UNIVERSIDAD CARLOS III DE MADRID. DEPARTAMENTO DE INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA. DOBLE GRADO ADE E INGENIERÍA INFORMÁTICA
ESTRUCTURA DE COMPUTADORES
8 de enero de 2018 **Examen final**

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

Ejercicio 1 (1puntos). Sea un computador de 32 bits que direcciona la memoria por bytes y que incluye un banco de 32 registros. Se pide:

a) ¿Qué es el contador de programa y qué tipo de registro es?
b) ¿Cuántos MB de memoria principal es capaz de direccionar este computador?
c) ¿Cuántos bits se pueden almacenar en un registro y en una posición de memoria?
d) ¿Cómo se representa el valor -2.5 en este computador, si se utiliza el estándar IEEE 754 de simple precisión? Exprese el resultado en hexadecimal.

Ejercicio 2 (3 puntos). Considere la rutina `Insertar`. Esta rutina acepta **cuatro** parámetros de entrada:

- La dirección de inicio de una matriz (se almacena por filas) de números de tipo `float`, de dimensión `NxN`
- La dirección de inicio de un vector de números de tipo `float`, de dimensión `N`.
- La dimensión de la matriz y del vector (`N`).
- Un número `j`.

La función inserta el vector pasado como segundo argumento en la fila `j` de la matriz. También inserta dicho vector en la columna `j` de la matriz. La inserción implica copiar el vector en la fila y columna correspondiente. Si el valor de `j` está fuera de rango, la función devuelve -1 y no realiza ninguna inserción, en caso contrario devuelve 0 y realiza la inserción en la fila y columna correspondiente.

Se pide:

a) Codifique correctamente la rutina `Insertar` anteriormente descrita. Esta rutina deberá obligatoriamente invocar a otras dos funciones, que deberán también desarrollarse:
   a. Una función que permitirá insertar un vector en una fila.
   b. Una función que permitirá insertar un vector en una columna.

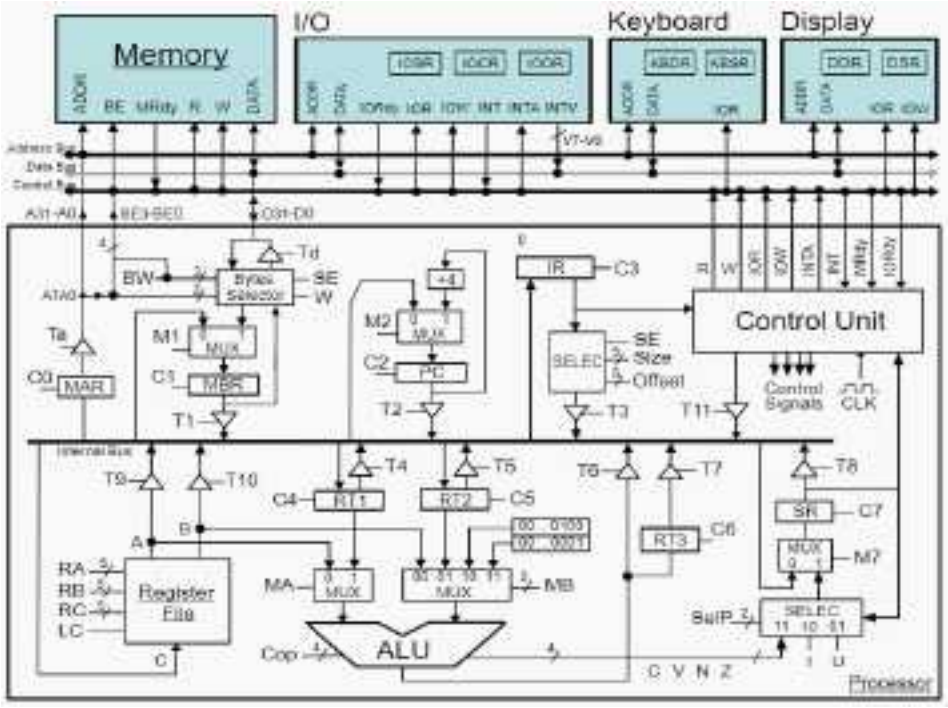
   A de seguirse estrictamente el convenio de paso de parámetros del MIPS visto en clase.

b) Dada la siguiente definición de matriz de dimensión 4x4 y vector de 4 elementos:

