Trabajo Nro. 2 de MIPS

Tema: Evaluación de Expresiones
Integrante: Facundo Ferrari

Repositorio de GitHub de las resoluciones pedidas en el práctico aquí.

```
.data
dato1: .word 30
dato2: .word 40
res:   .space 1
        .text
main:  lw $t0,dato1($0) # cargar dato1 en t0
     lw $t1,dato2($0) # cargar dato2 en t1
     slt $t2,$t0, $t1 # poner a 1 $t2 si t0<t1
     sb $t2,res($0) # almacenar $t2 en res
```

Cuestión 1.1: ¿Qué valor se carga en la posición de memoria res?

El valor que se carga en la posición de memoria es el número 1. Ya que la evaluación de si 30 es menor que 40 es positiva. Por lo tanto la instrucción guarda un 1 en la posición res como está indicado.

```
slt $t2,$t0, $t1 # poner a 1 $t2 si t0<t1
```

Cuestión 1.2: ¿Qué valor se carga ahora en la posición de memoria res?

Cambiando dato1 por 50 y dato2 por 20, tenemos como resultado que ahora en la posición de memoria res se encuentra un 0. Porque anteriormente **slt** vimos que en base a los valores que se pongan en los registros, nos va a arrojar un 0 o un 1 dependiendo si se cumple o no la condición de **menor que**. Así que como antes se cumplía la condición y nos arrojaba un 1 en la posición de memoria deseada, como ahora sabemos que no se cumple, aparece un 0 en dicha posición. Y lo vemos reflejado en **$t2** nuevamente, ya que esa parte del código no sufrió cambios.

```
dato1:      .word 50
dato2:      .word 20
```

Código Modificado.

Cuestión 1.3: ¿Qué comparación se ha evaluado entre dato1 y dato2?

```
slt rd,rs,rt
```

Como está escrito en la introducción del trabajo, la instrucción de comparación **slt** corresponde a una comparación del estilo menor estricto que.

Cuestión 1.4: Modifica el código anterior para evaluar la siguiente condición res(1)fl(dato1 = dato2).

```
.data
dato1: .word 50
dato2: .word 20
res:   .space 1
        .text
main:  lw $t0,dato1($0) # cargar dato1 en t0
     lw $t1,dato2($0) # cargar dato2 en t1
     seq $t2,$t0, $t1 # poner a 1 $t2 si t0==t1
     sb $t2,res($0) # almacenar $t2 en res
```

Cuestión 1.5: Resuelve la cuestión anterior utilizando la pseudoinstrucción sge.

```
     .data
dato1:    .word 20
dato2:    .word 50
```

```
res:      .space 1
     .text
main:     lw $t0, dato1($0) # cargar dato1 en t0
     lw $t1, dato2($0) # cargar dato2 en t1
     sge $t2, $t0, $t1 # poner a 1 $t2 si t0>=t1
     sle $t3, $t0, $t1 # poner a 1 $t3 si t0<=t1
     and $t4, $t2, $t3
     sb $t4,res($0) # almacenar $t2 en res
```

Cuestión 1.6: ¿Qué valor se carga en la posición de memoria res?

```
       .data
dato1: .word 30
dato2: .word 40
res:    .space 1
       .text
main: lw $t0,dato1($0) # cargar dato1 en t0
      lw $t1, dato2($0) # cargar dato2 en t1
      slt $t2, $t0, $t1 # poner a 1 $t2 si t0<t1
      bne $t0,$t1,fineval # si t0<>t1 salta a fineval
      ori $t2,$0,1 # poner a 1 t3 si t0=t1
fineval: sb $t2,res($0) # almacenar $t2 en res
```

En la posición de memoria **res** se carga un 1. Esto es porque si miramos las evaluaciones que tiene el programa, nos vamos a encontrar que **$t2** queda seteado en 1 por el **slt** e inmediatamente luego del **bne** salta **fineval** ya que se cumple la condición de que **$t0** y **$t1** son distintos. Por lo tanto no llegaría a evaluarse la instrucción original.

