**Trabajo del tema 2: Ensamblador MIPS**

**Nombre:** Supervielle Sánchez, Mª Teresa

**Problema 1:** Swap (0,5 puntos) Programar una función llamada swap que tome como parámetros dos punteros a enteros de tamaño palabra (Word). La función intercambiará los contenidos de las dos posiciones de memoria apuntadas por dichos punteros. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento.

.data #inicialización de los dos números a intercambiar

p1: .word 3

p2: .word 5

.text

.globl main

main:

lw $s0, p1 #almacenamos en s0 el valor de p1

lw $s1, p2 #almacenamos en s1 el valor de p2

move $a0, $s0 #instrucciones simples para imprimir los valores que tienen al comienzo

li $v0, 1

syscall

move $a0, $s1

li $v0, 1

syscall

jal swap #llamada a la función swap

move $a0, $s0 #instrucciones simples para imprimir los valores que tienen tras la función

li $v0, 1

syscall

move $a0, $s1

li $v0, 1

syscall

li $v0, 10 #finalización del programa

syscall

swap: #etiqueta de la función

li $t0, 0 #auxiliar

move $t0, $s0 #aux = p1

move $s0, $s1 #p1 = p2

move $s1, $t0 #p2 = aux

jr $ra #vuelve al main

**Descripción:** No hace falta introducir ningún número, sólo ejecutar el programa. En este no he encontrado ninguna dificultad.

**Problema 2:** Codificación XOR (1 punto) Escribir un programa que pida al usuario una cadena de caracteres y la codifique haciendo a cada carácter (excepto el terminador) una operación XOR con el valor 63 decimal. La cadena resultante se imprime después por pantalla. A la hora de probar el programa (como usuario), comprobar que, si se le da una cadena C1 como entrada y se obtiene una cadena C2 como resultado, si la siguiente vez se le da C2 como entrada, se obtiene un resultado C3==C1 (esto se debe a que a XOR a = 1)

.data

C1: .space 100 #almacena la cadena que introducimos

C2: .space 100 #almacena la cadena traducida

cadena\_introduce: .asciiz "Introduzca una cadena: "

cadena\_xor: .asciiz "La nueva cadena seria: "

.text

.globl main

main:

li $t0, 0 #inicialización del iterador i

la $a0, cadena\_introduce #instrucciones simples para introducir la cadena C1

li $v0, 4

syscall

li $v0, 8

la $a0, C1

li $a1, 100

syscall

for: #for para recorrer la cadena carácter a carácter

lb $s0, C1($t0) #almacenamos en $s0, el carácter C1[i]

beqz $s0, fin\_for #si es el final, termina el for

xor $s0, $s0, 63 #instrucción XOR entre 63 y el carácter

sb $s0, C2($t0) #almacenamos en C2[i] el carácter $s0

addi $t0, $t0, 1 #i++

j for #salto al for

fin\_for:

la $a0, cadena\_xor #instrucciones simples para imprimir la cadena C2

li $v0, 4

syscall

la $a0, C2

li $v0, 4

syscall

li $v0, 10 #fin del programa

syscall

**Descripción:** Sólo habría que introducir la cadena. Mi principal dificultad fue no darme cuenta de que los caracteres miden un byte; y en la comprobación de la cadena aparece un 5.

**Problema 3:** Codificación XOR 2 (0,5 puntos) Cambiar el programa anterior para que la codificación se haga en una función auxiliar y no en el programa principal. Esta función toma como único parámetro un puntero a cadena ASCII con terminador nulo (.asciiz) y no devuelve resultados.

