2.1 En la parejas de registros R5:R4 y R7:R6 se almacenan dos números de 32 bits que se quieren sumar (LSW en el registro par). Realizar un programa en ensamblador que haga la suma y la almacene en R5:R4. ¿Qué modificaciones hay que realizar al programa para que se acepten números con signo o sin signo?

SOLUCIÓN

```
            add.w  r6,r4          ;Sumar palabras de menor peso
            addc.w r7,r5          ;Sumar palabras de mayor peso
```

La solución anterior soporta números con y sin signo.

2.2 Realizar en ensamblador una subrutina que permita comparar dos números de 32 bits (con o sin signo) almacenados en los pares R5:R4 (fuente) y R7:R6 (destino). El resultado debe quedar en el SR y no se debe alterar ningún registro.

SOLUCIÓN

La comparación se realiza igual que una resta, pero sin almacenar el resultado. La solución a este problema sería muy sencilla si existiese una instrucción CMPC (CoMParar con Carry). Dado que no es así, habrá que proceder de otra forma: primero se comparan las palabras de mayor peso; si son distintas, se sale, quedando el resultado en el CCR; en cambio, si son iguales, el veredicto vendrá dado por la comparación de las palabras de menor peso.

```
cmp32       cmp.w  r5,r7          ;Comparar MSW
            jne    cmp32fin       ;Si son distintas, hemos terminado
            cmp.w  r4,r6          ;Si no, comparar las LSW
cmp32fin    ret
```

2.3 Escriba una subrutina en ensamblador (*hexa7seg*) que devuelva la representación en código de siete segmentos del nibble de menor peso de la entrada. Los requisitos son:

```
Entradas:
            R4: Dígito hexadecimal en nibble de menor peso
Salidas:
            R5.B: Código de siete segmentos.
Se conserva:
            Todo
```

SOLUCIÓN

La subrutina almacena los códigos de siete segmentos en una tabla en memoria.

```
;-------------------------------------------------------------------------------
; hexa7seg                                                              v1.0
;
; Obtiene el código de 7 segmentos del nibble de menor peso de la entrada.
; Entradas:
;   R4: Dígito hexadecimal en nibble de menor peso
; Salidas:
;   R5.B: Código de siete segmentos.
; Se conserva:
;   Todo
;-------------------------------------------------------------------------------
hexa7seg    push.w r4             ;Conservar R4
            and.w  #0x000F,r4     ;Dejar sólo el nibble menos significativo
            mov.b  Tab7Seg(r4),r5;Leer código
            pop.w  r4             ;Recuperar R4
            ret                   ;Vuelta al llamador
;-------------------------------------------------------------------------------
; Tab7seg                                                               v1.1
; Tabla de códigos de siete segmentos
; bit       76543210
; segmento -gfedcba
;--------------------Control de versiones
```

```
; La tabla no ha cambiado, sólo la descripción de la misma que era -abcdefg
;-------------------------------------------------------------------------
Tab7Seg     .BYTE  0b00111111    ;0
            .BYTE  0b00000110    ;1
            .BYTE  0b01011011    ;2
            .BYTE  0b01001111    ;3
            .BYTE  0b01100110    ;4
            .BYTE  0b01101101    ;5
            .BYTE  0b01111100    ;6
            .BYTE  0b00000111    ;7
            .BYTE  0b01111111    ;8
            .BYTE  0b01100111    ;9
            .BYTE  0b01110111    ;A
            .BYTE  0b00011111    ;B
            .BYTE  0b01001110    ;C
            .BYTE  0b00111101    ;D
            .BYTE  0b01001111    ;E
            .BYTE  0b01000111    ;F
```

Versión mejorada que evita usar la pila para no modificar R4.

```
;-------------------------------------------------------------------------
; hexa7seg                                                          v1.1
;
; Obtiene el código de 7 segmentos del nibble de menor peso de la entrada.
; Entradas:
;   R4: Dígito hexadecimal en nibble de menor peso
; Salidas:
;   R5.B: Código de siete segmentos.
; Se conserva:
;   Todo
;-------------------------------------------------------------------------
hexa7seg    mov.w  r4, r5         ;Conservar R4
            and.w  #0x000F,r5     ;Dejar sólo el nibble menos significativo
            mov.b  Tab7Seg(r5),r5;Leer código
            ret
```

2.4 Desde un programa en ensamblador se desea llamar a la función `printf` de C. A dicha subrutina se le pasan los siguientes parámetros:

```
printf ("Total de la cuenta: %d € (%d ptas)\n", euros, pesetas);
```

Ambas variables son de tipo `int` y se encuentran en las posiciones de memoria $1000 y $2000 respectivamente. Genere el *trozo de código* necesario para realizar la llamada desde código ensamblador del M68000.

