Assembly programming
Solved exercises

Exercise 1. Given the following program in RISC-V32.

```
.text
 main:
                    ...
                    li   a0,  5
                    jal  ra funcion1

                    li   a7, 1
                    ecall

                    li   a7, 10
                    ecall

funcion1:           li  t0, 10
                    bgt a0, t0, et1
                    li  t0, 10
                    add a0, t0, a0
                    j   et2
      et1:          li  t0, 8
                    add a0, t0, a0
      et2:          jr  ra
```

Indicate in a reasoned way the value that is printed on the screen (first system call of the previous code).

Solution:

The function is passed as an argument in the register `a0` the value `5`. If this value is higher than (`bgt`) 10 it is jumped to the label `et1`. As it is not the case, then `10 (t0)` is added to `5` and the result is stored in `a0`, which is the printed value, i.e. `15`.

Exercise 2. Consider a function called `AddValue`. Three parameters are passed to this function:

- The first parameter is the starting address of an array of integers.
- The second parameter is an integer value that indicates the number of components in the array.
- The third parameter is an integer value.

The `AddValue` function modifies the array, adding the last value as the third parameter to all the array components. You are asked to:

a) Indicate in which registers each of the parameters must be passed to the function.
b) Program using the RISC-V_{32} code of the above function.
c) Given the following program fragment:

```
.data
        v:   .word    7, 8, 3, 4, 5, 6
.text
  main:
```

Include in the `main` above, the necessary assembly sentences to invoke the `AddValue` function implemented in section b) so that it adds to the components of array `v` defined in the data section, the number 5. Implement after the function call, the assembly instructions for printing all the elements of the array.

Solution:

a) The starting address is passed in a0, the number of items in a1 and the value to be added in a2.

b)

```
AddValue:          li    t0, 0
                   mv    t1, a0

      loop  :      bgt   t0, a1, end
                   lw    t2, 0(t1)
                   add   t2, t2, a2
                   sw    t2, 0(t1)
                   addi  t0, t0, 1
                   addi  t1, t1, 4
                   j     loop
      end:         jr    ra
```

c) the `main` function is:

```
.data
        v: .word  7, 8, 3, 4, 5, 6
.text
 main:      addi  sp, sp, 4
            sw    ra, 0(sp)

            la    a0, v
            li    a1, 6
            li    a2, 5
            jal   ra, AddValue
```

```
            li    v0, 1
            li    t0, 0
            mv    t1, a0
loop2:      bgt   t0, a1, end2
            lw    a0, 0(t1)
            ecall

            addi  t0, t0, 1
            addi  t1, t1, 4
            j     loop2
end2:
            lw    ra, 0(sp)
            addi  sp, sp, 4

            li   a7, 10
            ecall
            jr    ra
```

Exercise 3. Given the following RISC-V32 program:

```
.text
       main:
                     li   a0,  5
                     jal  ra f1

                     li   a7, 1
                     ecall

                     li   a7, 10
                     ecall

         f1:         li  t0, 10
                     bgt a0, t0, et1

                     mv   t0, a0
                     li   t1, 0
         b1:         beq  t0, x0, fin
                     add  t1, t1, t0
                     addi t0, t0, -1
                     j    b1
        fin:         mv   a0, t1
                     j    et2

         et1:        li a0, 0
         et2:        jr ra
```

Indicate in a reasoned way the value returned by the function `f1` in the register `a0` and printed on the screen (first system call of the previous code).

Solution.

Function `f1` performs the following functionality:

```
if (a0 <= 10)
{
       t0 = a0;
       t1 = 0;
       while (t0 != 0)
       {
              t1 = t1 + t0;
              t0 = t0 - 1;
       }
       return t1;
}
else
{
       return 0;
}
```

Since the value of `a0` is 5 and is less than 10, the function performs the addition of the values 5, 4, 3, 2, 1 and returns 15.

Exercise 4. Given the following RISC-V32 program:

```
.data
            a: .word 5
            b: .word 10
.text
main:
            li    t0  1
            la    t1, a
            lw    t1, 0(t1)
            la    t2, b
            lw    t2, 0(t2)
  label1:   bgt   t0, t1, label2
            addi  t2, t2, 2
            addi  t0, t0, 1
            j     label1
  label2:   la    t0, a
            sw    t0, 0(t0)
            la    t2, b
            sw    t2, 0(t2)
```

Indicate the values stored in `t0`, `t1` y `t2` and the memory address `a` and `b` when the program finished the execution.

Solution:

t0	t1	t2	a	b	
-	-	-	5	10	
1	-	-	5	10	
1	5	10	5	10	
1	5	12	5	10	
2	5	12	5	10	
2	5	14	5	10	
3	5	14	5	10	
3	5	16	5	10	
4	5	16	5	10	
4	5	18	5	10	
5	5	20	5	10	
6	5	20	5	10	end (t0 > t1)
6	5	20	**6**	**20**	

Exercise 5. Give the following program.

```
.text
 main:
                   li   a0, 5
                   jal  ra, funcion
                   li   a7, 1
                   ecall

                   li   a7, 10
                   ecall
funcion:
                   mv   t0, a0
                   li   t1, 0
         bucle:    beq  t0, 0, fin
                   add  t1, t1, t0
                   addi t0, t0, -1
                   j    bucle
          fin:     mv   a0, t1
                   jr   ra
```

It is requested:

a) Indicate in a reasoned way the value that is printed on the screen (first call to the system of the previous code).
b) If in the register `a0`, used to pass parameters to the function, values are represented in one's complement, What range of numbers could be passed to the function?