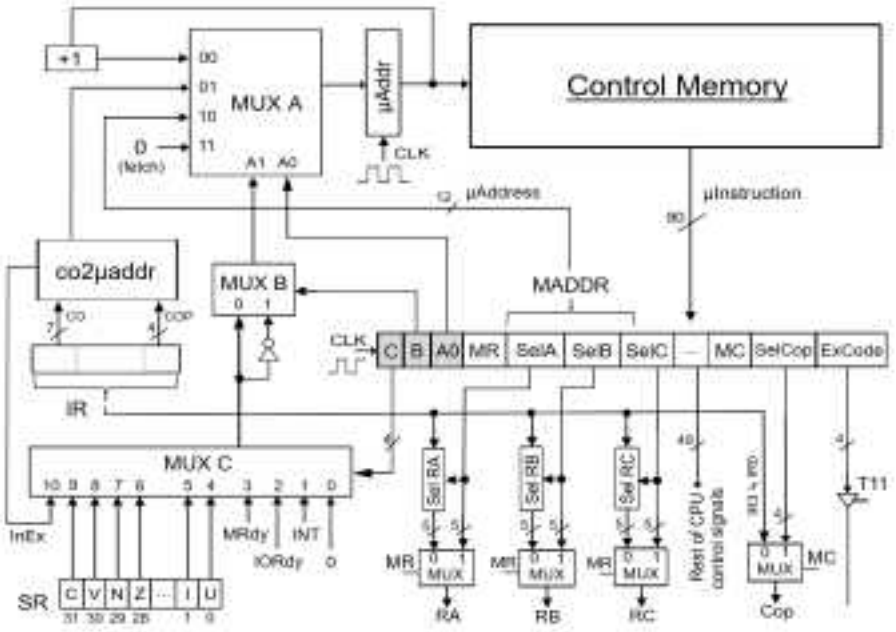
```
.data
  Matriz:  .float     0.0, 0.1, -0.2, 4.0
                      1.0, 1.1, 1.2, 3.0
                      2.0, 2.1, 2.2, 2.9
                      0.0, -2.3, 4.5, 8.0
  Vector: .float  4.0, 5.0, 6.0, 7.0
```

Codifique el fragmento de código que permite invocar correctamente a la función `Insertar` de forma que se inserte el vector anterior en la fila y columna 2.

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) Especifique las operaciones elementales y señales de control necesarias para ejecutar la instrucción máquina `strchr Ra Rb Rc` (incluya el ciclo de *fetch*). Esta instrucción recibe en el registro indicado por `Rb` la dirección no nula de una cadena de caracteres terminada en el valor cero y en el registro indicado por `Rc` el código ASCII de un carácter. La instrucción deberá devolver en `Ra` la dirección de la primera ocurrencia de dicho carácter en la cadena. El único registro que la instrucción modifica es el indicado por `Ra`.

b) Decida razonadamente qué pasará si no aparece el carácter en la cadena pensando en el programador en ensamblador que use dicha instrucción.

NOTA: Asuma que R29 actúa como puntero de pila, que el puntero de pila apunta a cima de pila y que la pila crece hacia direcciones decrecientes de memoria.

Ejercicio 4 (1punto). Imagine que la empresa para la que ha desarrollado las instrucciones strlen_2 y skipasciicode_2 tiene un nuevo proyecto de sonómetro clínico "inteligente" que está desarrollando.
Hay un prototipo de dispositivo y el módulo de E/S asociado que emplea una arquitectura MIPS 32 con mapa de entrada y salida conjunto y técnica de entrada/salida programada.
El módulo de entrada/salida dispone de tres registros de 32 bits:

- **Registro de datos** (con dirección 0x104). Almacena el número de decibelios leídos.
- **Registro de control** (con dirección 0x108). Cuando se escribe en el registro el valor 0x123 se enciende el sensor. Cuando se escribe el valor 0x321 se apaga. Cuando se escribe el valor 0x111, se solicita que el sensor realiza una medición.
- **Registro de estado** (con dirección 0x100). Si el valor del registro es 0, no se ha completado ninguna medición. Si el valor es 1, se ha completado una medición y el controlador dispone del valor en el registro de datos. Si el valor es -1, se ha producido un error y ha de volver a apagarse el sensor y volverlo a encenderlo. Un valor de 0 indica que el dispositivo todavía no ha terminado su encendido y que todavía no ha finalizado el apagado. Un valor de 1 indica que ya se ha completado el encendido y el apagado.

