Tema 6: LENGUAJE ENSAMBLADOR

Grado en Informática

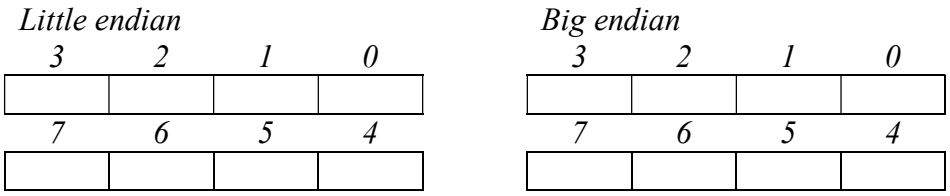
EJERCICIOS

Organización de memoria

M 1)

Distribuir los siguientes datos, de 4 bytes de tamaño cada uno, en las direcciones de memoria correspondientes.

Dato 1: 0xABCDEFFF, Dato 2: 0x01234567



M 2)

Dadas las siguientes directivas de datos, Indique cuál será el contenido de la memoria de datos, sabiendo que NULL representa el carácter nulo y que la máquina en cuestión almacena las palabras según el formato *Little Endian*. Indique claramente las zonas de memoria de contenido desconocido.

```
.data 0x10000028
.byte 3
.ascii "ABC"
.float 1.5
.word 0xFFFF
.half 19,38
```

31 24	23 16	15 8	7 0	Dirección

M 3)

Dadas las siguientes directivas de datos, Indique cuál será el contenido de la memoria de datos, sabiendo que NULL representa el carácter nulo y que la máquina en cuestión almacena las palabras según el formato *Little Endian*. Indique claramente las zonas de memoria de contenido desconocido.

```
.data 0x1000000C
.space 2
.half 0xF0
.asciiz "050"
.double 1.5
```

31 24	23 16	15 8	7 0	Dirección

M 4)

Dadas las siguientes directivas de datos, Indíquese cuál será el contenido de la memoria de datos, sabiendo que NULL representa el caracter nulo y que la máquina en cuestión almacena las palabras según el formato *Little Endian*. Indíquese claramente las zonas de memoria de contenido desconocido.

```
.data 0x10000000
.half 5,3
.byte 3
.data 0x10000011
.byte 5
```

31 24	23 16	15 8	7 0	Dirección

Juego de instrucciones

JI 1)

Dado el siguiente contenido de la memoria de datos:

Memoria de datos					Dirección
0x	6C	FF	FF	FF	0x10000000
0x	AB	77	80	44	0x10000004
31				0	

¿Qué valor tendrán los registros $5 y $6 tras ejecutarse las siguientes instrucciones?

```
lui $2, 0x1000
lh $5, 0($2)
lw $6, 4($2)
```

Solución

JI 2)

Dado el siguiente contenido de la memoria de datos:

Memoria de datos					Dirección
0x	12	34	56	78	0x10010008
0x	CB	00	88	00	0x1001000C
31				0	

¿Qué valor tendrán los registros $4 y $5 tras ejecutarse las siguientes instrucciones?

```
lui $3, 0x1001
lw $4, 12($3)
lb $5, 9($3)
```

Solución

Programación en ensamblador y código máquina

Prog 1)

Dado el código en lenguaje ensamblador MIPS R2000 que se muestra a continuación.

```
      .data 0x100000A0
      .byte 1,2,3
      .half 4
dato1:.word 8
dato2:.word 2
      .space 5
      .word 9
      .text 0x00400000
      .globl __start
```

```
__start:
      la $2, dato1
      la $3, dato2
      lw $8,0 ($2)
      lw $4,-4 ($3)
      add $9,$4,$8
      sw $9, 0 ($2)
      .end
```