Solution:

a) The function performs the addition of the numbers 5, 4, 3, 2, 1 and returns the result in the register a0, whose value is therefore 15, which is the result that is printed on the screen.
b) For a word of n bits, the range of representation of numbers in one's complement is $[-2^{n-1}+1, \ldots -1, 0, -0, 1 \ldots 2^{n-1}-1]$. In the case of RISC-V 32, the registers are 32 bits, so n = 32 and the representation range would be $[-2^{31}+1, \ldots, 2^{31}-1]$.

Exercise 6. Write a program using the RISC-V32 instruction set, which perform the addition of the squares of a series of numbers entered by the keyboard. To do this, the program will first ask for the number of numbers to read. Then, it will read those numbers, make the corresponding addition, and finally print the result.

Solution:

```
.data
  msg01: .asciiz "Amount of numbers to read: "
  msg02: .asciiz "introduce number: "
  msg03: .asciiz "The result is: "

.text
  main:
    # print message "Amount of numbers to read: "...
    la         a0 msg01
    li         a7 4
    ecall

    # read the amount of numbers to read
    li         a7 5
    ecall
    mv         t0 a0

    # if quantity of numbers to read is zero, finish.
    beq        x0, t0 f01

    # loop (t1: counter of numbers, t2: partial result)
    li  t1 0
    li  t2 0

   b01:
    # print "introduce number: "
    la  a0 msg02
    li  a7 4
    ecall

    # read the number
    li  a7 5
    ecall

    # Calculation of the square and partial addition
    mul        v0 a0 a0
    add        t2 t2 v0

    # loop
    addi       t1 t1 1
    blt        t1 t0 b01

    # print  "The result is: "...
    la  a0 msg03
    li  a7 4
    ecall
    # print the result
    mv         a0 t2
    li         a7 1
    ecall

f01:    li  a7 10
        ecall
```

Exercise 7. Write a function named `printScreen` that receives as a parameter in the register $a0 the address of a memory position containing a 32-bit integer array. The array elements represent the initial address of a string of characters (ending in zero), ending the array with the zero-value element. The function must print all strings and return in the `v0` register the number of characters printed on the screen.

Solution:

```
.data
   pan1: .asciiz "one\n"
   pan2: .asciiz "two and three"
   pan:  .word   pan1, pan2

.text
  PrintScreen:     mv     t4 a0
                   mv     t0 a0
                   li     t1 0

                   # loop
         IP_ini1: lw      t2 0(t0)
                  beq     x0 t2 IP_end1
                  mv      a0 t2
                  li      a7  4
                  ecall

                  # loop to count characters
         IP_ini2: lb     t3 0(t2)
                  beq    x0 t3 IP_end2
                  addi   t1 t1 1
                  addi   t2 t2 1
                  j      IP_ini2

         IP_end2: addi   t0 t0 4
                  j      IP_ini1

         IP_end1: mv     a0 t0
                  jr     ra


   main:           la     a0 pan
                   jal    ra PrintScreen


                   li     a7 1
                   ecall

                   li     a7 10
                   ecall
```

Exercise 8. Consider a function called `Vowels`. This function is passed as a parameter the start address of a string. The function calculates the number of times the character 'a' (in lower case) appears in the string. In the case of passing the null string the function returns the value -1. If the string has no 'a', the function returns 0:

a) Write using the RISC-V32 assembly a program for the `Vowels` function.
b) Indicate in which register the argument must be passed to the function and in which register the result must be collected.
c) Given the following program fragment:

```
.data
      str: .asciiz  "Hello"

.text
 main:
```

Include in the main above, the sentences in assembler necessary to be able to invoke the function Vocals implemented in the section a) and print by screen the value that the function returns. The objective is to print the number of times the character 'a' appears in the string "Hello".

Solution:

a) It is assumed that the string address is passed in the register $a0 and the result is returned in the register $v0

```
Vowels:     li    t0, -1      # counter of  a
            mv    t1, a0
            beq   x0, t1, fin
            li    t0, 0
            li    t2, 'a'
   bucle:   lbu   t3, 0(t1)
            beq   x0, t3, fin
            bne   t3, t2, noA
            addi  t0, t0, 1
      noA:  addi  t1, t1, 1
            j     bucle
      fin:  mv    a0, t0
            jr    ra
```

d) The arguments are passed in the `ax` records and the results in the `a0` and `a1` registers. In this case the string start address is passed in `a0` and the result is collected in `a0`.

e) The `main` function is:

```
.data
        str:  .asciiz    "Hola"
.text
 main:      addi  sp, sp, -4
            sw    ra, 0(sp)

            la    a0, cadena
            jal   ra, Vocales
            li    a7, 1
            ecall
            lw    ra, 0(sp)
            addi  sp, sp, 4

            li   a7, 10
            ecall
```

Exercise 9. Consider a 32-bit computer with 48 registers and 200 machine instructions. Indicate the format of the hypothetical instruction `beqz t1, t2, address` where t1 and t2 are registers and `address` represents a memory address.

