Fundamentos de Computadores 2018/19 (2º cuatrimestre)
Ejercicios – Hoja 1 (modulo 8)

1. Indica cuál es el resultado de ejecutar las siguientes instrucciones, dando el contenido final de los registros y posiciones de memoria para cada instrucción.

    1. add r3, r0, r1
    2. add r2, r2, #1
    3. sub r4, r1, r0
    4. sub r4, r0, r1
    5. mul r4, r0, r1
    6. or r3, r1, r0
    7. mov r4, #0
    8. lsr r2, r0, r4
    9. ldr r0, [r4]
    10. ldr r0, [r4,#-4]
    11. str r2, [r4,r3]
    12. str r2, [r0]

    Se supone que para cada instrucción a ejecutar el contenido de los registros y posiciones de memoria es el siguiente:

Registers	
r0	0x 0000 0016
r1	0x 0000 0054
r2	0x FFFF FFFF
r3	0x 0000 0000
r4	0x 0000 0004

Memory	
0x00	0x 0339 3826
0x04	0x EA00 63AF
0x08	0x 17FA 8912
0x0C	0x BC98 3304
0x10	0x 7845 F34A
0x14	0x 534B 4AAA

2. Escribe un programa en lenguaje ensamblador que implemente el siguiente bloque IF-THEN. Usa la sección *data* para asignar algún valor inicial a las variables.

```
if (x >= y) {
   x = x+2;
   y = y-2;
}
```

3. Escribe un programa en lenguaje ensamblador que implemente el siguiente bloque IF-THEN-ELSE. Usa la sección *data* para asignar algún valor inicial a las variables.

```
if (x >= y) {
   x = x+2;
   y = y+2;
}
else {
   x = x-2;
   y = y-2;
}
```

4. Escribe un programa en lenguaje ensamblador que implemente el siguiente código. Usa la sección *bss* para reservar espacio en memoria para las variables.

```
a = 81;
b = 18;
do {
   a = a-b;
} while (a > 0);
```

5. Escribe un programa en lenguaje ensamblador que implemente el siguiente código. Usa la sección *bss* para reservar espacio en memoria para las variables.

```
n = 5;
fprev = 1;
f = 1;
i = 2;
while (i <= n) {
   faux = f;
   f = f + fprev;
   fprev = faux;
   i = i+1;
}
```

6. Escribe un programa en lenguaje ensamblador que implemente el siguiente código. Usa la sección *data* para asignar algún valor inicial a las variables f y n, y la sección *bss* para reservar espacio de memoria para la variable i.

```
for (i=2; i<=n; i++) {
f=f+f;
}
```

7. El siguiente programa calcula el máximo común divisor de dos números a y b según el algoritmo de restas de Euclides. Traducirlo a ensamblador del ARM. Se supone en este caso que todo el programa consta de una sola Sección (*text*) con la siguiente estructura:
   - Los valores iniciales de a y b se declaran con directivas *.word*, al comienzo de la sección.
   - A continuación, se reserva espacio para la variable mcd, por medio de una directiva *.space*
   - Finalmente, se escriben las instrucciones del programa

```
int a=5, b=15, mcd;
While (a≠b){
   if (a>b)
      a=a-b;
   else
      b=b-a;
}
mcd=a;
```

8. Traduce la siguiente sentencia en C:

```
f=g+h+B[4]
```

a ensamblador, donde f, g y h son enteros contenidos en memoria y B es un vector de 10 componentes.

9. Tenemos un vector de 10 componentes almacenado en la posición de memoria etiquetada como V. Diseña un programa en ensamblador que sume uno a cada una de sus componentes. Ayuda: utiliza el siguiente pseudo-código de alto nivel:

```
#define N 10
int V[N]={12,1,-2,15,-8,4,-31,8,8,25};
for (i=0; i<N; i++)
   V[i] = V[i]+1;
```

10. Tenemos un vector de 6 componentes almacenado en la posición de memoria etiquetada como V. Diseña un programa en ensamblador que cuente el número de valores mayores que 0 que contiene. Ayuda: utiliza el siguiente pseudo-código de alto nivel:

```
#define N 6
int V[N]={14,1,-2,7,-8,4};
int count = 0;

for (i=0; i<N; i++) {
   if (V[i] > 0)
   count = count+1;
}
```

