

# Tarea 2

ARM Assembler

Integrantes: Francisco Muñoz P.  
Profesor: Pablo Guerrero P.  
Auxiliar: Matías Torrealba  
Ayudantes: Gabriel Chandía G.  
Gaspar Ricci  
Fecha de entrega: 10 de Diciembre 2018  
Santiago, Chile

# 1. P1

El código creado para las instrucciones fue implementado en c++ utilizando librerías estándar en conjunto con **sexpr** librería recomendada por el enunciado. Se adjuntara con la entrega un README con instrucciones simples para compilar.

Antes que nada, se fijan algunas ideas para ayudar a la creación del compilador:

- Se dejaran los registros r0, r1, r2 y r3 para pasarlos como parámetros a funciones segun lo establecido en ARM assembly.
- Los registros r4, r5, r6, r7, r8, r9, r10 y r11 serán utilizados para aplicar las instrucciones y al ser registros que deben ser preservados (segun lo establecido en ARM assembly), cada función los tendrá a entera disposición.
- Todo registro cuyo valor sea guardado en la pila queda libre para ser reutilizado, en el caso de ADD Y SUB, ambos registros utilizados quedan libres.
- Las instrucciones IF0 y FUN utilizaran etiquetas únicas para cada una (se llevara un contador dentro del compilador), pero IF0 no sera considerada como una función, pudiendo usar los valores que existan en los registros r4-r11.
- Se considerara que toda función (FUN inst\_list) termina en un (RETURN) de forma que siempre entregue algún valor valido en r0.
- Siempre que se use (APPLY), la pila debe tener en su tope algún valor seguido del llamado a la función.
- Se considerara que una vez llamada a una función, esta se pierde del stack y en caso de querer volver a llamarla el código la creara de nuevo con otra etiqueta.
- El valor utilizado en la instrucción if0 (que se compara con 0) no regresa a la pila y se pierde.
- Segun enunciado, se considera a (FUN inst\_list) como una instrucción que viene acompañada de una serie de otras instrucciones sin estar estas dentro de una lista, por ejemplo, (FUN (INT\_CONST 3) (ADD) (RETURN)).
- Por el contrario, (IF0 tb fb) tendrá dos listas de instrucciones, una para cada caso (se cumple o no la condición), por ejemplo, (IF0 ((INT\_CONST 3)) ((INT\_CONST 7))).

Teniendo en mente esto, la idea básica detrás del compilador en c++ es recibir las instrucciones SECD y traducirlas a lenguaje ARM assembly escribiendo el código correspondiente a un archivo el cual puede ser compilado de la forma tradicional y ejecutado.

Entonces, lo primero a considerar es la traducción a lenguaje ARM assembly de cada instrucción SECD:

- (INT\_CONST n): guarda el valor n en la pila, esto requiere de utilizar la instrucción **mov** para guardar el valor n en algún registro ri, y luego guarda su valor en la pila utilizando **stmfd** donde **fd** viene de *full descending*. Por ejemplo, la instruccion INT\_CONST 4 es:

```

1 mov r4, #3      @ guarda el valor 4 en r4
2 stmfd r13!, {r4} @ guarda el valor 4 de r4 en la pila

```

- **(ADD)**: carga los últimos 2 valores desde la pila con **ldmfd** y los deja en dos registros *ri* y *rj*, luego sumamos los valores utilizando **add**, dejando el resultado en *rj* para guardar esto en la pila con **stmfd**. La forma general de ADD es:

```

1 ldmfd r13!, {r4} @ carga el primer valor de la pila en r4
2 ldmfd r13!, {r5} @ carga el segundo valor de la pila en r5
3 add r5, r4, r5   @ suma ambos valores y los guarda en r5
4 stmfd r13!, {r5} @ guarda resultado en la pila

```

- **(SUB)**: lee los últimos 2 de igual forma, pero esta vez resta a *ri* el valor de *rj* dejando el resultado en *rj* para lo que se utiliza **sub** y guarda este valor en la pila. La forma general de SUB es:

```

1 ldmfd r13!, {r4} @ carga el primer valor de la pila en r4
2 ldmfd r13!, {r5} @ carga el segundo valor de la pila en r5
3 sub r5, r4, r5   @ r5 = r4 - r5
4 stmfd r13!, {r5} @ guarda resultado en la pila

