Práctica 10. Instrucciones de salto, lectura de cadenas y datos alineados.

1. Instrucciones de salto

El lenguaje ensamblador no dispone de las clásicas estructuras de control de flujo de programa (`if`, `for`, `while`...) presentes en los lenguajes de alto nivel. Normalmente, como casi siempre cuando se trata de programar en ensamblador, hay que utilizar mecanismos de bajo nivel para implementar estas estructuras "a mano".

Las instrucciones que permiten saltar a otro punto del programa en función del valor de un registro, o bien en función del resultado de comparar dos registros entre sí son las instrucciones de *salto condicional.* También se ofrece la posibilidad de saltar a un punto determinado del código sin necesidad de que se cumpla ninguna condición. Las instrucciones que lo permiten son las de *salto incondicional.*

Por último, hay un tercer tipo de instrucciones de salto. Son las instrucciones de *llamadas a subrutinas*, que utilizaremos en la práctica siguiente.

En general, las instrucciones de salto harán referencia a otras instrucciones a las que se quiera "saltar". Para hacer referencia a instrucciones del segmento de texto se utilizarán únicamente etiquetas que precedan a las instrucciones de destino (las cuales estarán en el segmento de texto).

- Instrucciones de salto incondicional

Jump: `j etiqueta`
La instrucción comienza con la letra `j`. Salta incondicionalmente a la instrucción especificada por la etiqueta.

- Instrucciones de salto condicional

Saltar si igual: `beq $rs, $rt, etiqueta`
Compara el contenido de `$rs` con el de `$rt`. Si ambos registros tienen el mismo valor, el flujo del programa continúa en la instrucción referenciada mediante la `etiqueta`. Si no, el flujo del programa continúa en la siguiente instrucción.

Saltar si distinto: `bne $rs, $rt, etiqueta`
Compara el contenido de `$rs` con el de `$rt`. Si no son iguales, salta a `etiqueta`.

Saltar si igual a cero: `beqz $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si `$rs` vale cero, salta a `etiqueta`.

Saltar si distinto de cero: `bnez $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si el registro **no** vale cero, salta a `etiqueta`.

Saltar si mayor que cero: `bgtz $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si el registro es mayor que cero, salta a `etiqueta`.

Saltar si menor que cero: `bltz $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si el registro es menor que cero, salta a `etiqueta`.

Saltar si mayor o igual que cero: `bgez $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si el registro es mayor o igual que cero, salta a `etiqueta`.

Saltar si menor o igual que cero: `blez $rs, etiqueta`
Compara el contenido de `$rs` con el valor cero. Si el registro es menor o igual que cero, salta a `etiqueta`.

Saltar si mayor (pseudoinstrucción): `bgt reg1, reg2, etiqueta`
Compara el contenido de `reg1` con `reg2` y, si `reg1 > reg2`, salta a `etiqueta`.

Saltar si mayor o igual (pseudoinstrucción): `bge reg1, reg2, etiqueta`
Compara el contenido de `reg1` con `reg2` y, si `reg1 >= reg2`, salta a `etiqueta`.

Saltar si menor (pseudoinstrucción): `blt reg1, reg2, etiqueta`
Compara el contenido de `reg1` con `reg2` y, si `reg1 < reg2`, salta a `etiqueta`.

Saltar si menor o igual (pseudoinstrucción): `ble reg1, reg2, etiqueta`
Compara el contenido de `reg1` con `reg2` y, si `reg1 <= reg2`, salta a `etiqueta`.

2. Ejemplos

A continuación, se ofrecen tres listados a modo de ejemplo. En el Listado 1 se declara una cadena y, a continuación, se reserva memoria para almacenar 32 bits (4 bytes, el tamaño de un registro). Si realmente se va a utilizar este espacio para guardar el contenido de un registro, esta dirección tiene que estar alineada, es decir, tiene que ser múltiplo de 4. Sin embargo, no hay garantía de que sea así, porque cada cadena ocupa varios bytes y resulta muy pesado contarlos a mano. Se soluciona empleando la directiva `.align 2` antes de reservar los 4 bytes de espacio. Así, sea cual sea la dirección en la que termine la cadena, el simulador reserva el espacio a partir de la primera dirección múltiplo de 4 disponible.

