**Trabajo del tema 2: Ensamblador MIPS**

Nombre y apellidos: Jesús Rodríguez Heras

En colaboración con: Jaime Barea Paredes

**Problema 1.-** (0.5 puntos) Programar una función llamada *abs* que tome como parámetro un número entero con signo y devuelva el valor absoluto de ese mismo entero. Emplear esta función en un programa principal sencillo para demostrar su funcionamiento.

##### función absv #####

.globl absv

absv:

add $t0,$t0,$a0 # Utilizamos el registro $t0 para almacenar el número

bgez $t0,fin\_if # Estructura selectiva simple para que, en caso de que el número sea negativo, nos devuelva su valor absoluto (multiplicando por -1)

mul $t0,$t0,-1 # Multiplicamos el número negativo por -1. Esa sentencia nos guardará el resultado en $t0

move $v0,$t0 # Movemos el resultado de $t0 a $v0 para pasarlo a la función ppal

fin\_if:

jr $ra # Salto a la función ppal

##### Fin función absv #####

Al iniciar el programa, se pide al usuario que introduzca un número por teclado para calcular su valor absoluto. Para ello, el número es enviado a la función que, en caso de que el número sea negativo, lo multiplicará por “-1” para obtener su valor absoluto y devolverá en pantalla el valor absoluto del número. En caso de no ser negativo, el programa se salta la estructura selectiva y devuelve directamente el valor absoluto del número.

**Problema 2.-** (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.

La función recibe como parámetros dos punteros a enteros e intercambia las posiciones de memoria de dichos enteros. Luego, devuelve el control a la función principal que imprimirá los números en orden contrario debido a que se han cambiado la posición de memoria de ambos enteros.

##### función swap #####

.globl swap

swap:

move $t1,$a0 # Movemos el valor ubicado en $a0 a $t1 (el cual utilizamos como registro auxiliar

move $a0,$a1 # Movemos el valor de $a1 a $a0

move $a1,$t1 # Movemos el valor de $t1 a $a1

move $v0,$a0 # Movemos el valor de $a0 a $v0

move $v1,$a1 # Movemos el valor de $a1 a $v1

jr $ra # Salto a la función ppal

##### Fin función swap #####

**Problema 3.-** (1 punto) Programar una función llamada strupr que tome como parámetros dos punteros a cadena: orig y dest. La función hará en dest una copia de la cadena apuntada por orig, pero sustituyendo las letras minúsculas que aparezcan por letras mayúsculas. La función no devuelve resultados.

##### función strupr #####

.globl strupr

strupr:

li $t0, 0 # $t0 = i = índice/contador

strupr\_while:

add $t2,$a1,$t0 # Utilizamos el registro $t2 como índice para recorrer los vectores

lb $t1,0($t2) # Carácter = puntero auxiliar

beqz $t1,strupr\_fin # Bucle para controlar el recorrido por los vectores

blt $t1,97,fin\_if # Estructura condicional para establecer el límite entre mayúsculas y minúsculas

bgt $t1,122,fin\_if

sub $t1,$t1,32 # Restamos 32 al carácter ASCII para obtener la letra mayúscula

fin\_if:

add $t2,$a0,$t0 # Registro $t2 como índice para recorrer el vector

sb $t1,0($t2) # Puntero auxiliar = nuevo carácter

addi $t0,$t0,1 # Incrementamos el índice

j strupr\_while # Reiterar

strupr\_fin:

jr $ra # Salto a la función ppal

##### Fin función strupr #####

El programa recibe la cadena inicial y luego la copia en otro registro para realizar un decremento en el código ASCII de las letras minúsculas para transformarlas en mayúsculas. La estructura condicional del interior del bucle establece el límite entre mayúsculas y minúsculas para que el programa se ejecute correctamente.