Cuestión 1.7: Inicializa las posiciones de memoria dato1 y dato2 con los valores 50 y 20, respectivamente. Ejecuta de nuevo el programa, ¿Qué valor se carga en la posición de memoria res?

Ahora en la posición de memoria **res** se guarda un 0. Ya que como **$t0** no es menor que **$t1** se cumple el otherwise de la instrucción. Por lo tanto se almacena el 0. Luego se cumple la comparación de que son distintos entre sí, así que salta hacia fineval guardando directamente el valor 0 en la posición de memoria **res**.

Cuestión 1.8: Inicializa las posiciones de memoria dato1 y dato2 con los valores 20 y 20, respectivamente. Ejecuta de nuevo el programa, ¿Qué valor se carga en la posición de memoria res?

El resultado de ejecutar el código con los nuevos cambios es tener un 1 en la posición de memoria res. Esto es porque no se cumple ninguna de las condiciones anteriores a la condición de si es que son iguales o no entre si **$t0** y **$t1**. Por lo tanto tampoco salta directamente a **fineval**. Así que son iguales y por la condición de la anteúltima línea del código, se guarda 1 en **$t2** y por último se almacena el 1 en la posición **res** dentro de la memoria.

Cuestión 1.9: ¿Qué comparación se ha evaluado entre dato1 y dato2?

Como he dicho anteriormente, se evalúa la comparación de si son iguales o no entre **dato1** y **dato2**.

Cuestión 1.10: Evalúa esta comparación utilizando pseudoinstrucciones.

```
       .data
dato1: .word 20
dato2: .word 20
res:    .space 1
       .text
main: lw $t0,dato1($0) # cargar dato1 en t0
      lw $t1, dato2($0) # cargar dato2 en t1
      slt $t2, $t0, $t1 # poner a 1 $t2 si t0<t1
      bne $t0,$t1,fineval # si t0<>t1 salta a fineval
      seq $t2,$t0,$t1 # poner a 1 t3 si t0=t1
fineval: sb $t2,res($0) # almacenar $t2 en res
```

Cuestión 1.11: Modifica el código anterior para que la condición evaluada sea res(1) fl (dato1 >= dato2). No utilices pseudoinstrucciones.

```
       .data
dato1: .word 15
dato2: .word 20
res:    .space 1
```

```
        .text
main:  lw $t0,dato1($0) # cargar dato1 en t0
       lw $t1, dato2($0) # cargar dato2 en t1
       ori $t3, $0, 1
       ori $t4, $0, 1
       beq $t0, $t1, finalval # si son iguales salta a finalval
       slt $t4, $t1, $t0 # si t1 < t0 ==> t3 = 1, otherwise t3 = 0
finalval:
       and $t2, $t4, $t3
       sb $t2,res($0)
```

Cuestión 1.12: Modifica el código anterior utilizando pseudoinstrucciones.

```
        .data
dato1:  .word 20
dato2:  .word 20
res:    .space 1
        .text
main:  lw $t0,dato1($0) # cargar dato1 en t0
       lw $t1, dato2($0) # cargar dato2 en t1
       sge $t2, $t0, $t1
       sb $t2,res($0)
```

Evaluación de condiciones compuestas por operadores lógicos “and” y “or”.

Cuestión 1.13: ¿Qué valor se carga en la posición de memoria res? Para ayudarte en el análisis del código, dibuja el diagrama de flujo de éste.

En la posición de memoria res se guarda un 1. Ya que la idea del programa es retorna 1 si ambos números son iguales a 0 o retornar 0 si ambos números son distintos de 0.

Cuestión 1.14: Inicializa dato1 y dato2 con los valores 0 y 20, respectivamente. ¿Qué valor se carga en la posición de memoria res?