Solution:

The number of bits needed to encode 48 registers is 6 bits. To encode 200 instructions, 8 bits are needed (OC). To represent an address in a 32-bit computer, 32 bits are needed. Therefore, two words are necessary:

OC	t1	t2		address
8	6	6	12	32

Exercise 10. Consider a function called `func` that receives three integer type parameters and returns an integer type result, and consider the following fragment of the data segment:

```
.data
      a: .word 5
      b: .word 7
      c: .word 9

.text
```

Indicate the necessary code to call the previous function passing as parameters the values of the memory positions a, b and c. Once the function is called, the returned value must be printed

Solution:

```
la     t0, a
lw     a0, 0(t0)
la     t0, b
lw     a1, 0(t0)
la     t0, c
lw     a2, 0(t0)
jal    ra, func
li     a7, 1
ecall
```

Exercise 11. Indicate a RISC-V_{32} instruction that includes the relative addressing mode. What does this addressing consist of?

Solution:

```
lw t1, 20(t2)
```

The field with this addressing mode is 20(t2), which represents the memory address obtained by adding 20 with the address stored in the t2 register.

Exercise 12. Given the following fragment:

```
.data
      A1: .word 5, 8, 7, 9, 2, 4, 5, 9
      A2: .word 1, 4, 3, -8, 5, 6, 5, 9
     .align 2
      A3: .space 32

.text
```

Questions to be answered:

a) What does A1 represent? How many bytes does data structure A1 occupy (justify your answer)?

b) You want to implement a function whose prototype in a high-level language is the following:

```
void Merge(int  a[], int b[], int c[], int N)
```

This function receives four parameters, the first three are integer arrays and the fourth indicates the number of components of each of these arrays. The function is responsible for storing in each component i of c, the following value: `c[i] = max (a[i], b[i])`.

Write, using the RISC-V32, the code corresponding to this function.

a) Write the necessary code to call the function developed in the previous section, for arrays A1, A2 and A3 defined in the previous data section. Assume that A3 is the array where the maximum elements should be left.

Solution:

a) A1 represents an array of eight integers. If there are 8 integers, and being a 32-bit machine, each integer is represented by 4 bytes, it means a total of 8*4 = 32 bytes.

b) and c)

```
.data
   v1: .word 1, 2, 3, 4, 5
   v2: .word 5, 4, 3, 2, 1
   v3: .word 0, 0, 0, 0, 0

.text
.text
 max:
         mv   t0 a0
         li   t4 0

         mv   t1 a1
         mv   t2 a2
         mv   t3 a3

 bucle1: bge  t4 t0 fin1
         lw   t5 0(t1)
         lw   t6 0(t2)
         bgt  t5 t6 es5
         sw   t6 0(t3)
         j    next1
    es5: sw   t5 0(t3)

  next1: addi t1 t1 4
         addi t2 t2 4
         addi t3 t3 4
         addi t4 t4 1
```

```
          j   bucle1
    fin1: jr  ra

 main:
          li a0  5
          la a1 v1
          la a2 v2
          la a3 v3

         jal ra  max

         li  t0 0
 loop2:  bge t0 5 fin2

         lw a0 0(a3)
         li a7 1
         ecall

         addi a3 a3 4
         addi t0 t0 1
         j    loop2

 fin2:   li a7 10
         ecall
```

Exercise 13. Given the following code fragment written in C, write using the RISC-V 32 assembly, the code of the equivalent function.

```
int max (intA, int B)
{
      if (A > B)
            return A;
      else  return B;
}
```

Using the assembly above function, implement the code for the following function using the RISC-V 32.

```
void maxV (int v1, int v2, int v3, int N)
{
      int i;

      for (i = 0; i < N; i++)
            v3[i] = max(v1[i], v2[i]);
      return;
}
```

Solution:

```
     max:     bgt   a0, a1, then
              mv    a0, a1
 then:        jr    ra


  maxV:       addi  sp, sp, -4
              sw    ra, 0(sp)

              mv    t0, a0
              mv    t1, a1
              mv    t2, a2
              li    t3, 0
              mv    t4, a3

loop1:        bge   t3, t4, endLoop1
              lw    a0, 0(t0)
              lw    a1, 0(t1)
              jal   ra, max
              sw    a0, 0(t2)

              addi  t0, t0, 4
              addi  t1, t1, 4
              addi  t2, t2, 4
              addi  t3, t3, 1
              j     loop1

endLoop1:     lw    ra, (sp)
              add   sp, sp, 4
              jr    ra
```

Exercise 14. Consider the following function:

```
int  Substitute (String cadena, char c1)
```

The function replaces each occurrence of character c1 that appears in the string with the last character of the string. The function also returns the position of the last changed character in the string.

Example: If `str = "Hola mundo"` and `c1 = 'a'` The function must modify the string so that its new value is `"Holo mundo"`. The function would return for this case the value 3.

