ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

1º Ingeniero en Informática

EJERCICIOS DE LA PRÁCTICA 1: REPRESENTACIÓN DE LA INFORMACIÓN (TIPOS DE DATOS)

Noviembre de 2008

EJERCICIO 1

Modificar el segmento de datos del programa "codifica.s" (ver anexo) para que contenga los siguientes datos:

a) Las variables **cad1** y **cad2** deben contener los nombres y apellidos respectivos de los dos componentes del grupo. La primera cadena terminará con un espacio en blanco y la segunda con el retorno de carro. Al igual que en el programa original, la primera cadena debe ser de tipo **.ascii** y la segunda de tipo **.asciiz**.

b) Sustituir los valores de los bytes a partir de la etiqueta **ent8** con los siguientes valores: Día de nacimiento del primer componente del grupo, el mismo valor pero negativo, día de nacimiento del segundo componente del grupo, el mismo día pero negativo, y el valor decimal 199. Por ejemplo, si el primer alumno nación un 21/02/1982 y el segundo un 12/10/1981, habrá que colocar la secuencia **21, -21, 12, -12, 199**.

c) Sustituir los valores de tipo **.half** con la siguiente secuencia de valores: Repetir los cinco valores anteriores, y a continuación poner esos mismos valores al cuadrado (pero conservando el signo negativo en cada caso). Para el mismo ejemplo anterior habría que colocar los valores **21, -21, 12, -12, 199, 441, -441, 144, -144, 39601**.

d) Sustituir los valores de tipo **.word** por la misma secuencia de valores que para el tipo **.half**, pero añadiendo al final dos nuevos valores: El producto del primer día de nacimiento por el segundo y por mil, y el mismo valor pero negativo. Para el caso anterior, sería la secuencia **21, -21, 12, -12, 441, -441, 144, -144, 39601, 252000, -252000**.

e) Sustituir los valores de tipo **.float** con el último valor obtenido en el apartado anterior (pero positivo), dividido entre el año de nacimiento del primer alumno (en el caso anterior, sería 252000/1982 = 127.1442987). Después agregar el mismo valor pero negativo, y, por último, añadir el 0. Es decir, la secuencia quedaría, para el ejemplo de antes, así: **127.1442987, -127.1442987, 0.0**.

f) Sustituir los valores de tipo **.double** por los mismos valores de la etiqueta **real32**.

EJERCICIO 2

Cargar el programa en PCSpim y ejecutarlo. Después observar y describir detalladamente el segmento de datos, explicando cómo y dónde se han codificado todos los datos en él contenidos.

Las explicaciones dadas deben ajustarse al siguiente modelo de tabla:

Dato	**Tipo**	**Dirección / Rango Dir.**	**Contenido**	**Comentario**
"Alumno 1 "	ascii	0x10010000	0x41	Código ascii de la "A" (0x41=65_{10})
		0x10010001	0x6C	Código ascii de la "l" (0x6C=108_{10})
		…	…	(avanzamos 7 posiciones más, ya que la cadena "Alumno 1 " consume 9 bytes (1 carácter = 1 byte)
		0x10010008	0x20	Código ascii el espacio en blanco (0x20=32)
(…)	(…)	(…)	(…)	(…)