SOLUCIÓN

```
euros       .EQU   0x1000
pesetas     .EQU   0x2000

Texto       .CSTRING"Total de la cuenta: %d € (%d ptas)\n"

            mov.w  #Texto, r12    ;Primer parámetro  (cadena de formato) a R12
            mov.w  &euros, r13    ;Segundo parámetro a R13
            mov.w  &pesetas, r14 ;Último parámetro a R14
            call   #printf        ;Imprimir
```

2.5 Escriba la función *strlen* que calcula la longitud de una cadena de caracteres tipo C (terminadas con un carácter 0). La subrutina usa el convenio de llamada de C y atiende al prototipo siguiente:

```
int strlen (const char *s);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------
; int strlen (const char *s)                                        v1.0
;
; Calcula la longitud de una cadena.
;-------------------------------------------------------------------------
strlen      mov.w  r12, r13      ;1w1c, R13=s
strlenb     inc.w  r12           ;1w1c, avanzar puntero
            tst.b  -1(r12)       ;2w4c, es el final de la cadena?
            jnz    strlenb       ;1w2c, ...no, seguir leyecndo cadena
            sub.w  r13, r12      ;1w1c, Calcular tamaño restando punteros
            dec.w  r12           ;1w1c, El terminador de cadena no se contabiliza
            ret                  ;1w3c, Total 8w, [7(N+1)+6]c
```

La siguiente versión usa un registro adicional (R14), pero ocupa 1 palabra menos de código y el bucle es 2 ciclos más rápido (5 frente a 7) gracias a que usa el direccionamiento indirecto con postincremento (disponible sólo en el operando fuente) en lugar del indexado. Además, el código es más claro y directo:

```
;-------------------------------------------------------------------------
; int strlen (const char *s)                                        v2.0
;
; Calcula la longitud de una cadena.
;-------------------------------------------------------------------------
strlen      mov.w  r12, r13      ;1w1c, R13=s
strlenb     mov.b  @r12+, r14    ;1w2c, Leer carácter y avanzar puntero
            tst.b  r14           ;1w1c, es el final de la cadena?
            jnz    strlenb       ;1w2c, ...no, seguir leyecndo cadena
            sub.w  r13, r12      ;1w1c, Calcular tamaño restando punteros
            dec.w  r12           ;1w1c, El terminador de cadena no se contabiliza
            ret                  ;1w3c, Total 7w, [5(N+1)+6]c
```

2.6 Escriba la función *strcpy* que copia una cadena de caracteres de tipo C (terminadas con un carácter 0) apuntada por s1 a otra dirección. La subrutina usa el convenio de llamada de C y atiende al prototipo siguiente:

```
void strcpy (const char *s1, char *s2);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------
; void strcpy (const char *s1, char s2);                            v1.0
;
; Copia una cadena en otro sitio.
;-------------------------------------------------------------------------
strcpy      inc.w  r13           ;1w1c Avanzar s2
            tst.b  0(r12)        ;2w4c Fin de cadena?
            mov.b  @r12+, -1(r13);2w5c Copia carácter
            jnz    strcpy        ;1w2c El carácter 0 termina el bucle
            ret                  ;Total: 6w12Nc

;-------------------------------------------------------------------------
; void strcpy (const char *s1, char s2);                            v2.0
;
; Copia una cadena en otro sitio.
;-------------------------------------------------------------------------
strcpy      mov.b  @r12+, r14    ;1w2c Lee carácter y avanza puntero fuente
            mov.b  r14, 0(r13)   ;2w4c Copia carácter en destino
            inc.w  r13           ;1w1c Avanza puntero destino
            tst.b  r14           ;1w1c Era fin de cadena?
            jnz    strcpy        ;1w2c ...no, siguiente carácter
            ret                  ;Total: 6w10Nc
```