A. ¿Cuál es el contenido de la memoria de datos antes de la ejecución del programa?

31 24	23 16	15 8	7 0	**Dirección**

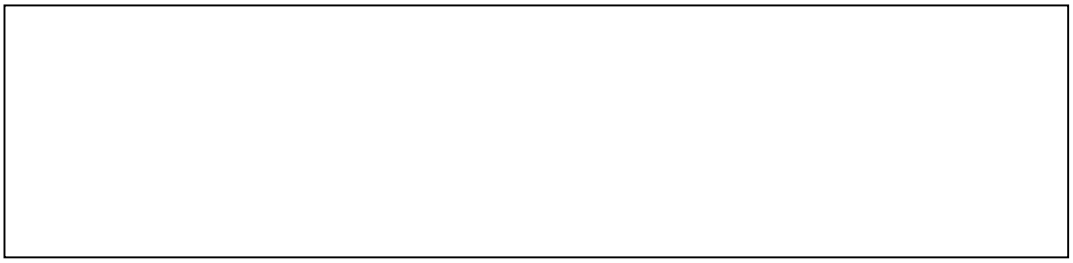
B. ¿Cuál es el contenido de la memoria de datos después de la ejecución del programa?

31 24	23 16	15 8	7 0	**Dirección**

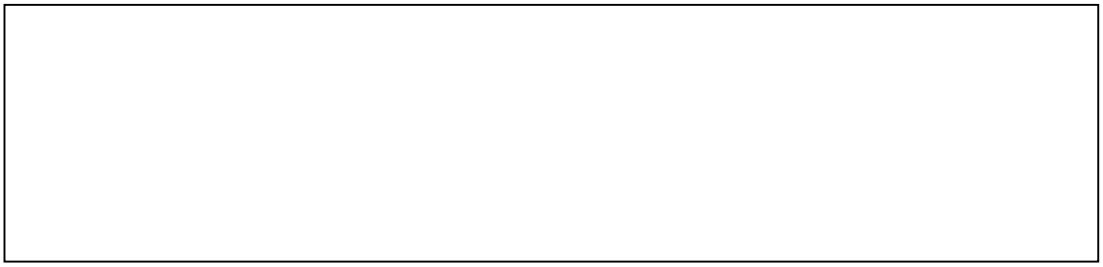
C. ¿Cuál será el valor almacenado en los siguientes registros después de la ejecución del programa?

Registro	**Valor**
$2	
$3	
$8	
$4	
$9	

D. Indique la secuencia de instrucciones por las que se traduciría la pseudoinstrucción `la $2, dato1`



E. Codifique la instrucción *`lw $4,-4 ($3)`*



Prog 2)

En un algoritmo de encriptación por bloques, se van cifrando bloques de un tamaño concreto. Para aumentar la seguridad del algoritmo, se suelen hacer operaciones previas entre bloques. Uno de los modos de operar se conoce como CBC y consiste en realizar una OR exclusiva entre el bloque de texto a cifrar y el bloque precedente. El siguiente código comprueba el funcionamiento de este modo.

```
.data 0x10000000
tam:        .half 8                 # Tamaño de bloque
BloqueA:    .asciiz "or bloqu"      # Bloque precedente
BloqueB:    .asciiz "es, se v"      # Bloque para cifrar
BloqueF:    .space 8          # Espacio para el resultado

.globl __start
.text 0x00400000
__start:
      # Lectura de los datos iniciales
      la $10, BloqueA   # Leemos la dirección del bloque precedente
      la $11, BloqueB   # Leemos la dirección del bloque para cifrar
      la $12, BloqueF   # Leemos la dirección del bloque resultado
      la $13, tam       # Leemos el valor del tamaño de bloque
      lh $14, 0($13)

      # bucle del algoritmo
bucle:
      beq $14, $0, fin
      lb $20,0($10)
      lb $21,0($11)
      xor $22,$20,$21
```

```
        sb $22,0($12)
        addi $10,$10,1
        addi $11,$11,1
        addi $12,$12,1
        addi $14,$14,-1
        j bucle
fin:    # Final del algoritmo
.end
```

A. Indique qué contenido tendrá la memoria de datos antes de que se ejecute el programa.

31 24	23 16	15 8	7 0	**Dirección**

B. Indique qué contenido tendrá la memoria de datos después de que se ejecute el programa.

31 24	23 16	15 8	7 0	**Dirección**

C. ¿Cuál será el valor almacenado en los siguientes registros después de la ejecución del programa?

Registro	**Valor**
$10	
$11	
$12	
$13	
$14	

D. Codifique la instrucción `j bucle`

E. Codifique la instrucción *`beq $14, $0, fin`*

Prog 3)

