UNIVERSIDAD CARLOS III DE MADRID. DEPARTAMENTO DE INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA. DOBLE GRADO ADE E INGENIERÍA INFORMÁTICA
ESTRUCTURA DE COMPUTADORES
10 de enero de 2020 **Examen final**

Para la realización del presente examen se dispondrá de **2:30 horas.**
NO se podrán utilizar libros, apuntes **ni** calculadoras (o dispositivos electrónicos) de ningún tipo.

Ejercicio 1 (1puntos).

En relación al estándar IEEE 754 de simple precisión:

a) Codifique en este estándar el valor decimal -17,25. Exprese el resultado final en hexadecimal.
b) Indique la diferencia (exprese esta diferencia como un valor decimal) que existe entre el valor normalizado más pequeño positivo que se puede representar en este estándar y el valor no normalizado positivo más grande que se puede representar en este este estándar.

Ejercicio 2 (3 puntos).

Considere la rutina `Sumar`. Esta rutina acepta **cinco** parámetros de entrada (se pasan en el mismo orden en el que se describen a continuación):

- La dirección de inicio de una matriz (se almacena por filas) de números de tipo `int`, de dimensión `MxN.`
- El número de filas de la matriz (`M`).
- El número de columnas de la matriz (`N`).
- Un valor entero (`fila`).
- Un número de fila o columna `i`.

Si el valor de `fila` es 0 la función suma los valores de todos los elementos situados en la columna `i`. En caso de que `fila` sea distinto de 0, la función suma los valores de todos los elementos situados en la fila `i`. La función devuelve dos valores:

- Si el argumento `i` se encuentra fuera de rango dependiendo del valor que tenga el argumento `fila` la función devolverá como primer resultado -1 (indicando el error) y 0 como segundo resultado.
- Si el argumento `i` es correcto (se encuentra dentro del rango de la fila o columna, dependiendo del valor que tenga el argumento `fila`) devuelve 0 como primer resultado y el valor de la suma de la fila o columna `i` como segundo resultado: si `fila` es igual a 0, devuelve la suma de la columna `i`; si `fila` es distinto de 0 devuelve la suma de la fila `i` como segundo resultado.

Se pide:

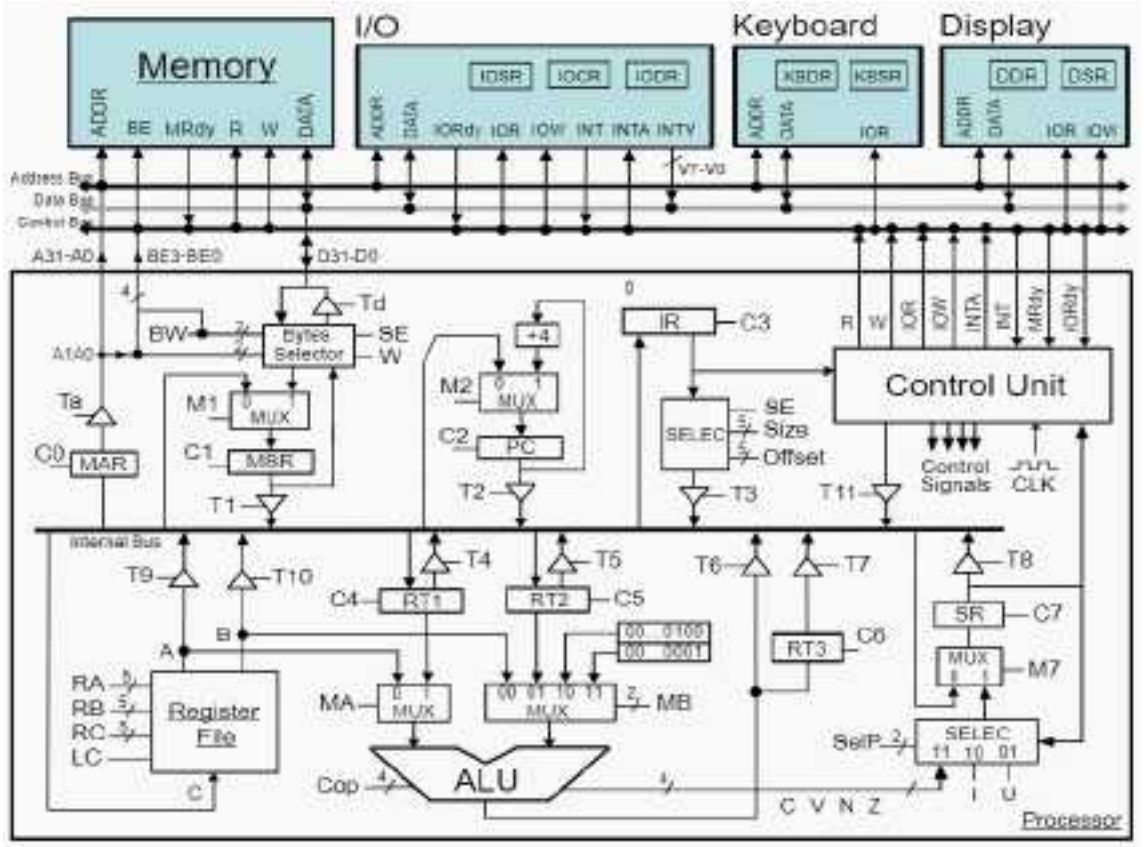
a) (2,5 puntos) Codifique correctamente la rutina `Sumar` anteriormente descrita. Ha de seguirse estrictamente el convenio de paso de parámetros del MIPS visto en la asignatura.

