Organización y Arquitectura de Computadores

Bitacora

17/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Mesitas de FIEC

Nos reunimos para comenzar el proyecto en sí, aunque cada miembro del grupo ya había estado investigando y practicando temas relacionados con el proyecto.

Definimos primero ciertas variables que nos iban a ser de utilidad como lo son el nombre del archivo .txt donde se escribirán los números aleatorios.

```
#PROYECTO PRIMER PARCIAL - ORGANIZACION Y ARQUITECTURA DE COMPUTADORES

# a) Usar el generador de números aleatorios de MARS para crear archivos que contenga números entre 0 y 1000 y
#    agruparlos de la siguiente forma (10, 20, 50, 100, 150, 200, 250, 300, 350, 400) el nombre para el archivo
#    debe ser "aleatorios.txt", este archivo será leido posteriormente por su programa.
        .data
fout:   .asciiz "aleatorios.txt"      # nombre del archivo de numeros aleatorios
coma:   .asciiz ","
buffer: .space 6                      #creo el buffer donde guardare el string a escribir en el archivo
```

Fig1.1 Muestra algunas variables del proyecto.

Decidimos comenzar creando una función que genere un numero aleatorio y lo retorne ya que será una de las más utilizadas en el literal a) del proyecto.

```
67  Random:
68          li $v0, 42      #llamada al sistema para generar aleatorios
69          li $a1, 1001    #seteo el limite superior del rango de aleatorios
70          syscall
71          add $v0, $a0, $zero     #guardo mi valor aleatorio en el registro $v0
72          jr $ra          #retorno el aleatorio
```

Fig1.2 Función Random.

Luego nos pusimos a investigar acerca de la escritura de archivos y nos topamos con la dificultad de que no podíamos escribir integers en el archivo, en distintos ejemplos que buscamos solamente se escribían cadenas de caracteres, así que optamos por buscar una función en la web que convirtiera nuestro numero aleatorio en un string para así poder escribirlo en el archivo.

La función la encontramos en la siguiente página: http://www.daniweb.com/software-development/assembly/code/435631/integer-to-string-in-mips-assembly.

```
76  ##############################################################################
77  #                           CONVERTIR INT A STRING                           #
78  ##############################################################################
79  # int ItoA(int, char*)
80  # arguments:
81  #    $a0 - integer to convert
82  #    $a1 - character buffer to write to
83  # return:  number of characters in converted string
84  #
85  ItoA:
86          bnez $a0, ItoA.non_zero  # first, handle the special case of a value of zero
87          nop
88          li   $t0, '0'
89          sb   $t0, 0($a1)
90          sb   $zero, 1($a1)
91          li   $v0, 1
92          jr   $ra
93  ItoA.non_zero:
94          addi $t0, $zero, 10     # now check for a negative value
95          li $v0, 0
96          bgtz $a0, ItoA.recurse
97          nop
98          li   $t1, '-'
99          sb   $t1, 0($a1)
.00         addi $v0, $v0, 1
.01         neg  $a0, $a0
.02 ItoA.recurse:
.03         addi $sp, $sp, -24
.04         sw   $fp, 8($sp)
.05         addi $fp, $sp, 8
.06         sw   $a0, 4($fp)
.07         sw   $a1, 8($fp)
.08         sw   $ra, -4($fp)
.09         sw   $s0, -8($fp)
.10         sw   $s1, -12($fp)
.11
.12         div  $a0, $t0       # $a0/10
.13         mflo $s0            # $s0 = quotient
.14         mfhi $s1            # s1 = remainder
.15         beqz $s0, ItoA.write
.16 ItoA.continue:
.17         move $a0, $s0
.18         jal ItoA.recurse
.19         nop
.20 ItoA.write:
.21         add  $t1, $a1, $v0
.22         addi $v0, $v0, 1
.23         addi $t2, $s1, 0x30 # convert to ASCII
.24         sb   $t2, 0($t1)    # store in the buffer
.25         sb   $zero, 1($t1)
.26
.27 ItoA.exit:
.28         lw   $a1, 8($fp)
.29         lw   $a0, 4($fp)
.30         lw   $ra, -4($fp)
.31         lw   $s0, -8($fp)
.32         lw   $s1, -12($fp)
.33         lw   $fp, 8($sp)
.34         addi $sp, $sp, 24
.35         jr $ra
.36         nop
```

Fig1.3 Función ItoA.