Dado el siguiente programa en ensamblador MIPS R2000:

```
      .data 0x10000000
estado:.byte 25
zonaA: .word xffffffff,0xffffffff,0xffffffff, 0xffffffff
tamanyo:.word 4

            .text 0x400400
            .globl __start
__start:
        la $6, estado
        lb $6, 0($6)
        la $7, zonaA
        la $8,tamanyo
        lw $9, 0($8)
        li $10, 0x00000000
        li $11, 0xaaaaaaaa
        beq $6,$0,accion0
accion1:
        beq $9,$0,fin
        sw $11, 0($7)
        addi $7,$7,4
        addi $9,$9,-1
        j accion1
accion0:
        beq $9,$0,fin
        sw $10, 0($7)
        addi $7,$7,4
        addi $9,$9,-1
        j accion0
fin:    .end
```

A. ¿Cuál es el contenido de la memoria de datos antes de la ejecución del programa?

31 24	23 16	15 8	7 0	**Dirección**

B. ¿Cuál es el contenido de la memoria de datos después de la ejecución del programa?

31 24	23 16	15 8	7 0	**Dirección**

C. Codifique la instrucción `j accion0`

D. Codifique la instrucción *`beq $6, $0, accion0`*

Prog 4)

Como parte de un algoritmo de realidad virtual, se quiere calcular el volumen de un prisma regular y comprobar si este volumen supera un cierto umbral. En caso de que el volumen supere el umbral, el programa escribe un 1 en el byte de memoria etiquetado como "res"; en otro caso escribe un cero. Asimismo, el volumen calculado se almacena en la dirección de memoria etiquetada como "vol". El código propuesto es el siguiente

```
.data 0x10000000
ladoA:      .half 100   # Arista A
ladoB:      .half 50    # Arista B
ladoC:      .half 25    # Arista C
umb:        .word 1500  # Umbral para la comparación
res:        .byte 0     # Espacio para resultado de comparación
vol:        .word 0     # Espacio para volumen prisma

.globl __start
.text 0x00400000
__start:
      # Lectura de los valores de las aristas
      la $20, ladoA
      lh $10,0($20)
      la $20, ladoB
      lh $11,0($20)
      la $20, ladoC
      lh $12,0($20)

      # Cálculo del volumen
      mult $10,$11
      mflo $13
      mult $12,$13
      mflo $13

      # Almacenamiento del volumen en memoria
      la $20, vol
      sw $13, 0($20)

      # Comparación con el umbral
      la  $20, umb
      lw  $14, 0($20)
      slt $15, $14, $13
      # Almacenamiento del resultado
      la $20, res
      sb $15, 0($20)
.end
```

A. Indique cuál es el estado del segmento de datos antes de que se ejecute el programa.

31 24	23 16	15 8	7 0	**Dirección**

B. Haga una propuesta de nueva declaración de datos que reduzca los espacios de memoria no usados a causa del alineamiento

C. ¿Cuál será el valor almacenado en los siguientes registros después de la ejecución del programa? En cualquier caso, indique en qué base están expresados los valores numéricos

Registro	**Valor**
$10	
$11	
$12	
$13	
$14	
$15	
$20	

D. Codifique la instrucción `slt $15, $14, $13`

Prog 5)

