array of characters) and finds how many digits (i.e. 0, 1, 2, …, 9) is in the string. The input argument of the function is the address of string (i.e. the base address of the array of characters). The function returns a number, which indicates how many digits is in the given string.

   (b) Write the main program, which calls this function using the example string: "Numbers are 0, 25 and 123 for this." to test it.
   Test your code using SPIM.

(a)

Note: ASCII codes of digits 0 to 9 are 0x30 to 0x39. So, our algorithm only needs to check if the ASCII code of the read character is in the range of 0x30 to 0x39.

```
func1:  # base address of the string is in reg. $a0
        # the result is returned in reg. $v0

        add     $t0, $a0, $0        # save $a0
        addi    $v0, $0, 0          # initialize $v0

L1:     lbu     $t1, 0($t0)         # read the character
        beq     $t1, $0, L3         # is it end of string?

        # check if it is number between 0 to 9
        slti    $t2, $t1, 0x30
        bne     $t2, $0, L2
        slti    $t2, $t1, 0x3A
        beq     $t2, $0, L2

        addi    $v0, $v0, 1         # increment no. of found numbers (digits)

L2:     addi    $t0, $t0, 1         # increment $t0 to point to the next char
        j       L1

L3 :    # exit the loop if end of string
        jr      $ra
```

(b)

```
        .data
 str1:  .asciiz   " Numbers are 0, 25 and 123 for this."

        .text
        .globl  main
main:
        la      $a0, str1           # array base address should be in $a0
        jal     func1

        add     $a0, $0, $v0
        ori     $v0,  $0, 1
        syscall                     # print the result

        jr      $ra
```

Note: Only the red lines are needed to get the full marks for this part.

3. **A** is a two-dimensional array (matrix) of double-precision floating-point numbers and **B** is an array of 32-bit integers. Write *the MIPS assembly instructions* to perform the following arithmetic operation:

$$C = \sum_{i=0}^{9} \sum_{j=0}^{4} (A[i][j] + B[j])$$

C is a double-precision floating-point number, which should be stored in register $s2.
Assume that the processor is ***big-endian***. The base addresses for arrays **A** and **B** are given in registers $s0 and $s1 respectively.

In order to do the floating-point operations using co-processor 1 instructions, elements of matrix B which are integer should be copied into one of FP registers and converted to double-precision floating-point numbers.

A matrix is a two-dimensional array. However, memory is organized as a one-dimensional array. So, a matrix can be stored in memory as either in form of row-major (where the first row is placed in memory followed by the next rows) or in form of column-major (where the first column is placed in memory followed by the next columns). Then the address of element A[i][j] should be calculated properly (depending on if row-major or column-major is used).

We assume that the base addresses of matrix A and array B are given in registers **$s0** and **$s1** respectively. Loop counters are registers **$t2** (for the outer loop) and **$t3** (for the inner loop) so they should be initialized properly as indicated below.
The address of C is assumed to be in register **$s2**. The processor is **big-endian**.

A is a two-dimensional array (matrix) of double-precision floating-point numbers so each element is 8 bytes.

Address of A[i][j] = base address of A + ((i * size of row) + j) * size of data = base address of A + ((i * size of row) + j) * 8

For example, we calculate the address of A[i][j] and put it into $t6, (we assumed base address of A is in register $s0, size of row (or no. of columns) is in $t1, i is $t2, and j is $t3)
Address of A[i][j] $t6 = $s0 + (($t2 * $t1) + $t3) * 8

B is an array of integer numbers so each element is 4 bytes.

Proper instructions can be used to calculate the address of A[i][j] and B[j].

```
# Here is the required MIPS instructions:
        ...
        # it is assumed that the base address of arrays A and B are in registers
        #  $s0 and $s1 respectively. Address for variable C is in register $s2

        addi    $t0, $0, 10       # no. of rows
        addi    $t1, $0, 5        # no. of columns

        # initialize C to 0.0  (which is in register pair  $f14,$15)
        mtc1    $0, $f10
        cvt.d.w $f14, $f10

        # initialize loop counters
        add     $t2, $0, $0       # i
        add     $t3, $0, $0       # j

 L1:    # calculate address for element A[i][j]
        # $t6  =   $s0 + (($t2 * $t1) + $t3) * 8
        mult    $t2, $t1
        mflo    $t4               # note HI will be 0 in this case
        add     $t4, $t4, $t3
        sll     $t4, $t4, 3
        add     $t6, $t4, $s0

        # calculate address for element B[j]
        # $t7  =   $s1 + $t3 * 4
        sll     $t7, $t3, 2
        add     $t7, $t7, $s1

        # read A[i][j] from memory (assuming big-endian)
        lwc1    $f6, 0($t6)
        lwc1    $f7, 4($t6)

        # read B[j] from memory
        lw      $t4, 0($t7)

        # convert B[j] to double precision floating point
        mtc1    $t4, $f8
        cvt.d.w $f10, $f8         # result in $f10,$f11 pair

        # perform the operation
        add.d   $f12, $f6, $f10   # $f12,$f13 = A[i][j] + B[j]
        add.d   $f14, $f14, $f12  # update C up to this point

        # increment the inner loop counter
        addi    $t3, $t3, 1
        bne     $t3, $t1, L1
        add     $t3, $0, $0       # j = 0

        # increment the outer loop counter
        addi    $t2, $t2, 1
        bne     $t2, $t0, L1

        # Now at this point the result is in $f14,$f15 pair to be written in C
        swc1    $f14, 0($s2)
        swc1    $f15, 4($s2)      # assuming big-endian
        ...
```

