



DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE
CC4301 ARQUITECTURA DE COMPUTADORES

Tarea 2

Profesor:
Pablo Guerrero

Auxiliar:
Pablo Polanco

Alumno:
Gabriel Azócar C.

Fecha:
29 de Noviembre de 2016

Explicación de la solución

Para obtener las instrucciones, se utiliza casi el mismo código que fue creado para la Mini Tarea 6, agregando algunas modificaciones.

Primero, se define una variable global **actual**, que representa el número del registro de número índice disponible. Por ejemplo, si ya usamos el registro r0, entonces actual está en 1, para que el siguiente registro a usar sea r1, así sucesivamente. Se hace lo mismo para los labels de condiciones, labels de funciones y algunos saltos extras.

Luego se generan las líneas de rigor, como la definición del main. Además se agrega una línea que nos ayudará a imprimir en la consola el resultado final. El código a generar es el siguiente:

.LC0:

```
.string "%u\n"
.text
.global main
.type main,%function
```

Luego, como tenemos 3 posibles instrucciones, sumar, restar o agregar un número al stack, tenemos tres casos:

1. INT_CONST num: Esta instrucción hace un push del número entregado en el stack, por lo que debe tener la siguiente estructura:

```
mov rX ,#num
tmfd r13!, rX
```

Donde en la primera línea guardamos el número en el registro número X (donde X es el número del registro que sigue para hacer push, por ejemplo si ya se hizo push de 0 y 1, entonces X tendría que ser 2) para luego, en la segunda línea, hacerle push al stack.

2. ADD: Esta instrucción obtiene los dos primeros elementos del stack (los que están en el tope) y los suma, para luego hacer push del resultado. Por lo tanto, el código a generar sería:

```
ldmfd r13!, rX
ldmfd r13!, r(X-1)
add r(X-1), rX, r(X-1)
stmfd r13!, r(X-1)
```

Donde en las dos primeras líneas hacemos pop del stack, en la tercera se hace la suma y en la última se hace el push correspondiente.

3. SUB: Esta instrucción obtiene los dos primeros elementos del stack y los resta, para luego hacer push del resultado. Por lo tanto, el código a generar sería:

```
ldmfd r13!, rX
ldmfd r13!, r(X-1)
sub r(X-1), rX, r(X-1)
stmfd r13!, r(X-1)
```

Donde la explicación es totalmente simétrica con la suma, solo cambiando la operación.

4. FUN: Esta instrucción se comporta diferente dependiendo del contexto: Si se está en la "primera pasada" de compilación, o en la segunda. En la primera, se encarga de generar el label para poder identificar a la función y el cuerpo de esta. En la segunda, se encarga de que se haga push correctamente con su label respectivo. Para identificarse usa el mismo principio que los registros, usa el número disponible y se nombra ".LX", con X el número disponible.
5. IF0: Esta instrucción es la más larga del código. Basicamente lo que hace es generar dos label para cada una de sus ramas posibles de ejecución (.condX y cond(X+1), con X un número asignado). Luego, decide a cual saltar dependiendo de la comparación dada. Una vez estas condiciones terminan su ejecución, vuelven al código principal a través de otro label llamado ".regresoX", con X un número asignado.
6. APPLY: Esta instrucción lo que genera es una llamada a la función que se encuentra penultima en el stack, donde el tope del stack es el argumento a entregar. Luego de esto, apila el resultado al stack.
7. RETURN: Esta instrucción es la que marca el final de una función, por lo que se usa como punto de termino de las definiciones de estas y también ayuda para reemplazar por el nombre de la función las instrucciones en el programa principal.

Estos casos se encuentran en **escribir y escribirFun**, dependiendo del caso.

Luego, para finalizar el programa, se imprime el resultado y vacía la pila, además de definir un label para poder cargar el número como string en r0. Las instrucciones son:

```
mov r1 , r0
ldr r0, .L1
bl printf
ldmfd sp!, pc
.L1:
    .word .LC0
```

Donde cargamos el resultado con ayuda de .L1, que a su vez usa .LC0. Una vez terminado todo, se vacía la pila con la última instrucción.