.data

C1: .space 100 #almacena la cadena que introducimos

C2: .space 100 #almacena la cadena traducida

cadena\_introduce: .asciiz "Introduzca una cadena:"

cadena\_xor: .asciiz "La nueva cadena seria:"

.text

.globl main

main:

li $t0, 0 #inicialización del iterador i

la $s0, C1 #cargamos la dirección de memoria de C1 en s0

la $a0, cadena\_introduce #instrucciones simples para introducir la cadena C1

li $v0, 4

syscall

li $v0, 8

la $a0, C1

li $a1, 100

syscall

jal xor2 #salto a la función xor2

la $a0, cadena\_xor #instrucciones simples para imprimir la cadena C2

li $v0, 4

syscall

la $a0, C2

li $v0, 4

syscall

li $v0, 10 #fin del programa

syscall

xor2: #etiqueta del programa xor2

lb $s0, C1($t0) #almacenamos en $s0, el carácter C1[i]

beqz $s0, fin\_xor2 #si es el final, termina el xor2

xor $s0, $s0, 63 #instrucción XOR entre 63 y el carácter

sb $s0, C2($t0) #almacenamos en C2[i] el carácter $s0

addi $t0, $t0, 1 #i++

j xor2 #salto al xor2

fin\_xor2:

jr $ra #salta al main

**Descripción:** Es igual que el programa 2, salvo que, en vez de hacer la codificación en el main, lo hacemos en una función aparte. Al igual que el anterior tiene el mismo problema con el 5.

**Problema 4:** strcat (1 punto) Programar una función llamada strcat que tome como parámetros dos punteros a cadena: orig y dest. La función hará una copia de la cadena apuntada por orig a partir de donde termina la cadena apuntada por dest, efectuando así la concatenación de ambas cadenas. La función no devuelve resultados. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento.

.data #inicialización de las dos cadenas iniciales, y reservamos el espacio para la cadena concatenada

orig: .asciiz "Bien"

dest: .asciiz " Holacomoestas"

cat: .space 400

cadena\_cat: .asciiz "Cadena concatenada:"

.text

.globl main

main:

li $t0, 0 #iterador i = 0 (que recorre la cadena destino y la cat, ya que origen se copiara a continuación de la de destino)

li $t1, 0 #iterador j=0 (que recorre la cadena origen para copiarla en cat)

jal strcat #salta a la función strcat

la $a0, cadena\_cat #instrucciones simples para imprimir la cadena cadena\_cat

li $v0, 4

syscall

la $a0, cat #instrucciones simples para imprimir la cadena cat

li $v0, 4

syscall

li $v0, 10 #fin del programa

syscall

strcat: #etiqueta de la función strcat

lb $s0, dest($t0) #almacenamos en s0 el carácter dest[i]

beqz $s0, origen #si es el final de la cadena, salta a origen

sb $s0, cat($t0) #almacenamos en cat[i], el carácter que hay en s0

addi $t0, $t0, 1 #i++

j strcat #saltamos a strcat

origen: #etiqueta de la función origen

lb $s1, orig($t1) #almacenamos en s1 el carácter orig[j]

beqz $s1, fin\_strcat #si es el final de la cadena, salta a fin\_strcat

sb $s1, cat($t0) #almacenamos en cat[j], el carácter que hay en s1

addi $t1, $t1, 1 #j++

addi $t0, $t0, 1 #i++

j origen

fin\_strcat:

jr $ra #volvemos al main

**Descripción:** No hace falta introducir nada, sólo ejecutar el programa. Como este ejercicio lo hice después del 2, ya no tenía problemas a la hora de recorrer las cadenas.

**Problema 5:** isdigit (0.5 puntos) Programar una función llamada isdigit que tome como parámetro un carácter ASCII. Se comprobará si su código ASCII está comprendido entre ‘0’ y ‘9’. Si el carácter dado está en ese intervalo (es por tanto un carácter numérico), la función debe devolver un valor entero 1. En caso contrario devolverá un valor 0. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento.

