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

Little endian

3	*2*	*1*	*0*
0xAB	**0xCD**	**0xEF**	**0xFF**
7	*6*	*5*	*4*
0x01	**0x23**	**0x45**	**0x67**

Big endian

3	*2*	*1*	*0*
0xFF	**0xEF**	**0xCD**	**0xAB**
7	*6*	*5*	*4*
0x67	**0x45**	**0x23**	**0x01**

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
'C'	**'B'**	**'A'**	**0x03**	**0x10000028**
0x3F	**0xC0**	**0x00**	**0x00**	**0x1000002C**
0x00	**0x00**	**0xFF**	**0xFF**	**0x10000030**
0x00	**0x26**	**0x00**	**0x13**	**0x10000034**
?	**?**	**?**	**?**	**0x10000038**
?	**?**	**?**	**?**	**0x1000003C**

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
0x00	0xF0	0x00	0x00	0x1000000C
NULL	'0'	'5'	'0'	0x10000010
?	?	?	?	0x10000014
0x00	0x00	0x00	0x00	0x10000018
0x3F	0xF8	0x00	0x00	0x1000001C
?	?	?	?	0x10000020

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
0x00	0x03	0x00	0x05	0x10000000
?	?	?	0x03	0x10000004
?	?	?	?	0x10000008
?	?	?	?	0x1000000C
?	?	0x05	?	0x10000010
?	?	?	?	0100000014

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

$5=0xFFFFFFFF
$6=0xAB778044

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

$4=0xCB008800
$5=0x00000056

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
?	0x03	0x02	0x01	0x100000A0
?	?	0x00	0x04	0x100000A4
0x00	0x00	0x00	0x08	0x100000A8
0x00	0x00	0x00	0x02	0x100000AC
0x00	0x00	0x00	0x00	0x100000B0
?	?	?	0x00	0x100000B4
0x00	0x00	0x00	0x09	0x100000B8

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
?	0x03	0x02	0x01	0x100000A0
?	?	0x00	0x04	0x100000A4
0x00	0x00	0x00	0x10	0x100000A8
0x00	0x00	0x00	0x02	0x100000AC
0x00	0x00	0x00	0x00	0x100000B0
?	?	?	0x00	0x100000B4
0x00	0x00	0x00	0x09	0x100000B8

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

Registro	Valor
$2	0x100000A8
$3	0x100000AC
$8	0x00000008
$4	0x00000008
$9	0x00000010

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

lui $1, 4096 # lui $1,0x1000
ori $2,$1,168 # ori $2, $1, 0x00A8

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

0x8C64FFFC

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
0x72	0x6F	0x00	0x08	0x10000000
0x6F	0x6C	0x62	0x20	0x10000004
0x65	NULL	0x75	0x71	0x10000008
0x73	0x20	0x2C	0x73	0x1000000C
NULL	0x76	0x20	0x65	0x10000010
0x00	0x00	0x00	0x00	0x10000014
0x00	0x00	0x00	0x00	0x10000018

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
0x72	0x6F	0x00	0x08	0x10000000
0x6F	0x6C	0x62	0x20	0x10000004
0x65	NULL	0x75	0x71	0x10000008
0x73	0x20	0x2C	0x73	0x1000000C
NULL	0x76	0x20	0x65	0x10000010
0x42	0x0C	0x01	0x0A	0x10000014
0x03	0x51	0x0A	0x1F	0x10000018

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

Registro	Valor
$10	0x1000000A
$11	0x10000013
$12	0x1000001C
$13	0x10000000
$14	0x00000000

D. Codifique la instrucción `j bucle`

0x08100008

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

0x11C0000A

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
?	?	?	0x19	0x10000000
0xFF	0xFF	0xFF	0xFF	0x10000004
0xFF	0xFF	0xFF	0xFF	0x10000008
0xFF	0xFF	0xFF	0xFF	0x1000000C
0xFF	0xFF	0xFF	0xFF	0x10000010
0x00	0x00	0x00	0x04	0x10000014

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
?	?	?	0x19	0x10000000
0xAA	0xAA	0xAA	0xAA	0x10000004
0xAA	0xAA	0xAA	0xAA	0x10000008
0xAA	0xAA	0xAA	0xAA	0x1000000C
0xAA	0xAA	0xAA	0xAA	0x10000010
0x00	0x00	0x00	0x04	0x10000014

C. Codifique la instrucción `j accion0`

0x08100110

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

0x10C00006

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
0x00	0x32	0x00	0x64	0x10000000
?	?	0x00	0x19	0x10000004
0x00	0x00	0x05	0xDC	0x10000008
?	?	?	0x00	0x1000000C
0x00	0x00	0x00	0x00	0x10000010
?	?	?	?	0x10000014

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

Con la actual disposición se pierden 5 bytes a causa del alineamiento. Para solucionarlo se puden mover las etiquetas de manera que los datos que provocan un alineamiento se agrupen, de esta forma la declaración quedará como:

```
umb:        .word 1500
vol:        .word 0
ladoA:      .half 100
ladoB:      .half 50
ladoC:      .half 25
res:        .byte 0
```

Existen más soluciones que ahorran un espacio, aunque la más eficiente es la que se ha expuesto.