Answer

a) Implement the code of the previous function using the RISC-V 32 assembly.

b) Given the following data segment fragment:

```
.data
      str: .asciiz "This is a string"
```

Indicate the code needed to invoke the above function by passing as parameter the string str and the character 'a'. Print the result that returns the function on the screen.

Solution:

a)

```
      Substitute:
                  mv    t0, a0
                  li    v0, 0     # position of the last changed character
                  lbu   t1, 0(t0)
                  bne   x0, t1, loop1
                  jr    ra

        loop1:    beq   x0, t1, end1
                  lbu   t2, 0(t0)
                  addi  t0, t0, 1
                  lbu   t1, 0(t0)
                  j     loop1

      # in $t2 is stored the last character of the string

         end1:    mv    t0, a0
                  lbu   t1, 0(t0)
        loop2:    beq   x0, t1, end2
                  bne   t1, t2, next
                  sw    a1, 0(t0)
                  addi  v0, v0, 1
        next:     addi  t0, t0, 1
                  lbu   t1, 0(t0)
                  j     loop2
         end2:    jr    ra
```

b)

```
      la    a0, Cad
      li    a1, 'a'
      jr    sustituir
      li    a7, 1
      ecall
```

Exercise 15. The following function code is to be developed using the RISC-V 32 assembly:

```
void reverse(char[] ori_str, char[] final_str);
```

which takes two strings as parameters. This function reverse the first string and saves the result in the second one. So, for example, calling this function with "`Hello World`" will store in `final_str` the string "`odnuM aloH`". Consider that the final string has at least the same space reserved as the source string. Also consider that the end of the string is indicated by '`\0`'.

a) Write the algorithm you will use to implement the above routine.
b) Develop, according to the algorithm described in section a), the content of the subroutine using the RISC-V 32 assembly.
c) Develop the function `void print(char[] string)` in RISC-V 32. This function takes as parameter a string of characters and prints it by screen. It must be strictly followed the convention of parameters passage described in the course.
d) Suppose you have the following data segment:

```
.data
      string1: .asciiz "Hello word"
.align 2
      string2: .space 11
```

develop, using the previous routines, a `main` routine that first returns `string1` and stores it in `string2`, and then prints the content of `string2` on the screen. To do this, consider the parameter passage convention.

Solution:

a) A possible algorithm:

- Point pointer 1 at the beginning of the source string
- Source string length = 0
- As long as it does not read a 0
  - Read character
  - Increase length by 1
  - Move source string pointer to next position
- Point pointer 1 at the beginning of the final string
- Back one position source string pointer
- As long as the length is greater than or equal to 0
  - Read character in source string
  - Write final string character
  - Move source string pointer to previous position
  - Move end string pointer to next position
  - Decrease length by 1
- Write end of string in final string

b) The corresponding code can be the following:

```
reverse:
      li    t0 0              # Counter of length
      li    t1 0              # to store characters
      mv    t2 a0
for1:                         # Length measurement
      lb    t1 0(t2)          # Loading a character
      beq   x0 t1 next        # Locate '\0'
      addi  t0 t0 1           # Length increase
      addi  t2 t2 1           # Increment pointer to string 1
      j     for1

next:
```

```
        addi  t2 t2 -1          # Returns to the last position of string1

for2:                           # Loop to reverse the string
        bgt   a0 $t2 end        # Finish reading string1
        lbu   t1 0(t2)          # Loading a character
        sb    t1 0(a1)          # Storage of a character
        addi  t2 t2 -1          # Decrement pointer of string1
        addi  a1 a1  1          # Increment pointer of string2
        j     for2

end:    sb    x0 0(a1)          # close string2
        jr    ra
```

c) The code of the `Print` function is:

```
print:  li  a7 4
        ecall

        jr  ra
```

d)

```
main:
        addi  sp sp -4
        sw    ra 0(sp)

        la    a0 string1
        la    a1 string2
        jal   ra reverse

        la    a0 string2
        jal   ra print

        lw    ra 0(sp)
        addi  sp sp 4

        jr    ra
```

Exercise 16. Consider the `Accounting` routine. This routine accepts two input parameters:

- An array of `floats`.
- The number of elements in the array
- The function returns three values:
  - The number of elements with a value equal to 0.
  - The number of elements corresponding to normalized values other than 0.
  - The number of elements corresponding to non-normalized values (NaN type values are not included).

The following are required:

a) Write in RISC-V32 the `Accounting` function described above. You can use the auxiliary routines that you consider appropriate. You have to strictly follow the convention of parameters passing and stack use, although using the stack frame register is unnecessary.

b) Given the following array:

```
.data
     array: .float 0.0, 0.1, -0.2, 1.0, 1.1, 1.2, 2.0, 2.1, 2.2
```

Encode the code fragment that allows you to correctly invoke the Count function and print the values returned by that function.