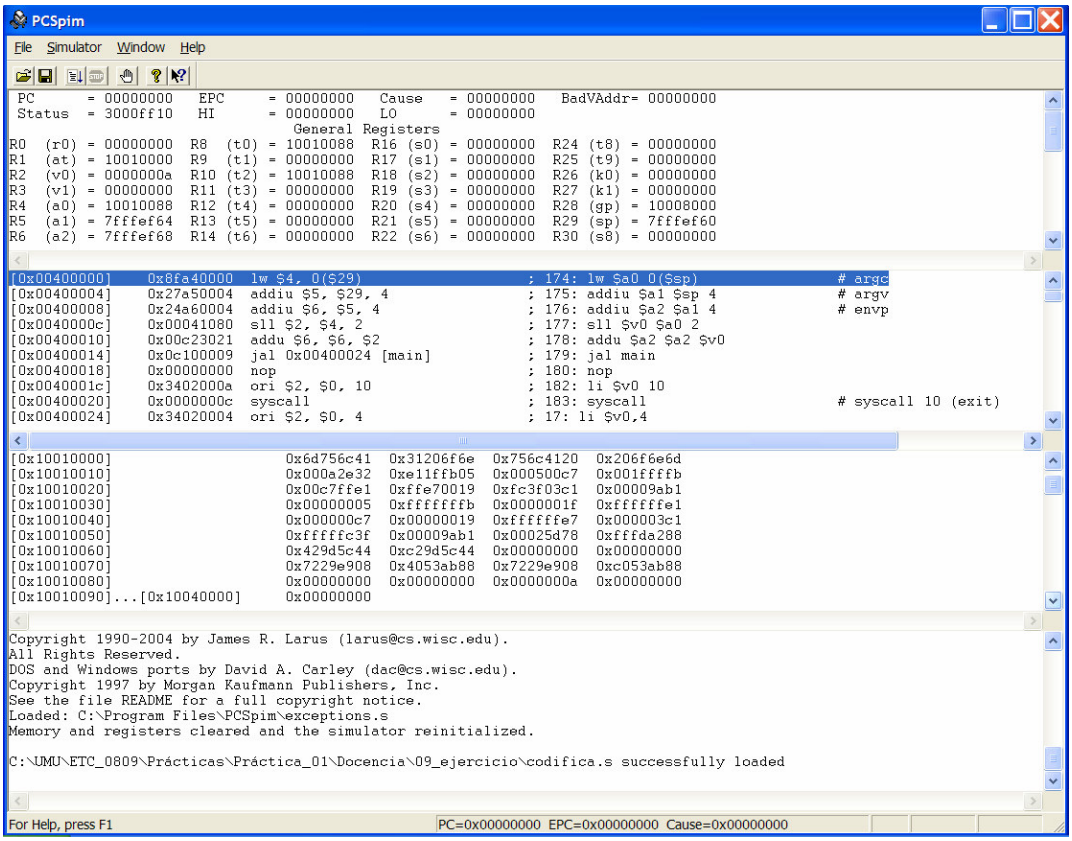
NOTA: No es necesario comentar los datos de la etiqueta real64.

SOLUCIÓN EJERCICIO 1

```
            .data

cad1:       .ascii  "Alumno 1 "
cad2:       .asciiz "Alumno 2.\n"
ent8:       .byte   5,-5,31,-31,199
ent16:      .half   5,-5,31,-31,199,25,-25,961,-961,39601
ent32:      .word   5,-5,31,-31,199,25,-25,961,-961,39601,155000,-155000
real32:     .float  78.6802030,-78.6802030,0.0
real64:     .double 78.6802030,-78.6802030,0.0
cr:         .asciiz "\n"

            .text

( … )
```

SOLUCIÓN EJERCICIO 2

- Volcado de pantalla de PCSpim:



- Explicación del segmento de datos:

Dato	Tipo	Dirección / Rango Dir.	Contenido	Comentario
"Alumno 1 "	ascii	0x10010000	0x41	Código ascii de la "A" (0x41=65_{10})
		0x10010001	0x6C	Código ascii de la "l" (0x6C=108_{10})
		…	…	(avanzamos 7 posiciones más, ya que la cadena "Alumno 1 " consume 9 bytes (1 carácter = 1 byte)
		0x10010008	0x20	Código ascii el espacio en blanco (0x20=32)
"Alumno 2.\n"	asciiz	0x10010009	0x41	Código ascii de la "A" (0x41=65_{10})
		…		(avanzamos 8 posiciones más, hasta el ".")
		0x10010011	0x2E	Código ascii del "." (0x2E=46_{10})
		0x10010012	0x0A	Código ascii del retorno de carro (0x0D=10_{10})
		0x10010013	0x00	Un dato del tipo "asiiz" siempre termina con el byte 0x00.
5	byte	0x10010014	0x05	
-5		0x10010015	0xFB	Bit más significativo (1): Signo -. Resto de bits (1111011): C_2 del valor absoluto 5.
31		0x10010016	0x1F	
-31		0x10010017	0xE1	
199		0x10010018	0xC7	El valor 199 se sale del rango de representación de los números de tipo "byte" (de -127 a +127). En este caso, 0xC7 es 199 en hexadedimal, pero al empezar dicho código por 1, el programa lo interpreta incorrectamente, como un número negativo (el -57).
5	half	0x1001001B – 0x10010019	0x000500	Primer byte (dirección 0x10010019): Relleno 0x00 necesario para permitir que nuestro "half" comience en una dirección par (alineamiento). Segundo byte (dirección 0x1001001A): Byte menos significativo del valor half a guardar (byte 0x05). Debe ser el byte menos significativo (y no el más significativo) porque el MIPS sigue el modo de almacenamiento "litte-endian". Tercer byte (dirección 0x1001001B): Byte más significativo del valor half a guardar (byte 0x00).
-5		0x1001001D – 0x1001001C	0xFFFB	El byte 0xFB es la codificación, en complemento a 2, del número -5 (ver antes el almacenamiento en memoria del -5 como byte). Ahora el -5 es un "half" (media palabra), lo que significa que hay que extender el bit de signo a las posiciones más signficativas.
31		0x1001001F – 0x1001001E	0x001F	
-31		0x10010021 – 0x10010020	0xFFE1	