11. Implementar un programa en ensamblador que calcule la sucesión de *Fibonacci* y la almacene en un vector V

de longitud arbitraria N. Esta sucesión infinita de números naturales queda definida como: V(0)=0, V(1)=1, V(i+2)=V(i+1)+V(i) (i=0,1,2,...). La dimensión del vector V se debe definir en el programa como una constante. Ayuda: utiliza el siguiente pseudo-código de alto nivel.

```
#define N 12
int V[N];
V[0] = 0;
V[1] = 1;
for (i=0; i< N-2; i++)
   V[i+2] = V[i+1] + V[i];
```

12. Dado un vector, A, de 12 componentes se desea generar otro vector, B, tal que B sólo contiene las componentes de A que son números pares mayores que cero. Ejemplo: Si A = (0,1,2,7,-8,4,5,12,11,-2,6,3), entonces B = (2,4,12,6). Ayuda: utiliza el siguiente pseudo-código de alto nivel

```
#define N 12
int A[N]={0,1,2,7,-8,4,5,12,11,-2,6,3};
int B[N];
int countB=0;
j=0;
for (i=0; i<N; i++) {
     if (A[i] > 0 && A[i] is even){
        B[j]=A[i];
        j++;
     }
}
countB=j;
```

13. Dados dos vectores, A y B, de 10 componentes cada uno se desea construir otro vector, C, tal que:

    `C(i) = |A(i) + B(9-i)|,  i = 0,..,9.`

    Escribe un programa en lenguaje de alto nivel que construya el vector C. Traduce el programa al lenguaje ensamblador del ARM.

14. Traduce el siguiente programa escrito en un lenguaje de alto nivel a lenguaje ensamblador. La orden swap(a, b) intercambia los valores de las variables a y b.

```
int a=13, b=16;

while (a>10){
   a=a-1;
   b=b+2;
}
if (a<b)
   swap (a, b);
else
   b= a-1;
```

15. Escribe un programa para el ensamblador del ARM que llame a una subrutina, `swap(int *a, int *b)`, encargada de intercambiar el contenido de dos posiciones de memoria. La subrutina recibirá como parámetros de entrada las posiciones de memoria correspondiente a `a` y `b` y deberá preservar el contenido de todos los registros que obligue el estándar de llamadas estudiado en clase.

    Cuestión: ¿Qué registros debemos utilizar dentro de la función para evitar tener que salvar y restaurar registros durante el proceso?

16. Escribe un programa para el ensamblador del ARM que cuente el número de 0's de un vector de longitud arbitraria. Emplea para ello una subrutina llamada `cuenta0s`, que reciba como parámetros de entrada toda la información necesaria para llevar a cabo la tarea.

17. Implementar el algoritmo de ordenación de la burbuja o *bubble sort* en ensamblador. Este sencillo algoritmo ordena los elementos de un vector de menor a mayor por medio de un procedimiento muy sencillo: recorre repetidas veces el vector, intercambiado posiciones sucesivas si V(i)>V(i+1), hasta que no se realiza ningún cambio. Se proporciona como ayuda el siguiente código de alto nivel:

```
do
   swapped=false
   for i from 0 to N-2 do:
      if V[i] > V[i+1] then
         swap( V[i], V[i+1] )
         swapped = true
      end if
   end for
while swapped
```

Nota. Usa una constante, N, para definir la longitud del vector.

18. Escribe un programa en lenguaje de alto nivel que llame a una función `fact` que calcule el factorial de un número no negativo, n, por medio de un bucle que realiza una secuencia iterativa de multiplicaciones. Traduce el programa al lenguaje ensamblador del ARM.

19. Escribe un programa en lenguaje de alto nivel que calcule el factorial de un número no negativo, n, usando recursividad, sabiendo que fact(n)=n*fact(n-1). Traduce el programa al lenguaje ensamblador del ARM.

