Guía de trabajos prácticos de TD2

M. Estefanía Pereyra

15 de junio de 2019

Trabajo Práctico: programación en assembler de ARM

Compilado:

En línea de comandos de la consola, sobre el directorio donde se encuentra el archivo fuente *.s

- arm-none-eabi-as -mcpu=arm7tdmi -v -g ex1.s -o ex1.o
- arm-none-eabi-ld -Ttext=0 -v -g ex1.o -o ex1.elf
- arm-none-eabi-objdump -S ex1.elf > ex1.lst

Depurado:

- arm-old-gdb –tui

en la interfaz de usuario,

- target sim
- file ex1.elf
- load
- list

Comandos del debugger,

- break *linea* o b *linea*. Break point (punto de parada) en la linea indicada.
- run o r. Run comienza la ejecución del programa.

- step o s. Ejecución paso a paso.
- continue o c. Ejecución continua hasta encontrar un break point o terminar el programa.
- info registers o i r. Lista el estado de los registros.
- info registers *registro*. Muestra el estado de un registro en particular.
- layout regs. Abre otra ventana con los registros.
- layout asm. Abre otra ventana con código desensamblado.
- x *VAR*. Muestra el contenido de una variable almacenada en memoria. (*VAR*:nombre de la variable)
- x\n *VECT*. Muestra n elementos almacenados en memoria desde la dirección *VECT*.
- Ctrl+x o. Cambia el foco de la ventana activa.
- Ctrl+c. Finaliza la ejecución de un programa.
- quit o q. Salir del depurador.

Ejercicio 1:

Considere el siguiente código escrito en lenguaje de alto nivel C. Asumir que las variables enteras (signadas) g y h están en los registros R0 y R1, respectivamente:

1.
```
if (g>=h)
  g = g+h;
else
  g = g-h;
```

2.
```
if (g<h)
  h = h+1;
else
  h = h*2;
```

a) Escriba el código equivalente en lenguaje ensamblador de ARM asumiendo que la ejecución condicional solo esta disponible para instrucciones de salto.

b) Escriba el código equivalente en lenguaje ensamblador de ARM asumiendo ejecución condicional en todas las instrucciones.

c) Compare la diferencia en densidad de código (número de instrucciones) entre (a) y (b).

Solución:

```
/* ============================
TEST CODE
*  ============================
*/
    mov r0, #4        @var1
    mov r1, #5        @var2

/*Modo1a: saltos condicionales*/

    cmp r0,r1         @g >= h?
    blt else
    add r0,r0,r1      @g = g+h
    b   modo2a
else:
    sub r0,r0,r1      @g = g-h

/*Modo2a: instrucciones condicionales*/

modo2a:
    cmp r0,r1         @g >= h?
    addge r0,r0,r1    @g = g+h
    sublt r0,r0,r1    @g = g-h

/*Modo1b: saltos condicionales*/

    cmp r0,r1         @g < h?
    bge else
    add r1,r1,#1      @h = h+1
    b   modo2b
else:
    lsl r1,r1,#1      @h = h*2

/*Modo2b: instrucciones condicionales*/

modo2b:
    cmp r0,r1         @g < h?
    addlt r1,r1,#1    @h = h+1
    lslge r1,r1,#1    @h = h*2
```

Ejercicio 2:

Realizar un programa que sume dos números de 64 bits cada uno, guardando el resultado en R5 R6 (64bits), los números a sumar estarán dispuesto en R1 R2(64bits) y R3 R4(64bits) respectivamente.

Solución:

```
/* ================================
TEST CODE
*  ================================
*/
reset:
    mov r0, #NUM11   @r0<-&NUM11
    ldr r1, [r0]     @r1=mem[r0]
    mov r0, #NUM12
    ldr r2, [r0]
    mov r0, #NUM21
    ldr r3, [r0]
    mov r0, #NUM22
    ldr r4, [r0]

    adds r5, r1, r3  @r5=r1+r3 (modifica flags)
    adc  r6, r2, r4  @r6=r2+r4 (suma con acarreo)

loop:   b loop

/* ================================
*  CONTANTES
*  ================================
*/
NUM11:   .word 0x1231AAAA
NUM12:   .word 0xB3423433
NUM21:   .word 0xFF232323
NUM22:   .word 0x23323111
```

Ejercicio 3:

Realizar un programa que analizando un bit en el registro R3 active una alarma si este bit es 0, el nro de bit a analizar está guardado en el registro R4, la alarma se activa escribiendo un 1 en el bit 16 de R6.