**Problema 4.-** (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.

##### función strcat #####

.globl strcat # La función strcat contará los caracteres de la segunda cadena y luego concatenará el contenido de la primera cadena con el contenido de la segunda cadena, uniéndolo todo en la segunda cadena.

strcat:

li $t0, 0 # $t0=contador de la longitud de la cadena

contador: # El contador nos sirve para saber cuál es la longitud de la cadena

add $t2,$a0,$t0 # Recorremos la primera cadena para ver su longitud

lb $t1,0($t2) # Carácter = puntero auxiliar

beqz $t1,fin\_contador # Bucle para recorrer la primera cadena

add $t0,$t0,1 # Incrementamos el registro $t0 con cada iteración para incrementar el valor de la longitud de la cadena

j contador # Reiteración

fin\_contador:

strcat\_while:

add $t1,$a1,$t4 # Recorremos la primera cadena

lb $t3,0($t1) # Carácter = puntero auxiliar

beqz $t3,strcat\_fin # Bucle para controlar el recorrido por los vectores

add $t1,$a0,$t0 # Recorremos la primera cadena

sb $t3,0($t1) # Puntero auxiliar = nuevo carácter

addi $t0,$t0,1 # Incrementamos el índice $t0

addi $t4,$t4,1 # Incrementamos el índice $t4

j strcat\_while # Reiteración

strcat\_fin:

jr $ra # Salto a la función ppal

##### Fin función strcat #####

La función recibe como parámetros dos punteros a las cadenas origen y destino. Primero mide la longitud de la segunda cadena (destino) y luego, a partir de su último carácter, copia la cadena origen en la cadena destino. La función principal imprimirá ambas cadenas concatenadas.

**Problema 5.-** (0.5 puntos) Programar una función llamada isdigit que tome como parámetro un carácter ASCII. Si el carácter dado es numérico (un dígito del 0 al 9), la función debe devolver un valor entero 1. En caso contrario devolverá un valor 0. La comprobación debe ser compacta y eficiente.

##### función isdigit #####

.globl isdigit

isdigit:

li $t0, 0 # $t0 = i = índice/contador

isdigit\_while:

add $t2,$a0,$t0 # Utilizamos el registro $t2 como índice para recorrer los vectores

lb $t1,0($t2) # Carácter = puntero auxiliar

beqz $t1,isdigit\_fin # Bucle para controlar el recorrido por los vectores

blt $t1,48,else # Estructura condicional para establecer el límite entre números y caracteres

bgt $t1,57,else

add $t1,$zero,49 # Sumamos el carácter ASCII 49 (número 1) al registro $t1 para que en caso de que haya un número escriba un 1

j fin\_if

else:

add $t1,$zero,48 # Sumamos el carácter ASCII 48 (número 0, al registro $t1 para que en caso de que haya un carácter escriba un 0

fin\_if:

add $t2,$a0,$t0 # Registro $t2 como índice para recorrer el vector

sb $t1,0($t2) # Puntero auxiliar = nuevo carácter

addi $t0,$t0,1 # Incrementamos el índice

j isdigit\_while # Reiterar

isdigit\_fin:

jr $ra # Salto a la función ppal

##### Fin función isdigit #####

La función recibe un puntero a la cadena origen formada por números y por letras. Esta función cambia los números por 1 y el resto de caracteres por 0. Para ello, la estructura condicional del interior del bucle establece el límite entre números y caracteres para que la función se ejecute correctamente.

**Problema 7.-** (1.5 puntos) Programar una función llamada strrev que reciba un puntero a cadena e invierta el orden de los caracteres de dicha cadena. No devuelve resultados. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento. Existen varias estrategias válidas para conseguir este comportamiento:

a) La función almacena uno a uno los caracteres en la pila, y seguidamente los recupera uno a uno y los va almacenando en memoria en el orden inverso al que tenían originalmente.

b) La función encuentra dónde está el último carácter (no nulo) de la cadena y obtiene así un puntero al último carácter. A partir de ahí lleva dos punteros que apuntan a los caracteres que hay que intercambiar entre sí. El intercambio de caracteres se hace usando la función swap del ejercicio anterior (convenientemente modificada para que maneje datos de tamaño byte).

