

Ejercicios del Tema 3. Fundamentos de la programación en ensamblador

Ejercicio 1. Dada la siguiente expresión de un lenguaje de alto nivel

```
int a = 6;
int b = 7;
int c = 3;
int d;
```

```
d = (a+b) * (a+b);
```

Indique un fragmento de código en ensamblador del MIPS 32 que permita evaluar la expresión anterior. El resultado ha de almacenarse en el registro \$t5.

Ejercicio 2. Escriba un programa en ensamblador del MIPS 32 para calcular la suma de los 100 primeros números naturales. El programa debe dejar el resultado en el registro \$v0.

Ejercicio 3. Dado el siguiente fragmento de programa

```
.data
    a: .word 10
    b: .word 5
.text
    li $t0, 2
    lw $t1, a
    lw $t2, b
label1: bgt $t0, $t1, label2
        addi $t2, $t2, 2
        addi $t0, $t0, 1
        b label1
label2: sw $t0, a
        sw $t2, b
```

Indique el valor que tienen los registros \$t0, \$t1 y \$t2 y las posiciones de memoria a y b al final de la ejecución del programa

Ejercicio 4. Modifique el programa anterior para imprimir el resultado por pantalla.

Ejercicio 5. Escriba un programa que lea dos números enteros A y B e indique si uno de ellos es múltiplo del otro.

Ejercicio 6. Escriba un programa en ensamblador del MIPS32 que lea un número N y muestre por pantalla lo siguiente:

```
1
1 2
1 2 3
1 2 3 4
.....
1 2 3 4 5 .... N
```

Ejercicio 7. Indique la secuencia de instrucciones del MIPS necesaria para ejecutar la siguiente sentencia en lenguaje C (asumiendo que a y b son variables de tipo int)

```
a = b + c + 100;
```

Ejercicio 8. Escriba un programa en ensamblador que lea dos números enteros. El programa debe imprimir el mayor de ellos.

Ejercicio 9. Escriba un programa en ensamblador del MIPS que lea un número e indique si el número es par o impar.

Ejercicio 10. Escriba un programa en ensamblador que lea un número N par y calcule la siguiente suma: $2 + 4 + 6 + \dots + N$. Asuma que siempre se introduce un número par. El programa imprimirá el resultado

Ejercicio 11. Escriba una función reciba en el registro \$a0 la dirección de comienzo de una cadena de caracteres, en el registro \$a1 el código ASCII de un carácter y en el registro \$a2 el código ASCII de otro. La función debe sustituir en la cadena todas las ocurrencias del carácter almacenado en \$a1 por el carácter almacenado en \$a2.

Ejercicio 12. Considere una función denominada `func` que recibe tres parámetros de tipo entero y devuelve un resultado de tipo entero, y considere el siguiente fragmento del segmento de datos:

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

.text
```

Indique el código necesario para poder llamar a la función anterior pasando como parámetros los valores de las posiciones de memoria a, b y c. Una vez llamada a la función deberá imprimirse el valor que devuelve la función.

Ejercicio 13. Dado el siguiente fragmento de programa

```
.data
    a: .word 10
    b: .word 5
    .align 2
    v: .space 800
.text

. . .
```

Si v representa un array de número enteros. Indique:

- El número de elementos del vector.
- Indique las instrucciones en ensamblador necesarias para ejecutar la siguiente sentencia de alto nivel `v[20] = v[30]`;
- Indique las instrucciones en ensamblador necesarias para ejecutar: `v[10] = b`;

Ejercicio 14. Dado el siguiente fragmento de código escrito en C, escriba utilizando el ensamblador del MIPS 32 el código de la función equivalente.

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

Utilizando la función en ensamblador anterior implemente el código de la siguiente función utilizando el ensamblador del MIPS 32.

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

```

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

```

Para el desarrollo de este ejercicio ha de seguirse la convención de paso de parámetros vista en el temario de la asignatura.

Ejercicio 15. Dado el siguiente fragmento de programa, escriba un programa equivalente utilizando el ensamblador del MIPS 32.

```

int vector[1024];          // variable global

int funcion1()
{
    int z;
    int i;

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

    z = suma(1024);
    print_int(z);

int suma(int n)
{
    int s = 0;
    int i;

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

    return s;
}

```

Ejercicio 16. Dado el siguiente fragmento de programa, escriba un programa equivalente utilizando el ensamblador del MIPS 32.