```

- **(FUN inst\_list)**: primero asigna un label a la nueva función a crear, luego asigna este label a un registro y lo guarda en la pila. Finalmente, realiza la implementación de la función según la lista de instrucciones. La función inicia guardando su parámetro de entrada en la pila y termina cargando el ultimo valor de la pila en su parámetro de salida (*r0* en ambos casos). Además, por precaución, cada vez que inicia una función, se guardan los valores de todos los registros que deben ser preservados (*r4* a *r11*) y al final se restauran sus valores. Por ejemplo, recibir **FUN ((INT\_CONST 3) (INT\_CONST 5) (ADD) (RETURN))** es en ARM assembly:

```

1 funN:
2     stmfd r13!, {r4, r5, r6, r7, r8, r9, r10, r11, lr} @ guarda los valores
3     stmfd r13!, {r0}   @ guarda el parametro de la funcion en el stack
4     mov r4, #3         @ instruccion INT CONST 3
5     stmfd r13!, {r4}
6     mov r4, #5         @ instruccion INT CONST 5
7     stmfd r13!, {r4}
8     ldmfd r13!, {r4}
9     ldmfd r13!, {r5}
10    add r5, r4, r5     @ instruccion ADD
11    stmfd r13!, {r5}
12    ldmfd r13!, {r0}   @ usa ultimo valor del stack como parametro de salida.
13    ldmfd r13!, {r4, r5, r6, r7, r8, r9, r10, r11, pc} @ restaura los valores
14 main:
15    ...
16    ldr r4, =funN      @ carga la etiqueta en r4
17    stmfd r13!, {r4}   @ la guarda en el stack
18    ...

```

- **(RETURN)**: es una instrucción que indica el retorno de una función, para ello lee el ultimo

valor guardado en la pila y lo carga al registro r0 que es considerado como el registro de salida. Para un ejemplo ver instrucción FUN.

- **(APPLY)**: indica la utilización de una función. Esta instrucción requiere que el ultimo valor guardado en la pila sea el parámetro para la función y que el segundo valor sea el puntero a la función. Carga estos valores en registros y salta a la función utilizando *blx*, luego, el resultado que entregue esta función lo guarda en la pila. La forma general de APPLY es:

```

1 ldmfd r13!, {r0}    @ carga el parámetro de la función en r0
2 ldmfd r13!, {r4}    @ carga el puntero a la función
3 blx r4              @ salta a la función
4 stmfd r13!, {r0}    @ guarda el resultado de la función en la pila

```

- **(IF0 tb fb)**: lee el ultimo valor en el stack, si es 0, ejecuta las instrucciones tb, en caso contrario ejecuta instrucciones fb, para realizar los saltos, se crean etiquetas según un numero indicador unico. Una forma general de esta instrucción es:

```

1 ldmfd r13!, {r4}    @ carga ultimo valor de stack en r4
2 cmp r4, #0          @ compara r4 con 0
3 bne if0N            @ si son distintos salta a if0N
4 @ en caso que sea igual ejecuta instrucciones tb
5 b if0N_end          @ y salta al resto de ejecuciones
6 if0N:
7 @ en caso de ser diferentes ejecuta instrucciones fb
8 if0N_end:
9 @ resto de instrucciones

```

En base a lo anterior, gracias a *sexp*, se puede recorrer iterativamente las instrucciones en múltiples capas. Una primera pasada entrega las instrucciones principales INT CONSTS, ADD, SUB, FUN, APPLY, IF0 o RETURN que estaran dentro de *main*, luego, en caso de recibir FUN o IF0, realiza una segunda pasada buscando las sub-instrucciones y repite el proceso tantas veces como sea necesario. Todo el código correspondiente a las sub-funciones se escribe antes que el código correspondiente a la función main. Además, se utilizara la función *printf* de c para entregar el resultado. El código permite funciones y condiciones if anidadas gracias al indicador único por label.

## 2. P2

Con el objetivo de verificar lo mejor posible un correcto funcionamiento del programa, se realizan 7 tests, de los cuales 5 son ejemplos básicos verificando que cada instrucción por separado funcione correctamente, el 6to corresponde al ejemplo entregado por el enunciado y el ultimo es un caso complejo donde utiliza funciones anidadas.

- **test basico INT\_CONST:** se le entrega el valor 2 y se espera que devuelva el mismo numero:

```
1 ((INT_CONST 2))
```

- **test basico ADD:** se le entrega los valores 45 y 23, y se suman, esperando que entregue como resultado 68:

```
1 ((INT_CONST 45) (INT_CONST 23) (ADD))
```

- **test basico SUB:** se le entrega los valores 12 y 24, y se restan, esperando que entregue como resultado 12:

```
1 ((INT_CONST 12) (INT_CONST 23) (SUB))
```

- **test basico IF0:** se le entrega el valor 0 y una instrucción IF0 que guarda 2 si cumple condición y 23 si no, esperando que entregue como resultado 2:

```
1 ((INT_CONST 0) (IF0 ((INT_CONST 2)) ((INT_CONST 23))))
```

- **test basico FUN:** se le entrega una funcion la cual recibe como valor de entrada un 2 y lo suma con un 4, esperando que entregue como resultado 6:

```
1 ((FUN (INT_CONST 4) (ADD) (RETURN)) (INT_CONST 2) (APPLY))
```