Una vez resuelto nuestro problema empezamos a escribir la parte inicial de la función para escribir en el archivo.

```
16  WriteInFile:
17          #Abro el archivo aleatorios.txt
18          li   $v0, 13       #llamada al sistema para abrir archivos
19          la   $a0, fout     #especifico el nombre del archivo
20          li   $a1, 1        #especifico que abro el archivo para escritura
21          li   $a2, 0
22          syscall            #abro el archivo y el file descriptor se guarda en $v0
23          add $s6, $v0, $zero#guardo el file descriptor en $s6
```

Fig1.4 Función WriteInFile.

Una vez abierto el archivo necesitamos generar los números aleatorios y que los escriba, para esto utilizamos nuestras funciones previamente implementadas (Random y ItoA), y todo esto dentro de un lazo que deberá repetirse dependiendo la cantidad de aleatorios que queramos. Así que para empezar a probar solamente generaremos 10 aleatorios y los escribiremos.

```
25          li $s1, 0       #i=0
26  For:
27          #Genero el numero aleatorio
28          addi $sp, $sp, -4
29          sw $ra, 0($sp)  #guardo la direccion de retorno enla pila
30          jal Random      #llamo a mi funcion para generar un numero aleatorio
31          lw $ra, 0($sp)  #cargo la direciion de retorno de la pila
32          addi $sp, $sp, 4
33          add $s0, $v0, $zero   #guardo el retorno de la funcion random en $s0
34          #Convierto el numero aleatorio generado a String
35          move $a0, $s0   #envio el numero aleatorio como parametro para convertirlo en string
36          la $a1, buffer  #envio como parametro el buffer donde almacenare el entero convertido a string
37          addi $sp, $sp, -4
38          sw $ra, 0($sp)  #guardo la direccion de retorno en la pila
39          jal ItoA        #llamo a la funcion para convetir a string
40          add $t0, $v0, $zero     #guardo en $t0 el numero de digitos del aleatorio
41          lw $ra, 0($sp)  #cargo la direccion de retorno de la pila
42          addi $sp, $sp, 4
43          #Escribo en el archivo aleatorios.txt
44          li   $v0, 15       #llamada al sistema para escritura en archivos
45          add $a0, $s6, $zero      #especifico el file descriptor
46          la $a1, buffer     #especifico la direccion del string que queremos escribir
47          li $a2, 5
48          syscall            #escribo en el archivo
49          li   $v0, 15       #llamada al sistema para escritura en archivos
50          la $a1, coma       #especifico la direccion del string que queremos escribir
51          li $a2, 1
52          syscall            #escribo en el archivo
53
54          addi $s1, $s1, 1        #i++
55
56          slti $t1, $s1, 10       #if($s1<$10) $t1=1; else $t1=0;
57          bne $t1, $zero, For     #if($s1<$10) sigue en el for; else sale del for
58
59          #Cierro el archivo
60          li   $v0, 16       #llamada al sistema para cerrar archivo
61          add $a0, $s6, $zero      #especifico el file descriptor
62          syscall            #cierro el archivo
63          jr $ra          #termino mi funcion
```

Fig1.5 Lazo for para escribir en el archivo.

Finalmente para terminar nuestra primera prueba creamos la función Main donde llamamos a la función de escritura de archivos, y posteriormente se detiene la ejecución del programa.

```
11  Main:
12          jal WriteInFile         #llamo a mi funcion escribir en un archivo
13          li $v0, 10              #terminar ejecucion
14          syscall
```

Fig1.6 Función Main.

19/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Mesitas de FIEC

En esta segunda reunión nos propusimos terminar el literal a) del proyecto, que consistía en hacer un programa que cree un archivo con los 10 grupos de números aleatorios agrupados de la forma indicada.

