Departamento de Informática
Escuela Técnica Superior de Ingeniería Informática
Paseo de Belén, 15. 47011 Valladolid

1 de febrero de 2011. 16h.

EXAMEN ORDINARIO 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 un procedimiento en lenguaje ensamblador de SPARC que tome como parámetros tres direcciones: el procedimiento debe construir, partiendo de las cadenas existentes en las dos primeras direcciones, una nueva cadena que comience en la tercera dirección. Esta última cadena se formará tomando alternativamente un carácter de cada una de las dos cadenas anteriores. Las cadenas se considerarán terminadas cuando se encuentre un carácter nulo. Si una de las cadenas es más corta que la otra en lugar de los caracteres que falten se pondrán espacios en blanco.

b) Utilizar el procedimiento anterior para escribir un programa que pida dos cadenas por teclado y escriba por pantalla la cadena resultante de tomar alternativamente una carácter de cada cadena.

```
        MOV N, R0
        MOV #A, R1
        CLR R2
        MOV #1, R3
Bucle:  MOV R3, (R1)+
        MOV R3, R4
        ADD R2, R3
        MOV R4, R2
        SOB R0, Bucle
        ...
```

Figura 1.

Solución a ambos apartados:

```
LF=10
ESPACIO=32
.data
Mensaje:   .asciz "Introduzca una cadena: "
Mensajer:  .asciz "Resultado de la mezcla: "
Cadena1:   .skip 128
Cadena2:   .skip 128
Resultado: .skip 256
           .global main
.text
main:
           save   %sp, -64, %sp
           set    Mensaje, %o0
           clr    %o1
           call   imprime_cadena
           nop
           set    Cadena1, %o0
           call   Lee_cadena
           nop
           set    Mensaje, %o0
           call   imprime_cadena
           nop
           set    Cadena2, %o0
           call   Lee_cadena
           nop
           set    Cadena1, %o0
           set    Cadena2, %o1
           set    Resultado, %o2
           call   mezclar_cadenas
           nop
           set    Mensajer, %o0
           call   imprime_cadena
           nop
           mov    %o2, %o0
```

```
            clr    %o1
            call   imprime_cadena
            nop
            ret
            restore

mezclar_cadenas:
! Parámetros:
!           i0: Cadena1
!           i1: Cadena2
!           i2: Cadena resultado
            save   %sp, -64, %sp
            clr    %l0
            clr    %l1
            clr    %l2
            clr    %l3
bucle:      ldub   [%i0+%l0], %l4
            tst    %l4
            bne    guarda1
            inc    %l0
            mov    ESPACIO,%l4
            mov    1, %l3
            dec    %l0
guarda1:    stb    %l4,[%i2+%l2]
            inc    %l2
            ldub   [%i1+%l1], %l4
            tst    %l4
            bne    guarda2
            inc    %l1
            cmp    %l3,1
            be     fin
            mov    ESPACIO, %l4
            dec    %l1
guarda2:    stb    %l4, [%i2+%l2]
            inc    %l2
            ba     bucle
fin:        stb    %g0, [%i2+%l2]
            ret
            restore

Lee_cadena:
! Parámetro: 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 (2.5 p.) Sea el fragmento de programa de PDP-11 mostrado en la figura 1:

a) Escribir ese programa en código máquina suponiendo que los símbolos `A`, `N` y `Bucle` representan, respectivamente, a las direcciones $002136_{(8}$, $002252_{(8}$ y $002346_{(8}$

Solución:

Dirección	Contenido
002330	016700
002332	177716
002334	012701
002336	002136
002340	005002
002342	012703
002344	000001
002346	010321
002350	010304
002352	060203
002354	010402
002356	077005

⊗

b) Determinar la evolución de los registros y posiciones de memoria modificados por el programa si la palabra situada en la dirección representada por el símbolo `N` contiene 6.

Solución:

	Registros						Direcciones de memoria					
Instrucción	PC	R0	R1	R2	R3	R4	002136	002140	002142	002144	002146	002150
MOV N, R0	002332	000006										
MOV #A, R1	002340		002136									
CLR R2	002342			000000								
MOV #1, R3	002346				000001							
MOV R3, (R1)+	002350		002140				000001					
MOV R3, R4	002352					000001						
ADD R2, R3	002354				000001							
MOV R4, R2	002356			000001								
SOB R0, Bucle	002346	000005										
MOV R3, (R1)+	002350		002142					000001				
MOV R3, R4	002352					000001						
ADD R2, R3	002354				000002							
MOV R4, R2	002356			000001								
SOB R0, Bucle	002346	000004										
MOV R3, (R1)+	002350		002144						000002			
MOV R3, R4	002352					000002						
ADD R2, R3	002354				000003							
MOV R4, R2	002356			000002								
SOB R0, Bucle	002346	000003										
MOV R3, (R1)+	002350		002146							000003		
MOV R3, R4	002352					000003						
ADD R2, R3	002354				000005							
MOV R4, R2	002356			000003								
SOB R0, Bucle	002346	000002										
MOV R3, (R1)+	002350		002150								000005	
MOV R3, R4	002352					000005						
ADD R2, R3	002354				000010							
MOV R4, R2	002356			000005								
SOB R0, Bucle	002346	000001										
MOV R3, (R1)+	002350		002152									000010
MOV R3, R4	002352					000010						
ADD R2, R3	002354				000015							
MOV R4, R2	002356			000010								
SOB R0, Bucle	002360	000000										

⊗

c) Escribir en un lenguaje de alto nivel un código con los mismos efectos que el mostrado.

Solución:

En lenguaje C, por ejemplo:

```
int a[6],i,j,k,l;
j=0;k=1;
for(i=0;i<6;i++)
 {a[i]=k;
  l=k;k=j+k;j=l;}
```

⊗

d) ¿Cuál puede ser el propósito de ese fragmento de código?

Solución:

El programa va almacenando a partir de la dirección A los términos de la sucesión de Fibonacci, o lo que es lo mismo, la sucesión en que cada término es la suma de los dos anteriores.

⊗

3 (1.5 p.) Un procesador con un reloj de 2 GHz tiene un CPI de 1,6: ¿Cuál será su velocidad en MIPS?

Solución:

Sabemos que el tiempo de ejecución de un programa viene dado por

$$t = N * CPI * \frac{1}{f}$$

Por definición, la velocidad en MIPS es el número de millones de instrucciones ejecutadas en un segundo; por tanto, la velocidad en MIPS será $N * 10^{-6}$ cuando t sea un segundo, es decir:

$$1 = MIPS * 10^6 * CPI * \frac{1}{f} \Longrightarrow MIPS = \frac{10^{-6} f}{CPI}$$

En nuestro caso:

$$MIPS = \frac{10^{-6} f}{CPI} = \frac{10^{-6} * 2 * 10^9}{1,6} = \frac{2000}{1,6} = 1250$$

⊗

4 (1.5 p.) En una máquina de 16 bits que trabaja en complemento a 2, los registros r_1 y r_2 contienen respectivamente 82B2H y 75B3H. En ella se ejecutan las instrucciones:

1. Sumar $r_1 + r_2$
2. Restar $r_1 - r_2$
3. Desplazar r_2 un lugar a la izquierda

a) Explicar **breve y razonadamente** cuáles serán los valores de los bits N, Z, V y C después de ejecutar cada una de esas instrucciones suponiendo que todas ellas actúan sobre los citados *flags*.

Solución:

1. $r_1 + r_2 = 82B2 + 75B3 = F865 \Longrightarrow Z = 0, N = 1, C = 1$ *y* $V = 0$ *porque los sumandos tienen signo diferente.*
2. $r_1 - r_2 = 82B2 - 75B3 = 82B2 + 8A4D = 10CFF \Longrightarrow Z = 0, N = 0, C = 1$ *y* $V = 1$ *porque los dos sumandos de la suma final son negativos y el resultado es positivo.*
3. $75B3 << 1 = EB66 \Longrightarrow Z = 0, N = 1, C = 0$ *y* $V = 1$ *porque el signo ha cambiado.*

⊗

b) Supóngase que después de la instrucción 2 (comparación) se ejecuta la instrucción `BLEQ` en un VAX o `BLE` de un PDP-11 (bifurcar si menor o igual en ambos procesadores) ¿Se comportarán ambas máquinas de igual modo en cuanto al cumplimiento de la condición de bifurcación? Si el comportamiento de las dos máquinas fuera distinto: ¿Cuál de los dos sería más correcto y por qué?

Solución:

Si miramos las tablas de instrucciones de ambos procesadores, vemos que en el VAX `BLEQ` *bifurca cuando* $Z \vee N == 1$*, mientras en el PDP-11 la instrucción* `BLE` *bifurca cuando* $Z \vee (N \oplus V) == 1$*, esto significa que el PDP-11 tiene en cuenta la situación en que la resta active el desbordamiento, en cuyo caso la condición de la bifurcación también se cumplirá correctamente. Por ello el comportamiento del VAX no será del todo correcto porque en caso de que haya desbordamiento en la resta, la condición no se evaluará satisfactoriamente.*

⊗

5 (1.5 p.) Escribir las mínimas instrucciones en lenguaje ensamblador de SPARC para intercambiar los registros $\%l3$ y $\%l4$ sin emplear ningún otro almacenamiento intermedio (ni registro ni memoria).

Solución:

```
!Supondremos que inicialmente %l3 contiene A y %l4 contiene B, recordemos que (x ⊕ y) ⊕ x = y
xor %l3, %l4, %l4 ! Ahora %l3 sigue conteniendo A y %l4 pasa a tener A ⊕ B
xor %l3, %l4, %l3 ! Después de esta instrucción, %l4 tiene A ⊕ B y %l3 pasa a tener B
xor %l3, %l4, %l4 ! Finalmente %l3 contiene B y %l4 contiene A
```

⊗