b) (0,5 puntos) Dada el siguiente segmento de datos:

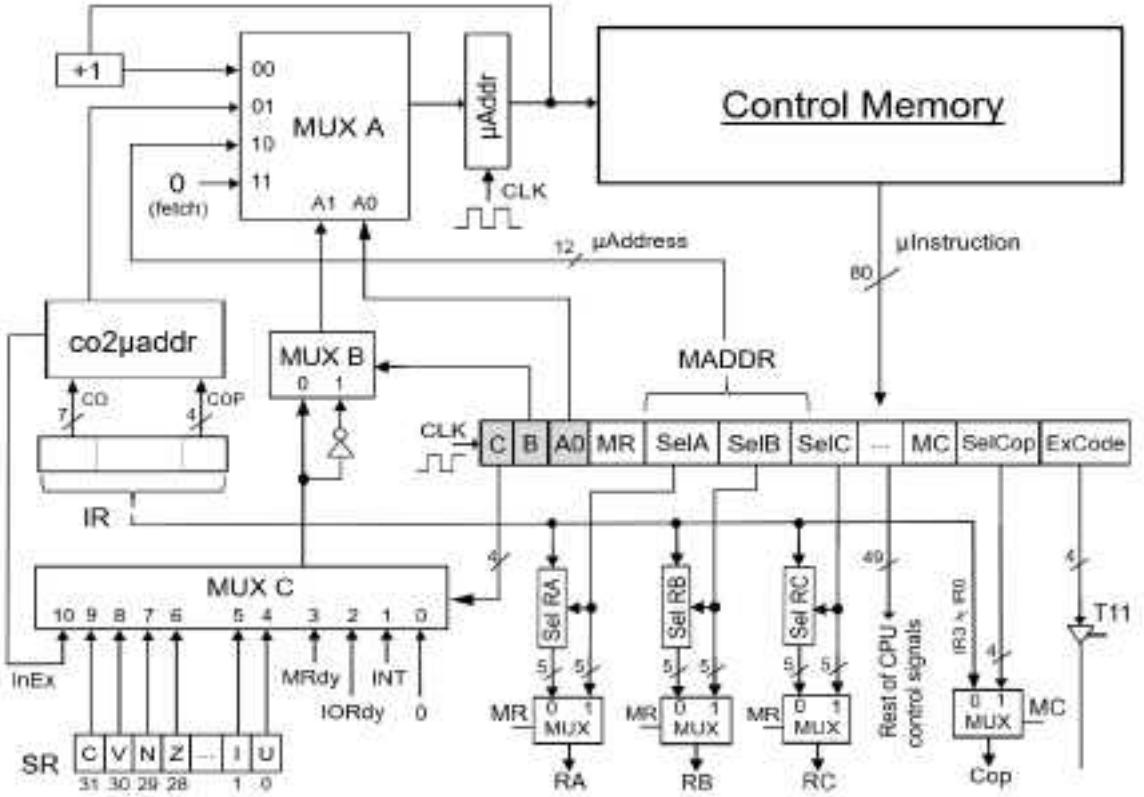
```
.data
  A:  .word       8, 4, 3, 3, 5,
                  2, 3, 0, 4, 5,
                  0, 0 ,1, 2, 3
  M: .word 3
  N: .word 5
```

Codifique el fragmento de código que permite invocar correctamente a la función `Sumar` para la matriz `A` de dimensión `MxN` y los valores de `fila=1` e `i=2`. A continuación se deben imprimir los dos valores devueltos por la función.

