# Práctica 2. Introducción al ensamblador MIPS.

## Objetivos.

- Operaciones básicas con instrucciones del ensamblador del MIPS.

## Fundamentos teóricos.

- Repasar la práctica anterior.
- Paginas de la 18 a la 23 del manual.

## Desarrollo.

**P1)** Rellena el campo *resultado* con el valor resultante de ejecutar el programa de la columna de la izquierda.

| Instrucción | Resultado |
|---|---|
| li $t1, 0xa | 0x0000000a |
| li $t2, 16 | 0x00000010 |
| addu $t0,$t1,0 | 0x0000000a |
| subu $t0,$t1,0xb | 0xffffffff |
| li $t1, 0xc | 0x0000000c |
| subu $t0,$t1,0xb | 0x00000001 |
| and $t0,$t1,0xff | 0x0000000c |
| or $t0,$t1,0x00f | 0x0000000f |
| sra $t0,$t1,2 | 0x0000003 |
| multu $t1,$t2 | 0x0000000 0x000000c0 |
| li $t1, 0x7FFFFFFE | 0x7FFFFFFE |
| add $t2, $t1, 5 | Arithmetic overflow |

**P2)** Rellena la siguiente tabla suponiendo que el contenido del registro $t1 = 3 y que $t2 = 0xFFFFFFFD.

| Instrucción | Resultado | Instrucción | Resultado |
|---|---|---|---|
| slt $t0,$t1,0x4 | 1 | sltu $t0,$t1,4 | 1 |
| slt $t0,$t1,0x3 | 0 | sltu $t0,$t1,2 | 0 |
| slt $t0,$t1,0x2 | 0 | sltu $t0,$t1,3 | 0 |
| slt $t0,$t2, -2 | 1 | sltu $t0,$t2, 0xFFFFFFFB | 0 |
| slt $t0,$t2, -3 | 0 | sltu $t0,$t2, 0xFFFFFFFE | 1 |
| slt $t0,$t2, 0xFFFFFFFC | 0 | sltu $t0,$t2, 0xFFFFFFFD | 0 |

**P3)** Escribe las instrucciones ensamblador necesarias para programar las siguientes operaciones.

i) Salta a la etiqueta “eti1”
ii) Salta a la etiqueta “eti1” si el registro $t1 es mayor que el registro $t0
iii) Salta a la etiqueta “eti1” si el registro $t1 es mayor que 0.
iv) Salta a la etiqueta “eti1”, guarda el registro $t2 con el valor 7, y retorna a la instrucción siguiente a la de salto

v) Si $t0 es mayor que $t1, salta a la etiqueta “eti1”, guarda el valor 5 en el registro $t1, y salta a la etiqueta “fin”. Si no, guarda el valor 4 en $t1 y salta a la etiqueta “fin”.
vi) Si el registro $f0 es menor que $f1, salta a la etiqueta “eti1” y guarda el valor 2.1 en $f2. Si no, guarda en $f2 el valor 3.4.

**i)**
```
j eti1
```

**ii)**
```
bgt     $t1, $t0, eti1
```

**iii)**
```
bgtz   $t1, eti1
```

**iv)**
```
jal eti1
…

eti1:
li $t2, 7
jr $ra
```

**v)**
```
bgt $t0, $t1, eti1
li $t1, 4
j fin

eti1:
li $t1, 5
j fin
```

**vi)**
```
c.lt.s $f0, $f1
bc1t   eti1
l.s $f2, dat1    # en segmento de datos poner (dat1:  .float     3.4)
…
(se podría poner un j seguir)

eti1:
l.s $f2, dat2     # en segmento de datos poner (dat2:  .float     2.1)
(y aquí añadir seguir: )
…
```

**P4)** *Partiendo del segmento de datos dado a continuación, escribir el código necesario para:*

*i) cargar los datos numéricos desde memoria en registros (a elegir por el usuario)*
*ii) imprimir la cadena “Hi there “ por consola.*