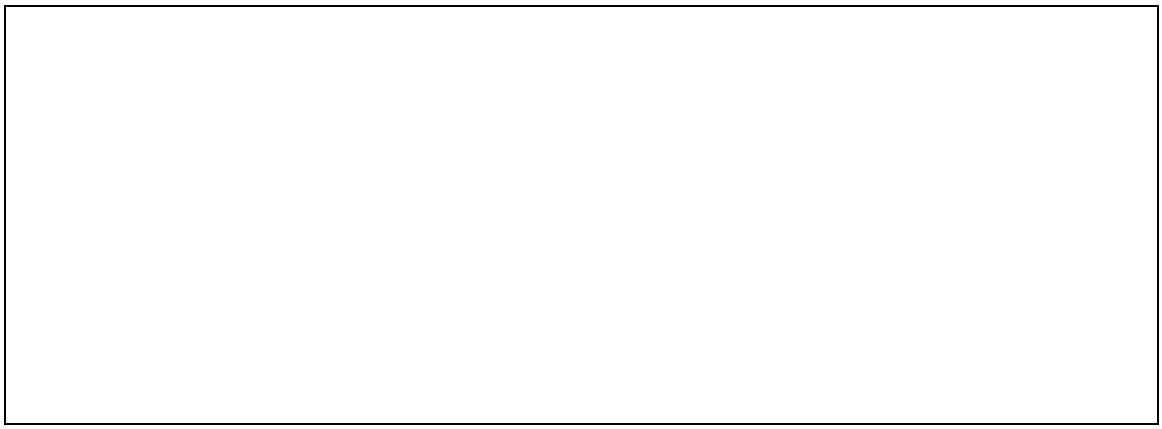
El código siguiente contiene una serie de errores. Identifíquelos

```
                .data 0x10000000
       vector: .byte 0x333, 0x88, 0x54, 0x77
                .word 0x55
                .half 0x44445555
                .space 18
                .text 0x400000
                .globl __start
                 __start:
                lui $10, 0x1000
                ori $10, $10, 0x0003
                lw $11, 0($10)
                lw $12, 1($10)
               .end
```

Prog 6)

¿Qué efecto tiene la ejecución del siguiente código?

```
.text 0x400800
.globl __start
__start:
  bucle:  lui $10, 0xFFFF
          andi $10, $10, 0xFFFF
          beq $10, $zero, bucle
          li $10, 0x12345678
.end
```



Prog 7)

Programe en ensamblador del MIPS R2000 el siguiente conjunto de operaciones:

resultado = datoa – datob + datoc - datod,

tenga en cuenta las siguientes consideraciones:

- Los datos “datoa”, “datob”, “datoc” y “datod” se definirán como enteros de 32 bits ubicados a partir de la posición de memoria 0x10002000
- Se debe reservar espacio para almacenar el “resultado” (entero de 32 bits) a partir de la posición 0x10001000.
- El programa guardará en “resultado” el valor final de la operación.

Prog 8)

Codifique en lenguaje ensamblador del MIPS R2000 un programa que realice la suma de dos variables de tipo short int (16 bits), “datoa” y “datob” y deje el resultado en “datoc”. El siguiente fragmento muestra en lenguaje de alto nivel lo que se desea hacer.

```
short int datoa := 9;
short int datob:=12;
short int datoc;

datoc := datoa+datob;
```

Realice la reserva de datos y las instrucciones precisas para acceder a las variables datoa y datob, sumarlas y guardar el resultado en datoc.

Prog 9)

Realice un programa en ensamblador del MIPS R2000 que calcule la operación:

"datoc = datoa * datob"

teniendo en cuenta las siguientes consideraciones:

- Los datos "datoa", y "datob" se definirán como enteros de 16 bits ubicados a partir de la posición de memoria 0x10000000
- Se debe reservar espacio para almacenar el resultado "datoc" (entero de 32 bits) a partir de la posición 0x10001000.

Prog 10)

Realice un programa en ensamblador del MIPS R2000 que calcule la operación

"datoa / datob",

y deje el valor de la división entera en "cociente", y el resto en la variable "resto", teniendo en cuenta las siguientes consideraciones:

- Los datos "datoa", "datob", "cociente" y "resto" se definirán como enteros de 32 bits ubicados a partir de la posición de memoria 0x10000000
- Se debe comprobar que "datob" no es cero para continuar con la división. Si lo es saltaremos a la etiqueta "DivisionEntreCero"

Prog 11)

Realice un programa en ensamblador del MIPS R2000 que realice la operación:

desultado = datoa + datob,

teniendo en cuenta las siguientes consideraciones:

- Los datos "datoa" y "datob" se definirán como enteros de 32 bits ubicados a partir de la posición de memoria 0x10000000
- Se debe reservar espacio para almacenar el "resultado" (entero de 32 bits) a partir de la posición 0x10001000.
- El programa guardará en "resultado" el valor final de la operación.
- El programa indicará en "haydesbordamiento" si la operación es correcta. Con un "1" se indicará que es incorrecta, o sea hay desbordamiento y con un "0" que es correcta, o sea no hay desbordamiento.
  NOTA: para saber si ha habido desbordamiento bastará con ver si el bit de signo de los sumandos son iguales y distintos del valor del resultado.