Ejercicio 3 (3 puntos). Dado el procesador WepSIM con la siguiente estructura:



Este procesador dispone de una Unidad de control representada por la siguiente figura:



Se pide:

a) (0,5 puntos) Indique un formato válido, justificando su respuesta, para la siguiente instrucción: `addm Rd, n, dirección`. Esta instrucción suma el valor entero `n` al contenido de la posición de memoria `dirección` y deja el resultado en el registro especificado `Rd`. La instrucción debe poder especificar cualquier posición de memoria en el procesador y representar cualquier valor entero.
b) (1,25 puntos) Especifique las operaciones elementales (incluido el *fetch*) necesarias para ejecutar la instrucción `lw Rd, direccion`. Esta instrucción almacena en el registro `Rd` el valor almacenado en la posición de memoria `dirección`. La instrucción ocupa dos palabras. En la primera palabra se codifica el código de operación y el registro, y en la segunda el campo `dirección`.
c) (1,25 puntos) Especifique todas las señales de control necesarias para ejecutar las siguientes cuatro operaciones elementales. Indique la solución en una hoja aparte, utilizando una estructura similar a la que se indica a continuación:

Reg ← RT2 (el campo Reg se encuentra en la instrucción entre los bits 25 y 21). Después se sigue con la siguiente microinstrucción.	
RT1 ← regOr1 + RT2 (el campo regOr1 se encuentra en la instrucción entre los bits 12 y 8. Esta operación actualiza el registro de estado. Después se sigue con la siguiente microinstrucción.	
If (SR(Z) ≠ 0) salta a la microdirección que se encuentra en la etiqueta utilizada en WepSim `salto`. En caso contrario sigue con la siguiente microinstrucción.	
MBR ← MP[MAR] (lectura de una palabra). Después se sigue con la siguiente microinstrucción.	

Ejercicio 4 (1,5 puntos). Considere un computador de 32 bits con 64 registros que direcciona la memoria por bytes. Responda a las siguientes preguntas:

a) ¿Qué cantidad de memoria (expresada en MB) puede direccionar este computador?
b) ¿Cuántos bits se utilizan para identificar a los registros en este computador?
c) ¿Qué es y para qué sirve el registro contador de programa?
d) Indique las acciones que realiza la unidad de control en el ciclo de reconocimiento de una interrupción.

Ejercicio 5 (1,5 puntos). Considere un computador con un ancho de palabra de 32 bits, que incluye una memoria caché de datos de 128KB, asociativa por conjuntos de 8 vías y líneas de 64 bytes. Considere que en este computador se ejecuta el siguiente fragmento de programa:

```
double    A[1024];
float     B[1024];
int       C[1024];

for (i=0; i < 1024; i++)
     C[i] = (int)A[i] + (int)B[i];
```

a) (0,25 puntos) Indique el número de líneas y conjuntos de la memoria caché de datos.
b) (1 punto) Indique la tasa de aciertos que produce la ejecución del fragmento anterior en la caché de datos, asumiendo que la caché de datos está inicialmente vacía.
c) (0,25 puntos) Si se cambia la caché de datos por una totalmente asociativa, qué repercusiones tendría sobre la tasa de aciertos calculada y sobre el rendimiento global de la memoria caché. Justifique su respuesta.

Soluciones

Ejercicio 1.

a) $-17{,}25_{(10} = -10001{,}01_{(2} = -1{,}000101 \times 2^4$ (de forma normalizada). El bit de singo es 1 (el número es negativo). El valor que se al almacena en el exponente es 4 + 127 = 131 = 10000011. La mantisa que se almacena (no se almacena el bit implícito) es 0001010000…..0000.

Su representación es 1100 0001 1000 1010 0….000000 = 0xC18A0000

b) El número positivo normalizado más pequeño es aquel normalizado que tiene la mantisa más pequeña (todo ceros) y el exponente más pequeño (1). Su representación en el estándar IEEE 754 es la siguiente:

0 00000001 000000000000

El valor decimal de este número es $1{,}0 \times 2^{-126} = 2^{-126}$

Por otra parte, el número no normalizado más grande se corresponde con una representación donde el exponente es 0 y la mantisa es la más grande (todos 1). Su representación es:

0 00000000 11111…1111111

El valor decimal de este número es $(1-2^{-23}) \times 2^{-126} = 2^{-126} - 2^{-149}$

Entre estos dos números no se puede representar en este estándar ningún otro número, son números consecutivos.