Como expliqué anteriormente, si ambos números son iguales a 0, el código retorna un 1. Pero en este caso tenemos 0 y 20 en **dato1** y **dato2** respectivamente. Así que se estaría almacenando un 0 en **res**.

Cuestión 1.15: Inicializa dato1 y dato2 con los valores 20 y 0, respectivamente. ¿Qué valor se carga en la posición de memoria res?

Misma explicación que la cuestión anterior pero con los valores invertidos.

Cuestión 1.16: Inicializa dato1 y dato2 con los valores 0 y 0, respectivamente. ¿Qué valor se carga en la posición de memoria res?

El valor que se carga es un 1. Por la explicación sobre qué hace el código que escribí anteriormente.

Cuestión 1.17: ¿Qué comparación compuesta se ha evaluado entre dato1 y dato2?

La comparación compuesta que se ha hecho es: ($t8 == 0) && ($t9 == 0)

Cuestión 1.18: Modifica el código anterior para que la condición evaluada sea: res (1) if ((dato1 <> 0) and (dato1 <> dato2)).

```
        .data
dato1:  .word 20
dato2:  .word 20
res:     .space 1
        .text
main:   lw $t8,dato1($0)
        lw $t9,dato2($0)
        ori $t0,$t0,1
        ori $t1,$t1,1
        bne $t8,$0,noIgual
        andi $t0,$0,1
noIgual: bne $t9,$t8,fineval
        and $t1,$t1,$0
fineval: and $t0,$t0,$t1
       sb $t0,res($0)
```

Cuestión 1.19, **Cuestión 1.20** y **Cuestión 1.21**:

El código devuelve un 1 en la posición de memoria **res** cuando se cumplen las condiciones que están en el enunciado que dice que modificaciones tenemos que hacer. De lo contrario, devuelve 0 en la posición **res**.

Cuestión 1.22: La función compuesta que se ha evaluado es ((dato1 <> 0) and (dato1 <> dato2)).
Cuestión 1.23:

```
      .data
dato1: .word 20
dato2: .word 30
res:  .space 1
      .text
main:  lw $t8,dato1($0)
      lw $t9,dato2($0)
      ori $t0,$t0,1
      ori $t1,$t1,1
      bne $t8,$0,noIgual
      andi $t0,$0,1
noIgual: ble $t8,$t9,fineval
      and $t1,$t1,$0
fineval: and $t0,$t0,$t1
      sb $t0,res($0)
```

Cuestión 1.24: Modifica el código anterior utilizando la pseudoinstrucción sle.

```
      .data
dato1: .word 40
dato2: .word 30
res:  .space 1
      .text
main:  lw $t8,dato1($0)
      lw $t9,dato2($0)
      ori $t0,$t0,1
      ori $t1,$t1,1
      bne $t8,$0,noIgual
      andi $t0,$0,1
noIgual: sle $t7, $t8, $t9
      andi $t7, $t7, 1
      beq $t7, $t0, fineval
```

```
      and $t1,$t1,$0
fineval: and $t0,$t0,$t1
      sb $t0,res($0)
```

Cuestión 1.25: ¿Qué valor se carga en la posición de memoria res? Si te sirve de ayuda dibuja el diagrama de flujo asociado al código que se quiere analizar.

```
      .data
dato1: .word 30
dato2: .word -20
res:   .space 1
      .text
main: lw $t8,dato1($0)
      lw $t9,dato2($0)
      and $t0,$t0,$0
      and $t1,$t1,$0
      slt $t0,$t8,$t9
      bne $t9,$0,fineval
      ori $t1,$0,1
fineval: or $t0,$t0,$t1
      sb $t0,res($0)
```

En la posición de memoria **res** se almacena un **0**. Ya que la comparación compuesta es si **dato1 < dato2** o **dato2 == 0**. Por lo tanto con estos valores ingresados no cumplen con estas dos condiciones para que **$t0** tenga valor 1.