199		0x10010023 – 0x10010022	0x00C7	En esta ocasión la codificación no se puede malinterpretar (al contrario, ver antes, de lo que sucedía cuando el valor 199 se intentaba codificar en un único byte)
25		0x10010025 – 0x10010024	0x0019	
-25		0x10010027 – 0x10010026	0xFFE7	El C_2 de 25 es, en binario, 00111. Si añadimos el bit de signo, la representación interna es 100111. Como son 16 bits, la extensión del bit de signo hace que la codificación definitiva sea 1111111111100111. Nótese que el primer bit de signo siempre debe estar justo a la izquierda del primero de los bits que resulte de obtener el complemento a 2 del valor absoluto del número a codificar.
961		0x10010029 – 0x10010028	0x03C1	
-961		0x1001002B – 0x1001002A	0xFC3F	
39601		0x1001002D – 0x1001002C	0x9AB1	Mismas circunstancias que en (2). Un tipo half admite valores entre $-(2^{15}-1)$ y $+2^{15}-1$ (32767). Por tanto, el programa interpretaría incorrectamente este codificación (como el valor negativo -25935).
5	word	0x1001002F – 0x1001002E	0x0000	El siguiente dato es un "word", lo que implica que debe comenzar en una dirección que sea múltiplo de 4. La siguiente dirección libre que es múltiplo de 4 es la 0x10010030. En consecuencia, los dos bytes que nos quedan hasta llegar a esa dirección (los que ocupan las posiciones 0x1001002E y 0x1001002F) deben rellenarse con ceros.
		0x10010033 – 0x10010030	0x00000005	
-5		0x10010037 – 0x10010034	0xFFFFFFFB	
31		0x1001003B – 0x10010038	0x0000001F	
-31		0x1001003F – 0x1001003C	0xFFFFFFE1	
199		0x10010043 – 0x10010040	0x000000C7	
25		0x10010047 – 0x10010044	0x00000019	
-25		0x1001004B – 0x10010048	0xFFFFFFE7	
961		0x1001004F – 0x1001004C	0x000003C1	
-961		0x10010053 – 0x10010050	0xFFFFFC3F	
39601		0x10010057 – 0x10010054	0x00009AB1	
155000		0x1001005B – 0x10010058	0x00025D78	Aquí no hay problemas con los límites de la representación. El mayor valor absoluto representable, 2147836487 ($2^{31}-1$), es muy superior al 155000.
-155000		0x1001005F – 0x1001005C	0xFFFDA288	
78.6802030	float	0x10010063 – 0x10010060	0x429D5C44	Representación en IEEE 754 de simple precisión (1 bit de signo, 8 bits para el exponente y 23 bits para la mantisa). No es necesario alinear los datos (no hay bytes de relleno antes de 0x10010060). (*)

-78.6802030		0x10010067 – 0x10010064	0xC29D5C44	
0.0		0x1001006B – 0x10010068	0x00000000	Representación (no normalizada) del 0.

(*) Codificación de 78.6802030:

- Parte entera (78):

$78 = 64 + 8 + 4 + 2$, luego $78_{(10} = 1001110_{(2}$

- Parte decimal (0.6802030):

Hemos de obtener, entre la parte entera y la parte decimal, un total de 23 bits para la mantisa; 7 de esos bits provienen de la parte entera, luego hemos de sacar 23-7=16 bits decimales. No obstante, al asumirse en la representación normalizada IEEE 754 de simple precisión la existencia de un 1 implícito a la izquierda del punto decimal, ganamos un bit más. Por tanto, hay que calcular 17 bits para la parte decimal.