Solución:

```
/* ================================
TEST CODE
*  ================================
*/
reset:
    mov r3, #0b1101   @registro a ser analizado
    mov r4, #1        @verifica el bit 2
    mov r6, #0        @registro salida de alarma

    mov r0, #0x01     @registro auxiliar
    lsl r0, r4

    and r3, r0        @verifica el bit 2
    cmp r3, #0        @compara que sea cero o no
    bne salir         @si no es 0 sale
    orr r6, #(1<<16) @si es 0 activa alarma

salir:
```

```
loop:   b loop

.end
```

Ejercicio 4:

Realizar un programa que dado un numero guardado en R1 lo multiplique por 10, de dos formas distintas, usando la instrucción mul y por corrimiento.

Solución:

```
/* ============================
TEST CODE
*  ============================
*/
reset:
    mov r1, #4              @valor a multiplicar

/* Modo: MUL*/
    mov r0, #10
    mul r2, r1, r0          @R2 = R1*10

/* Modo: corrimiento*/

    lsl r3, r1, #3          @R3 = R1*8
    add r3, r3, r1, lsl #1 @R3 = R3 + R1*2

loop:   b loop

.end
```

Ejercicio 5:

Realizar un programa que dado un numero guardado en R1 y otro en R2, calcule el resultado de elevar R1 a la potencia de R2 y guardar el resultado en R3 ($R3 = R1^{R2}$).

Solución:

```
/* ============================
TEST CODE
*  ============================
*/
reset:
    mov r1, #4      @valor a multiplicar
    mov r2, #4      @exponente

    mov r3, #1
    cmp r2, #0
    beq salir

potencia:
    mul r3, r1, r3  @R3 = R3*R1
    subs r2, r2, #1 @decrementa el exponente en 1
```

```
    bne  potencia   @vuelve a multiplicar por la base mientras r2 distinto de 0

salir:
loop:   b loop

.end
```

Ejercicio 6:

Realizar un programa que sume los primeros 100 números naturales.
Solución:

```
/* ============================
TEST CODE
*  ============================
*/

reset:

/* Metodo 1*/
    mov r0,#0       @i=0
    mov r1,#0       @sum=0

for1:
    add r1,r1,r0    @sum=sum+i
    add r0,r0,#1    @i=i+1
    cmp r0,#100     @i == 100?
    bne for1

/* Metodo 2*/
    mov r0, #99     @i=99
    mov r1, #0      @sum=0

for2:
    add r1,r1,r0    @sum=sum+i
    subs r0,r0,#1   @i = i-1
    bne  for2       @repetir loop

loop:   b loop

.end
```

Ejercicio 7:

Realizar un programa que copie el contenido de un vector almacenado en memoria en otro vector, asumiendo que este segundo vector tiene asignado el espacio suficiente en memoria para almacenar los datos. Considere el código escrito en C como guía:

```
int i;
```

```
int array1[100];
int array2[100];

for(i=0; i<100; i=i+1)
    array1[i] = array2[i];
```

a) Escriba el código equivalente en lenguaje ensamblador de ARM sin utilizar pre o post indexado o un registro escalado.

b) Escriba el código equivalente en lenguaje ensamblador de ARM utilizando pre o post indexado o un registro escalado.

c) Compare la diferencia en densidad de código (número de instrucciones) entre (a) y (b).

Solución:

```
/* ============================
TEST CODE
*  ============================
*/

reset:
/* Metodo sin pre/post indexado ni registro escalado */

    ldr r1,=ARRAY1   @r1<-&ARRAY1
    ldr r2,=ARRAY2   @r2<-&ARRAY2
    add r3,r1,#40    @r3 = fin del array
for:
    cmp r1,r3        @fin del array?
    bge metodo2      @salir
    ldr r0,[r1]      @r0=ARRAY1[i]
    str r0,[r2]      @ARRAY2[i] = r0
    add r1,r1, #4    @r1 apunta al siguiente elemento del array1
    add r2,r2, #4    @r2 apunta al siguiente elemento del array2
b for

/* Metodo con post-indexado */

metodo2:
    ldr r1,=ARRAY1   @r1<-&ARRAY1
    ldr r2,=ARRAY2   @r2<-&ARRAY2
    add r3,r1,#40    @r3 = fin del array
for2:
    cmp r1,r3        @fin del array?
    bge loop         @salir
    ldr r0,[r1],#4   @r0=ARRAY1[i] y actualiza r1
    str r0,[r2],#4   @ARRAY2[i] = r0 y actualiza r2
    b for2
loop:   b loop

/* ============================
```

```
CONSTANTES
*  ============================
*/
.balign 4
ARRAY1:  .word 1,2,3,4,5,6,7,8,9,10   @vector de 10 elementos de tipo word (32bits).
ARRAY2:  .space 40                    @reserva espacio en memoria para 40 bytes (10 words).

.end
```

