Departamento de Informática
Universidad de Valladolid

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

6 de febrero de 2009 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 (3p.) Sea la siguiente estructura de datos descrita en lenguaje Pascal:

```
var a: array [0..99] of
record
   x: integer;
   y: array [1..20] of char
end
```

Dirección	Contenido
016750	010140
016752	010701
016754	012701
016756	100144
016760	000000
016762	021527
016764	000000
016766	177777
016770	030200
016772	016705
016774	177764
016776	012700
017000	016750
017002	012701
017004	000004
017006	010502
017010	042702
017012	177760
017014	010220
017016	072527
017020	177774
017022	077107

Figura 1.

a) Escribir los directivos en lenguaje ensamblador de SPARC que reserven el espacio de memoria necesario para almacenar la citada estructura de datos.

Solución:

*Cada componente de la matriz tiene 1 entero y 20 caracteres, es decir, 24 bytes. Dado que son 100 componentes, el número de bytes necesarios para guardar todo el vector será 24*100 = 2400. Por tanto, los directivos necesarios para realizar la reserva serán:*

```
        .align    4
a:      .skip     2400
```

⊗

b) Escribir una función en el mismo lenguaje que tenga los siguientes parámetros:

- La dirección de la estructura.
- Dos índices con rango de 0 a 99, llamémosles i_1 e i_2

La función debe intercambiar las componentes i_1 e i_2 del vector `a`.

Solución:

```
intercambio:
              save      %sp, -64,%sp     ! i0: Dirección del vector
              mulx      %i1, 24,%l1      ! i1, i2: Índices de las componentes a intercambiar
              mulx      %i2, 24,%l2      ! l1, l2: Posición del comienzo de las componentes i1 e i2
              mov       6,%l0            ! l0: Índice del bucle: 6 palabras = 24 bytes
bucle:        ld        [%i0+%l1],%l3    ! l3, l4: Variables temporales de intercambio
              ld        [%i0+%l2],%l4
              st        %l3, [%i0+%l2]
              st        %l4, [%i0+%l1]
              inc       4,%l1
              deccc     %l0
              bgt       bucle
              inc       4,%l2
              ret
              restore
```

⊗

c) Utilizando la función escrita en el apartado anterior, escribir un fragmento de programa, en lenguaje ensamblador de SPARC, que reordene el vector a de forma que la componente con mayor valor del campo x quede la última, sin alterar el orden de las demás.

Solución:

```
maximoalfinal:
                set     a,%o0           ! Dirección del vector
                call    maximo          ! Calcula el índice con mayor x
                nop
saltar:         add     %o1, 1,%o2      ! A partir del máximo
                call    intercambio     ! intercambiamos cada registro con
                nop                     ! el siguiente
                cmp     %o1, 98
                blt     saltar
                inc     %o1             ! Ciclo adelantado
                ...

maximo:
                save    %sp,-64,%sp     ! Dirección del vector en i0
                clr     %i1             ! Devuelve el índice del máximo x en i1
                ld      [%i0],%l4       ! l4 = valor del máximo x
                clr     %l1             ! Índice del vector
otro:           inc     %l1
                mulx    %l1, 24,%l2
                ld      [%i0+%l2],%l3
                cmp     %l3,%l4
                ble     compara
                nop
                mov     %l3,%l4         ! Actualiza máximo e índice si es mayor
                mov     %l1,%i1
compara:        cmp     %l1, 99
                blt     otro
                nop
                ret
                restore
```

⊗

2 (2.5 p.) La memoria de un PDP-11 se encuentra en el estado mostrado en la figura 1 (todo en octal):

a) Decodificar el código existente a partir de la dirección $016772_{(8}$.

Solución:

```
        MOV A, R5         !A = PC + d = 016776(8 + 177764(8 = 016762(8
        MOV #16750, R0
        MOV #4, R1
Bucle:  MOV R5, R2
        BIC #177760, R2
        MOV R2, (R0)+
        ASH #-4., R5
        SOB R1, Bucle
```

⊗

b) Explicar los efectos del programa sobre los registros y la memoria de la máquina

Solución:

	Registros					Direcciones de memoria			
Instrucción	*PC*	*R5*	*R0*	*R1*	*R2*	*016750*	*016752*	*016754*	*16756*
MOV A, R5	*016776*	*021527*							
MOV #16750, R0	*017002*		*016750*						
MOV #4, R1	*017006*			*000004*					
MOV R5, R2	*017010*				*021527*				
BIC #177760, R2	*017014*				*000007*				
MOV R2, (R0)+	*017016*		*016752*			*000007*			
ASH #-4., R5	*017022*	*001065*							
SOB R1, Bucle	*017006*			*000003*					
MOV R5, R2	*017010*				*001065*				
BIC #177760, R2	*017014*				*000005*				
MOV R2, (R0)+	*017016*		*016754*				*000005*		
ASH #-4., R5	*017022*	*000043*							
SOB R1, Bucle	*017006*			*000002*					
MOV R5, R2	*017010*				*000043*				
BIC #177760, R2	*017014*				*000003*				
MOV R2, (R0)+	*017016*		*016756*					*000003*	
ASH #-4., R5	*017022*	*000002*							
SOB R1, Bucle	*017006*			*000001*					
MOV R5, R2	*017010*				*000002*				
BIC #177760, R2	*017014*				*000002*				
MOV R2, (R0)+	*017016*		*016760*						*000002*
ASH #-4., R5	*017022*	*000000*							
SOB R1, Bucle	*017024*			*000000*					

