Departamento de Informática
Universidad de Valladolid

Escuela Técnica Superior de Ingeniería Informática
Camino del Cementerio s/n. 47011 Valladolid

13 de julio de 2009. 9h.

EXAMEN EXTRAORDINARIO DE ORGANIZACIÓN DE COMPUTADORES

NOTA: *Los alumnos con las prácticas pendientes deben sacar una nota mínima de 2 en el primer problema para superar la parte práctica de la asignatura.*

1 (3 p.) a) Escribir una función en lenguaje ensamblador de SPARC que admita 3 parámetros: los dos primeros corresponderán a las direcciones de dos cadenas de caracteres y el tercero será la salida de la función. La función debe comparar las dos cadenas, carácter a carácter, y devolverá el resultado en el último parámetro según el siguiente criterio:

- Si la primera cadena es mayor que la segunda, devolverá 1.
- Si la primera cadena es menor que la segunda, devolverá -1.
- Si ambas cadenas son iguales, devolverá 0.

Se supondrá que el terminador de cadena es el carácter nulo (cuyo código ASCII es 0) y que las cadenas no tienen más de 100 caracteres cada una.

b) Utilizando la función anterior, escribir un programa en lenguaje ensamblador de SPARC que pida dos cadenas por teclado y escriba por pantalla en primer lugar la cadena menor y después la otra, según el orden dado por la función.

Solución a ambos apartados:

```
LF=10
.data
Mensaje1: .asciz "Introduzca la primera cadena: "
Mensaje2: .asciz "Introduzca la segunda cadena: "
Cadena1: .skip 100
Cadena2: .skip 100


.text
.global main
main:
        save    %sp,-64,%sp
        set     Mensaje1, %o0
        clr     %o1
        call    imprime_cadena
        nop
        set     Cadena1, %o0
        mov     %o0, %l0
        call    Lee_cadena
        nop
        set     Mensaje2, %o0
        clr     %o1
        call    imprime_cadena
        nop
        set     Cadena2, %o0
        mov     %o0, %l1
        call    Lee_cadena
        nop
        mov     %l0, %o0
        mov     %l1, %o1
        call    comparar
        nop
        tst     %o2
        ble     seguir
        nop
        mov     %l0, %l1
        mov     %o1, %l0
```

```
seguir:mov     %l0, %o0
       clr     %o1
       call    imprime_cadena
       nop
       mov     LF, %o0
       call    putchar
       nop
       mov     %l1,%o0
       clr     %o1
       call    imprime_cadena
       nop
       mov     LF, %o0
       call    putchar
       nop
       ret
       restore


comparar:
! Parámetros:
! i0: Cadena1
! i1: Cadena2
! i2: Resultado: Cadena1 mayor: 1, Cadena2 mayor: -1, iguales: 0
        save    %sp, -64, %sp
        clr     %l0
        mov     100,%l1
bucle:  ldub    [%i0+%l0], %l3
        ldub    [%i1+%l0], %l4
        cmp     %l3, %l4
        bl      menor
        nop
        bg      mayor
        nop
        tst     %l3
        be      iguales
        nop
        inc     %l0
        deccc   %l1
        bg      bucle
        nop
iguales:clr     %i2
        ret
        restore
menor:  mov     -1, %i2
        ret
        restore
mayor:  mov     1, %i2
        ret
        restore
```

```
Lee_cadena:
! Parámetros:
! i0: Dirección donde se guardará la cadena leída
        save    %sp,-64,%sp
        clr     %l1
bucle1:
        call    getchar
        nop
        stb     %o0,[%i0+%l1]
        cmp     %o0,LF
        bne     bucle1
        inc     %l1
        dec     %l1
        stb     %g0, [%i0+%l1]
        ret
        restore


imprime_cadena:
! Parámetros:
! i0: direccion de la cadena
! i1: longitud máxima,  si i1 es 0 se supone que la longitud maxima es 255
        save    %sp, -64, %sp
        tst     %i1
        bne     continuar
        clr     %l1
        mov     255, %i1
continuar:
        ldub    [%i0 + %l1], %o0
        tst     %o0
        be      retornar
        nop
        call    putchar
        inc     %l1
        deccc   %i1
        bne     continuar
        nop
retornar:
        ret
        restore
```

2 (3 p.) En la memoria de un VAX se encuentra el programa mostrado en la figura 1.

a) Codificar el citado programa en lenguaje máquina, suponiendo que los símbolos X, N, Bucle y Error representan, respectivamente, a las direcciones 0A9B10F0H, 0A9B10F4H, 0A9B1175H y 0A9B1201H. Debe procurarse que la codificación del programa ocupe la menor cantidad de memoria posible.