Dicha empresa le pide que desarrolle una rutina en ensamblador que se encargue de encender el sensor y realizar de forma indefinida la lectura de nivel de sonido.
El valor leído se imprime por pantalla (usando el servicio syscall con código 1). Cuando se detecta un error, ha de apagarse el sensor y volver a encenderlo para continuar el proceso de lectura. La función nunca finaliza su ejecución.

Ejercicio 5 (2 puntos). Considere el siguiente fragmento de código:

```
float  A[10000];
double B[10000];

for (i=0; i<10000; i++) {
     B[i] = A[i] + A[i];
}
```

Dicho código se ejecuta en una arquitectura 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.
Se pide:

a) Indique el número de líneas y conjuntos de esta memoria.
b) Indique la tasa de aciertos que produce la ejecución del fragmento de código anterior teniendo en cuenta solo accesos a los vectores A y B.
c) Modifique el fragmento de código anterior para reducir el número de accesos a memoria.

Guía rápida del ensamblador MIPS32

Instrucciones de bifurcación y salto

b etiqueta	Bif. incondicional a la instrucción que está en etiqueta.
beq Rsrc1, Src2, etiqueta	Bif. condicional si Rsrc1 es igual a Src2.
beqz Rsrc, etiqueta	Bif. condicional si el registro Rsrc es igual a 0.
bge Rsrc1, Src2, etiqueta	Bif. condicional si el registro Rsrc1 es mayor o igual a Src2 (con signo).
bgt Rsrc1, Src2, etiqueta	Bif. condicional si el registro Rsrc1 es mayor que Src2 (con signo).
ble Rsrc1, Src2, etiqueta	Bif. condicional si Rsrc1 es menor o igual a Src2 (con signo).
blt Rsrc1, Src2, etiqueta	Bif. condicional si Rsrc1 es menor que Src2 (con signo).
bne Rsrc1, Src2, etiqueta	Bif. condicional si Rsrc1 no es igual a Src2.
bnez Rsrc, etiqueta	Bif. condicional si Rsrc no es igual a 0.

Instrucciones aritméticas del coprocesador de coma flotante

add.s fd, fs, ft	Suma los registros fs y ft y almacena el resultado en fd (float)
add.d fd, fs, ft	Suma los registros fs y ft y almacena el resultado en fd (double)
div.s fd, fs, ft	Divide fs entre ft y deja el resultado en fd (float)
div.d fd, fs, ft	Divide fs entre ft y deja el resultado en fd (double)
mul.s fd, fs, ft	Multiplica los registros fs y ft y deja su resultado en fd. (float)
mul.d fd, fs, ft	Multiplica los registros fs y ft y deja su resultado en fd. (double)

Instrucciones de carga y almacenamiento del coprocesador de coma flotante

l.s fd, dirección	Carga en fd el valor del float (32 bits) que se encuentra a partir de la dirección especificada.
s.s fs, dirección	Almacena el registro fs a partir de la dirección indicada. (float)
li.s fd, valor	Carga en el registro fd del coprocesador matemático el valor (float)

El campo dirección se puede representar utilizando direccionamiento absoluto, indirecto de registro o relativo a registro

Instrucciones de transferencia de datos entre registros

move Rdest, Rsrc	Mueve el contenido del registro Rsrc al registro Rdest (del procesador general)
mfc1 Rdest, CPsrc	Mueve el contenido del registro CPsrc del coprocesador en coma flotante al registro de la CPU Rdest.
mtc1 Rsrc, CPdest	Mueve el contenido del registro Rsrc de la CPU al registro Cpdest del coprocesador en coma flotante.
mov.s fd, fs	Mueve el contenido del registro fs al registro fd. (float)

Instrucciones de comparación

En todas las instrucciones siguientes, Src2 puede ser un registro o un valor inmediato (de 16 bits).

seq Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 y Src2 son iguales, en otro caso pone 0.
sge Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es mayor o igual a Src2, y 0 en otro caso (para números con signo).
sgeu Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es mayor o igual a Src2, y 0 en otro caso (para números sin signo).
sgt Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es mayor que Src2, y 0 en otro caso (para números con signo).
sgtu Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es mayor que Src2, y 0 en otro caso (para números sin signo).
sle Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es menor o igual a Src2, en otro caso pone 0 (para números con signo).
sleu Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es menor o igual a Src2, en otro caso pone 0 (para números sin signo).
slt Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es menor a Src2, en otro caso pone 0 (para números con signo).
sltu Rdest, Rsrc1, Src2	Pone Rdest a 1 si Rsrc1 es menor a Src2, en otro caso pone 0 (para números sin signo).
sne Rdest, Rsrc1, Src2	Pone Rdest to 1 si el registro Rsrc1 no es igual a Src2 y 0 en otro caso.