Definimos dos strings nuevos que serán los corchetes que indicaran en inicio y el final de un grupo de aleatorios.

```
 9  corcheteIzq:    .asciiz "["
10  corcheteDer:    .asciiz "]"
```

Fig2.1 Nuevas variables.

También se definió un arreglo de integers donde tenemos almacenado la cantidad de número aleatorios que deberán generarse por cada uno de los 10 grupos.

```
13  grupos: .word 10, 20, 50, 100, 150, 200, 250, 300, 350, 400
```

Fig2.2 arreglo de la cantidad de elementos de los grupos.

Modificamos también la función *WriteInFile* para agregarle un lazo más, que será el que controlara que grupo es el que se está escribiendo y dependiendo de eso se escogerá el índice del arreglo que contiene la cantidad de número aleatorios que deberán generarse. También se le agrego la validación de que cuando ya se escriba el último número del grupo se finalice con corchete, para indicar que ahí termina el grupo.

Entonces la función indicada quedo de la siguiente manera:

```
WriteInFile:
        #Abro el archivo aleatorios.txt
        li   $v0, 13       #llamada al sistema para abrir archivos
        la   $a0, file1     #especifico el nombre del archivo
        li   $a1, 1        #especifico que abro el archivo para escritura
        li   $a2, 0
        syscall            #abro el archivo y el file descriptor se guarda en $v0
        add $s6, $v0, $zero      #guardo el file descriptor en $s6

        la $s3, grupos  #guardo en $s3 la direccion base del arreglo
        li $s2, 0       #j=0

ForG:
        li   $v0, 15       #llamada al sistema para escritura en archivos
        add $a0, $s6, $zero      #especifico el file descriptor
        la $a1, corcheteIzq     #especifico la direccion del string que queremos escribir
        li $a2, 1
        syscall                 #escribo el corchete inicial

        sll $t5, $s2, 2         #multiplico el indice del arreglo por cuatro para obtener el offset=$t5
        add $t5, $t5, $s3       #sumo el offset a la direccion base del arreglo y obtengo la direccion del indice
                                #del arreglo=$t5
        lw $t6, 0($t5)          #guardo en $t6 el valor del indice del arreglo

        li $s1, 0       #i=0
        add $t4, $t6, -1        #temporal para verificar si ya se escribio el ultimo numero
For:
        #Genero el numero aleatorio
        addi $sp, $sp, -4
        sw $ra, 0($sp)  #guardo la direccion de retorno enla pila
        jal Random      #llamo a mi funcion para generar un numero aleatorio
        lw $ra, 0($sp)  #cargo la direciion de retorno de la pila
        addi $sp, $sp, 4
        add $s0, $zero, $zero
        add $s0, $v0, $zero   #guardo el retorno de la funcion random en $s0
                #li $v0, 1      #llamada al sistema para imprimir un integer
                #add $a0, $s0, $zero    #seteo el numero a imprimir
                #syscall
        #Convierto el numero aleatorio generado a String
        move $a0, $s0   #envio el numero aleatorio como parametro para convertirlo en string
        la $a1, buffer  #envio como parametro el buffer donde almacenare el entero convertido a string
        addi $sp, $sp, -4
        sw $ra, 0($sp)  #guardo la direccion de retorno en la pila
        jal ItoA        #llamo a la funcion para convetir a string
        add $t0, $v0, $zero     #guardo en $t0 el numero de digitos del aleatorio
        lw $ra, 0($sp)  #cargo la direccion de retorno de la pila
        addi $sp, $sp, 4
        #Escribo en el archivo aleatorios.txt
        li   $v0, 15       #llamada al sistema para escritura en archivos
        add $a0, $s6, $zero      #especifico el file descriptor
        la $a1, buffer     #especifico la direccion del string que queremos escribir
        add $a2, $zero, $t0
        #li $a2, 6
        syscall            #escribo en el archivo
        li   $v0, 15       #llamada al sistema para escritura en archivos
        beq $s1, $t4, Corchete
        la $a1, coma     #especifico la direccion del string que queremos escribir
        j Continue
Corchete:
        la $a1, corcheteDer     #especifico la direccion del string que queremos escribir
Continue:
        li $a2, 1
        syscall            #escribo en el archivo
        addi $s1, $s1, 1        #i++
        slt $t1, $s1, $t6       #if($s1<$t6) $t1=1; else $t1=0;
        bne $t1, $zero, For     #if($s1<$t6) sigue en el for; else sale del for

        addi $s2, $s2, 1        #j++
        slti $t7, $s2, 10       #if($s2<10) $t1=1; else $t1=0;
        bne $t7, $zero, ForG    #fin del for grande

        #Cierro el archivo
        li   $v0, 16       #llamada al sistema para cerrar archivo
        add $a0, $s6, $zero      #especifico el file descriptor
        syscall            #cierro el archivo
        jr $ra          #termino mi funcion
```