2.7 Escriba la función *strcat* que añade la cadena de caracteres *s2* a otra cadena *s1*. La subrutina usa el convenio de llamada de C y atiende al prototipo siguiente:

```
void strcat (const char *s1, char *s2);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------
; void strcat (const char *s1, char *s2);                          v1.0
;
; Añade la cadena s2 al final de s1.
;-------------------------------------------------------------------------
strcat     ;Buscar el final de s1
strcatBuc1 inc.w  r12
           tst.b  -1(r12)       ;Es el final del cadena?
           jnz    strcatBuc1    ;...no. Seguir buscando

           ;Copiar s2 al final de s1. Ojo r12 apunta después de final cadena
strcatBuc2 inc.w  r12           ;Avanzar s1
           mov.b  @r13+, -2(r12);Copia carácter. Ojo al -2 del destino
           tst.b  -1(r13)       ;Se ha coopiado el fin de cadena de s2?
           jnz    strcatBuc2    ;...no. Otro carácter más
           ret

;-------------------------------------------------------------------------
; void strcat (const char *s1, char *s2);                          v2.0
;
; Añade la cadena s2 al final de s1.
;-------------------------------------------------------------------------
strcat     ;Buscar el final de s1
           mov.b  @r12+, r14    ;Leer carácter y avanzar puntero
           tst.b  r14           ;es el final de la cadena?
           jnz    strcat        ;...no. Seguir buscando

           ;Copiar s2 al final de s1. Ojo r12 apunta después de final cadena
strcatBuc  mov.b  @r13+, r14    ;Lee carácter de s2 y avanzar puntero
           mov.b  r14, -1(r12)  ;Copia carácter en destino
           inc.w  r12           ;Avanza puntero destino
           tst.b  r14           ;Fin de cadena?
           jnz    strcatBuc     ;...no. Otro carácter más
           ret
```

2.8 Escriba la función *strchr* que busca el carácter *c* en la cadena de caracteres *s*. Devuelve la dirección de la primera ocurrencia o *NULL* (0) si no encontró el caracter. La subrutina usa el convenio de llamada de C y atiende al prototipo siguiente:

```
char *strchr (const char *s, char c);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------
; char *strchr (const char *s, char c);                            v1.0
;
; Busca un carácter en una cadena. Devuelve la dirección de la primera
; ocurrencia o NULL en caso contrario.
;-------------------------------------------------------------------------
strchr     mov.b  @r12+, r14    ;Leer un carácter
           tst.b  r14           ;Fin de cadena?
           jz     strchrErr     ;...sí, salir
           cmp.b  r13, r14      :...no. Es el carácter buscado?
           jne    strchr        ;...no, siguiente carácter
           dec.w  r12           ;...sí, devolver dirección del carácter
           jmp    strchrFin
strchrErr  clr.w  r12           ;Devolver NULL por no encontrado
strchrFin  ret
```

2.9 La subrutina *wc* sigue el convenio de llamada de C, recibe un puntero a una cadena de caracteres `s` y devuelve una palabra larga que contiene, en la palabra menos significativa, el número de letras de la cadena, y en la más significativa, el número de palabras. Las cadenas sólo podrán contener letras y espacios. No se tendrá en cuenta el caso de que haya caracteres de control (Escape, Intro, etc.), dígitos y otros caracteres imprimibles.

```
unsigned long wc (const char *s);
```

Ejemplo: `wc (" El  perro de San Roque ya no tiene rabo ")`, devuelve `$0009001E`, esto es, 9 palabras y 30 letras.

a) Explique con palabras el algoritmo que va a emplear.
b) Escriba la subrutina en ensamblador del M68000.

SOLUCIÓN

Contar el número de letras de una cadena de caracteres en las que sólo hay letras y espacios en blanco, es sencillo: basta hacer un bucle que se pare al final de la cadena y recorra los caracteres uno a uno. Si el carácter es distinto de blanco, se incrementa un contador inicializado a 0 antes del bucle (`R14` en nuestro caso). El contaje de palabras es un poco más difícil. Para hacerlo se dispondrá de una variable temporal (en nuestro caso `R15`), que hará las veces de bandera. Por defecto la bandera vale cero y se pone a 1 cada vez que se crea que estamos en una palabra, es decir, cuando encontramos una letra. La variable en la que se van contando las palabras (`R13` en nuestro caso) se incrementará cada vez que encontremos un terminador de palabra, que puede ser un blanco o el final de cadena.

En la implementación siguiente, se hace un bucle que recorre los caracteres de `s`. Para cada carácter, si es no blanco, se incrementa `R14` y se hace `R15=1`. Si es blanco, se suma `R15` a `R13` (contabilizando la posible palabra) y borrando `R15` (indicando que no hay palabra en curso).

Hay que tener cuidado de contabilizar la última palabra cuando se encuentra el final de cadena.

```
;-------------------------------------------------------------------------------
; unsigned long wc (const char *s);                                      v2.0
;
; Cuenta el número de letras y de palabras de s. Devuelve una palabra larga
; compuesta por el número de letras (palabra menos significativa) y el de
; palabras (palabra más significativa). Se supone que sólo hay letras y
; blancos.
;-------------------------------------------------------------------------------
wc          clr.w  r13           ;r13: contador de palabras
            clr.w  r14           ;r14: contador de letras
            clr.w  r15           ;Si r15=0, no hay palabra empezada
wcBuc       mov.b  @r12+, r11    ;Leer carácter
            tst.b  r11           ;¿Fin de cadena?
            jeq    wcFin         ;...sí, salir
            cmp.b  #' ', r11     ;¿es espacio?
            jeq    wcEspacio     ;...sí, posible nueva palabra
            inc.w  r14           ;...no, incrementar contador de letras
            mov.w  #1, r15       ;  e indicar que hay una palabra empezada
            jmp    wcBuc
wcEspacio   add.w  r15, r13      ;Contabilizar palabra
            clr.w  r15           ;Desactivar flag
            jmp    wcBuc
wcFin       add.w  r15, r13      ;Contabilizar palabra
            mov.w  r14, r12      ;Número de letras en LSW
            ret
```