Solution:

a) A possible solution could be:

```
IsZero:          # check if f12 (float type argument) is 0.0
            fclass.s  t0, fa0, fa0
            li    t1, 4
            beq   t1, t0, true1    #  es +0
            li    a0, 0        # 0: not zero
            jr    ra
true1:      li    a0, 1        # 1: is zero
            jr    ra

IsNormalized:    #check if f12
                 # (float argument) is normalized
                 # is normalized if:   0  < exponent < 255
            fmv.x.w     t0, fa0
            li    t1, 0x7F800000    # se obtiene el exponente
            beq   x0, t1, falso1
            li    t2, 255
            beq   t1, t2, falso1
            li    a0, 1             # 1: es normalizado
            jr    ra
falso1:     li    a0, 0             # 0: no es normalizado
            jr    ra

IsNotNormalized:
                 #check if f12 (float argument)
                 # is not normalized
                 # exponent equal to 0 and mantissa <> 0

            fmv.x.w     t0, fa0
            li    t1, 0x7F800000   # se obtiene el exponente
            beq   x0, t1, comprueba
            li    a0, 0 #0: no es no normalizado
            jr    ra
```

```
check:     li    t1, 0x007FFFFF  # obtaine mantissa
           beq   x0, t1, falso2
           li    a0, 1 # 1: es no normalizado
           jr    ra
falss2:    li    a0, 0 # 0: no es no normalizado
           jr    ra


Accounting:
           addi  sp, sp, -20
           sw    s0, 16(sp)
           sw    s1, 12(sp)  # number of zeros
           sw    s2,  8(sp)  # number of normalized
           sw    s3,  4(sp)  # number of non-normalized
           sw    ra,  0(sp)

           li    s0, 0   # index for loop
           li    s1, 0         #
           li    s2, 0
           li    s3, 0

loop:      bgt   s0, a1, fin
           lflw  fa0, (a0)   # next element of array

           jal   IsZero
           add   s1, s1, a0

           flw   fa0, (a0)
           jal   IsNormalized
           add   s2, s2, v0

           lflw  fa02, (a0)
           jal   IsNotNormalized
           add   s3, s3, a0

           addi  s0, s0, 1
           addi  a0, a0, 4   # prepare the next
                                   # element of the array
           j     loop

      fin: mv    a0, s1      # number of zeros
           mv    a1, s2      # number of normalized
           mv    t0, s3      # number of non-normalized

           lw    s0, 16(sp)
           lw    s1, 12(sp)
           lw    s2,  8(sp)
           lw    s3,  4(sp)
           lw    ra,   (sp)
           addi  sp, sp, 20

           # number of non-standards at the top of the stack
           addi  $sp, $sp, -4
           sw    t0, (sp)
           jr    ra
```

b) The necessary fragment to invoke the previous function is:

```
# passing arguments
la    a0, matriz
```

```
li    a1,12
jal   Accounting


li    $a7, 1
ecall       # print the number of zeros

mv   a0, a1
ecall       # print the number of normalizaed

# the third result is extracted from the top of the stack
lw    a0, (sp)
addi  sp, sp, 4
ecall       # print the number of non-normalized
```

Exercise 17. Consider the **Add** routine. This routine accepts three input parameters:

- A square matrix of integer type numbers.
- A vector of integer type numbers
- An integer, which represents the dimension of the matrix and the vector

The function adds to each row of the matrix the vector passed as the second argument.

a) Encode in the MIP32 assembly the Add routine described above. You can use the auxiliary function that you consider appropriate. You have to strictly follow the convention of parameter passing and stack use, although it is not necessary to make use of the stack frame register.

b) Given the following definition:

```
.data
     matrix: .word 8, 4, 5
             .word 0, 9, 7
             .word 4, 4, 1

     vector: .word 0, 1, 3
```

Write the code fragment that allows you to correctly invoke the Add function and print the values returned by that function.

Solution:

a)

```
     Add:     mv      t0, a0
              li      t3, 1

        B1:   mv      t1, a1
              li      t2, 1
        B2:   bgt     t2, a2, end2
              lw      t4, 0(t0)
              lw      t5, 0(t1)
              add     t5, t5  t4
              sw      t5, 0(t0)
              addi    t0, t0 4
              addi    t1, t1 4
              addi    t2, t2 1
              j        B2
      end2:   addi    t3, t3 1
              bgt     t3,  a2, end1
              j        B1
      end1:   lw      a0, 0(a0)
              jr       ra
```

b)

```
              # input parameters in $a0, $a1 y $a2
              la      a0, matriz
              la      a1, vector
              li      a2, 3

              # call to function Add
              jal     ra, Add

              li      a7, 1
              ecall
```

Exercise 18. Consider a 32-bit integer array with f rows and c columns, stored in memory row by row. Write, using the MIPS-32 assembly, the XCH routine that accepts the following parameters (in this order):

- The starting address of the matrix
- The number of rows in the matrix
- The number of columns in the matrix
- A number that identifies row i
- A number that identifies row j

This function is responsible for exchanging all the elements that are in rows i by those of the row j, so that in the direction of memory where the k-th element of the row i is placed the k-th element of the row j and vice versa. The function does not return any result.

a) Indicate the passing parameter convention for the XCH routine, that is, where and how the arguments are passed to this function.
b) Code correctly the XCH routine described above. To do this, first write the pseudocode of the Solution used. Consider also that there is not to do error control.
c) Given the following matrix definition:

```
.data
     matrix: .word 00, 01, 02
             .word 10, 11, 12
             .word 20, 21, 22
             .word 30, 31, 32
```

write the code fragment that allows, using the XCH function, to exchange row 1 with row 3 of the previously defined array