Fig2.3 Función WriteInFile modificada.

22/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Casa de Victor

En esta tercera reunión que se realizó en la casa de nuestro compañero Víctor Rodríguez con la finalidad de avanzar en los siguientes enunciados del proyecto. Nos concentramos en la investigación y elaboración de los algoritmos de ordenamientos solicitado el cual se pudo realizar el algoritmo de ordenamiento por burbuja **(bubble sort).**

La parte inicial del código consiste en establecer los registros necesarios para realizar el ordenamiento y cargar en un registro el arreglo definido. **SLoop** en esencia es la parte más trabajo realiza ya que es la encargada de mantenernos en el lazo y realiza las comprobaciones los elementos del arreglo.

```
20  Sort:
21                  addi    $t0, $zero, 0           #initiate counter
22                  addi    $t1, $zero, 0           #initiate register
23                  addi    $t2, $zero, 0           #initiate register
24                  addi    $t4, $zero, 0           #initiate register
25                  addi    $t5, $zero, 0           #initiate register
26                  addi    $t6, $zero, 0           #initiate register
27                  addi    $t7, $zero, 0           #initiate register
28                  addi    $s2, $zero, 0           #set/reset swap flag
29                  sll     $t1, $t0, 2             #$t0 * 4 as offset
30                  add     $t1, $t1, $s0           #load the array into $t1
31                  addi    $t2, $t1, 4             #load it to use to compare
32          SLoop:
33                  addi    $t0, $t0, 1             #increment counter
34                  beq     $t0, $s1, ExitLoop      #check if it ever branched to swap and exit
35                  lw      $t6, 0($t1)             #put $t1 into $t6
36                  lw      $t7, 0($t2)             #put $t2 into $t7
37                  bgt     $t6, $t7, Swap          #send the intgers to swap if $t1 < $t2
38                  sll     $t1, $t0, 2             #$t0 * 4 as offset
39                  add     $t1, $t1, $s0           #load the array into $t1
40                  addi    $t2, $t1, 4             #add 4 and load the array into $t2
41                  j       SLoop                   #start loop over again
```

Fig3.1 Inicialización de variables y SLoop.