Prog 12)

Realice un programa en ensamblador del MIPS R2000 que realice la operación:

desultado = datoa - datob,

teniendo en cuenta las siguientes consideraciones:

- Los datos "datoa" y "datob" se definirán como enteros de 32 bits ubicados a partir de la posición de memoria 0x10000000
- Se debe reservar espacio para almacenar el "resultado" (entero de 32 bits) a partir de la posición 0x10001000.
- El programa guardará en "resultado" el valor final de la operación.
- El programa indicará en "haydesbordamiento" si la operación es correcta. Con un "1" se indicará que es incorrecta, o sea hay desbordamiento y con un "0" que es correcta, o sea no hay desbordamiento.
  ATENCIÓN: Sólo puede haber desbordamiento en una resta si los signos de los operandos son distintos. En este caso, habrá desbordamiento si el signo del resultado es distinto del signo del minuendo (datoa).

Prog 13)

Dadas las siguientes directivas de datos:

```
            .data 0x10000000
tira:       .asciiz "ABC"
tira_res:   .space 3
```

Escriba un programa en ensamblador que partiendo de la cadena de caracteres almacenada en la dirección ***tira,*** almacene la misma cadena pero con los caracteres en minúscula a partir de la dirección ***tira_res***. Las instrucciones del programa deberán ubicarse a partir de la dirección de memoria 0x00400000.

Nota: dada la tabla ASCII, se observa que si se suma 32 al código ASCII de un carácter alfabético en mayúsculas se obtiene el código del mismo carácter en minúsculas.

Ejercicios genéricos

G 1)

Teniendo en cuenta el formato de instrucción visto en las transparencias, ¿cuántas instrucciones de tipo R, I y J puede tener el MIPS?

G 2)

Si todas las instrucciones de tipo R tienen el código de operación 0 ¿cuántas instrucciones de tipo R como máximo puede tener el MIPS?

G 3)

En las instrucciones *sll* y *srl*, ¿cuál es el desplazamiento máximo que se puede poner? ¿Sería interesante tener más desplazamiento?

G 4)

Si el banco de registros del MIPS tuviese 64 registros de 32 bits, ¿qué cambios implicaría en el formato de las instrucciones de tipo R?

G 5)

Si el tamaño de los registros del banco de registros del MIPS pasase a ser de 64 bits, manteniéndose 32 registros, ¿qué cambios implicaría en el formato de las instrucciones de tipo R?

G 6)

Si el banco de registros del MIPS tuviese 64 registros de 32 bits, ¿qué cambios implicaría en el formato de las instrucciones de tipo I?

G 7)

¿Cuál es la distancia máxima a la que puede hacer un salto condicional?

G 8)

¿Por qué se codifica en complemento a dos el dato inmediato en las instrucciones de salto condicional?

G 9)

¿Por qué se codifica el salto en palabras en las instrucciones de salto condicional? ¿Qué distancia máxima se alcanzaría si se codificase en bytes y no en palabras?

G 10)

¿Cuál es el desplazamiento máximo teórico que se puede alcanzar con una instrucción de carga o almacenamiento?

G 11)

¿Por qué las instrucciones de carga y almacenamiento tienen el desplazamiento codificado en bytes?

G 12)

El hecho de no poder saltar a una dirección que no comience por 0x0…. ¿hace que el MIPS se deje fuera parte de la zona de memoria destinada al código? ¿por qué?

G 13)

¿Por qué la instrucción *jr* no se considera de tipo J?

G 14)

El tamaño de dirección del MIPS es de 32 bits. ¿En qué afectaría a las instrucciones de tipo J un aumento de tamaño de dirección del MIPS a 64 bits?