El Listado 1 lee un número del teclado y lo almacena en el segmento de datos:

Listado 1.

```
        .data           # comienzo del segmento de datos

cad1:   .asciiz "Introduzca un número entero: "
        .align 2 # como el siguiente dato es un espacio de tamaño palabra de 32
                 # bits, tomamos la precaución de alinearlo. Esto es así porque
                 # nada nos garantiza que la cadena anterior termine
                 # exactamente en la dirección anterior a una múltiplo de 4
result: .space 4        # reservamos sitio para guardar 4 bytes
                        # (un registro completo)

        .globl main     # declaración de la etiqueta main como global

        .text           # comienzo del segmento de texto

main:

        li $v0, 4       # llamada al sistema que imprime la cadena que se
        la $a0, cad1    # encuentra en la dirección especificada por la
        syscall         # etiqueta 'cad1:'

        li $v0, 5       # llamada al sistema read_int
```

```
        syscall
        move $t0, $v0   # guardamos en $t0 el entero leído

        sw $t0, result  # almacenamos en memoria el contenido de $t0

        li $v0, 10      # llamada al sistema de fin del programa
        syscall
```

El Listado 2 lee de la memoria dos valores, calcula su suma y la almacena en la memoria:

Listado 2.

```
        .data           # comienzo del segmento de datos

num1:   .word 0x1924  # valor 1
num2:   .word 2304    # valor 2
result: .space 4      # espacio para almacenar el resultado
        .globl main   # declaración de la etiqueta main como global

        .text         # comienzo del segmento de texto

main:
        lw $t0, num1    # cargamos desde la memoria en $t0 uno de los valores
        lw $t1, num2    # cargamos desde la memoria en $t1 el otro valor
        add $t1,$t0,$t1 # calculamos la suma en $t1
        sw $t1, result  # almacenamos en memoria el resultado de la suma

        li $v0, 10      # llamada al sistema de fin del programa
        syscall
```

En el ejemplo anterior no ha habido necesidad de alinear los datos. Esto es así porque el segmento de datos contiene dos `.word` (4 bytes cada una) y un espacio para otros 4 bytes. Evidentemente, dado que el segmento de datos comienza por defecto en la dirección `0x10010000`, el primer `.word` comenzará en esta dirección, el segundo `.word` comenzará a partir de `0x10010004`, y el espacio `result` comenzará a partir de `0x10010008`. Todas ellas son direcciones múltiplo de 4.

En el Listado 3 se lee una cadena por teclado. Previamente hay que reservar espacio para la cadena con la directiva `.space`. Después se calcula su longitud y se muestra el resultado por pantalla. El carácter `\0` corresponde con el valor 0 almacenado en el registro `$zero`.

Listado 3.

```
        .data           # comienzo del segmento de datos

cad:      .asciiz "Introduce una cadena"
cadleida: .space 100  # Reservo sitio para 100 letras
cadresult: .asciiz "La longitud es "
        .globl main   # declaración de la etiqueta main como global

        .text         # comienzo del segmento de texto

main:
        li $v0, 4   # llamada al sistema para imprimir cadena cad
        la $a0, cad
        syscall

        li $v0, 8         # llamada al sistema para leer cadena y
        la $a0, cadleida  # almacenarla en cadleida
        li $a1, 100       # de como máximo 100 caracteres
        syscall

        li $t0, 0     # Inicializo mi contador de la longitud
        la $t1, cadleida   # Creo un puntero a la cadena leída (a la 1ª letra)

  bucle:
        lb $t2, ($t1)   # Leo una letra de la cadena (el contenido del puntero)
        beq $t2, $zero, fin   #Compruebo si es el carácter \0 (he acabado)
```

```
        addi $t1, $t1, 1        #Muevo el puntero a la dir. Siguiente (sgte. Letra)
        addi $t0, $t0, 1        #contador++;
        j bucle                 #sigo recorriendo la cadena

fin:
        li $v0, 4
        la $a0, cadresult       # Imprimo la cadena cadresult
        syscall

        li $v0, 1
        move $a0, $t0           # Imprimo el resultado (el contador, es decir $t0)
        syscall

        li $v0, 10              # Fin del main
        syscall
```

3. Ejercicios

1. Realice un programa que lea una cadena por teclado y calcule cuántas letras ‘a’ tiene, mostrando el resultado por pantalla. Para ello hay que declarar la letra ‘a’ en el segmento de datos como `letra: .byte 'a'`

2. Realice un programa que pida un número *n* por teclado y calcule la suma 1+ 2 + 3 + … + n y la muestre por pantalla.

3. Considérese el siguiente fragmento de código que representa el segmento de datos:

```
.data

datos1: .half 4,7,2
        .align 2
        .word 1, -5
        .byte 4
        .align 1
        .half 3
        .align 2
        .word -3
        .space 2
        .byte 'c'
datos2: .byte 2,3,5,7
```

   Escriba un programa que, utilizando los modos de direccionamiento realice las siguientes operaciones sin modificar el segmento de datos:

   - Calcular la suma del primer **word** colocado a partir de la etiqueta `datos1` y del **half** que tiene valor 3 y colocar este resultado en el espacio de 2 bytes resaltado en negrita. Imprimir y comprobar el valor.
   - Utilizando la etiqueta `datos2`, almacenar el carácter ASCII ‘e’ en el byte resaltado en negrita e inicializado a 4. Imprimir y comprobar el valor.