La parte del **Swap**, básicamente se encarga de invertir los elementos del arreglo cuando el valor actual recorrido es mayor al valor siguiente del arreglo. Y para corroborar que la función realiza correctamente el ordenamiento, mostramos por pantalla a través de **Print** el arreglo original y **PrintSorted** el arreglo ordenado.

```
42          Swap:
43                  lw      $t4, 0($t1)             #load $t6 into $t4
44                  lw      $t5, 0($t2)             #load $t7 into $t5
45                  sw      $t5, 0($t1)             #swap $t4 into $t7
46                  sw      $t4, 0($t2)             #swap $t5 into $t6
47                  addi    $s2, $s2, 1             #add 1 to $s2 to check if the program ever came here
48                  j       SLoop                   #jump back to the SLoop
49          ExitLoop:
50                  bgtz    $s2, Sort               #start sort over again if the flag $s2 is set
51                  j       PrintSorted             #jump to print the sorted array
52  Print:
53                  la      $a0, DisplayInitial     #the title to display the initial array
54                  li      $v0, 4                  #the value to print a string
55                  syscall                         #call the function
56                  addi    $t0, $zero, 0           #initiate counter
57                  sll     $t1, $t0, 2             #$t0 * 4 as offset
58                  add     $t1, $t1, $s0           #load the array into $t1
59          PLoop:
60                  lw      $a0, 0($t1)             #load the integer to print in $a0
61                  li      $v0, 1                  #command to print an integer
62                  syscall                         #call the command to print the integer
63                  la      $a0, Space              #print a space between numbers
64                  li      $v0, 4                  #load 4 into $v0 to print a string
```

```
71                  la      $a0, NewLine            #create a new line
72                  syscall                         #call the function
73                  jr      $ra                     #return to calling function
74  PrintSorted:
75                  la      $a0, DisplaySorted      #the title to display the initial array
76                  li      $v0, 4                  #the value to print a string
77                  syscall                         #call the function
78                  addi    $t0, $zero, 0           #initiate counter
79                  sll     $t1, $t0, 2             #$t0 * 4 as offset
80                  add     $t1, $t1, $s0           #load the array into $t1
81          PLoop2:
82                  lw      $a0, 0($t1)             #load the integer to print in $a0
83                  li      $v0, 1                  #command to print an integer
84                  syscall                         #call the command to print the integer
85                  la      $a0, Space              #print a space between numbers
86                  li      $v0, 4                  #load 4 into $v0 to print a string
87                  syscall                         #call the command
88                  addi    $t0, $t0, 1             #increment counter
89                  sll     $t1, $t0, 2             #$t0 * 4 to offset
90                  add     $t1, $t1, $s0           #load next element in the array
91                  bne     $t0, $s1, PLoop2        #keep looping until $t0 == $s1
```

Fig3.2 Funciones de BubbleSort y PrintSorted.

23/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Mesitas de FIEC

En esta cuarta reunión se empezó a trabajar en el algoritmo de búsqueda por el método de inserción.

En esta primera parte del código realizamos la inicialización de lo registros, modulamos el algoritmo usando la pila. La etiqueta **for_compare** se encarga de la verificación de ciclo iterativo, luego capturamos el siguiente elemento del array para la comparación.

```
1124  InsertionSort:
1125          addi $sp, $sp, -32
1126          sw $t0, 28($sp)
1127          sw $t1, 24($sp)
1128          sw $t2, 20($sp)
1129          sw $t3, 16($sp)
1130          sw $t4, 12($sp)
1131          sw $t5, 8($sp)
1132          sw $t6, 4($sp)
1133          sw $t7, 0($sp)
1134
1135          move    $t5, $s0        #carga el array
1136          move    $t0, $s1        # tamaño del array
1137          li      $t1, 1          #constante
1138          li      $t7, 4          #constante para offset
1139  for_compare:
1140          bge     $t1, $t0, end_for       # condicion t1>=t0 (acum>=tam_array)
1141          addi    $t2, $t1, -1            #aux para index
1142          mul     $t4, $t1, $t7           # t4 toma un valor del sgt
1143          add     $t4, $t5, $t4           # t4 se iguala al sgt elemento del array
1144          lw      $t3, 0($t4)             # se toma el elemento de t4 en t3
```

Fig4.1 Inicialización de variables y etiqueta de comparación.

El algoritmo entra al **while;** realiza validaciones y toma el elemento anterior para compararlo con el elemento siguiente mostrado en el figura anterior. Realiza la comparación e intercambia los valores según sea el caso, el ciclo se repite hasta q la condición del **end_for** sea verdadera.

```
1148
1149  while:
1150          blt     $t2, 0, end_while       #condicion para tomar el sgt elemento
1151          mul     $t4, $t2, $t7           #reinicia t4 al anterior
1152          add     $t4, $t4, $t5           #t4 toma nuevamente el array
1153          lw      $t6, 0($t4)             #t6 toma el anterior valor de t4
1154          ble     $t6, $t3, end_while     #condicion t6<=t3
1155                                          #FALSA
1156          sw      $t6, 4($t4)             #t6 toma el anterior valor de t4
1157          addi    $t2, $t2, -1            #t2 disminuye
1158          j       while
1159  end_while:                              #VERDADERA
1160          mul     $t4, $t2, $t7           #t4 se reinicia
1161          add     $t4, $t5, $t4           #t4 toma el valor del array
1162          sw      $t3, 4($t4)             #t3 toma el segundo valor de t4
1163          addi    $t1, $t1, 1             #t1 se acumul
1164          j       for_compare
1165
1166  end_for:
```

Fig4.2 Etiqueta while de comparación de elementos.