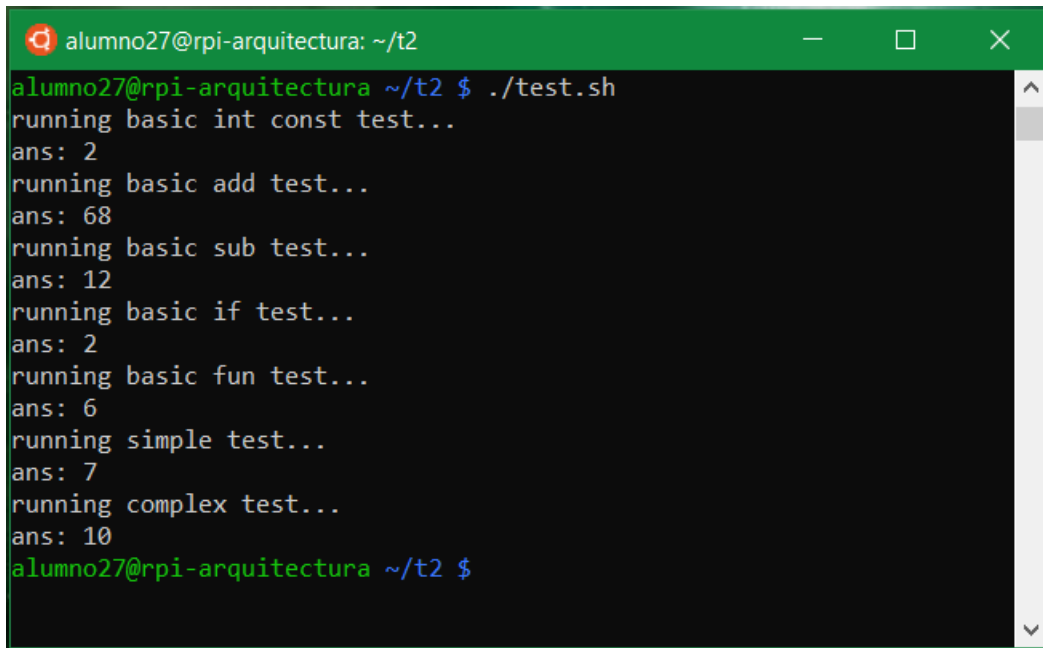
- **test simple:** ejemplo de instrucciones dado por el enunciado de la tarea, las instrucciones entregan primero un 2, luego una función que entrega 5 o 7 dependiendo de si el parámetro que reciba de entrada sea 0 o distinto de 0, se espera que estas instrucciones den como resultado 7:

```
1 ((INT_CONST 2) (FUN (IF0 ((INT_CONST 5)) ((INT_CONST 7))) (RETURN)) ↗
  ↘ (INT_CONST 0) (APPLY) (ADD))
```

- **test complejo:** un caso complejo donde se realizan llamados de funciones con condiciones if y de funciones dentro de funciones. No se explicara instrucción a instrucción por ser un caso mas extenso, pero el lector puede fácilmente deducir que el resultado esperado es 10:

```
1 ((INT_CONST 3) (INT_CONST 0) (IF0 ((FUN (IF0 ((INT_CONST 5)) ↗
  ↘ ((INT_CONST 7))) (RETURN)) (INT_CONST 18) (APPLY)) ((INT_CONST ↗
  ↘ 3) (SUB))) (FUN (INT_CONST 6) (ADD) (FUN (INT_CONST 2) (SUB) ↗
  ↘ (RETURN)) (INT_CONST 1) (APPLY) (ADD)) (INT_CONST 7) (APPLY) ↗
  ↘ (SUB) (ADD))
```

Los archivos con las instrucciones en SECD se darán con la entrega junto con instrucciones de como correrlos con el compilador, por simplicidad, se genera un script en bash que se encarga de ejecutar automáticamente todas las compilaciones. Los resultado obtenidos luego de compilar se muestran en fig. 1, donde se observa que los valores coinciden con lo esperado lo que sugiere un correcto funcionamiento del programa.

A terminal window with a green title bar. The title bar text is 'alumno27@rpi-arquitectura: ~/t2'. The terminal content shows the execution of a script './test.sh'. The script runs several tests: 'basic int const test...', 'basic add test...', 'basic sub test...', 'basic if test...', 'basic fun test...', 'simple test...', and 'complex test...'. Each test is followed by 'ans:' and a numerical result. The results are: 2, 68, 12, 2, 6, 7, and 10 respectively. The prompt 'alumno27@rpi-arquitectura ~/t2 \$' is visible at the bottom.

```
alumno27@rpi-arquitectura: ~/t2
alumno27@rpi-arquitectura ~/t2 $ ./test.sh
running basic int const test...
ans: 2
running basic add test...
ans: 68
running basic sub test...
ans: 12
running basic if test...
ans: 2
running basic fun test...
ans: 6
running simple test...
ans: 7
running complex test...
ans: 10
alumno27@rpi-arquitectura ~/t2 $
```

Figura 1: Resultados de los Test