INTRODUCCIÓN A LOS COMPUTADORES

PRÁCTICA 3.2

El objetivo de esta sesión es repasar lo visto hasta ahora y continuar profundizando en el lenguaje ensamblador del MIPS con nuevas instrucciones y estructuras.

1. Teclea el siguiente programa:

```
.data                # Espacio para datos en memoria.
cadena: .asciiz "hola mundo"


.text                # Texto obligatorio para comenzar un programa
.globl main          # indicando la rutina principal
main:                # comienzo del programa

la $a0, cadena          # Guardamos en $a0 la dirección de la cadena
addi $v0, $0, 4        # Impresión de la cadena en pantalla
syscall                 # llamada al sistema

addi $v0, $0, 10        # Finalización del programa
syscall                 # llamada al sistema
```

Repaso:

Las cadenas de texto están compuestas por una sucesión de letras u otros caracteres rematadas, normalmente, por un byte con valor 0x00. Cada carácter ocupa 1 byte y está codificado, normalmente también, en código ASCII. Las cadenas de texto no se pueden introducir en un registro porque en los 32 bits del registro no cabrían todas las letras. En su lugar, gestionamos las cadenas mediante su dirección en memoria, que es la dirección del primer carácter de la cadena.

- ¿A partir de qué dirección de memoria se almacena la cadena *hola mundo*? ¿Dónde está situada, en el segmento de datos o en el segmento de texto?
- ¿En qué dos instrucciones se traduce la pseudo-instrucción `la`?
- ¿En qué dirección de memoria se encuentra la instrucción `addi`? ¿Cuál es la codificación binaria de esta instrucción? Identifica los campos dentro del formato al que pertenezca esta instrucción. ¿Cómo se codifica el inmediato dentro de la instrucción?
- ¿Cuál es el contenido de los registros `$a0` y `$v0` al final de la ejecución? Indícalo en hexadecimal, binario y decimal.

Instrucciones de salto:

2. Realizar saltos incondicionales con la instrucción `j`. Para indicar a qué instrucción se salta es necesario utilizar una etiqueta. Así, haciendo uso de esta instrucción `j`:

   - Modifica el programa anterior para que imprima la cadena indefinidamente
   - ¿En qué dirección de memoria se almacena la nueva instrucción de salto?
   - ¿A qué dirección de memoria apunta la etiqueta destino del salto?
   - ¿Cuál es la codificación binaria de la instrucción de salto? ¿Cómo se codifica la dirección de salto dentro de la instrucción? Calcula la dirección destino del salto a partir de su codificación binaria y del contador de programa, y comprueba que el resultado coincide con la dirección de la etiqueta destino. ¿Qué modo de direccionamiento utiliza esta instrucción de salto?

3. Realizar saltos condicionales con las instrucciones `beq` y `bne`. Estos son útiles para implementar estructuras de control tales como if-else y bucles. Para implementar un bucle podemos usar un registro que iremos decrementando hasta que llegue a 0.

   - Modifica el programa inicial para que la cadena se imprima solo 3 veces
   - ¿En qué dirección de memoria se almacena la nueva instrucción de salto?
   - ¿A qué dirección de memoria apunta la etiqueta destino del salto?
   - ¿Cuál es la codificación binaria de la instrucción de salto? ¿Cómo se codifica la dirección de salto dentro de la instrucción? Calcula la dirección destino del salto a partir de su codificación binaria y del contador de programa, y comprueba que el resultado coincide con la dirección de la etiqueta destino. ¿Qué modo de direccionamiento utiliza esta instrucción de salto?

Recorrer y sumar un vector de números:

4. Carga en el **Simula3MS** el código *P31b.s* del CampusVirtual. Completa el código para que sume todos los elementos de un vector dado de tamaño *size* y deja el resultado en memoria (*suma*):

```
.data
numeros: .word 8, 7, 9, 6, 2, 3
size: .word 6
suma: .word 0


.text
.globl main

main:

        la $a0, —           # Puntero al vector numeros
        la $a1, size        # Cargamos la direccion del tamaño
        lw —,0( —)          # Cargamos el tamaño del vector en $a1

        # lazo para hacer la suma
        add $t0, $0, $0     # Inicializamos $t0 a 0, para usar como sumatorio

lazo:
        —                   # cargamos en $t1 cada número, usando el puntero
        add $t0, $t0, $t1   # sumamos el nuevo número cargado en la instrucción anterior
        —                   # movemos el puntero para que apunte al siguiente número
        addi $a1, $a1, -1   # decrementamos el número de elementos restantes
        —                   # salto condicional, comprueba si ya se han sumado todos

        la $a3, suma        # cargamos la dirección donde dejar el resultado
        —                   # almacenamos el resultado en memoria

exit:
        addi $v0, $0, 10    # finalización del programa
        syscall
```

Poner en práctica todo lo anterior:

5. Escribe un código que cambie los 1 de un vector por 10. La salida del siguiente ejemplo sería $\{10, 2, 5, 6, 10, 7, 8, 10, 9\}$

```
.data
numeros: .word 1, 2, 5, 6, 1, 7, 8, 1, 9
size: .word 9


.text
.globl main

main:

—
exit:
        addi $v0, $0, 10 # finalización del programa
        syscall
```