.data #guardamos espacio para el carácter, y las cadenas que nos avisaran si es letra o numero

caracter: .space 1

cadena\_introduce: .asciiz "Introduce un caracter:\n"

cadena\_num: .asciiz "\nEste caracter es un numero"

cadena\_let: .asciiz "\nEste caracter es una letra"

.text

.globl main

main:

la $a0, cadena\_introduce #instrucciones simples para imprimir la cadena introduce

li $v0, 4

syscall

li $v0, 12 #instrucciones simples para introducir por teclado el carácter

syscall

move $a0, $v0

jal isdigit #llamada a la función isdigit

beqz $v0, letra #como si es letra devuelve 0, pues salta a la función letra, si no, sigue adelante

la $a0, cadena\_num #instrucciones simples para imprimir la cadena numero

li $v0, 4

syscall

j fin #salta a fin, porque si siguiera se imprimiría también la cadena de letra

letra:

la $a0, cadena\_let #instrucciones simples para imprimir la cadena letra

li $v0, 4

syscall

fin:

li $v0, 10 #fin del programa

syscall

isdigit: #etiqueta de la función isdigit

blt $a0, 48, fin\_isdigit #si el carácter está comprendido entre el 48

bgt $a0, 57, fin\_isdigit #y el 57 (0-9 en la tabla ascii)

li $v0, 1 #devuelve 1, si no, salta a fin\_isdigit

j fin\_atras

fin\_isdigit:

li $v0, 0 #como es letra devuelve 0

fin\_atras:

jr $ra #volver al main

**Descripción:** Solo haría falta introducir un carácter. Mi problema en este programa fue que imprimía tanto la cadena de letra como la de número; ya que no le había puesto el salto a fin, y seguía.

**Problema 6:** ASCII a entero (0.5 puntos) Programar una función llamada valor que tome como parámetro un carácter ASCII numérico (un dígito del 0 al 9) y devuelva un entero con el valor entero correspondiente a ese dígito. Ejemplo: si se le da como parámetro el carácter ASCII ‘3’ (valor numérico 51hex) debe devolver un entero de valor 3. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento. Se debe emplear la función isdigit del ejercicio anterior para comprobar que el carácter dado es un dígito válido. En caso de error se debe devolver un valor numérico -1.

.data #cadenas instructivas

cadena\_introduce: .asciiz "Introduzca una caracter: "

cadena\_hexa: .asciiz "\nEl valor numerico en hexadecimal es: "

.text

.globl main

main:

li $t0, 0 #inicializar el contador

la $a0, cadena\_introduce #instrucciones básicas para imprimir por pantalla la cadena introduce

li $v0, 4

syscall

li $v0, 12 #introducir por teclado el carácter

syscall

move $a0, $v0

jal isdigit #salta a función isdigit

beq $v0, -1, fin #si se devuelve -1 es una letra, por lo tanto, no se devuelve el valor hexadecimal

la $a0, cadena\_hexa #instrucciones básicas para imprimir la cadena hexa

li $v0, 4

syscall

move $a0, $t0 #imprime el valor en hexadecimal correspondiente en la tabla ascii

li $v0, 1

syscall

fin:

li $v0, 10 #fin del programa

syscall

isdigit: #etiqueta de la función isdigit

blt $a0, 48, fin\_isdigit #el funcionamiento de isdigit esta explicada en el ejercicio anterior

bgt $a0, 57, fin\_isdigit

j valor #salta a valor

valor:

beq $t0, $a0, fin\_atras #mientras que el contador sea distinto al número hexadecimal introducido

addi $t0, $t0, 1 #el contador seguirá sumando

j valor

fin\_isdigit: #esto pasa cuando es letra

li $v0, -1

fin\_atras:

jr $ra #volver al main

**Descripción:** Sólo habría que introducir el carácter. A la hora de programar, sólo era reutilizar la función anterior y poner un contador, por lo tanto, no habría problema.