Ejercicio 8:

Realizar un programa que sume los word de un vector VECTW sin signo en las posiciones indicadas por un segundo vector VECTB de 20 byte,el resultado guardarlo en R1.

ejemplo

VECTB = 1,3,2,0,2.....

VECTW = 1,111,222,333,444,555,666,.....

se suman 111+333+222+1+222...

Ejercicio 9:

Realizar un programa que dada una cadena con terminación nula guardada en VECT, la pase a mayúscula, guardando el resultado en el mismo vector VECT. nota:la cadena de entrada solo contendrá valores alfabéticos o espacio en blanco ('a' - 'z','A' - 'Z').

Ejercicio 10:

Dado un vector de 16 words con signo, realizar un programa que encuentre el elemento del vector mas cercano a la media del mismo.

Ejercicio 11:

Realizar una serie de subrutinas que realicen diferentes comparaciones, devolviendo $R0 = 0$ si estas comparaciones fueron falsas y $R0 = 1$ si fueron verdaderas, las comparaciones son las siguientes.

- $R1 < 100$ y $R1 > 20$
- $R1 < 100$ o $R2 > 20$
- $R1 = 10$ o $R1 = 15$ o $R1 = 20$
- $R1 = 10$ y $R2 = 15$ y $R3 = 20$

nota: los datos en R1, R2, y R3 pueden estar inicializados al comienzo del programa o ser leídos de variables almacenadas en memoria.

Solución:

```
/* ============================
TEST CODE
*  ============================
*/

reset:
    mov r1,#10
    mov r2,#15
    mov r3,#20

    bl comp1
    bl comp2
    bl comp3
    bl comp4


loop:   b loop


@ ------------------------------------------------------------
@ r1<100 y r1>20

comp1:
    mov r0,#0       @ NO
    cmp r1,#100     @ r1<100 ?
    bhs comp1_no    @ NO
    cmp r1,#20      @ r1>20 ?
    ble comp1_no    @ NO
    mov r0,#1       @ SI cumple las dos
comp1_no:
    mov pc,lr       @ retorna a la direccion almacenada en LR

@ ------------------------------------------------------------
@ r1<100 o r2>20

comp2:
    mov r0,#1       @ SI
    cmp r1,#100     @ r1<100 ?
    blo comp2_s     @ SI, entonces sale
    cmp r2,#20      @ r1>20 ?
    bhi comp2_si    @ SI, entonces sale
    mov r0,#0       @ NO, cambia R0 y sale
comp2_si:
    mov pc,lr       @ retorna a la direccion almacenada en LR

@ ------------------------------------------------------------
@ r1=10 o r1=15 o r1=20

comp3:
    mov r0,#1       @ SI
    cmp r1,#10      @ r1 == 10?
    beq comp3_si    @ SI, entonces sale
    cmp r1,#15      @ r1 == 15?
```

```
    beq comp3_si    @ SI, entonces sale
    cmp r1,#20      @ r1 == 20?
    beq comp3_si    @ SI, entonces sale
    mov r0,#0       @ NO, cambia R0 y sale
comp3_si:
    mov pc,lr       @ retorna a la direccion almacenada en LR


@ ------------------------------------------------------
@ r1=10 y r2=15 y r3=20

comp4:
    mov r0,#0       @ NO
    cmp r1,#10      @ r1 == 10?
    bne comp4_no    @ NO, entonces sale
    cmp r2,#15      @ r1 == 15?
    bne comp4_no    @ NO, entonces sale
    cmp r3,#20      @ r1 == 20?
    bne comp4_no    @ NO, entonces sale
    mov r0,#1       @ SI, cambia R0 y sale
comp4_no:
    mov pc,lr       @ retorna a la direccion almacenada en LR


.end
```

Ejercicio 12:

a) Escriba una *función* en C para buscar un número con signo en un vector de WORDS:

```
int findNumber(int array[], int size, int number)
```

Donde *size*: es el tamaño del vector, *array*: especifica la dirección base del vector y *number*: número que se desea buscar. La función retorna el índice donde se encuentra por primera vez el número en el vector. Si el número no se encuentra en el vector retorna -1.

b) Escriba el código equivalente de la *función* en lenguaje ensamblador de ARM.

c) Realice un programa en lenguaje ensamblador de ARM que utilice la función *findNumber* para buscar un número almacenado en memoria en un vector de 16 words también almacenado en memoria.