##### función strrev #####

.globl strrev

strrev:

li $t0,0 # Contador

bucle:

add $t2,$a0,$t0 # Recorremos la cadena

lb $t1,0($t2) # Caracter = puntero auxiliar

blez $t1,fin\_bucle # Bucle para controlar el recorrido del vector

add $sp,$sp,-1 # Reservamos tamaño en pila

sb $t1, 0($sp) # Cargamos el caracter en la pila de tamaño byte

add $t0,$t0,1 # Incrementamos el contador

j bucle # Reitera

fin\_bucle:

bucleimp:

lb $t1,0($sp) # Caracter = puntero auxiliar

add $sp,$sp,1 # Implementamos un hueco en pila de tamaño byte

blez $t1,fin\_bucleimp # Bucle para controlar el recorrido del vector

add $t2,$a0,$t7 # Recorremos la cadena

sb $t1,0($t2) # Caracter = puntero auxiliar

add $t7,$t7,1 # incrementamos el contador

j bucleimp # Reitera

El programa recibe un puntero a una cadena e invierte los caracteres de la cadena. Para ello, en mi caso utilizo la opción a) aportada por el enunciado la cual consiste en almacenar los caracteres en la pila uno a uno y luego los saca de ella, con lo cual, quedan escritos en orden contrario. La función devuelve el control a la función principal que luego imprimirá la cadena con los caracteres invertidos.

fin\_bucleimp:

jr $ra # Salto a la función ppal

##### Fin función strrev #####

**Problema 10.-** “Calculadora”. Programar un programa que pida al usuario una operación aritmética. Esta operación se recibirá en forma de cadena de caracteres. El programa evaluará la expresión introducida y presentará por pantalla el resultado del cálculo. Se recomienda hacer un diseño incremental, partiendo de un funcionamiento básico y evolucionando en versiones subsiguientes hacia funcionamientos más complejos. Puntuación:

- 3 puntos si responde correctamente a las cuatro operaciones básicas. En esta primera versión se permite que los números introducidos tengan que ser de un solo dígito y que solamente se introduzca una operación a la vez. Ej:

Introduzca una operación:

> 2\*7

= 14

Introduzca una operación:

- +1 punto si admite varias operaciones encadenadas. En esta versión todas las operaciones se ejecutarán de izquierda a derecha sin tener en cuenta orden de precedencia. Ej:

Introduzca una operación:

> 2+1\*5

= 15

Introduzca una operación:

Pista: Una vez introducida la cadena, recorrerla con un puntero+índice. Mediante bloques de ejecución condicional, actuar de manera diferente según si se encuentra un dígito, un operador o un espacio.

- +1 punto si muestra un mensaje de error cuando la cadena introducida no está bien formada (por ej. contiene algún signo no soportado por el programa, o tiene sintaxis inadecuada). Ej:

Introduzca una operación:

> 2+1 3\*5

Error. Introduzca una operación:

> 3\*3ñ

Error. Introduzca una operación:

> 34/2

= 17

Introduzca una operación:

- +1 punto si además admite números de varias cifras. Se sugiere haber realizado antes el problema 17 y utilizar aquella función en este programa.

- +2 puntos si admite paréntesis y los procesa correctamente (un solo nivel de paréntesis, no anidados).

- +1 punto si admite paréntesis anidados y los procesa correctamente.

Pista: tomar la cadena que queda entre los paréntesis y procesarla en una llamada recursiva.

- +2 puntos si tiene en cuenta orden de precedencia de operaciones: multiplicación/división se deben operar antes que suma/resta.

- Retos adicionales fuera de puntuación:

a) Introducir operaciones nuevas, p.ej. potenciación o raíz cuadrada.

b) Añadir funciones trigonométricas u otras funciones científicas.

c) Añadir funcionalidad de memoria (como la de calculadoras tradicionales: memorizar el resultado de la operación anterior y posteriormente recuperarlo a petición del usuario).