Solución:

Dirección	Contenido	Comentarios
0A9B116B	DE	MOVAL
0A9B116C	AF	d(PC)
0A9B116D	82	d_X
0A9B116E	53	R3
0A9B116F	D0	MOVL
0A9B1170	AF	d(PC)
0A9B1171	82	d_N
0A9B1172	54	R4
0A9B1173	D4	CLRL
0A9B1174	55	R5
0A9B1175	D5	TSTL
0A9B1176	63	(R3)
0A9B1177	19	BLSS
0A9B1178	06	d_{Salida}
0A9B1179	C0	ADDL2
0A9B117A	83	(R3)+
0A9B117B	55	R5
0A9B117C	F5	SOBGTR
0A9B117D	54	R4
0A9B117E	F6	d_{Bucle}
0A9B117F	C3	SUBL3
0A9B1180	54	R4
0A9B1181	CF	d(PC)
0A9B1182	70	d_N (2 bytes)
0A9B1183	FF	
0A9B1184	54	R4
0A9B1185	13	BEQL
0A9B1186	7A	d_{Error}
0A9B1187	C6	DIVL2
0A9B1188	54	R4
0A9B1189	55	R5

⊗

```
         MOVAL X, R3
         MOVL N, R4
         CLRL R5
Bucle:   TSTL (R3)
         BLSS Salida
         ADDL2 (R3)+, R5
         SOBGTR R4, Bucle
Salida:  SUBL3 R4, N, R4
         BEQL Error
         DIVL2 R4,R5
           ...
Error:
```

Figura 1.

b) Explicar qué cambios provocará ese programa en la memoria y en los registros.

Solución:

El programa carga en el registro R3 la dirección X, en R4 el número (doble palabra) contenido en la dirección N y pone a cero el registro R5. Ya dentro del bucle, se van sumando en R5 las componentes del vector X (dobles palabras), apuntado por R3, hasta que se encuentre una componente negativa. En cada pasada del bucle el registro R3 se incrementa en 4. Después se calcula en R4 el número de componentes y se divide la suma hallada en R5 por ese número de componentes (media aritmética).

⊗

c) Explicar cuál puede ser el propósito del programa.

Solución:

El programa calcula la media aritmética de las componentes del vector de dobles palabras que comienza en la dirección X hasta que encuentre una componente negativa que se interpreta como terminador. Si no se encuentra el terminador, calcula la media del número de componentes que se halla en la dirección N.

⊗

d) Calcular el tiempo de ejecución del programa si el tiempo de acceso a cada byte de memoria es de 20 ns., suponiendo que la condición de la instrucción BLSS no se cumple nunca.

Solución:

En la siguiente tabla se puede ver el número de bytes accedidos por cada una de las instrucciones (para el acceso al código es necesario mirar la codificación del programa en lenguaje máquina expuesta anteriormente):

Instrucción		*Bytes accedidos*		
		Código	*Datos*	*Total*
	`MOVAL X, R3`	*4*		*4*
	`MOVL N, R4`	*4*	*4*	*8*
	`CLRL R5`	*2*		*2*
`Bucle:`	`TSTL (R3)`	*2*	*4*	*6*
	`BLSS Salida`	*2*		*2*
	`ADDL2 (R3)+, R5`	*3*	*4*	*7*
	`SOBGTR R4, Bucle`	*3*		*3*
`Salida:`	`SUBL3 R4, N, R4`	*6*	*4*	*10*
	`BEQL Error`	*2*		*2*
	`DIVL2 R4,R5`	*3*		*3*

Teniendo en cuenta que varias de las instrucciones están dentro de un bucle, el número de bytes accedidos en total será:

*N^o de accesos fuera del bucle+ n^o de accesos dentro del bucle * número de pasadas.*

Llamando n al número contenido en la dirección N (ya que se supone que la bifurcación BLSS no se cumple nunca), tendremos que el número total de bytes accedidos será $29 + 18n$, por lo que el tiempo total de acceso vendrá dado por $(29 + 18n) * 20ns$.

