Unit 3. Assembly programming
Exercises

Exercise 1. Write a program, using the MIPS 32 assembly language, to calculate the sum of the first 100 numbers. The result must be stored in register $v0.

Exercise 2. Modify the program of the exercise 1 in order to print the result.

Exercise 3. Write a program that reads two integer numbers A and B. The program must to indicate if one of these numbers is multiple of the other one.

Exercise 4. Write a program, using the MIPS32 assembly language that reads a number N and prints the following:

1
1 2
1 2 3
1 2 3 4
…..
1 2 3 4 5 …. N

Exercise 5. What is the sequence of MIPS instructions that allow implementing the following sentence in C language? (a and b are int variables)

a = b + c + 100;

Exercise 6. Write a program that reads two numbers. The program must print what number is the largest.

Exercise 7. Write a program to read a number. The program must indicate if the number is odd or even.

Exercise 8. Write a function with three arguments. In register $a0 receives the init address of a string, in register $a1 receives the ASCII code of a char, and in register $a2 receives the ASCII code of other char. The function must replace in the string any occurrence of the char stored in $a1 by the char stored in register $a2.

Exercise 9. Consider the following program. Write an equivalent program, using the MIPS 32 assembly language.

```
int array[1024];        // variable
global

int function1()
{
  int z;
  int i;

  for (i = 0; i < 1024; i++)
     array[i] = i;

  z = sum(1024);
  print_int(z);
}
```

```
int sum(int n)
{
     int s = 0;
     int i;

     for (i = 0; i < n; i ++)
         s = s + vector[i];

  return s;
}
```

Exercise 10. Write a program, using the MIPS 32 assembly language, similar to the following program written in C.

```
void function1 ()
{
  int z;
  int array[1024];// local variable

  for (i = 0; i < 1024; i++)
     array[i] = i;

  z = sum(array, 1024);
  print_int(z);
}

int sum(int array[],  int n)
{

     int s = 0;
     int i;

     for (i = 0; i < n; i ++)
         s = s + array[i];

      return s;
}
```

Exercise11. Let be a 16 bit computer, with byte addressing and a set of 60 machine instructions. The computer has 8 registers. Show the instruction format for the following hypothetical instruction `ADDV  R1, R2, M`, where R1 and R2 are registers, and M is memory address.

Exercise 12. A 32 bit computer with byte addressing, has 64 machine instructions, and 128 registers. Given the instruction SWAPM addr1, addr2 that swaps the content of the memory address addr1 and addr2.

a) What is the memory space addressable by this computer?
b) What is the instruction format?
c) Write a program fragment, using the MIPS 32 assembly language, equivalent to the previous instruction.
d) If the above instruction must occupy one word, what is the range of addresses that can be addressed by addr1 and addr2?

Exercise 13. Given the following program:

```
.data:
      .align 2                  # align next data to 2^2
      array:  .space 1024       # space reserved for an array of integers of  32 bits
```

Reply:

a) What is the number of components of the previous array?
b) Which are the assembly instructions needed to execute the following high level sentence:
c) `vector[8]  = 30;`
d) Write a program, using the MIPS 32 assembly language, which read a number and copy this number in all components of the above array.

Exercise 14. Given the following program.

```
.text
                .globl main
 main:
                li   $a0,  5
                jal  function
                move $a0, $v0
                li   $v0, 1
                syscall
```

```
                li   $v0, 10
                syscall
function:
                move $t0, $a0
                li   $t1, 0
         loop : beq  $t0, 0, end
                add  $t1, $t1, $t0
                sub, $t0, $t0, 1
                b    loop
          end:  move $v0, $t1
                jr $ra
```

a) What is the value printed by this program (first system call in the program).
b) If register $a0, used for argument passing, stores a value in one-complement, What is the range of numbers that can be stored in this register?

Exercise 15. Let be a function called `Vowels`. This function receives as parameter the init address of a string. The function calculates the number of "`a`" in the string. When the null string is passed, the function returns -1.

a) Write, using the MIPS 32 assembly language, this function.
b) What are the registers used to pass the argument and receive the result.
c) Given the following program:

```
.data
       string:  .asciiz    "Hello"
.text
       .globl main

 main:
```

Include in the `main` function, the assembly instructions needed to invoke the `Vowels` function, and print the value returned by the function.