Solución:

```
/* ================================
*  TEST CODE
*  Buscar numbero con signo en vector de WORDS
*  ================================
*/
@R0=direccion base del array
```

```
@R1=numero de elementos en el array
@R2=numero buscado
@R3=i
@R4=VECT[i]

reset:
        ldr r0,=VECT
        mov r1,#16
        mov r2,#NUM
        ldr r2, [r2]
        bl  ENCONTRARNUM
        b   SALIR


ENCONTRARNUM:
        push {r4}               @ resguarda r4
        mov r3, #0              @ i=0
FOR:
        cmp r3, r1              @ i<size?
        bge VOLVER

        ldr r4,[r0, r3, lsl #2] @ r4 = VECT[i]
        cmp r4, r2              @ VECT[i] == NUM ?
        addne  r3, r3, #1       @ si no es igual i++
        bne FOR
        mov r0, r3              @ retorna i
        pop {r4}                @ recupera r4 del stack
        mov pc, lr              @ retorna

VOLVER:
        mov r0, #-1             @ retorna -1
        pop {r4}                @ recupera r4 del stack
        mov pc, lr              @ retorna a la direccion almacenada en LR

SALIR:
loop:   b loop

/* ============================
*  CONTANTES
*  ============================
*/
        .balign 4
VECT:   .word 10,-10,200,40,50,60,70,80,2,-5,100,89,-43,23,15,86
NUM:    .word 80

        .end
```

Ejercicio 13:

La función *strcpy* copia una cadena de caracteres *src* a una cadena de caracteres *dst*.

```
void strcpy(char dst[], char src[]){
  int i = 0;
  do{
```

```
    dst[i] = src[i];
  }while(src[i++]);
}
```

a) Implemente la función *strcpy* en lenguaje ensamblador de ARM. Use el registro R4 para *i*.

b) Diagrame la estructura del *stack* antes, durante y luego del llamado a la función *strcpy*. Asumir $SP = 0xBEFFF000$ justo antes de la llamada a la función *strcpy*.

Solución:

```
/* ================================
*  TEST CODE
*  copia cadena de caracteres
*  ================================
*/
@R0=direccion base del array dst
@R1=direccion base del array src
@R4=i
reset:
        ldr r0,=DST
        ldr r1,=SRC
        bl  STRCPY
        b   SALIR


STRCPY:
        push {r4}               @ resguarda r4
        mov r4, #0              @ i=0
FOR:
        ldrb r2,[r1, r4]        @ r2 = SRC[i]
        strb r2,[r0, r4]        @ DST[i] = r2
        cmp r2, #0              @ SRC[i] == 0 ? (fin de la cadena)
        addne  r4, r4, #1       @ si no es igual i++ (r4 incrementa en 1 byte por ser datos tipo caracteres)
        bne FOR

VOLVER:
        pop {r4}                @ recupera r4 del stack
        mov pc, lr              @ retorna

SALIR:
loop:   b loop

/* ================================
*  CONTANTES
*  ================================
*/
        .balign 4
SRC:    .asciz "cadena de caracteres con terminacion nula"
DST:    .space 50
        .end
```

Stack antes del llamado a *strcpy* (a), durante el llamado a *strcpy* (b), luego del llamado a *strcpy* (a):

a)

Dirección	Datos
$SP = 0xBEFFF000 \rightarrow$	Dato desconocido

b)

Dirección	Datos
	Dato desconocido
$SP = 0xBEFFEFFC \rightarrow$	R4

Ejercicio 14:

Considere el código en lenguaje ensamblador de ARM siguiente. Las funciones *func1*, *func2*, *func3* son funciones no hojas (llaman a otras funciones) y la función *func4* es una función hoja. No se muestra el código de cada función, se indica mediante un comentario qué registros se utilizan durante el llamado de cada una.

```
0x00091000   func1:      ;func1 usa R4-R10
             ...
0x00091020   BL func2
...
0x00091100   func2:      ;func2 usa R0-R5
             ...
0x0009117C   BL func3
...
0x00091400   func3:      ;func3 usa R3, R7-R9
             ...
0x00091704   BL func4
...
0x00093008   func4:      ;func4 usa R11-R12
             ...
0x00093118   MOV PC, LR
```

a) Cuantas palabras tiene el bloque de stack correspondiente a cada función.

b) Diagrame la estructura del *stack* luego del llamado a la función *func4*. Indique qué registros están almacenados en el stack y señale cada bloque del stack correspondiente a cada función. Asumir $SP = 0xBFFFFF04$ justo antes de la llamada a la primera función.