20. (Junio 2015) Responde a las siguientes cuestiones, suponiendo que el vector *V* está almacenado a partir de la dirección de memoria *0x0C000000* y que el código se encuentra a continuación de los datos. El código almacena en la variable *CuentaTotal* el sumatorio de todos los números positivos que contiene el vector. Para realizar este sumatorio, se utilizan dos subrutinas: *Cuenta* y *Averigua*. El estudiante sólo necesita saber de la subrutina *Averigua* que como parámetro de entrada recibe el entero *V[i]* y como resultado devuelve: (1) el valor de *V[i],* si *V[i]* es positivo; (2) 0, si *V[i]* es negativo.

```
.data
V:  .word   1,2,-3,4,5,9,17,-15,20,12
N:  .word   10
.bss
CuentaTotal:    .space    4
.text
start:      ldr   r0,=V
            ldr   r1,=N
            bl    Cuenta
            ldr   r1, =CuentaTotal
            str   r0, [r1]
end:        b end

Cuenta:     PRÓLOGO_1
            mov   r2, #0
            mov   r5, #0
            ldr   r4, [r1]
bucle:      cmp   r2, r4
            beq   fin_bucle
            ldr   r3, [r0, r2, lsl #2]
            bl    Averigua
            add   r5, r5, r0
            add   r2, r2, #1
            b     bucle
fin_bucle:  move  r0, r5
            EPÍLOGO_1
            mov pc, lr
```

a) Indicar el valor de la dirección de la etiqueta *Cuenta*, es decir, a qué dirección salta la instrucción *bl Cuenta*. Razona la respuesta.

b) La subrutina *Cuenta* no es correcta considerando el estándar de llamadas a procedimientos de ARM.

Modificar el código de dicha subrutina para que respete este estándar.
c) Codificar el prólogo y el epílogo de la subrutina *Cuenta*, teniendo en cuenta las modificaciones del apartado b.
d) Escribe el código ensamblador correspondiente a la subrutina Averigua (incluye comentarios).

21. (Septiembre 2015) Dados dos puntos P1(x1,y1) y P2(x2,y2), la distancia de Chebishev entre ellos puede calcularse con el siguiente algoritmo:

```
int chebyshev(x1, y1, x2, y2)
{
  int d1, d2;
  d1 = abs(x1-x2)
  d2 = abs(y1-y2)
  if (d2 > d1)
      d1 = d2;
  return d1;
}
```

a) Codificar en ensamblador la subrutina chebishev(x1, x2, y1, y2), que recibe cuatro parámetros enteros con signo que se corresponden con las coordenadas *(x1, x2)* e *(y1, y2)* de dos puntos P1 y P2 y devuelva la distancia de Chebyshev que separa P1 y P2. En su cuerpo, invocará a la subrutina absoluto. Téngase en cuenta que la subrutina no es una subrutina hoja.
b) Codificar en ensamblador un programa que almacene en un vector D la distancia de Chebishev desde un punto P a todos los puntos de un vector V de N puntos, donde P, V y D serían variables globales. El vector V tendrá 2N enteros de forma que el i-ésimo punto tendrá coordenadas (x, y) = (V[2*i], V[2*i + 1]). Un código C equivalente sería:

```
#define N, ...

int Px, Py; //coordenadas x e y del punto P
int V[2N];  //Vector con N puntos V=[x0,y0,x1,y1,...]
int D[N];   //Vector de N distancias

void main(void)
{
  int i;
  for (i=0; i < N; i++)
    D[i] = chevishev(Px, Py, V[2*i], V[2*i + 1]);
}
```

* Nota: Se debe respetar el convenio visto en clase para llamadas a subrutinas.