**Problema 7:** Repeticiones (1.5 puntos) Escribir un programa en ensamblador MIPS que:

1. Tenga declarado en memoria un vector de 10 enteros tamaño media palabra.
2. Recorra el vector, contando el número de veces que ocurre que un elemento y el siguiente sean iguales.
3. Guarde el resultado (el número de repeticiones) en el registro v0.
4. Finalice con una llamada a la función del sistema número 10 (exit).

.data #inicializamos el vector

vector: .half 1, 2, 3, 4, 5, 5, 5, 5, 8, 8

cad\_total: .asciiz "El numero total de repeticiones es: "

.text

.globl main

main:

li $t0, 0 #contador del iterador

li $v0, 0 #contador de las repeticiones

repeticiones:

sll $t1, $t0, 1 #en t1, llevamos el iterador i del desplazamiento (1 porque es half)

lh $s0, vector($t1) #guardamos en s0 el valor de vector[i]

lh $s1, vector + 2($t1) #guardamos en s1 el valor de vector [i+1]

bne $s0, $s1, rep #si ambas son distintas salta a rep

addi $v0, $v0, 1 #como son iguales el contador se incrementa a 1

rep:

addi $t0, $t0, 1 #contador del iterador + 1

blt $t0, 9, repeticiones #mientras que este contador sea menor que 9 salta a repeticiones

move $t2, $v0 #lo movemos a t2, ya que v0, obtendrá ahora otro valor y se perdería

la $a0, cad\_total #se imprime la cadena total

li $v0, 4

syscall

move $a0, $t2 #imprime las repeticiones

li $v0, 1

syscall

li $v0, 10 #se acaba el programa

syscall

**Descripción:** No hace falta introducir nada, la función recorre el vector y compara la posición i, con la siguiente. Si ambas son iguales se suma uno al contador. El problema que he tenido con este programa, que no sé si estará bien solventado, es el tema de no utilizar punteros móviles.

**Problema 8:** Repeticiones 2 (1 punto) Modificar el programa anterior para que el proceso de recorrer el vector y contar repeticiones se realice en una función auxiliar. A la función se le pasan dos parámetros: (1) un puntero al inicio del vector, y (2) el número de elementos del vector. La función devuelve como resultado el número de repeticiones encontradas.

.data #inicializamos el vector

vector: .half 1, 2, 3, 4, 5, 5, 5, 5, 8, 8

cad\_total: .asciiz "El numero total de repeticiones es: "

.text

.globl main

main:

li $t0, 0 #contador del iterador

la $s0, vector #almacenamos en s0 la dirección de memoria del vector

move $a0, $s0

jal repeticiones #salta a repeticiones

move $t2, $v0 #lo movemos a t2, ya que v0, obtendrá ahora otro valor y se perdería

la $a0, cad\_total #se imprime la cadena total

li $v0, 4

syscall

move $a0, $t2 #imprime las repeticiones

li $v0, 1

syscall

li $v0, 10 #fin del programa

syscall

repeticiones:

addi $sp, $sp, -4 #PUSH

sw $s0, 0($sp)

inicio:

sll $t1, $t0, 1 #en t1, llevamos el iterador i del desplazamiento (1 porque es half)

lh $s0, vector($t1) #a partir de aquí, funciona como el programa anterior

lh $s1, vector + 2($t1)

bne $s0, $s1, rep

addi $v0, $v0, 1

rep:

addi $t0, $t0, 1

blt $t0, 9, inicio

lw $s0, 0($sp) #POP

addi $sp, $sp, +4

jr $ra #vuelve al main

**Descripción:** Como es una mejora del programa anterior, mi problema ha seguido siento el mismo, por lo tanto, he usado la pila.

**Problema 9:** Comprobar números primos (1.5 puntos) Escribir un programa que pida al usuario un número natural n y le conteste si este número es primo o no. Para la comprobación se debe usar una función primo que compruebe el resto (instrucción REM) de dividir el número n entre cada natural mayor que 1 y menor que n/2 (o que la raíz cuadrada de n). Si alguno de los restos es cero, el número no es primo. En caso contrario, sí es primo. No es necesario mejorar el programa evitando comprobaciones innecesarias.