2.10 Realice una subrutina llamada *Maximo* que busca la palabra mayor del vector de enteros sin signo apuntado por R12 y que tiene una longitud de R13 elementos. El resultado debe devolverse en R12. No altere ningún registro.

SOLUCIÓN

```
;-------------------------------------------------------------------------
; Maximo                                                           v1.3
;
; Encuentra el máximo de un vector de enteros sin signo (16 bits)
; Entradas: R12: Dirección del vector
;           R13: Número de elementos
; Salidas:  R12: Máximo
; Se conserva: Todo
; Versiones:
; v1.2 (2/11/21). Corregido mov.w -1(r12), r14 por mov.w -2(r12), r14
; v1.3 (7/10/25). Añadido jlo MAxFin para salir si R13=0
;-------------------------------------------------------------------------
Maximo     push.w r14           ;Conservar registros
           push.w r13
           mov.w  @r12+, r14    ;Tomar primer elemento como máximo inicial
           dec.w  r13           ;Primer elemento leído. Descontar
           jz     MaximoFin     ;Había 1 elemento? Sí, salir
           jlo    MaximoFin     ;Había 0 elementos? Sí, salir
MaximoBuc  cmp.w  @r12+, r14    ;Siguiente elemento, es mayor?
           jhs    NoCambiar     ;...no, siguiente elemento
           mov.w  -2(r12), r14  ;...sí, nuevo máximo temporal
NoCambiar  dec.w  r13
           jnz    MaximoBuc
MaximoFin  mov.w  r14, r12      ;R12 = Máximo
           pop.w  r13           ;Recuperar registros
           pop.w  r14
           ret
```

2.11 Escriba una subrutina en ensamblador que siga el convenio de llamada de C y que busque un elemento en un vector *p* ordenado de *n* palabras. Devuelve un puntero a la primera ocurrencia o NULL si no se encuentra o la lista está vacía.

```
int *BuscaBin (int elem, int *p, int n);
```

SOLUCIÓN

Dado que el vector se encuentra ordenado, es más eficiente hacer una búsqueda binaria que una búsqueda secuencial desde el principio. La búsqueda binaria consiste en tomar el elemento central del vector y compararlo con el elemento buscado. Si coinciden, se ha terminado la búsqueda. En caso contrario, si el elemento central es menor que el buscado, se deshecha la primera mitad del vector y se repite el proceso con la segunda. Finalmente, si el elemento central fuera mayor que el buscado, se desharía la segunda mitad del vector y se repetiría el proceso sólo en la primera mitad.

En el algoritmo se trabajará con tres índices: `i` será el índice al elemento inicial del vector (inicialmente `i=0`; en el código `R15`); `f` será el índice al elemento final del vector (inicialmente `f=0`; en el código `R14`). El elemento central `c` (`R11` en el código) se calcula de forma simple como `c=(i+f)/2`. El algoritmo tiene dos condiciones de terminación: que se encuentre el elemento, ya discutido y que no se encuentre. Este extremo se dará cuando se crucen los índices `i` y `c`, es decir, cuando `i>c`.