22. (Junio 2016) Dado un vector, A, de 3*N componentes, se desea obtener un nuevo vector, B, de N componentes donde cada componente de B es la suma módulo 32 de una tripleta de elementos consecutivos de A. Es decir:
B[0] = (A[0]+A[1]+A[2]) mod 32, B[1] = (A[3]+A[4]+A[5]) mod 32, etc
Se pide:

a) Escribir un programa en lenguaje ensamblador del ARM que implemente el cálculo descrito de acuerdo con el siguiente código C equivalente:

```
#define N 4
int A[3*N] = {una lista de 3*N valores},
int B[N];
int i, j=0;
void main (void)
{
  for (i=0; i<N; i++){
    B[i] = sum_mod_32(A, j, 3);
    j=j+3
  }
}
```

donde la subrutina sum_mod_32(V,p,m) devuelve como resultado la suma módulo 32 de m elementos consecutivos del vector V tomados a partir de la posición p.

b) Escribir el código ensamblador de la subrutina sum_mod_32, de acuerdo al siguiente código C equivalente:

```
sum_mod_32 (int A[ ], int j, int len)
   {
      int i, sum=0;
      for (i=0; i<len; i++)
         sum = sum + A[j+i];
      sum=mod_power_of_2(sum,5);
      return sum;
   }
```

donde la subrutina mod_power_of_2(num, exp), siendo num un entero positivo y exp un entero mayor que 0 y menor que 32, devuelve como resultado el valor de (num mod 2^exp). Por ejemplo, si invocamos a la función como: mod_power_of_2(34, 5), la función devolvería como salida: ((34) mod (2^5))=(34 mod 32)=2.

Nota.- En todos los apartados se debe respetar el estándar de llamadas a subrutinas estudiado en clase, y las variables pueden ubicarse en registros o en memoria (global o pila según corresponda).

23. (Septiembre 2016) Dado un vector **A** de **N** enteros de 32bit sin signo, se desea calcular su Código de Redundancia Cíclico o **CRC** siguiendo el algoritmo de Fletcher. Dicho algoritmo genera como salida dos enteros sin signo que servirán para comprobar la integridad de los datos. El algoritmo de Fletcher de 64bits se puede implementar a partir del siguiente código:

```
void Fletcher64( unsigned int data[], int length, unsigned int crc[] ) {
    unsigned int sum1 = 0;
    unsigned int sum2 = 0;
    int index;

    for ( index = 0; index < length; index++ ) {
        sum1 = sum_mod64(sum1, data[index]);
        sum2 = sum_mod64(sum1 , sum2);
    }
    crc[0] = sum1;
    crc[1] = sum2;
}
```

donde el primer argumento es el vector de datos, el segundo es la longitud del vector y el tercero es el vector que contiene el resultado (es decir, los dos enteros sin signo que componen el CRC). Se supone que la rutina `unsigned int sum_mod64(unsigned int A, unsigned int B)` está ya implementada.

a) Escribe el código ensamblador de la rutina `Fletcher64`.

b) Escribe un programa en lenguaje ensamblador del ARM que, invocando a la rutina `Fletcher64`, calcule el CRC de los siguientes vectores:

V={0x12340000, 0x00005678}, y

W={0xAB000000, 0x00CD0000, 0x0000EF00, 0x00000011}

Debes llamar a la rutina `Fletcher64` dos veces. La primera llamada retornará el CRC de V y la segunda el CRC de W.

Nota.- En todos los apartados se debe respetar el estándar de llamadas a subrutinas estudiado en clase, y las variables pueden ubicarse en registros o en memoria (global o pila según corresponda).

24. (Junio 2013). Supongamos que definimos que un número natural es “bonito” si es menor que cien mil y además su valor puede obtenerse como una suma de números naturales de la forma 1+2+3+4+5+…
Se pide:
    a) Escribir un programa en lenguaje ensamblador del ARM tal que dado un número natural N decida si es o no bonito. El programa escribirá en la variable B un 1 si el número es bonito y un 0 en caso

contrario.

b) Convertir el código anterior en una subrutina que reciba como entrada un número natural N y devuelva como salida un 1 si el número N es bonito y un 0 si no lo es. Escribir un programa en lenguaje ensamblador del ARM que llame a la subrutina, y tal que dado un vector A de M números naturales sea capaz de hallar cuántos números bonitos hay en el vector. El programa debe almacenar la cantidad de números bonitos hallada en la variable "cuenta_bonitos".