Exercise 16. Given the following program.

```
.text
                .globl main
 main:
                li   $a0,  5
                jal  function
                move $a0, $v0
                li   $v0, 1
                syscall

                li   $v0, 10
                syscall

function:       li $t0, 10
                bgt $a0, $t0, et1
                li $t0, 10
                add $v0, $t0, $a0
                b et2
     et1:       li $t0, 8
                add $v0, $t0, $a0
     et2:       jr $ra
```

What is the value printed by this program?

Exercise 17. Given a function called `AddValue`,with trhee parameters:

- The first one is the init address of an array.
- The second one is an integer value, which represent the number of components of the array.
- The third one is an integer value.

This function modifies the array, adding the value passed in the third argument to all components of the array.

a) What are the registers used to pass the arguments?
b) Write the code of this function
c) Given the following program:

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

 main:
```

Include in the main function, the assembly instructions needed to invoke the function `AddValue` implemented in b) in order to add the value 5 to all components of the array `v` defined in the previous data section. Then, the program must write all components of the array `v`.

Exercise 18. A 32-bit computer with memory addressed by bytes has 64 registers. We want to design the format for the instructions of this computer, assuming:

a) The computer has 115 instructions.
b) The execution model in register-register.
c) The computer has the following types of instructions:
    - Instructions for transferring data between memory and registers. These instructions use two types of addressing modes: direct, and base-register addressing
    - Arithmetic and logic instructions.
    - Branch instructions. There are two types of instructions: unconditional to a memory address, and conditional branches. For conditional branches, the condition is expressed in a register, and the branch is indicated with PC-relative addressing.

a) Design the format for the instructions of this computer, having into account that all instructions occupy one word.
b) If the displacement used in the instructions with base-register addressing uses two-complement, what is the maximum value for these displacements?
c) If the address used in the unconditional branch instructions use binary code (unsigned), what is the range of addresses for the branch?
d) What is the model of this computer, RISC or CISC?

Exercise 19. A 16-bits computer has a memory addressed by bytes. The computer has 16 registers, and a set of 180 machine instructions. Among these instructions, we can find the following:

`LOADI`	`R, imm`	Load in register R the immediate value imm
`STORE`	`R, addr`	Store in memory (address addr), the value stored in register R
`ADDI`	`R1, R2, imm`	Add the value stored in R2 to imm, and store the result in R1
`ADD`	`R1, R2,`	Add R1 to R2, and the result is stored in R1
`BNEZ`	`R, addr`	If the content of R is 0, branch to addr

The addresses and immediate values are integers of 16 bits. Given the following assembly code fragment:

```
LOADI       R1, 0
LOADI       R2, 3
LOADI       R3, 1
```

```
ETI:  ADD         R1, R3
      ADDI        R3, R3, 2
      ADDI        R2, R2, -1
      BNEZ        R2, ETI
      STORE       R3, 500
```

Answer:

a) Design a possible format for the previous instructions.
b) If the above program is loaded in memory in the address 1000, how many words use this program? What is the content of these memory locations?
c) What is the behavior of this program? What is the content of registers R1, R2, and R3, when finish the execution of this program?

Exercise 20. A 64-bit computer with a memory addressed by bytes and 256 registers, has 190 machine instructions. Consider the following instructions:

- `LOAD Reg, addr`, which stores the content of register reg in the memory address addr .
- `STORE Reg1, (Reg2)`, which stores the content of register Reg1 in the memory address stored in register Reg2.
- `BEQZ Reg1, addr`, if the content of register Reg1 is zero, next instruction to execute is the instruction stored in addr.

Reply:

a) Describes the addressing modes using in these instructions.
b) Indicate a possible instruction format for theses instructions, assuming that all instructions ocuppy one Word.
c) What is the maximum value that can be encoded in LOAD and BEQZ instructions?

Exercise 21. Write a program to read three numbers (A, B, C) and implement the following high level code.

```
if (A == 2 &&  B == 3 || C != 4)
            imprimir (1);
else
      imiprimir (2);
```

Exercise 22. Translate to assembly the following functions.

```
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]);
}
```

Exercise 23. Translate to assembly the following functions:

```
inf f (int k)
{
      int v[100];
      int i = 0;
      int s = 0;

      for (i = 0; i < k; i = i +2)
            v[i] = g(k + 2);

      for (i = 0; i < k; i = i + 2)
            s = s + v[i];

      return (s);
}

int g(int k)
{
            if (k % 2 == 0)
                  return (k * k + 2);
            else
                  return k * (-1);
}
```