Cuestión 1.26: Inicializa dato1 y dato2 con los valores -20 y 10, respectivamente, ejecuta de nuevo el código, ¿Qué valor se carga en la posición de memoria res?

El valor que se carga en la posición de memoria **res** es 1.

Cuestión 1.27: Inicializa dato1 y dato2 con los valores 10 y 0, respectivamente, ejecuta de nuevo el código, ¿Qué valor se carga en la posición de memoria res?

Nuevamente, el valor que se carga en la posición de memoria **res** es un 1.

Cuestión 1.28: Inicializa dato1 y dato2 con los valores 20 y 10, respectivamente, ejecuta de nuevo el código, ¿Qué valor se carga en la posición de memoria res?

El valor que se carga en la posición de memoria **res** es 0.

Cuestión 1.29: ¿Qué comparación compuesta se ha realizado?

La operación compuesta que se hace es la siguiente: ($t8 < $t9) || ($t9 == 0)

Cuestión 1.30 y 1.31:

```
      .data
dato1: .word 20
dato2: .word 0
res:   .space 1
      .text
main: lw $t8,dato1($0)
      lw $t9,dato2($0)
      and $t0,$t0,$0
      and $t1,$t1,$0
      sle $t0,$t8,$t9
      ble $t8,$0,fineval
      ori $t1,$0,1
fineval: or $t0,$t0,$t1
      sb $t0,res($0)
```

Control de flujo condicional

Cuestión 2.1: Identifica la instrucción que evalúa la condición y controla el flujo de programa. Compárala con la condición del programa descrito en pseudocódigo.

La instrucción que evalúa la condición y controla el flujo de programa es la siguiente: "beq $t1,$0,finsi". Como podemos ver, es bastante diferente a lo que sería la pseudoinstrucción que tenemos en el otro código "if ($t1!=0) $t2=$t0/$t1;" Pero a nivel funcional estamos haciendo exactamente lo mismo. Solo que de maneras diferentes.

Cuestión 2.2: Identifica el conjunto de instrucciones que implementan la estructura condicional *if-then*.

```
Si:     beq $t1,$0,finsi        #si $t1 = 0 finsi

finsi:  add $t3,$t0,$t1         #$t3=$t0+$t1
```

```
add $t2,$t3,$t2          #$t2=$t3+$t2
sw $t2,res($0)           #almacenar en res $t2
```

Cuestión 2.3: ¿Qué valor se almacena en la variable res después de ejecutar el programa?

En la variable res, después de ejecutar el programa se almacena un 0x00000047, que es equivalente a 71 en el sistema decimal.

Cuestión 2.4: Si dato2 es igual 0 ¿Qué valor se almacena en la variable res después de ejecutar el programa? Dibuja el diagrama de flujo asociado a la estructura de control implementada en el fragmento de código anterior.

En la variable res se almacena 0x00000028 que es igual a 40 en el sistema decimal.

Cuestión 2.5: Implementar el siguiente programa descrito en pseudocódigo

```
      .data
dato1:      .word 40
dato2:      .word 1
res:  .space 4
      .text
main:
      lw $t0,dato1($0)#cargar dato1 en $t0
      lw $t1,dato2($0)#cargar dato2 en $t1
      and $t2,$t2,$0         #t2=0

Si:
      bgt $t1,$0,entonces    #si $t1 > 0 entonces

entonces:
      div $t0,$t1            #t0/$t1
      mflo $t2          #almacenar LO en $t2

finsi:
      add $t3,$t0,$t1        #$t3=$t0+$t1
      add $t2,$t3,$t2        #$t2=$t3+$t2
      sw $t2,res($0)         #almacenar en res $t2
```

Estructura del control *if-then* con condición compuesta.