.data

pregunta: .asciiz "¿Qué operación desea realizar?\n> "

resultado: .ascii "= "

.text

ppal:

la $a0, pregunta # Cargamos la dirección de la etiqueta pregunta

li $v0, 4 # Imprimimos la cadena

syscall # Se imprime la cadena

li $v0, 8 # Leemos la operación introducida por el usuario

li $a1, 8

syscall

jal calculadora # Función calculadora

la $a0, resultado # Cargamos la dirección de la etiqueta pregunta

li $v0, 4 # Imprimimos la cadena

syscall # Se imprime la cadena

move $a0,$v1 # Movemos el resultado de $v0 a $a0 para introducirlo en el syscall

li $v0, 1 # Imprimimos el resultado de la operación

syscall

li $v0,10

syscall # Fin del programa

################### Fin programa ppal ###################

##### función calculadora #####

.globl calculadora

calculadora:

li $t0, 0 # $t0 = i = Indice

calculadora\_while:

add $t2,$a0,$t0 # Utilizamos el registro $t2 como índice para recorrer el vector

lb $t1,0($t2) # Caracter = puntero auxiliar

beqz $t1,calculadora\_fin # Bucle para controlar el recorrido por el vector

beq $t1,43,suma # Sentencia condicional para la suma

beq $t1,45,resta # Sentencia condicional para la resta

beq $t1,42,producto # Sentencia condicional para el producto

beq $t1,47,division # Sentencia condicional para la división

add $t0,$t0,1 # Incremento del contador

j calculadora\_while # Reitera

##############################################################

suma:

add $t2,$a0,$t7 # Utilizamos el registro $t2 como índice para recorrer el vector

lb $t1,0($t2) # Caracter = puntero auxiliar

beqz $t1,calculadora\_fin # Bucle para recorrer la operación inicial en busca de los dígitos de la suma

blt $t1,48,fin\_if\_suma # Estructura condicional para para establecer el límite ASCII entre números y el resto de caracteres

bgt $t1,57,fin\_if\_suma

bnez $t3,else\_suma # Si el registro $t3 está ocupado, lo almacenará en $t4

sub $t3,$t1,48 # Restamos 48 a $t3 para obtener el número en decimal a partir del código ASCII

add $t7,$t7,1 # Incrementamos el contador

j suma # Salta a suma

else\_suma: # Almacena el número en $t4 si el registro $t3 está ocupado

sub $t4,$t1,48 # Restamos 48 a $t4 para obtener el número en decimal a partir del código ASCII

add $t5,$t3,$t4 # Ejecutamos la suma

move $v1,$t5 # Movemos el contenido del registro $t5 a $v1 para pasarlo a la función ppal

fin\_if\_suma:

add $t7,$t7,1 # Incrementamos el contador

j suma

###############################################################

resta:

add $t2,$a0,$t7 # Utilizamos el registro $t2 como índice para recorrer el vector

lb $t1,0($t2) # Caracter = puntero auxiliar

beqz $t1,calculadora\_fin # Bucle para recorrer la operación inicial en busca de los dígitos de la resta

blt $t1,48,fin\_if\_resta # Estructura condicional para para establecer el límite ASCII entre números y el resto de caracteres

bgt $t1,57,fin\_if\_resta

bnez $t3,else\_resta # Si el registro $t3 está ocupado, lo almacenará en $t4

sub $t3,$t1,48 # Restamos 48 a $t3 para obtener el número en decimal a partir del código ASCII

add $t7,$t7,1 # Incrementamos el contador

j resta # Sala a resta

else\_resta: # Almacena el número en $t4 si el registro $t3 está ocupado

sub $t4,$t1,48 # Restamos 48 a $t4 para obtener el número en decimal a partir del código ASCII

sub $t5,$t3,$t4 # Ejecutamos la resta

move $v1,$t5 # Movemos el contenido del registro $t5 a $v1 para pasarlo a la función ppal

fin\_if\_resta:

add $t7,$t7,1 # Incrementamos el contador

j resta

###############################################################

producto:

add $t2,$a0,$t7 # Utilizamos el registro $t2 como índice para recorrer el vector

lb $t1,0($t2) # Caracter = puntero auxiliar

beqz $t1,calculadora\_fin # Bucle para recorrer la operación inicial en busca de los dígitos del producto

blt $t1,48,fin\_if\_producto # Estructura condicional para para establecer el límite ASCII entre números y el resto de caracteres

