Organización de Computadoras
Guía de Ejercicios Nº 2 - ASM MIPS

1. Escribir la instrucción para asignar el valor 0x4512 al registro **$t0** y ejecutarla.
   a) Copiar el código generado y pasarlo a binario.
   b) Agrupar los bits de acuerdo al tipo de instrucción e identificar los operandos y elementos .

2. Escribir las instrucciones necesarias para almacenar el valor 0x12345678 en el registro **$v1**
   a) Ejecutar paso a paso lo escrito y verificar que funcione correctamente

3. Copiar el siguiente código:

```
.text
    lui $a0,0x1111
    ori $a0,$a0,0x7fff
    lui $a1,0x1111
    ori $a1,$a1,0x8000
```

y compararlo con

```
.text
    lui $a2,0x1111
    addi $a2,$a2,0x7fff
    lui $a3,0x1111
    addi $a3,$a3,0x8000
```

   a) ¿Qué código se genera? ¿A qué conclusión llega? Investigue por qué.
   b) ¿Qué resultado almacenan los registros cuando lo ejecuta paso a paso?
   c) De acuerdo a lo visto ¿cuál de las 2 maneras es la más adecuada para almacenar cualquier valor constante de 32bits?

4. Escribir las instrucciones necesarias para restarle 5 a 20, usando la instrucción **sub** y la instrucción **addi**.
   a) Identifique los códigos generados en ambos casos. ¿Cómo se representan los valores, en particular si hay negativos?

5. Realizar un programa que cargue el valor 234 en el registro **$t0** y lo almacene en la primera posición del segmento de datos (0x10010000)

6. ¿Tiene sentido esta instrucción `add $t1,$t1,$t1` ? ¿Qué hace?

7. Indique para qué sirve esta instrucción: `and $t0,$t0,$0`

8. Codificar en ASM las siguientes expresiones aritméticas de C++ (considerar `int a,b,c,d,e;` y que ningún valor será 0)

```
a)  a = b;
b)  a = b + c;
c)  a = a + 1;
d)  a = c + 2;
e)  a = b + c + d + e;
f)  a = e - c;
g)  a = c + (b - d);
h)  a = (b + c) - (d + e);
i)  a = b * c;
j)  a = b / c;
k)  a = 3 * e;
l)  a = (b - c) * (d - e);
m)  a = b * c * d;
n)  a = (b + c) * (d / e);
```

9. Copiar el programa. Antes de ejecutarlo analizar cada instrucción y tratar de calcular el resultado. Luego verificar los valores obtenidos ejecutando paso a paso el código

```
.text
    ori $t0,$0,0x2476   # inicializo $t0
    ori $t1,$0,0x00FF   # inicializo $t1
    ori $t2,$0,0x8000   # inicializo $t2

    and $s0,$t0,$t1
    andi $s1,$t0,0xAAAA
    or  $s2,$t2,$t0
    ori $s3,$t1,1
    addi $s4,$t1,1
    xor $s5,$t0,$t0
    xori $s6,$s1,343
    xori $s7,$s6,343
    nor $a0,$s1,$t0
```

10. Copiar el programa. Antes de ejecutarlo analizar cada instrucción y tratar de calcular el resultado. Luego verificar los valores obtenidos ejecutando paso a paso el código.

```
.text
    ori $t0,$0,0x2476   # inicializo $t0
    ori $t1,$0,0x00FF   # inicializo $t1
    ori $t2,$0,0x8000   # inicializo $t2
    ori $t3,4           # inicializo $t3
    lui $t4,0xC301

    sll $s0,$t0,16
    sra $s1,$t1,1
    srlv $s3,$t4,$t3
    srav $s2,$t4,$t3
    sll $t5,$t3,1
    add $t5,$t5,$t3
    sllv $s5,$t0,$t5
```

11. Escribir un algoritmo para multiplicar un número de 16 bits que está en la parte baja de **$a0** por 17 sin utilizar las instrucciones **mult** y otro algoritmo que lo multiplique por 24. Dejar los resultados en **$a1**.

12. Poner en 0 los bits 10, 11 y 15 de **$t0**

13. Setear en 1 los 4 bits menos significativos de **$t1**