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	0x00000064
$11	0x00000032
$12	0x00000019
$13	0x0001E848
$14	0x000005DC
$15	0x00000001
$20	0x1000000C

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

0x01CD782A

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)   # $10+0 = 0x10000003
            lw $12, 1($10)
           .end
```

SOLUCIÓN:

Error 1: el dato 0x333 no es un BYTE, debería declararse con otra directiva como .HALF o .WORD

Error 2: el dato 0x44445555 no es media palabra (HALF), debería declararse con la directiva .WORD

Error 3: la dirección de acceso a memoria utilizada por la instrucción `lw $11, 0($10)` que accede a una palabra en memoria no es una dirección múltiplo de 4 por lo que la ejecución de dicha instrucción finalizaría el programa con error. La dirección que ocasiona el error de ejecución es: 0x10000003 que no es una dirección válida para acceso a una palabra en memoria.

Prog 6)

¿Qué efecto tiene la ejecución del siguiente código?

```
.globl __start
__start:
bucle:lui $10, 0xFFFF              # $10=0xFFFF0000
      andi $10, $10, 0xFFFF        # $10=0xFFFFFFFF
      beq $10, $zero, bucle        # $10<>0, por lo tanto no se salta
      li $10, 0x12345678           # $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.

Solución posible:

```
   .data 0x10002000
datoa: .word 9    # Se pueden inicializar con otros valores
datob: .word 2
datoc: .word 3
datod: .word 1

   .data 0x10001000
resultado: .word 0

   .globl __start
   .text 0x00400000
__start:
      #Carga de datos desde memoria a registros
       la $8, datoa
       lw $8, 0($8)        # $8 = datoa
       la $9, datob
       lw $9, 0($9)        # $9 = datob
       la $10, datoc
       lw $10, 0($10)      # $10 = datoc
       la $11, datod
       lw $11, 0($11)      # $11 = datod

     # Cálculos aritméticos, acumulando en registro $2
       sub $2, $8, $9      # $2 = datoa-datob
       add $2, $2, $10     # $2 = (datoa-datob)+datoc
       sub $2, $2, $11     # $2 = (datoa-datob+datoc)-datod

     # Escritura de resultado final desde registro a memoria
       la $7, resultado
       sw $2, 0($7)        # resultado =(datoa-datob+datoc)-datod
     .end
```

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.

Solución posible:

```
        .data 0x10000000
datoa:  .half 9    # Se pueden inicializar con otros valores
datob:  .half 12
datoc:  .half 0

        .globl __start
        .text 0x00400000
__start:  # Carga de datos desde memoria a registros
        la $2, datoa
        lh $2, 0($2)        # $2 = datoa
        la $3, datob
        lh $3, 0($3)        # $3 = datob

        #Operación de suma
        add $2, $2, $3      # $2 = datoa + datob

        #Escritura de resultados, desde registros a memoria
        la $3, datoc
        sh $2, 0($3)        # c = (datoa + datob)
    .end
```

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.

Solución posible:

```
        .data 0x10000000
datoa:  .half 9    # Se pueden inicializar con otros valores
datob:  .half 12
       .data 0x10001000
datoc:  .word 0

  .globl __start
```

```
        .text 0x00400000
__start:
    # Carga de datos desde memoria a registros
       la $2, datoa
       lh $2, 0($2)        # $2 = datoa
       la $3, datob
       lh $3, 0($3)        # $3 = datob

     # Operación de multiplicación
       mult $2, $3         # HI=0 y LO = datoa* datob
                           # porque datoa y datob ocupan 16 bits
       mflo $2             # $2 = LO = datoa * datob

     # Escritura de resultados, desde registros a memoria
       la $3, datoc
       sw $2, 0($3)        # datoc = (datoa*datob)
     .end
```

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"

Solución posible:

```
        .data 0x10000000
datoa: .word 12    # Estos valores pueden ser otros
datob: .word 6
cociente: .word 0
resto:    .word 0

   .globl __start
   .text 0x00400000
__start:
    # Carga de datos desde memoria a registros
       la $2, datoa
       lw $2, 0($2)        # $2 = datoa
       la $3, datob
       lw $3, 0($3)        # $3 = datob

    # Realizo la operación de división
       beq $3,$0,DivisionEntreCero
       div $2, $3          # HI=resto y LO = $2/$3

    # Almaceno el cociente en memoria
       mflo $2             # $2 = cociente
       la $3, cociente
```

```
  sw $2,0($3)

# Almaceno el resto en memoria
  mfhi $2            # $2 = resto
  la $3, resto
  sw $2,0($3)
.end
```

Prog 11)

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

resultado = 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.