Solution:

a) XCH function receives 5 parameters:
   - The starting address of the matrix is passed in \$a0
   - The number of rows in the matrix in \$a1
   - The number of columns in the matrix in \$a2
   - A number that identifies row i in \$a3
   - A number that identifies row j at the top of the stack.

b) A possible pseudocode for the function is the following:

```
void XCH(int m[][], int f, int c, int i, int j) {
   int k, aux;

   if (i == j)
         return;

         for (k = 0; k < c; k++) {
               aux = m[i][k];
               m[i][k] = m[j][k]
               m[j][k] = aux;
         }
         return;
   }
```

A possible assembly fragment would be the one shown below. We are going to use the register \$t0 to store the address of the element m[i][0] , that is, the first element in the row i and the register \$t1 for the address of the first element in the row j, m[j][0]. In general, the element m[k][0] is stored in the memory address $m + k \times c \times 4$

```
XCH:         # in $t4 we store the fifth argument j
             bne   a3, a4, no_iguales
             jr    ra    #row i and j equal, finish
```

```
no_equals:  li    t0, 4
            mul   t0, t0, a2
            mul   t0, t0, a3
            add   t0, t0, a0
            # in $t0 is stored the start address of m[i][0]

            li    t3, 4
            mul   t1, a2, t3
            mul   t1, t1, t4
            add   t1, t1, a0
            # in $t1 is stored the start address of m[j][0]

            # now go through row i and j and exchange the values
            li    $t3, 0     # index used to go through the rows
      loop: bge   t3, a2, end
            lw    t4, 0(t0)
            lw    t5, 0(t1)
            sw    t4, 0(t1)
            sw    t5, 0(t0)
            addi  t0, t0, 4  # address of the next element in the row i
            addi  t1, t1, 4  # address of the next element in the row j
            addi  t3, t3, 1
            j     loop
      end:  jr    $ra
```

c) Considering that the parameters are passed in the following way:

- The starting address of the matrix is passed in \$a0
- The number of rows in the matrix in \$a1
- The number of columns in the matrix in \$a2
- A number that identifies row i in \$a3
- A number that identifies row j at the top of the stack

The necessary fragment to invoke the previous function is:

```
la    a0, matriz
li    a1, 3
li    a2, 3
li    a3, 1
li    a4, 1

jal   ra XCH
```

Exercise 19. Consider a function called `AddOddNumbers` Addition that receives three parameters:

- The first is the starting address of an array of integers.
- The second is an integer value that indicates the number of components in the array.
- The third is an integer value.

The `AddOddNumbers` function modifies the array, adding the value passed as a third parameter to all its odd components. Consider that the vector cannot be empty.

Questions to be answered:

a) Indicate in which registers each of the parameters must be passed to the function.
b) Implement the code of the `AddOddNumbers` function using the MIPS32 assembly and commenting all the lines.
c) Given the following program fragment:

```
.data
      v: .word 3, 4, 5, 9, 6, 4, 1, 8, 2, 7
.text
.globl main
      main:
```

Include in the `main` the necessary assembly statements to invoke the `AddOddNumbers` function so that it adds the number 3 to the odd components of the array `v` defined in the data section, and then print all the elements of the vector.

Solution.

a) The starting address is passed in \$a0, the number of elements in \$a1 and the value to be added in \$a2

b)

```
AddOddNumbers:
            li    t0, 0
            mv    t1, a0
loop:       bgt   t0, a1, end
            lw    t2, 0(t1)
            rem   t3, t2, 2
            beq   x0, t3, next
            add   t2, t2, a2
            sw    t2, 0(t1)
next:
            addi  t0, t0, 1
            addi  t1, t1, 4
            j     loop
end:
            jr    ra
```

c) the body of main is:

```
.data
      v: .word 3, 4, 5, 9, 6, 4, 1, 8, 2, 7

.text
.globl main

main:
addi  sp, sp, -4
sw    ra, 0(sp)


la    a0, v
li    a1, 6
```

```
     li    a2, 5

     jal   ra, AddOddNumbers

     li    t0, 0
     mv    t1, a0

loop2:
     bgt   t0, a1, end2

     lw    a0, 0(t1)
     li    a7, 1
     ecall

     addi  t0, t0, 1
     addi  t1, t1, 4
     j     loop2

end2:
     lw    ra, 0(sp)
     addi  sp, sp, 4

     li    a7, 10
     ecall

     jr    ra
```

Exercise 20. Consider the `CountingZeros` routine. This routine accepts three input parameters:

- A matrix of float type numbers.
- The number of rows in the matrix (N)
- The number of columns in the matrix (M)

The function returns the number of the row (between 0 and N-1) that has more values equal to 0.

Questions to be answered:

a) Implement the function described above. You can use the auxiliary routines that you consider appropriate.
b) Given the following matrix 3x3:

```
.data
     M: .float    0.0, 0.1, -0.2,
                  1.0, 1.1, 1.2,
                  2.0, 2.1, 2.2
```

Implement the code fragment that allows you to correctly invoke the `CountingZeros` function and print the values returned by that function.