Solución:

b)Stack antes del llamado a las funciones:

Dirección	Datos
$SP = 0xBFFFFF04 \rightarrow$	Dato desconocido

Stack luego del llamado a la función *func4*:

	Dirección	Datos	
	$0xBFFFFF04$	Dato desconocido	
	$0xBFFFFF00$	LR = ?	stack frame func1, 8 palabras
	$0xBFFFFEFC$	R10	
	$0xBFFFFEF8$	R9	
	$0xBFFFFEF4$	R8	
	$0xBFFFFEF0$	R7	
	$0xBFFFFEEC$	R6	
	$0xBFFFFEE8$	R5	
	$0xBFFFFEE4$	R4	
	$0xBFFFFEE0$	LR = $0x00091024$	stack frame func2, 3 palabras
	$0xBFFFFEDC$	R5	
	$0xBFFFFED8$	R4	
	$0xBFFFFED4$	LR = $0x00091180$	stack frame func3, 4 palabras
	$0xBFFFFED0$	R9	
	$0xBFFFFECC$	R8	
	$0xBFFFFEC8$	R7	
SP =	$0xBFFFFEC4$	R11	stack frame func4, 1 palabras

Ejercicio 15:

Dada una cadena de caracteres con terminación nula la cual contiene palabras separadas por espacio, codificar en assembler de ARM una función denominada *aminuscula*, que pase a minúscula todas las letras con excepción de la primera de cada palabra que deberá ser pasada a mayúscula, además, las palabras de menos de 3 letras deberán quedar en minúscula.

Ejemplo:

entrada

VECT = .[El] mUndo dE Hoy",0

salida

VECT = .[el] Mundo de Hoy",0

La función *aminuscula* recibe como parámetro la dirección base de la cadena de caracteres y no tiene ningún valor de retorno, los datos se sobrescriben en la cadena original.

Ejemplos de código:

```
/* ================================
*  TEST CODE
*  Buscar el Mayor dentro de un vector de bytes y lo almacena en r3
*  ================================
*/

reset:
        mov r0,#8
```

```
        ldr r1,=VECT
        ldrsb r3,[r1],#1
        subs r0,#1
        beq  loop

otro:   ldrsb r2,[r1],#1
        cmp r3,r2
        bge nomayor
        mov r3,r2
nomayor:
        subs r0,#1
        cmp r0,#0
        bne otro



loop:   b loop

/* ================================
*  CONTANTES
*  ================================
*/
VECT:   .byte 10,-10,200,40,50,60,70,80
        .balign 4

        .end


/* ================================
*  TEST CODE
*  leer 8 elementos de un vector y calcular su promedio y lo almacena en memoria
*  ================================
*/
reset:
        mov r1,#0
        mov r2,#0
        mov r3,#VECT
otro:   ldrsb r4,[r3],#1
        add r2,r4
        add r1,#1
        cmp r1,#8
        bne otro
        lsr r2,#3
        strb r2,RESUL
loop:   b loop
        .ltorg

/* ================================
*  CONTANTES
*  ================================
*/
VECT:   .byte 10,-20,30,40,50,60,70,80
RESUL:  .byte 0
        .balign 4
        .end
```

```
/* ================================
*  TEST CODE
*  Guarda en r4 la longitud de la palabra mas larga
*  ================================
*/

reset:

        mov r4,#0
        ldr r1,=VECT
        mov r3,#0
otro:   ldrb r2,[r1],#1
        cmp r2,#0
        beq espacio
        cmp r2,#' '
        beq espacio
        add r3,#1
        bne noespacio
espacio:
        cmp r3,r4
        bls nomayor
        mov r4,r3
nomayor:
        mov r3,#0
        cmp r2,#0
        beq salir

noespacio:
        b otro
salir:


loop:   b loop

/* ================================
*  CONTANTES
*  ================================
*/
VECT:   .asciz "hola mundo de hoy"
        .balign 4


        .end
```