.data #cadenas instructivas, y guardamos el espacio para el numero

numero: .space 4

cad\_introduce: .asciiz "Introduce un numero: "

cad\_primo: .asciiz "\nEs primo"

cad\_noprimo: .asciiz "\nNo es primo"

.text

.globl main

main:

la $a0, cad\_introduce #instrucciones para imprimir la cadena introduce

li $v0, 4

syscall

li $v0, 5 #introducir un numero por teclado

syscall

sw $v0, numero

jal es\_primo #salta a la función es\_primo

beqz $t0, no\_primo #como si no es primo, devuelve 0, salta a no primo

la $a0, cad\_primo #si devuelve 1 si es primo, por lo tanto, imprime esa cadena

li $v0, 4

syscall

j fin #salta a fin

no\_primo:

la $a0, cad\_noprimo #imprime la cadena no primo

li $v0, 4

syscall

fin: #final del programa

li $v0, 10

syscall

es\_primo: #etiqueta de la función es primo

beqz $v0, no\_lo\_es #contamos con que el 0, 1, y 4 no son primos

beq $v0, 1, no\_lo\_es

beq $v0, 4, no\_lo\_es

li $t1, 2 #x

li $t2, 2 #2

div $v0, $t2 #dividimos el numero entre 2, para hacer una de las comparaciones

mflo $a0 #nos quedamos el cociente y la almacenamos a a0

for\_primo:

bgt $t1, $a0, fin\_for # si x > numero/2 se acaba el bucle

rem $t2, $v0, $t1 #si no, almacenamos en t2, el resto del número entre x

div $v0, $t2 #seguimos dividiendo entre el resto y almacenando el cociente

mflo $a0

bnez $t2, fin\_if #hasta que t2, no sea 0, que, por lo tanto, si es primo

j no\_lo\_es #si en algún momento el resto es 0, no es primo

fin\_if:

addi $t1, $t1, 1 #x+1

j for\_primo #volvemos a for primo

fin\_for:

j si\_lo\_es

no\_lo\_es: #no primo, devuelve 0

li $t0, 0

jr $ra

si\_lo\_es: #si primo, devuelve 1

li $t0, 1

jr $ra

**Descripción:** Se introduce el número que queramos comprobar si es primo o no. Para comprobar si este funcionaba correctamente, hice uso del programa siguiente. En ejecución no hubo problemas, pero tanto en este como en el siguiente estaba la dificultad de hallar los números primos.

**Problema 10:** Criba (2 puntos) Escribir un programa que encuentre e imprima por pantalla los números primos menores que N=100 por el método de la criba de Eratóstenes. Se permite usar versiones simplificadas que hagan más comprobaciones de las necesarias, pero el resultado final debe ser correcto.

.data #solo es la cadena para imprimir la coma

coma: .asciiz ","

.text

.globl main

main:

li $t0, 0 #i = 0

li $s0, 2 #el 2, 3, 5, 7 son los divisores utilizados para criba, por lo tanto, lo almacenamos en s0, s1, s2 y s3

li $s1, 3

li $s2, 5

li $s3, 7

for:

bgt $t0, 100, fin\_for #cuando i>100, se termina el bucle

jal criba #salta a la criba

addi $t0, $t0, 1 #se incrementa el iterador del bucle

j for #salta al bucle

fin\_for:

li $v0, 10 #fin del programa

syscall

criba:

beq $t0, 2, cambio #como estos ya sabemos que son primos, saltamos a cambio

beq $t0, 3, cambio

beq $t0, 5, cambio

beq $t0, 7, cambio

rem $t2, $t0, $s0 #empezamos las comparaciones, a ver si el numero t0, es múltiplo de s0, s1, s2 o s3