bgt $t1,57,fin\_if\_producto

bnez $t3,else\_producto # Si el registro $t3 está ocupado, lo almacenará en $t4

sub $t3,$t1,48 # Restamos 48 a $t3 para obtener el número en decimal a partir del código ASCII

add $t7,$t7,1 # Incrementamos el contador

j producto # Salta a producto

else\_producto: # Almacena el número en $t4 si el registro $t3 está ocupado

sub $t4,$t1,48 # Restamos 48 a $t4 para obtener el número en decimal a partir del código ASCII

mul $t5,$t3,$t4 # Ejecutamos el producto

move $v1,$t5 # Movemos el contenido del registro $t5 a $v1 para pasarlo a la función ppal

fin\_if\_producto:

add $t7,$t7,1 # Incrementamos el contador

j producto

###############################################################

Este programa consiste en una calculadora que realiza las cuatro operaciones básicas.

division:

add $t2,$a0,$t7 # Utilizamos el registro $t2 como índice para recorrer el vector

lb $t1,0($t2) # Caracter = puntero auxiliar

beqz $t1,calculadora\_fin # Bucle para recorrer la operación inicial en busca de los dígitos de la división

blt $t1,48,fin\_if\_division # Estructura condicional para para establecer el límite ASCII entre números y el resto de caracteres

bgt $t1,57,fin\_if\_division

bnez $t3,else\_division # Si el registro $t3 está ocupado, lo almacenará en $t4

sub $t3,$t1,48 # Restamos 48 a $t3 para obtener el número en decimal a partir del código ASCII

add $t7,$t7,1 # Incrementamos el contador

j division # Salta a división

else\_division: # Almacena el número en $t4 si el registro $t3 está ocupado

sub $t4,$t1,48 # Restamos 48 a $t4 para obtener el número en decimal a partir del código ASCII

div $t5,$t3,$t4 # Ejecutamos la división

move $v1,$t5 # Movemos el contenido del registro $t5 a $v1 para pasarlo a la función ppal

fin\_if\_division:

add $t7,$t7,1 # Incrementamos el contador

j division

calculadora\_fin:

jr $ra # Salto a la función ppal

##### Fin función calculadora #####

En primer lugar tenemos una estructura principal que muestra por pantalla una pregunta diciéndole al usuario que operación desea realizar. El usuario tendrá que introducir la operación en modo cadena de caracteres (Ej.: 3+2).

Una vez que el programa tiene dicha cadena de caracteres en memoria procede a llamar a la función calculadora.

Dentro de la función calculadora podemos observar un bucle para recorrer el vector que pasa la función principal como parámetro. Luego encontramos una estructura selectiva que se encarga de recorrer la cadena en busca del operador aritmético que esté empleando el usuario. Con ello tenemos 4 casos:

1. Suma: Si el código ASCII leído por la estructura selectiva es 43, la función salta a la etiqueta “suma” que se encarga de buscar los números en la cadena mediante sus códigos ASCII y luego transformar los números de caracteres ASCII a dígitos decimales para poder operar con ellos. Guarda los números en los registros $t3 y $t4 (con una condición para que antes se llene el registro $t3). Luego ejecuta la suma y devuelve el resultado a través del registro $v1.

2. Resta: Si el código ASCII leído por la estructura selectiva es 45, la función salta a la etiqueta “resta” que se encarga de buscar y guardar los números de la misma forma que lo hace la etiqueta “suma”. Luego ejecuta la resta y devuelve el resultado a través del registro $v1.

3. Producto: Si el código ASCII leído por la estructura selectiva es 42, la función salta a la etiqueta “producto” que buscará y guardará los números de la misma forma que las otras dos etiquetas. Ejecutará el producto y devolverá el resultado a través del registro $v1.

4. División: Si el código ASCII leído por la estructura selectiva es 47, la función salta a la etiqueta “división” que buscará y guardará los números de la misma forma que las otras tres etiquetas anteriores. Luego ejecutará la división y devolverá el resultado a través del registro $v1.

También he de destacar que, en caso de recibir una operación con paréntesis, el programa es capaz de admitirlos y procesarlos correctamente (solamente un nivel de paréntesis).