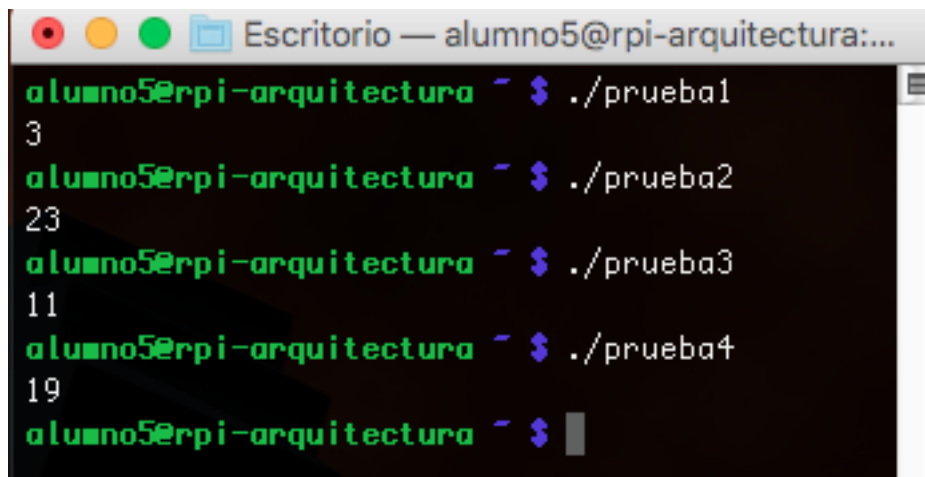
Pruebas

Las pruebas fueron las siguientes:

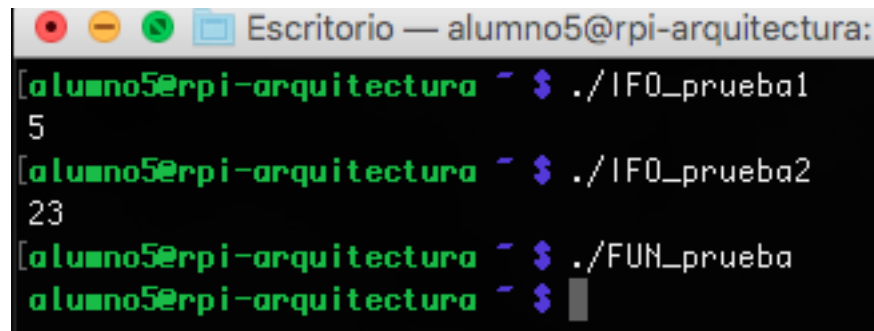
1. ((INT_CONST 1) (INT_CONST 2) (ADD)): Se generó el código que se muestra en el archivo adjunto **prueba1.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento de una suma simple.
2. ((INT_CONST 15) (INT_CONST 38) (SUB)): Se generó el código que se muestra en el archivo adjunto **prueba2.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento de una resta simple.
3. ((INT_CONST 3) (INT_CONST 8) (INT_CONST 6) (ADD) (SUB)): Se generó el código que se muestra en el archivo adjunto **prueba3.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento del ejemplo del enunciado.
4. ((INT_CONST 6) (INT_CONST 7) (INT_CONST 8) (INT_CONST 9) (ADD) (SUB) (ADD)): Se generó el código que se muestra en el archivo adjunto **prueba4.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento de un calculo un poco más largo.
5. ((INT_CONST 0) (IF0 ((INT_CONST 5)) ((INT_CONST 6)))): Se generó el código que se muestra en el archivo adjunto **IF0_prueba1.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento de un IF0 simple.
6. ((INT_CONST 1) (IF0 ((INT_CONST 5)) ((INT_CONST 6) (INT_CONST 7) (ADD))) (INT_CONST 10) (ADD)): Se generó el código que se muestra en el archivo adjunto **IF0_prueba2.s**. Esta prueba fue diseñada para mostrar el correcto funcionamiento de un IF0, cuando la comparación es falsa, las intrucciones son un poco más complejas y al finalizar se agrega una suma posterior.
7. ((FUN (INT_CONST 1) (INT_CONST 2) (ADD) (RETURN)) (APPLY)): Se generó el código que se muestra en el archivo adjunto **FUN_prueba.s**. Se trató de emular la prueba 1 de este listado, pero por alguna razón el resultado no se imprime, como se verá en la siguiente sección.

Resultados

A continuación se muestran las salidas de las pruebas anteriormente mencionadas, sacadas desde la Raspberry Pi que fue suministrada para la tarea. Se muestran las ejecuciones con su respectivo nombre. Se puede apreciar que FUN_prueba no imprime nada.



```
Escritorio — alumno5@rpi-arquitectura:~  
alumno5@rpi-arquitectura ~ $ ./prueba1  
3  
alumno5@rpi-arquitectura ~ $ ./prueba2  
23  
alumno5@rpi-arquitectura ~ $ ./prueba3  
11  
alumno5@rpi-arquitectura ~ $ ./prueba4  
19  
alumno5@rpi-arquitectura ~ $
```



```
Escritorio — alumno5@rpi-arquitectura:~  
[alumno5@rpi-arquitectura ~ $ ./IF0_prueba1  
5  
[alumno5@rpi-arquitectura ~ $ ./IF0_prueba2  
23  
[alumno5@rpi-arquitectura ~ $ ./FUN_prueba  
alumno5@rpi-arquitectura ~ $
```

Muestra de funcionamiento