Es interesante también notar que para acceder a un elemento del vector hay que convertir los índices (`IND`) en desplazamientos (`DES`) dentro del vector. Puesto que los elementos son de tamaño palabra, sabemos que `DES = IND * 2`.

```
;-------------------------------------------------------------------------
; int *BuscaBin (int elem, int *p, int n);                         v1.0
;
```

```
; Busca elem en el vector ordenado p de n elementos
;-------------------------------------------------------------------------------
BuscaBin    push.w r10
            dec.w  r14           ;R14 = f = n-1
            jn     BuscaBinNoE   ;Si f<0, vector vacío. Salir con error
            clr.w  r15           ;R15 = i = 0
BuscaBinBuccmp.w  r15, r14       ;Comparar índices
            jlo    BuscaBinNoE   ;R14<R15? se han cruzado, no encontrado
            mov.w  r15, r11      ;R11 = i
            add.w  r14, r11      ;R11 = i + f
            rra.w  r11           ;R11 = c = (i+f)/2
            mov.w  r11, r10      ;R10 = c
            rla.w  r10           ;R10 = desplazamiento del índice c
            add.w  r13, r10      ;R10 = dirección del elemento central
            cmp.w  @r10, r12     ;Comparar elem con central
            jeq    BuscaBinEnc   ;Iguales? Encontrado
            jlo    BuscaBinMen   ;Menor? recortar vector
                                 ;...no, mayor, recortar vector
            ;Mayor, recortar vector
            mov.w  r11, r15      ;R15 = i = c
            inc.w  r15           ;R15 = i = c+1
            jmp    BuscaBinBuc   ;Seguir buscando

            ;Menor, recortar vector
BuscaBinMenmov.w  r11, r14       ;R14 = f = c
            dec.w  r14           ;R14 = f = c-1
            jmp    BuscaBinBuc   ;Seguir buscando

            ;Iguales, encontrado. Devolver dirección
BuscaBinEncmov.w  r10, r12       ;Se devuelva la dirección del elemento
            jmp    BuscaBinFin   ;Salir

            ;No encontrado. Devolver NULL
BuscaBinNoEclr.w  r12            ;Devolver NULL
BuscaBinFinpop.w  r10
            ret
```

2.12 Escriba una subrutina en ensamblador que siga el convenio de llamada de C y que clasifique ascendentemente un vector de bytes mediante el algoritmo de la burbuja.

```
void Burbuja (char *p, int n);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------------
; void Burbuja (char *p, int n);                                        v1.0
;
; Clasifica ascendentemente el vector p de n elementos usando algoritmo
; de la burbuja.
;-------------------------------------------------------------------------------
Burbuja     dec.w  r13           ;R13 = n-1 pares de elementos
            jlo    BurbFin       ;Menos de 1 elemento? Salir
            push.w r10           ;Registro auxiliar para intercambios
BurbIni     mov.w  r12, r14      ;R14=p. Conservar R12
            mov.w  r13, r15      ;R15=n-1. Conservar R13
            clr.w  r11           ;R11=0 es flag de no modificado
BurbBuc     cmp.b  @r14+, 0(r14) ;Elementos bien ordenados? (Ojo direccionamiento)
            jge    BurbNoCamb    ;...sí. No hacer nada
            dec.w  r14           ;...no. Intercambiar en memoria
            mov.b  @r14+, r10
            mov.b  @r14, -1(r14)
            mov.b  r10, 0(r14)
            mov.w  #1, r11       ;R11=1 indica que hay cambios
BurbNoCamb  dec.w  r15           ;Un par menos por procesar
            jnz    BurbBuc
```

```
            tst.w  r11          ;Hay cambios?
            jnz    BurbIni      ;...sí. Hacer otra pasada
            pop.w  r10          ;...no. Vector ordenado
BurbFin     ret
```

2.13 (23/24 Conv1) La subrutina *bcd2bin* usa el convenio de llamada de C y atiende al siguiente prototipo:

```
long int bcd2bin (unsigned long int bcd);
```

bcd2bin convierte el número que se pasa en formato BCD empaquetado de 32 bits (un dígito binario cada cuatro bits), a binario natural (32 bits). En caso de error devuelve -1. Ejemplos de entradas y su salida:

- BCD: 0000 0000 0000 0000 0000 0000 1001 0101 (95 en BCD)
  Bin: 0000 0000 0000 0000 0000 0000 0101 1111 (95 en binario)
- BCD: 0000 0000 0000 0000 0000 0000 1011 0000
  Bin: 1111 1111 1111 1111 1111 1111 1111 1111 (Error).

a) Explique con palabras el algoritmo que va a emplear.
b) Escriba la subrutina en ensamblador.

SOLUCIÓN

a) El siguiente algoritmo procesa los dígitos BCD en orden de peso decreciente y los va acumulando en el par R_{13}:R_{12}. Si el BCD es $x_7x_6x_5x_4x_3x_2x_1x_0$ se implementa la fórmula:

$$n = ((((((0 \cdot 10 + x_7) \cdot 10 + x_6) \cdot 10 + x_5) \cdot 10 + x_4) \cdot 10 + x_3) \cdot 10 + x_2) \cdot 10 + x_1) \cdot 10 + x_0$$

Es decir, se entra en un bucle que, comenzando por el BCD más significativo, se multiplica el acumulado por 10 y se suma un nuevo dígito.

La multiplicación por 10 se hace con un sencillo algoritmo a base de desplazamientos y sumas.