4. Using function ***fibonacci(n)*** the sequence of numbers 1, 1, 2, 3, 5, 8, 13, 21, ... for values of ***n*** equal to 1, 2, 3, 4, 5, 6, 7, 8, … can be generated.

   The result for each ***n*** is calculated as follows:

   fibonacci(1) = 1, fibonacci(2) = 1,
   fibonacci(n) = fibonacci(n-1) + fibonacci(n-2) for n > 2

   (*Note that the function is called recursively).*

```
int fibonacci(int n)
{
    if (n < 1)      return 0;

    else if (n == 1 || n == 2)
        return 1;
    else
        return (fibonacci(n - 1) + fibonacci(n - 2));
}
```

   (a) Write the MIPS assembly code for function ***fibonacci***.

   (b) Write the main program, which prints the Fibonacci numbers up to the calculated value for ***n*** on SPIM Console. (For example, if **n** is 10, then the following sequence will be printed).

   1, 1, 2, 3, 5, 8, 13, 21, 34,

Test your code using SPIM.

(a)

```
fibonacci:
        # $a0: n

        # set up the stack frame (as this is a recursive function)
        addi    $sp, $sp, -16
        sw      $ra, 12($sp)    # save $ra
        sw      $s0, 8($sp)     # save $s0
        sw      $a0, 4($sp)     # save $a0

        addi    $v0, $0, 0
        addi    $s0, $0, 0

        # check if n < 1 then return 0
        slti    $t0, $a0, 1
        beq     $t0, $0, L1
        addi    $v0, $0, 0
        j       L3

        # check if n == 1 or n==2
L1:     addi    $t0, $0, 3
        slti    $t1, $a0, 3
        beq     $t1, $0, L2
        addi    $v0, $0, 1
        j       L3

L2:     addi    $a0, $a0, -1
        jal     fibonacci       # call recursively (n-1)
        add     $s0, $s0, $v0   # update the value up to this point
        addi    $a0, $a0, -1
        jal     fibonacci       # call recursively (n-2)
        add     $v0, $v0, $s0   # update the value up to this point

L3:   # release the stack frame
        lw      $ra, 12($sp)    # restore $ra
        lw      $s0, 8($sp)     # restore $s0
        lw      $a0, 4($sp)     # restore $a0
        addi    $sp, $sp, 16

        jr      $ra
```

(b)

```
        .data
str1:   .asciiz  "Fibonacci sequence is: "
str2:   .asciiz  ", "

        .text
         .globl  main
main:
        # set up the stack frame
        addi    $sp, $sp, -16
        sw      $ra, 12($sp)    # save $ra
        sw      $s1, 8($sp)     # save $s1
        sw      $s0, 4($sp)     # save $s0

        # print the initial message
        la      $a0, str1
        ori     $v0,  $0, 4
        syscall

        addi    $s1, $0, 10     # the test number is in $s1
        addi    $s0, $0, 1      # initialize the loop counter $s0

L0:     add     $a0, $0, $s0
        jal     fibonacci
        add     $a0, $0, $v0
        ori     $v0,  $0, 1
        syscall                 # print the result
        la      $a0, str2
        ori     $v0,  $0, 4
        syscall
        addi    $s0, $s0, 1
        bne     $s1, $s0, L0

        # pop the stack frame
        lw      $s0, 4($sp)     # restore $s0
        lw      $s1, 8($sp)     # restore $s1
        lw      $ra, 12($sp)    # restore $ra
        jr      $ra
```

Note that the part shown in red colour is not required to get the full mark for this question.