Solution:

a) A possible implementation would be:

```
CountingZeros:        # stack (store ra)
                      addi  sp sp -4
                      sw    ra 0(sp)

                      # t60 -> row with more zeros
                      # t0 -> number of zeros in row
                      li     t6 -1
                      li     t0 -1

                      li    t1 0
            b1:       beq   t1 a1 end1
                      li    t2 0
                      li    t4 0 # number of zeros in row
            b2:       beq   t2 a2 end2

                      # t3 -> address of the element
                      mul   t3 t1 a2
                      add   t3 t3 t2
                      mul   t3 t3 4
                      add   t3 t3 a3

                      # t4 is increased if zero
                      lw    t3 0(t3)
                      and   t3 t3 0x7FFFFFFF
                      bne   t3 x0 nozero
                      addi  t4 t4 1
       nozero:
                      bgt   t4 t0 nogreater
                      mv    t0 t4
                      mv    t6 t1
      nogreater:
                      addi  t2 t2 1
                      j     b2
             end2:    addi  t1 t1 1
                      j     b1
```

```
end1:
      lw    ra 0(sp)
      addu  sp sp 4
      mv    a0 t6
      jr    ra
```

d) The code to invoke the function is:

```
# argument passing
la    a0, M
li    a1, 3
li    a2, 3

jal   CountingZeros

li    a7, 1
ecall
```

Exercise 21. Consider the `AddExponents` routine. This routine accepts four input parameters:

- Matrix A of float type numbers.
- Matrix B of float type numbers.
- Matrix C of numbers of type int.
- The number of rows and columns of the three matrixes (N). Matrixes are assumed to be square.

The routine stores in the element (i,j) of the matrix C the addition of the exponents (the real exponents of the number, that is, eliminating the excess or bias that is introduced when it is represented in floating-point) of the numbers A_{ij} and B_{ij}, that is:

```
C[i,j] = Real exponent of A[i,j] + real exponent if B[i,j].
```

Assume that A and B store only normalized numbers or numbers with a value of 0. Note that the value 0 corresponds to exponent 1.

The function returns as value the number of C elements that have taken the value 0.

Questions to be answered:

a) Write the routine described above. You can use the auxiliary routines that you consider appropriate.
b) Write the code to invoke the above function for the following matrixes defined in the data segment:

```
.data
     A:     .float      0.0, 0.1, -0.2,
                        1.0, 1.1, 1.2,
                        2.0, 2.1, 2.2

     B:     .float      0.0, 0.1, -0.2,
                        4.0, 1.1, 1.2,
                        2.0, 8.1, 2.2

     A:     .word       0, 0, 0,
                        0, 0, 0,
                        0, 0, 0
```

Solution:

a) A possible solution would be:

```
EsCero:     #check if $f12 (float argument) is 0.0
            fclass.s    t0, fa0, fa0
            li     t1, 4
            beq    t1, t0, true1
            li     a0, 0       # 0: no es cero
            jr     ra
true1:      li     a0, 1       # 1: es cero
            jr     ra


Exponent:    # returns the exponent of a standardized number other
            # than 0
            fmv.x.w     t0, fa0
            li          t1, 0x7F800000  # obtain the exponent
            srl         a0, t1, 20     # shift to obtain
            jr          ra

  AddExponents:
            # stack (store $ra)

            addi   sp sp -12
            sw     ra 0(sp)
```

```
                sw     s0 4(sp)
                sw     s1 8(sp)

                li     s1 0         # s1 number of elements of C with value 1
                li     t1 0         # t1 -> i

        b1:     beq    t1 a3 end1
                li     t2 0         # t2 -> j

        b2:     beq    t2 a3 end2
                flw    fa0, 0(a1)
                jal    ra, Iszero
                bne    a0,  x0,  no1
                li     t5, 1
                j      cont1
       no1:     jal    ra Exponent
                addi   t5, a0, -127
     cont1:     flw    fa0, 0($a2)
                jal    ra Iszero
                bne    a0,  x0,  no2
                li     t6, 1
                j      cont2
       no2:     jal    ra Exponent
                addi   t6, v0, -127

  cont2:        add    s0, t5, t6
                bne    s0, x0, nozero
                addi   s1, s1, 1
    nozero:     sw     s0, 0(a2)
                addi   a0, a0, 4
                addi   a1, a1, 4
                addi   a2, a2, 4
                addi   t2, t2, 1
                j      b2
     end2:      addi   t1, 1
                j      b1
     end1:      mv     a0, s1
                lw     ra 0(sp)
                lw     s0 4(sp)
                lw     s1 8(sp)
                addi   sp sp 12
                jr     ra
```

f) The necessary fragment to invoke the previous function i:

```
# Arguments passing
la     a0, A
la     a1, B
la     a2, C
li     a3, 3

jal    ra, AddExponents

li     a7, 1
ecall
```

Exercise 22. We have a computer whose data bus has a size of 16 bits, the size of its registers is also 16 bits and it handles an instruction that has three fields: a 5-bit operation code, a field for a 3-bit register and a field for an 8-bit address. Using this information, you are asked to (Reason all answers):

a) What is the maximum number of instructions this computer can have in its instruction set?
b) How many general-purpose registers does the machine have?
c) If we use direct addressing, how much memory can be addressed with this instruction format. Explain what direct addressing is.
d) If we use indirect addressing, how much memory can be addressed with this instruction format. Explain what indirect addressing consists of.
e) What is the range of addressing with relative addressing to the base register for this instruction format?