```

void funcion1 ()
{
    int z;
    int vector[1024];      // variable local

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

    z = suma(vector, 1024);
    print_int(z);
}

int suma(int vector[], int n)
{
    int s = 0;
    int i;

    for (i = 0; i < n; i++)

```

```

        s = s + vector[i];
    }
    return s;
}

```

Ejercicio 17. Considere un computador de 32 bits con un juego de 140 instrucciones máquina y un banco con 64 registros.

Considere la siguiente instrucción máquina: sumar R1, R2, dirección. La instrucción suma el contenido del registro R1, con el contenido del registro R2 y deja el resultado en la posición de memoria dada por dirección.

Se pide:

- Indique de forma razonada un posible formato para la instrucción anterior.
- Utilizando el ensamblador del MIPS 32, indique un fragmento de código que sea equivalente a la instrucción sumar anterior.

Ejercicio 18. Sea un computador de 16 bits, que direcciona la memoria por bytes y que incluye un repertorio con 60 instrucciones máquina. El banco de registros incluye 8 registros. Indicar el formato de la instrucción ADDV R1, R2, M, donde R1 y R2 son registros y M es una dirección de memoria.

Ejercicio 19. Sea un computador de 32 bits, que direcciona la memoria por bytes. El computador incluye 64 instrucciones máquina y 128 registros. Considere la instrucción SWAPM dir1, dir2, que intercambia el contenido de las posiciones de memoria dir1 y dir2. Se pide:

- Indicar el espacio de memoria direccionable en este computador.
- Indicar el formato de la instrucción anterior.
- Especifique un fragmento de programa en ensamblador del MIPS 32 equivalente a la instrucción máquina anterior.
- si se fuerza a que la instrucción quepa en una palabra, qué rango de direcciones se podría contemplar considerando que las direcciones se representan en binario puro.

Ejercicio 20. Dada la siguiente sección de datos de un programa en ensamblador:

```

.data:
    .align 2                # alinea el siguiente dato a un límite de 22
    vector: .space 1024    # espacio reservado para un vector de números enteros de 32 bits

```

Se pide:

- Indique de forma razonada el número de componentes del vector.
- Indique las instrucciones en ensamblador necesarias para ejecutar la siguiente sentencia de un lenguaje de alto nivel: `vector[8] = 30;`
- Escriba un fragmento de código en ensamblador que lea un número entero y asigne ese número entero a todas las componentes del vector anterior.

Ejercicio 21. Dado el siguiente fragmento de programa en ensamblador.

```

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

    li   $v0, 10
    syscall

funcion:

```

```

        move $t0, $a0
        li   $t1, 0
bucle: beq  $t0, 0, fin
        add  $t1, $t1, $t0
        sub, $t0, $t0, 1
        b    bucle
fin:    move $v0, $t1
        jr  $ra

```

Se pide :

- Indicar de forma razonada el valor que se imprime por pantalla (primera llamada al sistema del código anterior).
- Si en el registro `$a0`, que se utiliza para el paso de parámetros a la función, el valor que se almacena se representa en complemento a uno, ¿qué rango de números podrían pasarse a la función?

Ejercicio 22. Considere una función denominada `Vocales`. A esta función se le pasa como parámetro la dirección de inicio de una cadena de caracteres. La función calcula el número de veces que aparece el carácter 'a' (en minúscula) en la cadena. En caso de pasar la cadena nula la función devuelve el valor -1. En caso de que la cadena no tenga ninguna 'a', la función devuelve 0. Se pide:

- Programar utilizando el ensamblador del MIPS32 el código de la función `Vocales`.
- Indique en qué registro se ha de pasar el argumento a la función y en qué registro se debe recoger el resultado.
- Dado el siguiente fragmento de programa:

```

.data
cadena: .asciiz "Hola"
.text
.globl main

main:

```

incluya en el `main` anterior, las sentencias en ensamblador necesarias para poder invocar a la función `Vocales` implementada en el apartado a) e imprimir por pantalla el valor que devuelve la función. El objetivo es imprimir el número de veces que aparece el carácter 'a' en la cadena "Hola".

Ejercicio 23. Dado el siguiente fragmento de programa en ensamblador.

```

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

    li   $v0, 10
    syscall

funcion:
    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

```

Se pide :

- a) Indicar de forma razonada el valor que se imprime por pantalla (primera llamada al sistema del código anterior).

Ejercicio 24. Considere una función denominada `SumarValor`. A esta función se le pasan tres parámetros:

- a. El primer parámetro es la dirección de inicio de un vector de números enteros.
- b. El segundo parámetro es un valor entero que indica el número de componentes del vector.
- c. El tercer parámetro es un valor entero.

La función `SumarValor` modifica el vector, sumando el valor pasado como tercer parámetro a todas las componentes del vector. Se pide:

- d) Indique en qué registros se han de pasar cada uno de los parámetros a la función.
- e) Programar utilizando el ensamblador del MIPS32 el código de la función `SumarValor`.
- f) Dado el siguiente fragmento de programa:

```

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

main:

```

incluya en el `main` anterior, las sentencias en ensamblador necesarias para poder invocar a la función `SumarValor` implementada en el apartado b) de forma que sume a las componentes del vector `v` definido en la sección de datos, el número 5. Implemente a continuación de la llamada a la función, las sentencias en ensamblador que permitan imprimir todas las componentes del vector.

Ejercicio 25. Sea un computador de 32 bits con 48 registros y 200 instrucciones máquina. Indique el formato de la instrucción hipotética `beqz $t1, $t2, dirección` donde `$t1` y `$t2` son registros y `dirección` representa una dirección de memoria.

Ejercicio 26. Indique una instrucción del MIPS que incluya el modo de direccionamiento relativo a registro base. ¿En qué consiste este direccionamiento?

Ejercicio 27. Sea un computador de 32 bits con direccionamiento de la memoria por bytes. El computador dispone de 64 registros. Se quiere diseñar un formato para las instrucciones de este computador teniendo en cuenta:

- a) El computador dispone de 115 instrucciones de máquina.
- b) El modelo de ejecución del computador es registro-registro.
- c) Dispone de los siguientes tipos de instrucciones:
 - Instrucciones de transferencia entre un registro y la memoria. Para estas instrucciones se permiten dos tipos de direccionamiento: direccionamiento directo y direccionamiento relativo a registro base.
 - Instrucciones aritméticas y lógicas
 - Instrucciones de bifurcación. Existen dos tipos de instrucciones de bifurcación: incondicional a una dirección de memoria y bifurcaciones condicionales. Para las bifurcaciones condicionales, la condición se expresa en un registro y el salto se indica con direccionamiento relativo a contador de programa.

Se pide:

- a) Diseñe el formato de instrucciones de este computador, teniendo en cuenta que todas las instrucciones deben caber en una palabra.

- Asumiendo que el desplazamiento utilizado en las instrucciones con direccionamiento relativo a registro base utiliza complemento a 2. Indique el valor máximo de los desplazamientos que se pueden realizar.
- Si la dirección utilizada en las instrucciones de bifurcación incondicional se representan en binario natural, indique el rango de direcciones posibles para el salto.
- Según el juego de instrucciones diseñado, ¿qué modelo de computador sería, RISC o CISC? Razone su respuesta en tres líneas como mucho.

Ejercicio 28. Sea una CPU de 16 bits, que se encuentra conectada a una memoria, que se direcciona por bytes. El banco de registros incluye 16 registros de propósito general visibles al usuario. El juego de instrucciones de este computador incluye 180 instrucciones de máquina entre las que se encuentran las siguientes:

LOADI	R, inm	Carga en el registro R el valor inmediato inm
STORE	R, dir	Almacena el contenido de R en la dirección de memoria dir
ADDI	R1, R2, inm	Suma el contenido de R2 con el valor inm y deja el resultado en R1
ADD	R1, R2,	Suma el contenido de R1 con el de R2 y deja el resultado en R1
BNEZ	R, dir	Si el contenido del registro R es distinto de 0 salta a dir

Las direcciones y los valores inmediatos que se incluyen en las instrucciones son enteros de 16 bits. Dado el siguiente fragmento de código:

```

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

```

Se pide:

- Indique un formato posible para las instrucciones anteriores.
- Suponiendo que el programa anterior se carga en la posición de memoria 1000. ¿Cuántas palabras de memoria ocupa el programa anterior? Indique el contenido de esas palabras de memoria.
- ¿Qué hace el programa anterior y qué resultado genera? Indique el contenido de los registros R1, R2 y R3 una vez acabada la ejecución de fragmento de código anterior.

Ejercicio 29. Considere un computador de 64 bits que direcciona la memoria por bytes, con un banco de 256 registros y que dispone de un repertorio de instrucciones compuesto por 190 instrucciones. Dada las tres siguientes instrucciones:

- LOAD Reg, direccion, que almacena el contenido de una posición de memoria en el registro Reg.
- STORE Reg1, (Reg2), que almacena en el registro Reg1 el dato almacenado en la posición de memoria almacenada en Reg2.
- BEQZ Reg1, direccion, instrucción de salto condicional. Si el contenido del registro Reg1 es cero, la siguiente instrucción a ejecutar será la almacenada en direccion.

Se pide:

- Indique el modo o modos de direccionamiento presentes en cada una de las instrucciones anteriores.
- Indique un posible formato para cada una de las instrucciones anteriores, asumiendo que en este computador todas las instrucciones ocupan una palabra.
- De acuerdo al formato indicado en el apartado anterior, ¿cuál es el valor máximo de la dirección que se puede especificar en la instrucción LOAD y BEQZ?

Ejercicio 30. Considere una CPU de 16 bits que dispone de un banco de 6 registros, un espacio de direccionamiento de 64KB y se encuentra conectada a una memoria, que se direcciona por bytes y que requiere tres ciclos para las operaciones de lectura y escritura. La unidad aritmético-lógica es capaz de realizar 20 operaciones aritméticas y lógicas (sumar, restar, multiplicar, dividir, incrementar, disminuir en uno, etc.).

Se pide:

1. Asumiendo un código de operación de longitud fija, diseñar el formato de instrucción que permita construir un juego de instrucciones con al menos:
 - a. 40 instrucciones aritmético-lógicas tipo registro a registro.
 - b. 5 Instrucciones de direccionamiento absoluto.
 - c. 5 Instrucciones de direccionamiento relativo a registro base con un desplazamiento máximo de 128 bytes.
 - d. 2 Instrucciones de direccionamiento indirecto absoluto.

Ejercicio 31. Se dispone de un computador de 32 bits que direcciona la memoria por bytes y que consta de un banco de 32 registros (R0, R31). El computador dispone del siguiente juego de instrucciones.

Instrucción	Descripción	Formato
add R1, R2	$R1 \leftarrow R1 + R2$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 0 0 0 R1 R2 X X X X X X X X X X X X X X X X </div>
sub R1, R2	$R1 \leftarrow R1 - R2$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 0 0 1 R1 R2 X X X X X X X X X X X X X X X X </div>
AND R1, R2	$R1 \leftarrow R1 \text{ and } R2$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 0 1 0 R1 R2 X X X X X X X X X X X X X X X X </div>
OR R1, R2	$R1 \leftarrow R1 \text{ OR } R2$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 0 1 1 R1 R2 X X X X X X X X X X X X X X X X </div>
lw R1, (R2)	$R1 \leftarrow \text{MP}[R2]$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 1 0 0 R1 R2 X X X X X X X X X X X X X X X X </div>
sw R1, (R2)	$\text{MP}[R2] \leftarrow R1$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 0 0 0 1 0 1 R1 R2 X X X X X X X X X X X X X X X X </div>
b desp	$\text{PC} \leftarrow \text{PC} + \text{desp}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 0 0 0 X X X X X X X X X X desp </div>
j R2	$\text{PC} \leftarrow R2$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 0 0 1 X X X X X R2 X X X X X X X X X X X X X X X X </div>
beq R1, R2, desp	$\text{Si } R1 == R2$ $\text{PC} \leftarrow \text{PC} + \text{desp}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 0 1 0 R1 R2 desp </div>
bneq R1, R2, desp	$\text{Si } R1 \neq R2$ $\text{PC} \leftarrow \text{PC} + \text{desp}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 0 1 1 R1 R2 desp </div>
bgt R1, R2, desp	$\text{Si } R1 > R2$ $\text{PC} \leftarrow \text{PC} + \text{desp}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 1 0 0 R1 R2 desp </div>
blt R1, R2, desp	$\text{Si } R1 < R2$ $\text{PC} \leftarrow \text{PC} + \text{desp}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 0 0 1 0 1 R1 R2 desp </div>
lj, R1, numero	$R1 \leftarrow \text{numero}$	<div style="text-align: center; font-size: small; margin-bottom: 5px;">31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</div> <div style="border: 1px solid black; padding: 2px; display: flex; justify-content: space-between;"> 1 1 0 1 0 1 R1 X X X X X numero </div>

Donde numero y desp se representan en complemento a dos. Dado el siguiente fragmento de programa

dirección	contenido
0x00000010	11010100100000000000000000000001
0x00000014	11010100001000000000000000000000
0x00000018	110101000100000000000000000001010
0x0000001C	11010100011000000000000000000000
0x00000020	100100000010001000000000000001100
0x00000024	00000000001001000000000000000000
0x00000028	00000000011001000000000000000000
0x0000002C	10000000000000001111111111110000
0x00000030	11010100001000000000000010000000
0x00000034	00010100011000010000000000000000
0x00000038	
....	
0x00000080	00000000000000000000000000000000
0x00000084	00000000000000000000000000000000

Considerando que el valor del PC es 0x00000010. Se pide:

- a) Obtenga el programa ensamblador correspondiente.
- b) Ejecute el programa anterior hasta que el contador de programa valga 0x00000038.
- c) Una vez calculado indique el contenido de los registros que se han modificado en el programa y el contenido de las posiciones de memoria 0x00000080 y 0x00000084.