Para extraer los dígitos BCD se colocan en el par R_{15}:R_{14} y se van rotando hacia la izquierda de 4 en 4 bits para colocar cada dígito BCD en los 4 lsb. Cada dígito se extrae con una operación AND.

b)

```
;-------------------------------------------------------------------------
; long int bcd2bin (unsigned long int bcd)                          v1.1
;
; Convierte el número que se pasa en formato bcd empaquetado a binario natural.
; En caso de error (algún dígito BCD ilegal), devuelve -1.
; -----Control de versiones-------
; v1.1 Optimizada la rotación de 4 bits a la izquierda de R15:R14
;-------------------------------------------------------------------------
bcd2bin     ;Salvar registros
            push   r9
            push   r10

            mov.w  r12, r14
            mov.w  r13, r15      ;R15:R14 = bcd
            clr.w  r12
            clr.w  r13           :R13:R12 = acumulador = 0

            mov.w  #8, r9        ;R9 contador de dígitos BCD
            jmp    bcd2binPV     ;Saltar primera multiplicación por 10

            ;Acumulador *= 10
bcd2binBuc2 rla.w  r12
            rlc.w  r13           ;R13:R12 = 2X
            mov.w  r12, r10
```

```
            mov.w  r13, r11      ;R11:R10 = acumulador = 2X
            rla.w  r10
            rlc.w  r11           ;R11:R10 = 4X
            rla.w  r10
            rlc.w  r11           ;R11:R10 = 8X
            add.w  r10, r12
            addc.w r11, r13      ;R13:R12 = 8X + 2X = 10X

            ;Obtener dígitos BCD por la izquierda y colocarlos en los 4LSB
bcd2binPV   mov.b  #4, r11       ;R11 contador de bits
bcd2binBuc1 bit.w  #BITF, r15    ;Copiar msb en carry
            rlc.w  r14           ;Desplazar R14 a la izq 1 bit metiendo C por dcha
            rlc.w  r15           ;Desplazar R15 a la izq 1 bit metiendo C por dcha
            dec.b  r11           ;Otra vuelta más
            jnz    bcd2binBuc1

            ;Poner a 0 todo menos los 4 LSB
            mov.w  r14, r11      ;R11 = bcd rotado (sólo palabra baja)
            and.w  #0xF, r11     ;Dejar sólo los 4 LSB
            cmp.b  #10, r11      ;Es mayor que 9?
            jhs    bcd2binErr    ;...sí, salir con error

            ;Acumular dígito. Acumulador = 10*acumulador + dígito
            add.w  r11, r12      ;Sumar dígito
            adc.w  r13           ;Sumar posible acarreo

            ;Procesar otro dígito
            dec.w  r9
            jnz    bcd2binBuc2

            jmp    bcd2binFin    ;Terminado. Salir

bcd2binErr  ;Se ha encontrado un dígito BCD erróneo. Salir con -1
            mov.w  #-1, r12
            mov.w  #-1, r13      R13:R12 = -1 (Error)

            ;Recuperar registros
bcd2binFin  pop    r10
            pop    r9
            ret
```

2.14 Escriba en ensamblador la subrutina *bin2bcd* que usa el convenio de llamada de C y que atiende al siguiente prototipo:

```
long int bin2bcd (unsigned int bin);
```

bin2bcd convierte el número que se pasa en formato binario natural (sin signo), a bcd empaquetado.

SOLUCIÓN

Para convertir un número binario a BCD (o decimal, es lo mismo), hay que hacer divisiones sucesivas por 10. En cada división se obtiene un resto, que será un dígito BCD del resultado, y un cociente, que se empleará en posteriores divisiones. El algoritmo termina cuando el cociente se hace cero. El resultado está compuesto por los sucesivos restos, tomados en orden inverso (esto es, el primero que sale es el menos significativo). Veamos un ejemplo:

12345	10	
5	1234	10

4	123	10		
	3	12	10	
		2	1	10
			1	0

Sin embargo, una forma más eficiente de hacer la tarea consiste en el *algoritmo de Horner*, que se basa en la definición de representación posicional de cantidades. Se inicializa una variable acumuladora a 0. Los bits del número en binario natural se extraen uno a uno empezando por el más significativo. Se incorporan al acumulador multiplicando por 2 el mismo y sumando dicho bit. Dicho proceso se hará usando aritmética decimal, por lo que, después de 16 iteraciones, tendremos el número convertido. Aunque el bucle dé 16 pasadas, éstas sólo constan de un desplazamiento y una suma con acarreo, por lo que su eficiencia es mucho mayor que la del algoritmo anterior.

```
;-------------------------------------------------------------------------
; long int bin2bcd (unsigned int bin)                                v1.0
;
; Convierte el número que se pasa en formato binario natural, a bcd
; empaquetado.
;-------------------------------------------------------------------------
bin2bcd    mov.w  r12, r14      ;Entrada a R14. Dejar libre salida en R12
           clr.w  r12           ;Inicializar resultado R13:R12(32 bits)
           clr.w  r13
           mov.w  #16, r15      ;R15 variable control bucle (16 vueltas)
bin2bcdBuc rla.w  r14           ;msb a carry
           dadd.w r12, r12      ;R13:R12 = 2*R13:R12 + C (suma BCD)
           dadd.w r13, r13
           dec.w  r15           ;Un bit menos por procesar
           jnz    bin2bcdBuc    ;Siguiente bit
           ret
```