Cuestión 2.6:

VARIABLES
    ENTERO: dato1 = 40; dato2 = 30; res;
INICIO
    if(dato1 != 0 || dato2 != 0)   res = dato1/dato2;
    res = res + dato1 + dato2;
FIN

Cuestión 2.7: Identifica la (las) instrucción(es) que evalúa(n) la condición y controla(n) el flujo de programa y compárala(s) con la condición del programa descrito en pseudocódigo.

Estas son las instrucciones que evalúan las condiciones y controlan el flujo del programa:

```
beq $t1,$0,finsi        #si $t1=0 saltar finsi
beq $t0,$0,finsi        #si $t0 =0 saltar finsi
```

Y estas son las instrucciones que hacen lo mismo que lo anterior pero en el pseudocódigo:

```
if(dato1 != 0 || dato2 != 0)
```

Cuestión 2.8: Identifica el conjunto de instrucciones que implementan la estructura condicional if-then. Dibuja el diagrama de flujo asociado con esta estructura de control.

Si:

```
beq $t1,$0,finsi        #si $t1=0 saltar finsi
beq $t0,$0,finsi        #si $t0 =0 saltar finsi
finsi:
add $t3,$t0,$t1             #t3=t0+$t1
add $t2,$t3,$t2             #$t2=t3+$t2
sw $t2,res($0)          #almacenar en res $t2
```

Cuestión 2.9: Al ajecutar el programa ¿Que se almacena en la variable res?

Al ejecutar el programa, en la variable res se almacena 0x00000047 que es el número 71 en el sistema decimal.

Cuestión 2.10: Si dato1=0, ¿Que valor se almacena en la variable res después de ejecutar el programa? Si dato2=0 ¿Que valor se almacena en la variable res?

Cuando dato1 = 0, el valor que se almacena en la variable res es 0x0000001e, que básicamente es el valor de dato2. Y cuando dato2 = 0, el valor que se guarda en la variable res es el valor de dato1, o sea 40, en hexadecimal 0x00000028.

Cuestión 2.11: Implementa el siguiente programa descrito en pseudocódigo:

```
.data
```

```
dato1: .word 0
dato2: .word 40
res: .space 4
     .text
main:
     lw $t0,dato1($0)      #cargar dato1 en t0
     lw $t1,dato2($0)#cargar dato2 en $t1
     and $t2,$t2,$0        #pone a 0 $t2
Si:
     bgt $t1,$0,finsi#si $t1>0 saltar finsi
     bge $t0,$0,finsi#si $t0>=0 saltar finsi
entonces:
     div $t0,$t1           #t0/$t1
     mflo $t2         #almacenar LO en t2
finsi:
     add $t3,$t0,$t1       #t3=t0+$t1
     add $t2,$t3,$t2       #$t2=t3+$t2
     sw $t2,res($0)        #almacenar en res $t2
```

Estructura de control *if-then-else* con condición simple.

Cuestión 2.12: Describe en lenguaje algorítmico el equivalente a este programa en ensamblador:

VARIABLES
    ENTERO: dato1 = 30; dato2 = 40; res;
INICIO
    if(dato1 >= dato2) res = dato1;
    else res = dato2;
FIN

Cuestión 2.13: ¿Qué valor se almacena en res después de ejecutar el programa? Si dato1=35, ¿qué valor se almacena en res después de ejecutar el programa?

Después de ejecutar el programa, en la variable res se almacena un 0x00000028, que es el 40 en número decimal, ya que como se puede ver en la cuestión de arriba, lo que se termina guardando en res es el número más grande puesto en las variables. Por lo tanto, si cambiamos el valor de dato1 a 35 nos va a seguir dando 40 hasta que cambiemos por un valor más grande.

Cuestión 2.14: Identifica en el lenguaje máquina generado por el simulador el conjunto de

instrucciones que implementan la pseudoinstrucción bge.

El conjunto de instrucciones que implementa el simulador para poder user la pseudoinstrucción bge son slt y beq.

