FUNDAMENTOS DE COMPUTADORES II
20 de mayo de 2022. Examen ordinario

1)(2puntos) El siguiente código en ensamblador suma los elementos de cada columna de una matriz A de NxN elementos de tipo entero y almacena los resultados en un vector V de N elementos, siendo:

$$V[j] = \sum_{i=0}^{N-1} A[i][j]$$

En el programa dado se hacen llamadas a la subrutina SumaColumna, que calcula y almacena en la posición correspondiente del vector el resultado de la suma de una columna de la matriz (la especificada en el último parámetro de la llamada) :void SumaColumna (int matriz[N][N], int vector[N], int N, int columna)

```
        .global start
        .Extern _stack
        .equ N, 4

        .data
A:  .word 0, 1, 2, 3,-3,-2,-1, 0, 0, 1, 2,3,-3,-2,-1,0

        .bss
V:  .space 4 * N

        .text
start:  Ldr sp,=_stack
        mov fp, #0

        ldr r0, =A
        ldr r1, =V
        mov r2, #0

Bucle:  cmp r2, #N
        bge Fin
        [                    ]
        [                    ]
        [                    ]

        bl SumaColumna

        [                    ]
        [                    ]
        [                    ]

        add r2, r2, #1
        b Bucle
Fin:    b .
```

a) Completa el código anterior, en los lugares señalados, para que las llamadas a SumaColumna se realicen adecuadamente. Debe respetarse el estándar de llamadas a subrutinas visto en clase.
b) Indica qué valores tiene el Vector V después de ejecutar el programa.
c) Codifica la subrutina SumaColumna utilizada en el código anterior, respetando el estándar estudiado.

2) (3,5 puntos) Para acelerar la ejecución de una serie de programas que utilizan continuamente multiplicaciones simples, se ha decidido incluir la instrucción MULS Rd, Rs, Rt (Multiplicación Simplificada) en la nueva versión del repertorio de instrucciones para la próxima revisión del microprocesador MIPS. Para ello, los ingenieros hardware han desarrollo un nuevo módulo llamado AluMULS, con las siguientes características:

- El módulo AluMULS sólo es capaz de multiplicar.
- El nuevo módulo tiene 2 entradas (Rs y Rt) de números de 32 bits y una salida de 32 bits, que contiene el resultado de multiplicar los 16 bits menos significativos de los registros fuente.
- A diferencia de la ALU, en la que el resultado de la operación tarda 1 ciclo en calcularse, en el módulo AluMULS el resultado de la multiplicación tarda 2 ciclos en obtenerse.
- En el nuevo módulo, las señales de entrada tienen que mantenerse estables durante los dos ciclos que dura la operación para que el resultado sea correcto.

a) Diseña las modificaciones necesarias dentro de la ruta básica del microprocesador MIPS.
b) Modifica el diagrama de estados del controlador para que sea capaz de ejecutar esta nueva instrucción, teniendo en cuenta que el OPCODE de MULS es 111000
c) Indica los cambios específicos que necesita la tabla de verdad del controlador.

d) En caso de que queramos ejecutar la instrucción MULS R3, R2, R1 (tipo R), muestra los valores de todos los registros modificados al final de cada fase de la ejecución, asumiendo que el estado del procesador antes de ejecutar esta instrucción es: PC=0xC00000E4, R1=0x00000002, R2=0x0000000A, R3=0x00000004; y que los campos SHAMT y FUNCT son 0.

3)(3,5 puntos) El siguiente código calcula el valor absoluto de la diferencia entre dos vectores:

$$W(i) = |U(i) - V(i)|$$

```
.global start
.equ N, 16
.data
U: .word  16, 14, 12,10, 8, 6, 4, 2,0,-2,-4,-6,-8,-10,-12,-14
V: .word -15,-13,-11,-9,-7,-5,-3,-1,1, 3, 5, 7, 9, 11, 13, 15
.bss
W: .space 4*N
.text
abssub:
    ldr r0, =U
    ldr r1, =V
    ldr r2, =W
    mov r3, #N
    mov r4, #0
abssub_for0:
    cmp r4, r3
    bge abssub_for0_end
    ldr r5, [r0, r4, lsl#2]
    ldr r6, [r1, r4, lsl#2]
    sub r5, r5, r6
    cmp r5, #0
    bge abssub_for0_if0
    neg r5, r5  @(r5 <= -r5)
abssub_for0_if0:
    str r5, [r2, r4, lsl#2]
    add r4, r4, #1
    b abssub_for0
abssub_for0_end:
    b .
```

En el script de enlazado, declaramos que la sección .data comienza en la dirección 0x0040, y después se colocan inmediatamente las secciones .bss y .text. La memoria principal es de 128Kbytes, y la cache de 256 bytes. El tamaño de bloque es de 32 bytes.

a) Indica el formato de dirección para MP y para cache.
b) Indica los rangos de direcciones que ocupan en memoria los vectores U, V y W. ¿Cuántos bloques ocupan?
c) Indica el rango de direcciones que ocupa en memoria el código. ¿Cuántos bloques ocupa?
d) Indica el contenido de la memoria cache tras una primera iteración del bucle, mostrando el valor de la etiqueta de los marcos, y explicando qué información contiene cada uno.
e) Calcula el número total de fallos que se producen al ejecutar el programa completo, contando que la instrucción b . se llega a ejecutar una única vez.
f) Si suponemos que el tiempo de acceso a memoria principal (TMP) es de 20ns, el tiempo de acceso a cache (TCache) es de 1ns, y la penalización por fallo en caché son 500ns, ¿es más ventajoso usar la jerarquía con caché, o únicamente una memoria principal?
g) Si aumentamos el valor de N a 32, ¿qué partes de los datos y el programa compiten por bloques en la cache?.

4) (1 punto) La ejecución de cierto código tarda 500ns en un procesador MIPS multiciclo, como el estudiado en clase, cuya frecuencia de funcionamiento es 2 GHz. Este código está formado por un total de 250 instrucciones, de las cuales el 10% son loads (CPI=5), el 40% son instrucciones aritméticas (CPI=4), y el resto de las instrucciones son stores (CPI=4) e instrucciones de salto (CPI=4 si el salto se toma y CPI=3 si el salto no se toma). De los saltos de los que consta el programa, el 75% de ellos se toman. Determina:

a) El número de ciclos promedio por instrucción.
b) La fracción de instrucciones de store sobre el total de las instrucciones del programa.
c) El número de instrucciones de salto que no se han tomado.