2.15 Se desea realizar una subrutina que permita calcular el producto de dos números de 32 bits sin signo. Se usará el algoritmo tradicional de multiplicación, como en el ejemplo:

			1	1	0	1	
		x		1	0	1	
			1	1	0	1	
		0	0	0	0		
	1	1	0	1			
1	0	0	0	0	0	1	

La subrutina se llamará mul32x32 y seguirá el siguiente prototipo:

```
unsigned long long mul32x32 (unsigned long a, unsigned long b);
```

SOLUCIÓN

```
;-------------------------------------------------------------------------
; unsigned long long mul32x32 (unsigned long a, unsigned long b);       v1.1
;
; Multiplica dos números sin signo de 32 bits y deja un resultado de 64.
; Control de versiones.
; v1.1 Añadido código para detectar de forma temprana que R9:R8 se hace 0 para
; salir y ser más eficiente.
;-------------------------------------------------------------------------
mul32x32   push.w r4
```

```
            push.w r5
            push.w r6
            push.w r7
            push.w r8
            push.w r9
            mov.w  r12, r4       ;R5:R4 = A
            mov.w  r13, r5
            clr.w  R6            ;R7:R4 = A (64 bits)
            clr.w  R7
            mov.w  r14, r8       ;R9:R8 = B
            mov.w  r15, r9
            clr.w  r12           ;R15:R12 = 0 (resultado)
            clr.w  r13
            clr.w  r14
            clr.w  r15
            mov.w  #32, r11      ;Procesar 32 bits
mul32Buc    ;Desplazar un bit a la derecha B (R9:R8)
            rra.w  r9
            rrc.w  r8
            jnc    mul32NoSum    ;Si el bit saliente es 0, no acumular
            ;Sumar A desplazado al acumulador R15:R12 += R7:R4
            add.w  r4, r12
            addc.w r5, r13
            addc.w r6, r14
            addc.w r7, r15
mul32NoSum ;Desplazar A (R7:R4) un bit a la izquierda
            rla.w  r4
            rlc.w  r5
            rlc.w  r6
            rlc.w  r7
            tst.w  r8            ;R8!=0?
            jnz    mul32Sig      ;...sí, seguir procesando
            tst.w  r9            ;...no. R8=0. R9 también?
            jz     mul32Fin      ;...sí, hemos terminado
mul32Sig    dec.w  r11           ;...no. Siguiente bit. Quedan?
            jnz    mul32Buc      ;...sí, procesar
mul32Fin    pop.w  r9
            pop.w  r8
            pop.w  r7
            pop.w  r6
            pop.w  r5
            pop.w  r4
            ret
```

2.16 La subrutina `traspone` acepta un puntero a una tabla cuadrada de enteros (`m`) de `nxn` elementos.

```
void *traspone (int *m, int n);
```

La función realiza la trasposición de la tabla (cambia filas por columnas y viceversa) y la deja en el mismo sitio (ver ejemplo, aunque tenga en cuenta que la dimensión no tiene por qué ser de 4x4).

Antes

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Después

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

a) Explique con palabras el algoritmo que va a emplear.
b) Escriba la subrutina en ensamblador del MSP430.

SOLUCIÓN

El proceso de trasposición consiste en intercambiar los elementos que están sobre la diagonal principal con los que están por debajo. Es decir $M_{ij} \leftrightarrow M_{ji}$, donde $i \neq j$. En C sería una cosa como la siguiente:

```
for (i = 0; i <= n-2; i++)
  for (j = i+1; j < n; j++)
  {
    temp = M[i][j];
    M[i][j] = M[j][i];
    M[j][i] = temp;
  }
```

En ensamblador resulta más eficiente hacer un acceso secuencial a las semifilas y semicolumnas. En el siguiente código se guarda en `R12` la dirección de las semifilas y en `R11` el de las semicolumnas.

Para encontrar la dirección del siguiente elemento de una semifila, basta con sumar 2 (el tamaño del elemento) al puntero (`R12`), o aprovechar el direccionamiento indirecto con postincremento. Para hacer lo mismo con la semicolumna, hay que sumar `2n` al puntero (`R11`). Como `2n` se usa mucho, se guarda en un registro (`R13`).