Soluciones

Ejercicio 1

a) El contador de programa es un registro de control que almacena la dirección de la siguiente instrucción a ejecutar.
b) El computador es capaz de direccionar $2^{32} / 2^{20} = 2^{12}$ MB.
c) En cada registro se almacenan 32 bits (el ancho de palabra). En cada posición de memoria se almacena 1 byte (direcciona por bytes).
d) $-2.5_{(10} = 10.1_{(2} = 1.01 \times 2^{1}{}_{(2}$

Signo = 1 (negativo)
Exponente = 127+1 = $128_{(10} = 10000000_{(2}$
Mantisa = 010000…. 000000 (23 bits)

El número es 1 10000000 0100000…..000000 = 0xC0200000

Ejercicio 2

a)

```
insertar: # comprobar valores fuera de rango
          li   $v0, -1
          blt  $a3, $0, error
          bgt  $a3, $a2, error

          # se guarda la dirección de retorno y los registros $ai
          # en la pila
          addi $sp, $sp, -20
          sw   $ra ($sp)
          sw   $a0, 4($sp)
          sw   $a1, 8($sp)
          sw   $a2, 12($sp)
          sw   $a3, 16($sp)

          jal  InsertarFila

          # restaurar los registros $ai por si la función
          # anterior los modificó
          lw   $a0, 4($sp)
          lw   $a1, 8($sp)
          lw   $a2, 12($sp)
          lw   $a3, 16($sp)

          jal  InsertarCol

          lw   $ra ($sp)
          lw   $a0, 4($sp)
          lw   $a1, 8($sp)
          lw   $a2, 12($sp)
          lw   $a3, 16($sp)
          addi $sp, $sp, +20

          li   $v0, 0
error:    jr   $ra
```

```
   InsertarFila: #asume que j no está fuera de rango

              # se obtiene la dirección de inicio de la fila j
              li    $t0, 4
              mul   $t0, $t0, $a2
              mul   $t0, $t0, $a3

              add   $a0, $a0, $t0  # a0 apunta a la fila j

              li    $t0, 0
   bucle1:    bge   $t0, $a2, fin1
              lw    $t1,($a1)
              sw    $t1, ($a0)
              addi  $a1, $a1, 4
              addi  $a0, $a0, 4
              addi $t0, $t0,1
              b     bucle1
   fin1:      jr    $ra

   InsertarCol: #asume que j no está fuera de rango

              # se obtiene la dirección de inicio del
              # primer elemento de la columna j

              li    $t0, 4
              mul   $t0, $t0, $a3

              add   $a0, $a0, $t0

              # entre cada elemento de la columna j hay
              # que avanzar N*4 bytes
              li    $t1, 4
              li    $t1, $t1, $a2

              li    $t0, 0
   bucle2:    bge   $t0, $a2, fin2
              lw    $t1,($a1)
              sw    $t1, ($a0)
              addi  $a1, $a1, 4
              addi  $a0, $a0, t1  # Avanza a la siguiente col.
              addi $t0, $t0,1
              b     bucle2
   fin2:      jr    $ra

b) main:
              addi  $sp, $sp, -20
              sw    $ra ($sp)
              sw    $a0, 4($sp)
              sw    $a1, 8($sp)
              sw    $a2, 12($sp)
              sw    $a3, 16($sp)

              la    $a0, Matriz
              la    $a1, Vector
              li    $a2, 4
              li    $a3, 2
              jal   Insertar
```

```
        lw    $ra ($sp)
        lw    $a0, 4($sp)
        lw    $a1, 8($sp)
        lw    $a2, 12($sp)
        lw    $a3, 16($sp)
        addi  $sp, $sp, 20

        jr    $ra
```