Solución posible:

```
  .data 0x10000000
datoa: .word 9    # Se pueden inicializar con otros valores
datob: .word 2
resultado: .word 0
haydesbordamiento: .byte 0

  .globl __start
  .text 0x00400000
__start:
 #Carga de datos desde memoria a registros
  la $8, datoa
  lw $8, 0($8)       # $8 = datoa
  la $9, datob
  lw $9, 0($9)       # $9 = datob

# Cálculos aritméticos, acumulando en registro $2
  sub $2, $8, $9     # $2 = datoa-datob

# Escritura de resultado final desde registro a memoria
  la $7, resultado
  sw $2, 0($7)

# Comprobando si ha habido desbordamiento:
#  Sabemos que sumar dos valores enteros en complemento
#  a dos, si tienen el mismo signo podría desbordar
```

```
          #  la suma, pero si son de distinto signo no, porque uno
          #  resta al otro. La operación de resta es al revés.
            li $3,0x80000000   # Máscara de bit de signo
            and $10,$8,$3      # Filtro bit de signo en $10 de datoa
            and $11,$9,$3      # Filtro bit de signo en $11 de datob
            beq $10, $11, fin  # Si son iguales, al restarlos
                               # seguro que no hay desbordamiento
          # Si sigo por aquí, son distintos los signos a datoa y datob
          # Ahora tengo que comprobar si el signo del resultado es
          # distinto del signo del datoa. Si lo es hay desbordamiento.-
            and $12, $2,$3     # Filtro bit de signo en $12 de resultado
            beq $12, $10,fin   # Signos iguales, no hay desbordamiento
           #Si sigo por aquí es que hay desbordamiento
           la $3, haydesbordamiento
           li $2,1
           sw $2, 0($3)
           j salir

fin: # No hay desbordamiento
       la $3, haydesbordamiento
       sw $0, 0($3)

salir:.end
```

Prog 12)

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.

Posible solución:

```
            .data 0x10000000

tira:       .asciiz "ABC"
tira_res:   .space 3

            .globl __start
          .text 0x00400000

__start:  la $10,tira      # almacenamos en $10 la dirección tira
          la $11,tira_res  # almacenamos en $11 tira_res
bucle:    lb $12,0($10)    # carga en $12 el código ASCII del
                               carácter
          beq $12,$0,fin   # si es el carácter 0 termina el bucle
```

```
            addi $12,$12,32 # suma 32 para obtener minúscula
            sb $12,0($11)   # escribir el carácter en tira_res
            addi $10,$10,1  #apuntamos al siguiente carácter de tira
            addi $11,$11,1  #apuntamos al siguiente byte de tira_res
            j bucle
fin:
           .end
```

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?

Los formatos I y J, no tienen código de función, por lo que el total de instrucciones que se pueden tener son 63 instrucciones de tipo I y J (ya que el código 0 lo emplean las instrucciones de tipo R), y 64 de tipo R. Es decir 127 instrucciones.

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?

El código de función es de 6 bits, por lo que el número de funciones posibles es de 2^6 es decir 64 instrucciones

G 3)

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

El máximo desplazamiento es de 32 bits, es decir 2^5, ya que 5 es el tamaño de campo de desplazamiento. Como los registros son de 32 bits, no es necesario desplazar más

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?

El problema sería que los campos que definen el número de registro deberían tener 6 bits, como es necesario definir 3 registros, serían necesarios 18 bits para los registros, por lo que habría que cambiar el tamaño de algunos de los campos restantes, o hacer el tamaño de la instrucción más grande.

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?

Sólo afectaría a las instrucciones de desplazamiento, que deberían contemplar la posibilidad de desplazar 64 bits internos dentro de un registro.

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?

Serían necesarios 6 bits para almacenar el número de registro al que se hace referencia, lo que dejaría menos espacio para el dato inmediato.

G 7)

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

La distancia vendrá definida por el número de palabras que se pueden almacenar en el campo inmediato. Como es de 16 bits, y está en complemento a dos, los saltos pueden ir desde 32767 instrucciones más adelante hasta 32768 instrucciones más atrás.

G 8)

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

Complemento a dos facilita el cálculo de la dirección destino, y permite números enteros positivos y negativos, por lo que el salto puede ser hacia delante o hacia atrás

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?

Lo que se calcula es la dirección de la instrucción a la que se debe saltar, y las instrucciones están siempre almacenadas en direcciones múltiplo de cuatro. Si se codificara en bytes, simplemente se reduciría el alcance del salto.

G 10)

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

Teóricamente no se puede desplazar más de 32767 o -32768, aunque hay métodos que permiten desplazamientos mayores (probar con el PCSPIM)

G 11)

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

Porque el desplazamiento se usa para calcular una dirección de memoria de datos, y el tipo de datos byte, o ASCII permite estar alineado en cualquier dirección, es decir no se puede suponer que sea dirección múltiplo de dos o cuatro.

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?

No pasa nada, ya que las direcciones de memoria de instrucciones, que son las direcciones a las que se puede saltar, van desde la 0x00400000 a la 0x0FFFFFFF, en dicho rango, los cuatro primeros bits, siempre son

G 13)

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

Porque no precisa almacenar la dirección de memoria dentro de la instrucción, ya que la dirección de memoria a la que se debe saltar está en un registro.

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?

Codificar la dirección sería muy complicado, ya que solo se tienen 26 bits para almacenar la dirección de salto. Se debería buscar otra forma de codificar las instrucciones de salto