Dada la dirección de una semifila, para encontrar la de la siguiente, hay que sumar `2*(n+1)`, mientras que para encontrar la de la semicolumna, hay que sumar `2*(n-1)`.

El resto de registros usados es `R15`, que es la variable de control del bucle externo (recorre desde `n-1` hasta 0) y `R14` que es la variable de control del bucle interno (recorre desde `R15` hasta 0).

```
;-------------------------------------------------------------------------
; void *traspone (int *m, int n);                                    v1.0
;
; Realiza la trasposición de la tabla (cambia filas por columnas y viceversa)
; y la deja en el mismo sitio.
;-------------------------------------------------------------------------
traspone   push.w r10           ;R10 es registro temporal de intercambio
           mov.w  r13, r15
           dec.w  r15           ;R15=n-1 (tamaño primera semifila)
           jz     trasFin       ;n=1, luego no hay nada que hacer
           jlo    trasFin       ;n<1, luego error
           rla.w  r13           ;R13=2*n
           incd.w r12           ;R12 apunta a principio primera semifila

trasBuc1   ;Este bucle intercambia los elementos que están por encima de la
           ;diagonal principal, con los que están por debajo

           ;Apuntar R11 al principio de semicolumna
           mov.w  r12, r11      ;R11 apunta a principio de semifila
           add.w  r13, r11
           decd.w r11           ;R11 apunta a principio de semicolumna

           push.w r12           ;Guardar dir de semifila
           mov.w  r15, r14      ;R14=contador de elementos de semifila
trasBuc2   ;Este bucle intercambia los elementos de una semifila con los
           ;de su semicolumna equivalente

           ;Intercambiar un elemento de semifila con semicolumna en memoria
           mov.w  @r12+, r10
           mov.w  @r11, -2(r12)
           mov.w  r10, 0(r11)

           add.w  r13, r11      ;Incrementar dirección de semicolumna
```

```
            dec.w  r14          ;Siguiente elemento en semifila
            jnz    trasBuc2

            ;Encontrar dirección de siguiente semifila
            pop.w  r12          ;Recuperar dirección de semifila
            add.w  r13, r12
            incd.w r12          ;R12 apunta principio de sig semifila

            dec.w  r15          ;Siguiente semifila
            jnz    trasBuc1
            pop.w  R10
            ret
```

2.17 Se desea realizar una subrutina `int suma(char *sf, char *sd, int n)` que permita calcular la suma de dos números enteros sin signo de `n` bytes. El primer sumando se encuentra almacenado a partir de la dirección `sf` y el segundo a partir de la dirección `sd` (formato normal de MSP430, *little endian*). El resultado se dejará almacenado en el mismo sitio que el segundo sumando. La subrutina sigue el convenio de llamada de C y devuelve un código de estado: -1 en caso de error (desbordamiento o n<1); 0 si todo fue bien.

a) Explique con palabras el algoritmo.
b) Escriba la subrutina en ensamblador del MSP430.

SOLUCIÓN

La solución consiste en hacer un bucle que dé n iteraciones y vaya sumando los bytes de cada vector empezando por el menos significativo. La complicación viene del hecho de que en el bucle hay que incrementar el puntero destino y decrementar el contador de bytes. Ambos detruyen el bit de carry que habría que preservar para la suma de la siguiente iteración. Para evitar su destrucción, se usará un registro (R15) que lo guardará temporalmente.

```
;-------------------------------------------------------------------------
; int suma(char *sf, char *sd, int n);                                 v1.1
;
; Suma dos números de n bytes.
;----------------
; Control de cambios:
; v1.1 Se ha cambiado la inicializadión del SR temporal en R15 que se hacía
; con clr.w r15 por dos instrucciones:
;    clrc
;    jmp PrimeraVez
; ya que la opción anterior tenía el efecto colateral de deshabilitar las
; interrupciones.
;-------------------------------------------------------------------------
suma        cmp.w  #1, r14       ;n<1?
            jlo    sumaErr       ;...sí, salir con error
            clrc                 ;Asegurar que C=0
            jmp    PrimeraVez    ;Empezar suma
sumaBuc     mov.w  r15, sr       ;Recuperar SR
PrimeraVez  addc.b @r12+, 0(r13) ;Sumar empezando por LSB
            mov.w  sr, r15       ;Guardar SR
            inc.w  r13           ;Avanzar puntero destino
            dec.w  r14           ;Contbilizar elemento sumado
            jnz    sumaBuc       ;Quedan más? Seguir
            mov.w  r15, sr       ;Recuperar SR. Hubo desbordamiento?
            jc     sumaErr       ;...sí. Salir con error
            clr.w  r12           ;...no, salir sin error
            jmp    sumaFin
sumaErr     mov.w  #-1, r12      ;Salir con error
sumaFin     ret
```