Cuestión 2.15: implementa en ensamblador el siguiente programa descrito en lenguaje algorítmico:

```
      .data
dato1:      .word 30
dato2:      .word 40
res:  .space 4
      .text
main:
      lw $t0,dato1($0)#cargar dato1 en $t0
      lw $t1,dato2($0)#cargar dato2 en $t1
Si:
      ble $t0,$t1, sino     #si $t0<=$t1 ir a sino
entonces:
      sub $t0,$t0,$t1
      sw $t0,res($0)        #almacenar $t0 en res
      j finsi               #ir a finsi
sino:
      sub $t1,$t1,$t0
      sw $t1,res($0)        #almacenar $t1 en res
finsi:
```

Estructura de control *if-then-else* con condición compuesta.

Cuestión 2.16: Describe en lenguaje algorítmico el equivalente a este programa en ensamblador.

VARIABLES
    ENTERO: dato1 = 30; dato2 = 40; dato3 = -1; res;
INICIO
    Si (dato 3 < dato1) entonces:
        res = 1;
        Finsi;
    Si (dato3 <= dato2) Sino:

```
            res = 1;
      Entonces:
            res = 0;
      Finsi
FIN
```

Cuestión 2.17: ¿Qué valor se almacena en res después de ejecutar el programa? Si dato1=40 y dato2=30, ¿qué valor se almacena en res después de ejecutar el programa?

El valor que se guarda en la variable res es el 1 ya que se cumple la condición de que dato3 es menor que dato1, por lo tanto se salta a la etiqueta entonces que nos guarda el número 1 en res y por último saltamos a una etiqueta que sería el fin del programa.
Si cambiamos los valores que como pide la consigna, el valor que queda en res es el mismo que antes ya que la condición que se cumple para que ello pase es la primera de todas y sigue ocurriendo aunque hayamos cambiado los valores de dato1 y dato2

Cuestión 2.18: Implementa en ensamblador el siguiente programa descrito en lenguaje algorítmico:

```
      .data
dato1:      .word 30
dato2:      .word 30
dato3:      .word 40
res:  .space 4
      .text
main:
      lw $t1,dato1($0)      #cargar dato1 en $t1
      lw $t2,dato2($0)      #cargar dato2 en $t2
      lw $t3,dato3($0)      #cargar dato3 en $t3
Si:
      bge $t3,$t1, entonces       #si $t3>=$t1 ir entonces
entonces:
      ble $t3,$t2,esUno     #si $t3<=$t2 ir a sino
      j sino                #ir a sino
esUno:
      addi $t4,$0,1         #$t4=1
      j finsi               #ir a finsi
sino: and $t4,$0,$0         #$t4=0
finsi: sw $t4,res($0)       #almacenar res
```

Estructura de control repetitiva *While*

Cuestión 1.19: Ejecuta paso a paso el programa anterior y comprueba detenidamente la función de cada una de las instrucciones que constituyen el programa ensamblador.

Primero se carga el address de memoria en el registro $t0 para luego ser referenciado más adelante y poder extraer caracter por caracter.
Y comienza el ciclo while con la etiqueta mientras la cual sera llamada una vez terminado un ciclo para que vuelva a ejecutarse y así hasta que $t1 sea igual a 0 y salte a finmientras que ahí concluye nuestro recorrido de cadena.
Dentro del ciclo evaluamos si $t1 es igual a 0, si no lo es, se avanza en el while sumando 1 en el contador de caracteres y avanzando en la cadena para poder agarrar el byte actual en la cadena al momento de estar recorriendola.
Una vez que $t1 tiene el valor 0, porque se ha terminado de recorrer la cadena, se llega a la etiqueta mientras que almacena $t2 en n para poder ver finalmente cuántos caracteres posee nuestra cadena.

Cuestión 2.20: ¿Qué valor se almacena en n después de ejecutar el programa?

El valor que se almacena en n después de ejecutar el programa es 4.