Nota: Se debe respetar el convenio del ARM visto en clase para llamadas a subrutinas. Además, en ambos apartados se deben incluir las directivas para reservar memoria y declarar las secciones (.data, .bss y .text) correspondientes.

25. (Septiembre 2013) Responde a las cuestiones suponiendo que el vector V está almacenado a partir de la dirección de memoria 0x0C000000, que el código se encuentra a continuación de los datos y que las pseudo-instrucciones ocupan el mismo espacio que las instrucciones.

```
# Codigo 1
.global start

.data
V: .word 12,21,13,14,5,9
N: .word 6

.bss
CuentaTotal: .space 4

.text
start:      ldr R0,=V
            ldr R2,=N
            ldr R1,[R2]
            mov R2,#0
            mov R3,#0
bucle:      cmp R2,R1
            beq fin_bucle
            ldr R4,[R0]
            and R4,R4,#1
            add R3,R3,R4
            add R2,R2,#1
            add R0,R0,#4
            b bucle
fin_bucle:  ldr R1,=CuentaTotal
            str R3, [R1]
            b .

.end
```

```
#Código 2
.global start

.data
V: .word 12,21,13,14,5,9
N: .word 6

.bss
CuentaTotal: .space 4

.text
start:      mov sp,#0x0C200000
            ldr R0,=V
            ldr R2,=N
            ldr R1,[R2]
            bl Cuenta
            ldr R2,=CuentaTotal
            str R0,[R2]
            b .

Cuenta:     PRÓLOGO_1
            mov R4,#0
            mov R5,#0
            mov R6,R0
            mov R7,R1
bucle:      cmp R4,R7
            beq fin_bucle
            ldr R0, [R6]
            bl Comprobar
            add R5,R5,R0
            add R4,R4,#1
            add R6,R6,#4
            b bucle
fin_bucle:  mov R0,R5
            EPÍLOGO_1
            mov pc,lr

Comprobar:  PRÓLOGO_2
            mov R4,#1
            and R0,R0,R4
            EPÍLOGO_2
            mov pc,lr

.end
```

a) ¿En qué dirección de memoria del código 1 está almacenada la variable N? ¿Y la primera instrucción del bucle para el código 1? Razona las respuestas.

b) ¿Cuál es el contenido final de la variable de memoria CuentaTotal en el código 1? ¿Y de los

registros R0, R1 y R2? Razona la respuesta.

c) Supongamos que queremos estructurar el código con subrutinas, y lo rescribimos en código 2. Completa el prólogo y epílogo de cada una de las dos subrutinas, explicando por qué incluyes cada instrucción.

26. (Junio 2014) Dado un vector V de N componentes se dice que es Melchoriforme si posee al menos un elemento Rubio. Un elemento V[i] es Rubio si satisface la siguiente expresión:

$$\sum_{j=0}^{N-1} \mathrm{v[j]} = 2 * \mathrm{v[i]}$$

Se pide:

a) Una subrutina SumaVector(V, N) que sume los N elementos del vector V, respetando el convenio de llamadas a subrutinas visto en clase.

b) Un programa que dado un vector V y su dimensión N decida si es Melchoriforme, utilizando la subrutina SumaVector.

27. (Septiembre 2014) Un vector V de N números naturales es noeliano si es una secuencia monótona creciente y sus elementos suman en total 45. Por ejemplo: 0-1-2-3-4-5-6-7-8-9 es noeliano porque 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 y 1+2+3+4+5+6+7+8+9=45. También: 3-5-5-7-10-15 es noeliano, ya que 3≤5≤5≤7≤10≤15 y 3+5+5+7+10+15=45. Se pide:

a) Escribir una subrutina en ensamblador de ARM Sum45(A, N) que reciba la dirección de comienzo de un vector A como primer parámetro, el número N de elementos del vector como segundo parámetro y devuelva 1 si su suma es 45 y 0 en otro caso. La subrutina debe programarse de acuerdo con el estándar de llamadas a subrutinas que hemos estudiado en clase.

b) Escribir un programa ARM que, utilizando la subrutina anterior, determine si un vector de entrada es noeliano o no.