⊗

3 (1 p.) Razonar por qué el código de Huffman es más eficiente para codificar los códigos de operación que otros códigos cuando puede generar códigos de operación con un número de bits mucho mayor que $\log_2 n$, siendo n el número de instrucciones del procesador.

Solución:

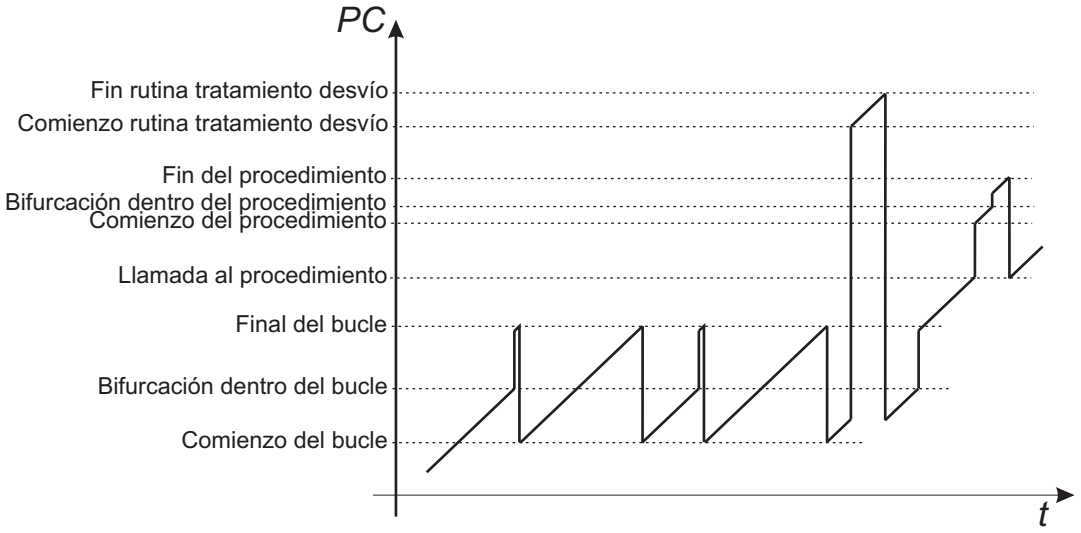
El código de Huffman asigna códigos de operación más cortos a las instrucciones más probables, normalmente mucho más cortos que $\log_2 n$; sin embargo, asigna códigos más largos a las menos probables, que pudieran ser mucho mayores que $\log_2 n$. La eficiencia del código es inversamente proporcional a la longitud **media** *de la palabra de código y en esta media pesarán mucho más las instrucciones más probables, con lo que la citada eficiencia será muy alta.*

⊗

4 (2 p.) Dibujar razonadamente la gráfica de variación del contador de programa respecto al tiempo para un programa con las siguientes incidencias, en el mismo orden en que están enumeradas. Se supone que entre dichas incidencias existe código pero éste no altera el flujo de las instrucciones:

1. Un bucle se repite 5 veces con un índice decreciente. Aproximadamente en la mitad del bucle existe una instrucción de bifurcación condicional cuya condición sólo se cumple cuando el índice es impar. El destino de esta bifurcación es la última instrucción del bucle. En la última pasada del bucle ocurre un desvío recuperable.
2. Llamada a un procedimiento. En el interior de ese procedimiento, se produce una bifurcación incondicional hacia adelante. Después de ella existe una bifurcación condicional hacia atrás cuya condición no se verifica.

Solución:



⊗

5 (1 p.) Explicar dónde quedarían enmarcadas las variables estáticas dentro de los criterios de clasificación de las variables de una función.

Solución:

Las variables estáticas, en cuanto a su ámbito de validez, son locales a una o varias funciones, sin embargo en cuanto a su vida son variables permanentes ya que su valor no se pierde cuando retorna la función. Es por ello que su espacio de memoria debe reservarse en el área global

⊗