Cuestión 2.21: Implementa en ensamblador el siguiente programa descrito en lenguaje algorítmico:

```
      .data
tira1:      .asciiz "hola"
tira2:      .asciiz "adios"
n:    .space 4
      .text
main:
      la $t0,tira1          #carga dir. tira1 en $t0
      la $t1,tira2          #carga dir. tira2 en $t1
      andi $t2,$t2, 0       #$t2=0
mientras:
      lb $t3,0($t0)         #almacenar byte en $t1
      lb $t4,0($t0)         #almacenar byte en $t1
      beq $t3,$0,finmientras      #si $t1=0 saltar a finmientras
      beq $t4,$0,finmientras      #si $t1=0 saltar a finmientras
      addi $t2,$t2, 1  #$t2=$t2+1
      addi $t0,$t0, 1  #$t0=$t0+1
      addi $t1,$t1, 1  #$t1=$t1+1
      j mientras            #saltar a mientras
finmientras:
```

```
    sw $t2,n($0)          #almacenar $t2 en n
```

Estructura de control repetitiva *for.*

Cuestión 2.22: Ejecuta paso a paso el programa y comprueba detenidamente la función que tiene cada una de las instrucciones que componen el programa en ensamblador.

A primera vista tenemos 4 partes principales en el código, la sección con las declaraciones, el inicio(main), el principio del for y el final del for.

En la parte de las declaraciones podemos ver que tenemos declarado al vector que queremos recorrer e ir sumando elemento a elemento y el res donde se va a almacenar el resultado final.

En la siguiente sección nos encontramos con el registro $t2 que está guardando la dirección del vector para poder referirnos a él y tomar sus elementos. También inicializamos en 0 el resultado final para tener algo sobre que sumar en la primera iteración. Y por último están las declaraciones del valor más chico donde comienza el for y el valor tope, que sería la cantidad de elementos que posee el vector a recorrer.

Las declaraciones nombradas anteriormente serán útiles ahora en la etiqueta **para** en la cual se encuentra creada la lógica para recorrer el vector. Primero se hace la evaluación típica que vemos en otros lenguajes, si $t0 < $t1 que siga avanzando con las iteraciones, sino, que salte a la etiqueta **finpara**, la cual nos cierra el ciclo for y nos guarda el valor de la suma de los elementos en la variable res para poder visualizarlo.

La primera línea del bucle se puede decir que realmente es la carga del elemento actual del array en el registro $t4. Luego se suma ese elemento tomado al registro con la suma acumulada en la siguiente línea(**add**) y por último se suma en 1 la variable(**$t0**) que va llevando la cuenta de las iteraciones y se suma en 4 el vector para poder pasar a recorrer el siguiente elemento en la próxima iteración. Y en el final se vuelve a llamar a la etiqueta **para** para volver a hacer una nueva iteración.

Cuestión 2.23: ¿Qué valor se almacena en res después de ejecutar el código anterior?

El valor que se almacena en res después de ejecutar el código es 41.

Cuestión 2.24: Implementa en ensamblador el siguiente programa descrito en pseudocódigo:

```
    .data
```

```
v1:   .word 6,7,8,9,10,-1,34,23
v2:   .space 4
      .text
main:
      la $t2,v1         #$t2=dirección de vector
      and $t3,$0,$t3          #$t3=0
      li $t0,0          #$t0=0
      li $t1,7          #$t1=5
para:
      bgt $t0,$t1,finpara   #si $t0>$t1 saltar finpara
      lw $t4,0($t2)         #carga elemento vector en $t4
      addi $t4,$t4, 1 #suma los elementos del vector
      sw $t4,v2($t3)
      addi $t2,$t2, 4 #$t2=$t2+4
      addi $t0,$t0, 1 #$t0=$t0+1
      addi $t3,$t3, 4
      j para                #saltar a bucle
finpara:
      sw $t3, v2($0)   #almacenar $t3 en res
```