La diferencia entre ellos es $2^{-126} – (2^{-126} - 2^{-149}) = 2^{-149}$

Ejercicio 2.

a) A esta función se le pasan 5 argumentos, los cuatro primeros se pasan en `$a0`, `$a1`, `$a2` y `$a3` y el quinto se pasa en la pila. La función devuelve dos valores, el primero se devuelve en `$v0` y el segundo en `$v1`.

```
Sumar:      lw    $t0, ($sp)        # se accede a la pila: argumento i

            beqz  $a3, columna      # fila = 0 -> sumar columna i

fila:       # en este caso fila <> 0 -> sumar fila i
            blt   $t0, $0, error    # i < 0
            bge   $t0, $a2, error   # i >= N
            # se calcula la dirección de inicio de la fila i
            muli  $t1, $a2, 4       #  N x 4
            muli  $t1, $t1, $t0     # (N x 4) x i
            add   $t1, $a0, $t1     #  $t1: dirección de inicio de fila i
            move  $t2, $a2          # número de elementos de la fila i a sumar
            move  $t3, 4            # distancia entre elementos de la fila
            b sumar

columna:    # en este caso fila = 0 -> sumar columna i
            blt   $t0, $0, error    # i < 0
            bge   $t0, $a1, error   # i>= M
            # se calcula la dirección de inicio de la columna i
```

```
            muli  $t1, $t0, 4       #  i x 4
            add   $t1, $a0, $t1     # $t1: dirección de inicio de columna i
            move  $t2, $a1          # número de elementos de la col. i a sumar
            muli  $t3, $a2, 4       # distancia entre elementos de la columna
                                    # 4 x N


sumar:      # se suman los elementos de la fila o columna
            # $t1: dirección de inicio del primer elemento
            # $t2: número de elementos a sumar
            # $t3: distancia entre un elemento y el siguiente. En el caso de
            #      sumar una fila la distancia es 4 (son elementos de tipo int).
            #      En caso de sumar una columna, la distancia es 4xN (hay que
            #      saltar N elementos de tipo entero.

            li    $t4, 0      # índice utilizado para recorrer la fila o col.
            li    $v1, 0      # $v1 almacena el valor de la suma
bucle:      bge   $t4, $t2, fin
            lw    $t5, ($t1)
            add   $v1, $v1, $t5
            addi  $t4, $t4, 1
            add   $t1, $t1, $t3     # dirección del siguiente elemento de
                                    # la fila i o columna i
            b     bucle

fin:        li    $v0, 0
            jr    $ra
error:      li    $v0, -1
            li    $v1, 0
            jr    $ra
```

b) Para invocar a la función se puede utilizar el siguiente fragmento de programa:

```
la    $a0, A
lw    $a1, M
lw    $a2, N
li    $a3, 1

# el valor de i (i=2) se pasa en la pila
addiu $sp, $sp, -4
li    $t0, 2
sw    $t0, ($sp)

jal   sumar

# se deja la pila como estaba inicialmente
addiu $sp, $sp, 4

# se imprime el primer valor
move  $a0, $v0
li    $v0, 1
syscall


# se imprime el segundo valor
move  $a0, $v1
syscall
```

Ejercicio 3.

a) El procesador WepSim es un procesador de 32 bits y según el esquema de la unidad de control emplea 7 bits para el código de operación. De acuerdo al enunciado el valor `n` debe poder especificar cualquier número entero y el campo `dirección` debe poder especificar cualquier dirección. Por tanto, cada uno de estos campos debe ocupar 32 bits (el ancho de palabra del computador). El código de operación ocupa 7 bits según el esquema mostrado para la unidad de control. En cuanto al campo `Rd`, se trata de un registro y se necesitan 5 bits para su codificación. Esta instrucción necesita, por tanto, tres palabras: en la primera se codifica el código de operación (bits 31 a 25) y el registro `Rd` (bits 24 a 20), en la segunda en campo `n` y en la tercera el campo `dirección`.

b) La instrucción `lw Rd dirección` ocupa dos palabras.

- *Fetch*:

Ciclo	Operaciones elementales
C0	MAR ← PC
C1	MBR ←MP[MAR] PC ← PC+4
C2	IR ← MBR
C3	Decodificar y salto a Código de operación.

- Operaciones elementales para la ejecución de la instrucción:

Ciclo	Operaciones elementales
C0	MAR ← PC
C1	MBR ←MP[MAR] PC ← PC+4
C2	MAR ← MBR
C3	MBR ←MP[MAR]
C4	Rd ← MBR

c) A continuación se indican las señales de control necesarias para estas operaciones elementales:

Reg ← RT2 (el campo Reg se encuentra en la instrucción entre los bits 25 y 21). Después se sigue con la siguiente microinstrucción.	T5, SelC= 10101, MR = 0, LC, A0=0, B = 0, C = 0
RT1 ← regOr1 + RT2 (el campo RegOr1 se encuentra en la instrucción entre los bits 12 y 8. Esta operación actualiza el registro de estado. Después se sigue con la siguiente microinstrucción.	SelA=01000, MR = 0, MB = 01, Cop = 1010 (SUMAR), T6, C4, SelP = 11, M7, C7, A0=0, B = 0, C = 0
If (SR(Z) ≠ 0) salta a la microdirección que se encuentra en la etiqueta utilizada en WepSim `salto`. En caso contrario sigue con la siguiente microinstrucción.	A0=0, C = 0111, B = 0, MADDR = salto
MBR ← MP[MAR] (lectura de una palabra). Después se sigue con la siguiente microinstrucción.	Ta, BW = 11, R, M1, C1, A0=0, B=, C=0

Ejercicio 4.

a) Como el computador tiene 32 bits, como mucho puede direccionar 2^{32} bytes = $2^{32} / 2^{20} = 2^{12}$ MB.
b) Se necesitan $\log_2 64 = 6$ bits.
c) El contador de programa es un registro de control del procesador que almacena la dirección de la siguiente instrucción a ejecutar. Se utiliza para acceder en memoria a las instrucciones que se tienen que ejecutar en el procesador.
d) La unidad de control realiza las siguientes acciones:
    - Comprueba se hay activada una señal de interrupción.
    - Si está activada:
        - Salva el contador de programa y el registro de estado. Se salvan en la pila o en registros especiales del procesador.
        - Pasa de modo usuario a modo núcleo (modificando el registro de estado).
        - Obtiene la dirección de la rutina de tratamiento de la interrupción.
        - Almacena en el contador de programa la dirección obtenida (de esta forma la siguiente instrucción será la de la rutina de tratamiento de la interrupción).

Ejercicio 5.

a) La caché de datos tiene un tamaño de 128 KB = 2^{17} bytes. Las líneas son de 64 bytes.
El número de líneas es de $2^{17} / 2^6 = 2^{11} = 2048$.
El número de conjuntos es de $2^{11} / 2^3 = 2^8 = 256$.

b) En cada iteración del bucle se produce los siguientes accesos a datos:
    - Una lectura de un elemento de tipo double (vector C). Este elemento ocupa 8 bytes de acuerdo al estándar IEEE 754 de doble precisión.
    - Una lectura de un elemento de tipo float (vector B). Este elemento ocupa 4 bytes de acuerdo al estándar IEEE 754 de simple precisión.
    - Una escritura de un elemento de tipo int (vector A). Este elemento ocupa 4 bytes en un computador de 32 bits como el del enunciado.

    En una línea de caché (64 bytes) caben 16 elementos de tipo int y de tipo float y 8 de tipo double.

    A continuación, se indica el patrón de accesos:

    i=0: Lectura de C[0]. Se produce un fallo. Se trae a caché una línea: C[0]…C[7].
    Lectura de B[0]. Se produce un fallo. Se trae a caché una línea: B[0]…C[15].
    Escritura en A[0]. Se produce un fallo. Se trae a caché una línea: A[0]…A[15].

    Desde i = 0 hasta i = 7 todos los accesos son aciertos, todos los elementos están en la caché.

    i=8: Lectura de C[8]. Se produce un fallo. Se trae a caché una línea: C[8]…C[15].
    Lectura de B[8]. Se produce un acierto. El elemento está en la caché.
    Escritura en A[8]. Se produce un acierto. El elemento está en la caché.

    Desde i=9 hasta i = 15 todos los accesos son aciertos. A partir de la iteración i=16, el patrón se repite, por tanto, basta con analizar lo que ocurre en las 16 primeras iteraciones (desde i=0 hasta i =15). En estas 16 iteraciones se producen 16x3 = 48 accesos a memoria, de los cuales 4 son fallos. Por tanto, de forma general, la tasa de fallos es de 4/48 y la tasa de aciertos de 44/48. Se ha considerado que los accesos a un double suponen un acceso a memoria. En caso de requerir dos accesos a memoria, el número de acceso sería de 16x4 = 64 accesos y la tasa de fallos sería 4/64.

c) Como la caché está estructurada en conjuntos de 8 líneas, se puede mantener siempre en caché líneas correspondientes a los tres vectores. Además, los datos no se reutilizan. Por tanto, no hay expulsiones y el comportamiento es similar a una caché totalmente asociativa (en cuanto a número de fallos). Una caché asociativa requeriría más bits para la etiqueta. Si la búsqueda de las etiquetas se hace en paralelo, el

rendimiento sería similar, pero se necesitaría una compleja circuitería para examinar en paralelo las etiquetas de todas las líneas de la caché. Si la búsqueda no se hace en paralelo, el rendimiento sería peor.