Al final se reinician las variables y se retorna el registro $ra.

```
65
66  end_for:
67          lw $t7, 0($sp)
68          lw $t6, 4($sp)
69          lw $t5, 8($sp)
70          lw $t4, 12($sp)
71          lw $t3, 16($sp)
72          lw $t2, 20($sp)
73          lw $t1, 24($sp)
74          lw $t0, 28($sp)
75          addi $sp, $sp, 32
76          jr $ra
77
```

Fig4.3 Retorno de registro $ra.

24/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Mesitas de FIEC

En esta quinta reunión se realizó el menú de usuario para mejor nuestro avance del proyecto a la hora de trabajar con todo el código que se estaba generando y facilitar el momento de probar nuestras funciones.

Primero se realizó la declaración de los las etiquetas en la sección **.data**

```
## LABEL DE LOS MENUS
Menu1: .asciiz "** Organizacion de Computadores *** \n"
Menu1.1: .asciiz "      1. Generar Randoms      \n"
Menu1.2: .asciiz "      2. Algoritmos de Ordamiento \n"
Menu1.3: .asciiz "      3. Salir        \n"
Menu: .asciiz "*********************************\n"
Menu2.0: .asciiz "***  Algoritmos de Ordanamiento ***\n"
Menu2.1: .asciiz "      1. BubbleSort               \n"
Menu2.2: .asciiz "      2. Inserción                \n"
Menu2.3: .asciiz "      3. Quicksort                \n"
Menu2.4: .asciiz "      4. Atras                    \n"
Op: .asciiz "Se Genero el archivo de numeros aleatorios \n"
OpcionBb: .asciiz "Ordenamiento por BubbleSort Completado, Se han generado los archivos bur_aleatorios.txt y bur_tiempos.tx
OpcionIn: .asciiz "Ordenamiento por Inserción Completado, Se han generado los archivos ins_aleatorios.txt y ins_tiempos.txt
OpcionQk: .asciiz "Ordenamiento por QuickSort Completado, Se han generado los archivos qui_aleatorios.txt y qui_tiempos.txt
```

Fig5.1 Etiquetas del Menú de Usuario.

El menú se dividió en dos partes, el menú exterior que contenía la opción de generar los números aleatorios y el segundo menú o menú interior que contenía las opciones para realizar los algoritmos de búsqueda.

La siguiente parte es la impresión por pantalla de las etiquetas, aquí mostramos una parte de la secuencia del código, así como la salida por consola del menú exterior.

```
################################
###    MENU PRINCIPAL
################################
menuPrincipal:
loopAtras:

la $a0, NewLine
li $v0,4
syscall

la $a0, Menu
li $v0,4
syscall

la $a0, Menu1
li $v0,4
syscall

la $a0, Menu
li $v0,4
syscall
```

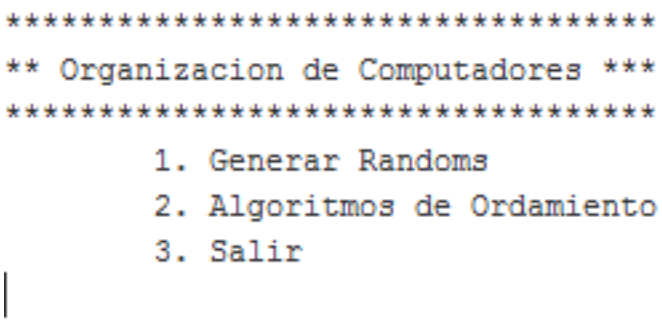

Fig5.2 Parte del código del menú exterior e imagen del menú por consola.