beqz $t2, fin

rem $t2, $t0, $s1

beqz $t2, fin

rem $t2, $t0, $s2

beqz $t2, fin

rem $t2, $t0, $s3

beqz $t2, fin

cambio: #aquí se imprime el número, ya que si es primo

move $a0, $t0

li $v0, 1

syscall

li $v0, 4 #imprime la coma

la $a0, coma

syscall

fin:

jr $ra #salta a main

**Descripción:** No hace falta introducir nada, sólo ejecutarlo. Mi principal problema fue que no tenía claro si inicializar un vector de 100 elementos y otro de copia con los primos; pero lo veía podo eficiente; así que decidí hacerlo por la sucesión de números sin necesidad de ningún vector.

**Problema 11:** Burbuja (3 puntos) Programar una función llamada bsort que implemente el algoritmo de ordenación de la burbuja. Recibe como parámetros:

* Un puntero p a un vector de enteros de tamaño Word.
* Un entero N.

.data #cadenas informativas

cad\_inicial: .asciiz "Vector inicial: "

cad\_coma: .asciiz ", "

cad\_final: .asciiz "\nVector final: "

vector: .word 2, 3, 4, 1, 5, 6, 7, 3, 4, 5

.text

.globl main

main:

li $t0, 0 #iterador que recorre el vector

la $a0, cad\_inicial #se imprime la cadena inicial

li $v0, 4

syscall

for\_1: #tanto for\_1 como for\_2 son bucles que recorren e imprime el vector, por lo tanto, no les voy a dar importancia

sll $t1, $t0, 2

bgt $t0, 9, seguir\_1

lw $a0, vector($t1)

li $v0, 1

syscall

la $a0, cad\_coma

li $v0, 4

syscall

addi $t0, $t0, 1

j for\_1

seguir\_1:

jal bsort #saltamos a la función bsort

li $t0, 0 #volvemos a poner el iterador a 0

la $a0, cad\_final

li $v0, 4

syscall

for\_2:

sll $t1, $t0, 2

bgt $t0, 9, seguir\_2

lw $a0, vector($t1)

li $v0, 1

syscall

la $a0, cad\_coma

li $v0, 4

syscall

addi $t0, $t0, 1

j for\_2

seguir\_2: #se acaba el programa

li $v0, 10

syscall

bsort: #etiqueta de la función bsort

li $t1, 10 #tamaño

li $t2, 1 #i

li $t3, 0 #j

for\_bsort1: #este es el bucle exterior que recorre el vector

bgt $t2, $t1, fin\_for1 #hasta que el t2 supere el tamaño del vector

sub $t4, $t1, $t2 #almacenamos en t4 la resta del tamaño - i

for\_bsort2: #este es el bucle interno que recorre el vector

sll $t5, $t3, 2 #t5 es el iterador que nos indique la posición del vector

bge $t3, $t4, fin\_for2 # j > tamaño - i

lw $s0, vector($t5) #almacenamos en s0, la posición vector[t5]

lw $s1, vector + 4($t5) #almacenamos en s1, la posición vector[t5+1]

blt $s0, $s1, fin\_if #comparamos ambos si s0 > s1, continua, si no, salta a fin\_if

move $t6, $s0 #aux = vector[t5]

move $s0, $s1 #vector[t5] = vector[t5 + 1]

move $s1, $t6 #vector[t5 + 1] = aux

sw $s0, vector($t5) #almacenamos a vector[t5] el nuevo valor de s0

sw $s1, vector + 4($t5) #almacenamos a vector[t5 + 1] el nuevo valor de s1

fin\_if:

addi $t3, $t3, 1 #j++

j for\_bsort2

fin\_for2:

li $t3, 0 #volvemos a poner j =0

addi $t2, $t2, 1 #i++

j for\_bsort1

fin\_for1:

jr $ra #vuelta a main

**Descripción:** Sólo habría que ejecutar el programa. Sin duda me parece el programa más complicado, no de programar si no, de como entender el funcionamiento del método de la burbuja.