$0.680203 \cdot 2 = 1.360406$ $0.360406 \cdot 2 = 0.720812$ $0.720812 \cdot 2 = 1.441624$
$0.441624 \cdot 2 = 0.883248$ $0.883248 \cdot 2 = 1.766496$ $0.766496 \cdot 2 = 1.532992$
$0.532992 \cdot 2 = 1.065984$ $0.065984 \cdot 2 = 0.131968$ $0.131968 \cdot 2 = 0.263936$
$0.263936 \cdot 2 = 0.527872$ $0.527872 \cdot 2 = 1.055744$ $0.055744 \cdot 2 = 0.111488$
$0.111488 \cdot 2 = 0.222976$ $0.222976 \cdot 2 = 0.445952$ $0.445952 \cdot 2 = 0.891904$
$0.891904 \cdot 2 = 1.783808$ $0.783808 \cdot 2 = 1.567616$

Y para el redondeo del último bit (regla del rendondeo al par):

$0.567616 \cdot 2 = 1.135232$ $0.135232 \cdot 2 = 0.270464$

La última secuencia, 10, representa la fracción 0.50. Por tanto, hay que obligar a que el número obtenido sea par (que acabe en 0). Como no es el caso (acaba en 1), hay que sumarle un 1 al bit decimal menos significativo.

Parte decimal: 10101110001000011
1
10101110001000100

Luego es: 0.10101110001000100

Así pues, $78.6802030_{(10} =$ $1001110.10101110001000100_{(2} =$
$1.00111010101110001000100 \cdot 2^6$ $_{(2}$

Bit de signo: 0, pues el valor a representar es positivo.
8 bits del exponente: 6 más el sesgo (127), o sea, $133_{(10} = 10000101_{(2}$
23 bits de mantisa: 00111010101110001000100

Valor codificado: 01000010100111010101110001000100

En hexadecimal: 0x429D5C44

ANEXO: codifica.s

```
            .data

cad1:       .ascii  "Primer mensaje... "
cad2:       .asciiz "Segundo mensaje.\n"
ent8:       .byte   70,250,-5,0,-128
ent16:      .half   -4000,4000,70,250,-5,128
ent32:      .word   10000000,-10000000,-4000,4000,70,250,-5,0,128
real32:     .float  3.14159,-1234.5678,123.4e-7
real64:     .double 3.14159,-1234.5678,123.4e-7
cr:         .asciiz "\n"

            .text

main:       li $v0,4
            la $a0,cad1
            syscall

            # Impresión del primer byte, el primer half y el
            # primer word, separados por un retorno de carro.

            li $v0,1
            lb $a0,ent8
            syscall
            jal imprime_cr      # El flujo de ejecución se desvía a la dirección
                                # "imprime_cr". Antes, se guarda en $ra el valor actual
                                # del contador de programa (registro PC) más 4 (PC
                                # apuntará a la dirección de la siguiente instrucción).
            li $v0,1
            lh $a0,ent16
            syscall
            jal imprime_cr      # Estamos REUTILIZANDO el código de "imprime_cr"
                                # (motivo de la existencia de los procedimientos).
            li $v0,1
            lw $a0,ent32
            syscall
            jal imprime_cr

            # Impresión de todos los float.

            la $t0,real32       # $t0 <- Dirección de inicio de los float.
            addi $t2,$t0,12     # $t2 <- Dirección final de los float, más 1.
buclefloat: li $v0,2            # Imprimir el dato apuntado por $t0.
            l.s $f12,0($t0)
            syscall
            jal imprime_cr      # Imprimir retorno de carro.
            addi $t0,$t0,4      # $t0 <- $t0 + 4
            bne $t0,$t2,buclefloat

            # Impresión de todos los double.

            la $t0,real64       # $t0 <- Dirección de inicio de los double.
            addi $t2,$t0,24     # $t2 <- Dirección final de los double, más 1.
bucledouble:li $v0,3            # Imprimir el dato apuntado por $t0.
            l.d $f12,0($t0)
            syscall
            jal imprime_cr      # Imprimir retorno de carro.
            addi $t0,$t0,8      # $t0 <- $t0 + 8
            bne $t0,$t2,bucledouble

            j final             # Salto incondicional a la dirección "final", para
                                # así eludir la ejecución de "imprime_cr".

imprime_cr: li $v0,4
            la $a0,cr
            syscall
            jr $ra              # Saltar a la dirección guardada en el registro $ra
                                # (en $ra está la dirección de la instrucción que
                                # sigue al "jal imprime_cr" que desvió la ejecución
                                # a este procedimiento).
final:      li $v0,10
            syscall
```