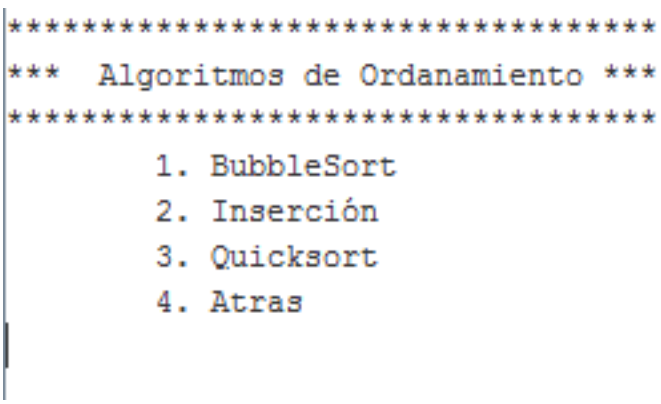

Fig5.3 Imagen del menú interior por consola.

El siguiente código muestra las validaciones realizadas para las opciones digitas por el usuario, solo realiza las indicadas en el menú y procede a su etiqueta correspondiente. Para cualquier otro número digitado permitirá al usuario digitar nuevamente.

```
74   la $a0, myOpcion        #cargando opcion 1
75   li $a1, 2               #permitiendo tipiear un digito
76   li $v0, 8               # opcion syscall para escribir
77   syscall
78   lw $s1, 0($a0)          #busco el valor digitado
79
80  ## bloque de controls
81   beq $s1,$t4, loopAtras          # pregunto por condicion = opcion 4
82
83   slt $t5, $s1,$t4                #condicion de ciclo
84   la $a0, NewLine
85   li $v0,4
86   syscall
87
88   beq $t5,$zero, loopInicio2      #condicion de numero equivocado
89
90   beq $s1,$t1, loopImprimir1      # pregunto por condicion = Bubblesort
91   beq $s1,$t2, loopImprimir2      # pregunto por condicion = Insertsort
92   beq $s1,$t3, loopImprimir3      # pregunto por condicion = Quicksort
93
```

Fig5.4 Bloque de control y validación de ingreso de digito menú interior.

27/06/2014 - Jose Romero, Victor Rodríguez, Fausto Mora - Mesitas de FIEC

Para nuestra sexta reunión nos pusimos a trabajar en el último algoritmo de ordenamiento, el Quicksort.

El algoritmo del Quicksort resulto ser el más completo de todos y nos llevó el doble de tiempo que los demás algoritmos de ordenamiento.

Lo podemos dividir en cuatro partes, inicialización, validación, partición e intercambio.

La inicialización la realizamos con el uso de la pila para poder modular la función.

```
QuickSort:

        addi    $sp, $sp, -20           # make room on stack for 5 registers
        sw      $s0, 4($sp)             # save $s0 on stack
        sw      $s1, 8($sp)             # save $s1 on stack
        sw      $s2, 12($sp)            # save $s2 on stack
        sw      $s3, 16($sp)            # save $s3 on stack
        sw      $ra, 20($sp)            # save $ra on stack

        move    $s0, $a0                # copy param. $a0 into $s0 (addr array)
        move    $s1, $a1                # copy param. $a1 into $s1 (low)
        move    $s2, $a2                # copy param. $a2 into $s2 (high)
```

Fig6.1 Inicialización de variables.

Las siguientes líneas de código realizan validaciones para determinar el pivote de nuestro array y guardar la posición.

```
if:
        blt     $s1, $s2, then          # if low < high
        j       endIf
then:
        move    $a0, $s0
        move    $a1, $s1
        move    $a2, $s2
        jal     partition
        move    $s3, $v0                # save pivotPosition
        move    $a0, $s0
        move    $a1, $s1
        addi    $a2, $s3, -1
        jal     QuickSort
        move    $a0, $s0
        addi    $a1, $s3, 1
        move    $a2, $s2
        jal     QuickSort
endIf:
        lw      $s0, 4($sp)             # restore $s0 from the stack
        lw      $s1, 8($sp)             # restore $s1 from the stack
        lw      $s2, 12($sp)            # restore $s2 from the stack
        lw      $s3, 16($sp)            # restore $s3 from the stack
```

Fig6.2 Determinación de ubicación de pivote.