Ejercicio 3

a)

Operaciones elementales	Señales de control
MAR <- PC	T2, C0
M[MAR] -> MBR, PC <- PC+4	Ta, R, BW=11, M1=1, C1, M2=1, C2
MBR -> RI	T1, C3
Decodificar y salto a c.o.	C=0, B=0, A0=1
RT2 <- SR	T8, C5
{MAR, Ra} <- Rb	MR=0, SelA=<Rb>, SelC=<Ra>, LC=1, T9, C0
bucle1: MBR <- Mem[Ra]	Ta, BW=0, R, M1, C1
RT1 <- MBR	T1, C4
RT1 * 1 -> flags	MA=1, MB=11, SelCOP=*, MC=1, SelP=11, M7, C7
bZ fin1	C=6, B=0, A0=0, MADDR=fin1
RT1 – Rc -> flags	MA=1, SelB=<Rc>, SelCOP=-, MC=1, M7, C7
bZ fin2	C=6, B=0, A0=0, MADDR=fin2
{MAR,Ra} <- Ra + 1	MR=0, SelA=<Ra>, SelC=<Ra>, LC=1, MB=11, SelCOP=+, MC=1, T6, C0
B bucle1	C=0, B=1, A0=0, MADDR=bucle1
fin1: Ra <- 0	MR=0, SelA=<R0>, SelC=<R1>, LC=1, T9, C0
fin2: SR <- RT2	MB=01, SelA=<R0>, SelCOP=+, MC=1, T6, C7
Salto a fetch	C=0, B=1, A0=0, MADDR=fetch

b) Dado que se espera devolver una dirección, podría usarse la dirección cero como indicador de que no está presente (si en las primeras direcciones de memoria no se almacena información del usuario, podría usarse como valor fuera del uso habitual de strchr).

Ejercicio 4

Una posible solución sería:

```
Sonómetro:
             # encender el hardware

  ini:       li    $t0, 0x123
             sw    $t0, 0x108

             # leer registro de control hasta que sea distinto de cero
  bwait:     lw     $t0,  0x100
             beqz   $t0,  bwait
```

```
medidas:   li    $t1, 0x111
           li    $t2, -1

medida:    sw    $t1, 0x108

bread:     lw    $t0, 0x100
           beqz  $t0, bread

           beq   $t0, $t2, error

           #leer el valor del registro de datos
           lw    $a0,  0x104

           # imprime el valor leído y volver a leer
           li    $v0  1
           syscall
           b medida

error:     # se apaga
           li    $t0, 0x321
           sw    $t0, 0x108

           #espera el apagado
boff:      lw    $t0, 0x100
           beqz $t0, boff

           #vuelve al principio
           b     ini

           jr    $ra
```

Ejercicio 5.

a) La caché tiene un tamaño de 128KB = 2^{17} bytes. Las líneas son de 64 bytes, por tanto, 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 línea de caché de 64 bytes se pueden almacenar 64/4 = 16 elementos de vector A (elementos de tipo float) y 64/8 = 8 elementos del vector B (elementos de tipo double).

Se va a considerar en primer lugar los accesos al vector A:

Iteración i = 0;

- Lectura de A[0] Fallo se trae a cache A[0]… A[15]
- Lectura de A[0] Acierto
- Lectura de A[1] Acierto
- Lectura de A[1] Acierto
- …. Se repite el patrón de accesos

Cada 16 iteraciones hay 32 accesos al vector A (dos lecturas) y un único fallo. En las 10000 iteraciones habrá 20000 accesos a A de los cuales fallos serán 20000 x 1/32 = 625.

A continuación, se van a analizar los accesos a B:

Iteración i = 0;

- Escritura de B[0] Fallo se trae a cache B[0]… B[7]
- Escritura de B[1] Acierto
- …. Se repite el patrón de accessos

Cada 8 iteraciones hay 8 accesos a B de los cuales uno es un fallo. En las 10000 iteraciones hay 10000 accesos, de los cuales 10000 x 1/8 = 1250 son fallos.

El número total de accesos es 30000 y el número de aciertos es 30000-(625+1250) = 28125. La tasa de aciertos es 28125/30000 = 0.9375

c) Una forma de recudir el número de accesos sería convertir el código en el siguiente:

```
for (i=0; i<10000; i++) {
     B[i] = 2* A[i];
}
```