Solution:

a) Since the operation code that this computer handles has 5 bits, the maximum number of instructions that the computer's instruction set can have is $2^5 = 32$.
b) The field that has the instruction in the statement has three bits to refer to the machine registers, then we can reference $2^3 = 8$ registers.
c) The field we can use to write a memory address has 8 bits, in the direct addressing the memory address we want to access has to be written in the instruction itself, then we can access from the address 0 to the $(2^8 – 1) = 255$.
d) In indirect addressing, the field of the instruction that contains the address actually has the address of the memory position where the real address of the data is, then when we read in memory the data that is in the address field of the instruction we are going to read 16 bits that really indicate the memory address where the data is, so in this case it is possible to address $2^{16} = 65536$ addresses.
e) Since the registers are 16-bit, they can be referenced from position 0 to 65535 (2^{16}-1), and since the address field is 8 bits, it can be represented from the 0 to 255 (2^8-1). Therefore, the range of addressing with relative addressing is from 0 (0+0) to 65790 (65535 + 255).

Exercise 23. Given a 32-bit computer and 16 registers, with an instruction that has the following format:

- 6-bit operation code.
- 1 bit indirection field.
- Two 4-bit fields to represent a register.
- A 17-bit field to represent an address.

The One Bit Indirection field indicates if the Address present in the instruction is an absolute address(0) or relative(1). It is requested:

a) Indicate what is the maximum number of instructions that computer may have in its set of instructions.
b) Is it possible to use absolute direct memory addressing mode with this format? Justify the answer. If it is, indicate what is the size of the addressable memory with this format and indicate what fields of the instruction would be filled and the possible values.
c) Is it possible to use direct addressing mode relative to register? Justify the answer. If it is, indicate what is the size of the addressable memory with this format and indicate which fields of the instruction would be filled and the possible values.
d) Can you use indirect addressing mode? Justify the answer. If so, indicate what is the size of the addressable memory with this format and indicate which fields of the instruction would be filled and the possible values.

Solution:

a) Since the Operation Code field has 6 bits it would be possible to have a 2^6 instructions = 64.
b) Yes. In the absolute direct memory addressing the address we want to access is in the instruction itself, in this case we have an address field with 17 bits, then we could access with this type of addressing to the addresses of $[0 , 2^{17}-1]$, the fields with value would be: OpCode with the corresponding Operation code, Indirection Bit to 0, and address field with the address where the data we want to use is located.
c) Yes. In the direct addressing related to register the address we want to access is within the register indicated in the instruction itself, in this case we have a registration field and therefore we can use this address. The number of addresses that we can access will be all the memory content, because the address we want to access is within the register, i.e., we can access the addresses of $[0 , 2^{32}-1]$, the fields with value would be: OpCode. with the corresponding Operation code, Indirection Bit to 0, and Reg. 1 with the register number containing the address we want to access.
d) Yes. In indirect addressing, the address we want to access is inside memory at the address indicated in the instruction itself, then we would have to access memory at the indicated address, there we would get the address where the data is and we would have to access memory again to access the data. The number of addresses we will be able to access will be the whole memory content, since the address we want to access is inside a memory address that will have 32 bits, that is, we can access the addresses of $[0 , 2^{32}-1]$. The fields with value would be: OpCode. with the corresponding Operation code, Indirection Bit to 1, and Address with the address where the data address we want to obtain is located

Exercise 24. Consider a 32-bit computer with an instruction format like the following:

- Instruction of type 1:
  - Operation code: 8 bits
  - Reg1: 6 bits
  - Reg2: 6 bits
  - Reg3: 6 bits
  - Not used field: 6 bits.
- Instruction of type 2:
  - Operation code: 8 bits
  - Reg1: 6 bits
  - Reg2: 6 bits
  - Immediate value 12 bits.
- Instruction of type 3:
  - Operation code: 8 bits
  - Reg1: 6 bits
  - Address: 18 bits.
- Instruction of type 4:
  - Operation code: 8 bits
  - Address: 24 bits.

Questions:

a) How many registers does this computer have?
b) How many instructions is the machine capable of executing?
c) Assuming that the immediate values are expressed in two's complement, what is the range of numbers that can be expressed in type 2 instructions?
d) Assuming that the computer is capable of executing instructions similar to those of MIPS, indicate in which type of instruction of the previous ones the following instructions could be coded, showing in which field each element of these instructions is stored.

```
lw  t1, 80(t2)
jal 90000
li  t1, 80
```

a) Indicate the addressing modes present in the above instructions.

Solution:

a) $2^6 = 64$ registers.
b) $2^8 = 256$ instructions.
c) $[-2^{11}..2^{11}-1]$
d) The instruction lw in a type 2 instruction; the instruction jal in a type 4 instruction; the instruction li in a type 2 instruction
e) In the instruction lw: register addressing mode and relative addressing; in the instruction jal direct address; and in li, register address and immediate addressing.