La siguiente parte es el núcleo del algoritmo del quicksort, realiza la comparación de las particiones y mediante ciclos la parte recursiva de la misma. Al final se hacen las restauraciones de pila y se retorna el registro $ra mediante un jr.

```
partition:
addi    $sp, $sp, -24           # make room on stack for 6 registers
sw      $s0, 4($sp)             # save $s0 on stack
sw      $s1, 8($sp)             # save $s1 on stack
sw      $s2, 12($sp)            # save $s2 on stack
sw      $s3, 16($sp)            # save $s3 on stack
sw      $s4, 20($sp)            # save $s4 on stack
sw      $ra, 24($sp)            # save $ra on stack
move    $s0, $a0                # copy param. $a0 into $s0 (addr array)
move    $s1, $a1                # copy param. $a1 into $s1 (low)
# initialize left, right, and pivot
move    $s2, $s1
move    $s3, $a2
li      $t4, 4
mul     $t0, $s1, $t4
add     $t0, $t0, $s0
lw      $s4, 0($t0)

whileQS:
        blt     $s2, $s3, whileBody
        j       endWhile
```

```
whileBody:
        while_2:
                li      $t4, 4
                mul     $t0, $s3, $t4
                add     $t0, $t0, $s0
                lw      $t1, 0($t0)
                bgt     $t1, $s4, whileBody_2
                j       endWhile_2
        whileBody_2:
                addi    $s3, $s3, -1
                j       while_2
        endWhile_2:
        while_3:
                blt     $s2, $s3, andTest
                j       endWhile_3
        andTest:
                li      $t4, 4
                mul     $t1, $s2, $t4
                add     $t1, $t1, $s0
                lw      $t2, 0($t1)
                ble     $t2, $s4, whileBody_3
                j       endWhile 3

         whileBody_3:
                 addi    $s2, $s2, 1

                 j       while_3
         endWhile_3:
         if_2:
                 blt     $s2, $s3, then_2
                 j       endIf_2
         then_2:

                 move    $a0, $t1
                 move    $a1, $t0
                 jal     swap
         endIf_2:

         j       whileQS
endWhile:
li      $t4, 4
mul     $t0, $s3, $t4
add     $t0, $t0, $s0
lw      $t1, 0($t0)
```

```
mul     $t2, $s1, $t4
add     $t2, $t2, $s0
sw      $t1, 0($t2)

sw      $s4, 0($t0)

move    $v0, $s3                # return right

lw      $s0, 4($sp)             # restore $s0 from the stack
lw      $s1, 8($sp)             # restore $s1 from the stack
lw      $s2, 12($sp)            # restore $s2 from the stack
lw      $s3, 16($sp)            # restore $s3 from the stack
lw      $s4, 20($sp)            # restore $s4 from the stack
lw      $ra, 24($sp)            # restore $ra from the stack
addi    $sp, $sp, 24            # restore stack pointer

jr      $ra
```

Fig6.3 Core del algoritmo quicksort.

La parte final del algoritmo corresponde a la etiqueta **swap,** que realiza el cambio de los registros del proceso anterior.

```
swap:
# $a0 contains address of operand1
# $a1 contains address of operand2
# $t0 contains temp
# $t1 contains value of operand2

# Since no subprograms are called, NO registers need to be saved and restored.

lw      $t0, 0($a0)
lw      $t1, 0($a1)

sw      $t1, 0($a0)
sw      $t0, 0($a1)

jr      $ra
```

Fig6.4 Función Swap.

BIBLIOGRAFIA

Convertir de Entero a String

- http://www.daniweb.com/software-development/assembly/code/435631/integer-to-string-in-mips-assembly

Bubble Sort

- http://codecellar.99k.org/BubbleSortMIPS.html

Quick Sort

- http://www.cs.uni.edu/~fienup/courses/copy-of-computer-organization/homework-solutions/hw6f98/hw6f98.mips

Scilab IDE 5.4.0

- http://www.scilab.org/download/5.4.1