*iii) Guardar 2 enteros (asumimos de tamaño word) solicitados por teclado a partir de la dirección de field1*

```
                .data   0x10020000
                .byte   0xab
                .half   0x1432
                .word   80000
variable_a:     .float 5.67e7
message1:       .asciiz “Hi there “
                .byte   0xff
field1:         .space 8
```

**i)**

```
lb      $t0, 0x10020000
lh      $t1, 0x10020002
lw      $t2, 0x10020004
l.s     $f0, variable_a
lb      $t3, message1 + 10
```

*ii)*

```
li $v0, 4
la $a0, message1
syscall
```

*iii)*

```
li $v0, 5
syscall
sw $v0, field1
syscall
sw $v0, field1 + 4
```

**P5)** Escribir las instrucciones (no directivas) necesarias para almacenar en memoria a partir de la dirección 0x10020000 los datos 4, 300, 0xdadacafe, 0.128, “salud”, ordenadamente, optimizando el espacio consumido, pero respetando el alineamiento.
NB: Para almacenar el valor decimal sí se permite el uso previo de directivas.

```
.text
li $t0, 4
sb $t0, 0x10020000
li $t1, 300
sh $t1, 0x10020002
li $t2, 0xdadacafe
sw $t2, 0x10020004
l.s $f0, dat1   # en segmento de datos poner (dat1:  .float     0.128)
s.s $f0, 0x10020008
li $t3, 0x73    # “s”
sb $t3, 0x1002000c
li $t3, 0x61    # “a”
```

```
sb $t3, 0x1002000d
li $t3, 0x6c    # “l”
sb $t3, 0x1002000e
li $t3, 0x75    # “u”
sb $t3, 0x1002000f
li $t3, 0x64    # “d”
sb $t3, 0x10020010
```

**P6)** Desarrollar un programa que calcule el volumen de un cilindro. Para ello, deberá:

i) Solicitar por consola al usuario el valor de la altura del cilindro
ii) Solicitar también el valor del radio de la base
iii) Hacer los cálculos correspondientes y sacar por consola el resultado del volumen.
iv) Si es volumen fuera mayor de 33.33, sacar por consola un mensaje informativo. Si fuera menor o igual, sacar el mensaje correspondiente.

NB: Asumir PI = 3.1415

```
.data

pi:                .float 3.1415
umbral:            .float 33.33
pedir_altura:      .asciiz "Introduce la altura del cilindro: "
pedir_base:        .asciiz "Introduce el radio del cilindro: "
cadena_volumen:    .asciiz "\n El volumen es: "
cadena_mayor:      .asciiz "\n El volumen es mayor de 33.33 "
cadena_menor:      .asciiz "\n El volumen es menor o igual a 33.33 "


        .text
        .globl __start

__start:

        la $a0, pedir_altura
        jal print_string

        jal read_float
        mov.s $f1, $f0              # en $f1 altura del cilindro

        la $a0, pedir_base
        jal print_string
```

```
        jal read_float
        mov.s $f2, $f0              # en $f2 radio del cilindro

        l.s $f3, pi
        mul.s $f4, $f2, $f2   # radio^2
        mul.s $f4, $f4, $f3   # pi*radio^2
        mul.s $f4, $f4, $f1   # a*pi*radio^2 → volumen en $f4

        la $a0, cadena_volumen
        jal print_string
        mov.s $f12, $f4
        jal print_float

        l.s $f5, umbral
        c.le.s  $f4, $f5
        bc1t    volumen_menor

        la $a0, cadena_mayor
        jal print_string
        j fin


volumen_menor:
        la $a0, cadena_menor
        jal print_string
        j fin

print_float:
        li $v0, 2
        syscall
        jr $ra


print_string:
        li $v0, 4
        syscall
        jr $ra

read_float:
        li $v0, 6
        syscall
        jr $ra

fin:    li $v0, 10
        syscall
        .end
```