⊗

c) Explicar brevemente el propósito del código.

Solución:

Extrae las cuatro cifras que se encuentran empaquetadas en cada 4 bits de la palabra situada en la dirección A (posiblemente codificadas en BCD) y las deposita en memoria sucesivamente a partir de la dirección $016750_{(8}$

⊗

d) Si cada acceso a una palabra de memoria emplea 50 nsg., calcular razonadamente el tiempo total de ejecución del programa sin tener en cuenta otros factores diferentes a los accesos a memoria.

Solución:

	Instrucción	*Accesos código*	*Accesos datos*	*Accesos totales*
	MOV A, R5	*2*	*1*	*3*
	MOV #16750, R0	*2*	*0*	*2*
	MOV #4, R1	*2*	*0*	*2*
Bucle:	*MOV R5, R2*	*1*	*0*	*1*
	BIC #177760, R2	*2*	*0*	*2*
	MOV R2, (R0)+	*1*	*1*	*2*
	ASH #-4., R5	*2*	*0*	*2*
	SOB R1, Bucle	*1*	*0*	*1*

En resumen: accesos fuera del bucle: 7; accesos dentro del bucle: 8.

Por tanto, el número total de accesos necesario para la ejecución del programa será: $7 + 4 * 8 = 39$

*Dado que cada acceso necesita 50 nsg., el tiempo total necesario para la ejecución de ese fragmento de programa será: 39*50 nsg. = 1950 nsg. = 1.95 μsg.*

⊗

3 (1.5 p.) Todas las instrucciones siguientes tienen el mismo efecto: borrar el registro $R2$ de un VAX. Sin embargo, requieren diferente número de bytes de memoria al codificarlas en código máquina. ¿Cuántos bytes requiere cada una? ¿Cuál es más eficiente? Razónese la contestación.

```
SUBL2 R2, R2     CLRL R2     MOVL #0, R2     XORL2 R2, R2
```

Solución:

Los bytes requeridos por cada instrucción son:

Instrucción	*Bytes requeridos*
`SUBL2 R2, R2`	*3*
`CLRL R2`	*2*
`MOVL #0, R2`	*3*
`XORL2 R2, R2`	*3*

Por tanto, la instrucción más eficiente es `CLR R2` que es la que menos accesos a memoria necesita, por lo que será más rápida.

⊗

De manera análoga, todas las instrucciones siguientes tienen el mismo efecto: borrar el registro $l2$ de un procesador con arquitectura SPARC. Explicar razonadamente cuál es más eficiente.

```
mov 0,%l2   xor%l2,%l2,%l2   or%g0,%g0,%l2   clr%l2   sub%l2,%l2,%l2
```

Solución:

En este caso todas serán equivalentes en cuanto a tiempo de ejecución, ya que todas las instrucciones de SPARC ocupan 32 bits.

⊗

4 (1.5 p.) Un computador tiene un procesador de 2.5 GHz. y ejecuta 1.500 MIPS: ¿Cuál será el número medio de ciclos por instrucción (CPI) de esta máquina?

Solución:

El CPI es, por definición, el número medio de ciclos empleado para ejecutar una instrucción. Si tomamos un cierto intervalo de tiempo, dividiendo el número de ciclos empleado entre el número de instrucciones empleadas obtendremos el CPI. Tomemos como intervalo de tiempo 1 segundo, en este caso, tendremos:

$$CPI = \frac{Ciclos\ \ empleados}{Instrucciones\ \ ejecutadas} = \frac{f}{10^6 MIPS} = \frac{2500 * 10^6}{1500 * 10^6} = \frac{25}{15} = 1{,}66$$

⊗

5 (1.5 p.) Explicar, clara y brevemente, la función del banco alternativo de registros de un Z-80.

Solución:

El banco alternativo del Z-80 se usa durante la ejecución de las rutinas de servicio de interrupción. Disponer de este banco evita el tener que salvar los registros en la pila cada vez que se produce una interrupción; bastará conmutar al banco de registros alternativo cuando comienza la rutina de servicio de interrupción y volver a conmutar al banco principal cuando se produzca el retorno.

⊗