

Instrucciones en el ARM (Segunda Parte)

Guillermo Steiner

Centro de Investigación en Informática para la Ingeniería
Universidad Tecnológica Nacional, F.R.C.

<http://ciiii.frc.utn.edu.ar>
Córdoba, Argentina



Técnicas Digitales II

Endianness

Endianness o Extremidad define el orden en que los bytes de un número son guardado en memoria, cuando el mismo supera el tamaño del byte.

De acuerdo a este orden, se establece tres configuraciones:

- Big-Endian 0x11223344 es guardado como {0x11,0x22,0x33,0x44}
- Little-Endian 0x11223344 es guardado como {0x44,0x33,0x22,0x11}
- Middle-Endian Adopta ambos formatos

Endianness

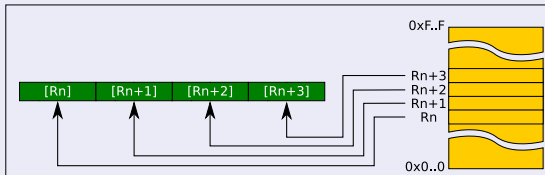
Endianness o Extremidad define el orden en que los bytes de un número son guardado en memoria, cuando el mismo supera el tamaño del byte.

De acuerdo a este orden, se establece tres configuraciones:

- Big-Endian 0x11223344 es guardado como {0x11,0x22,0x33,0x44}
- Little-Endian 0x11223344 es guardado como {0x44,0x33,0x22,0x11}
- Middle-Endian Adopta ambos formatos

Big-Endian

Se guardan en el orden natural, (el menos significativo aparece como el primer número en la secuencia), adoptado por Motorola



Endianness

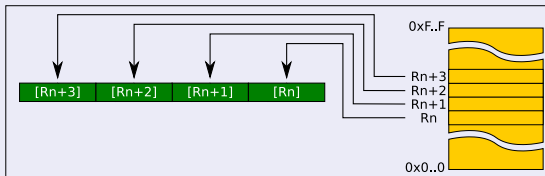
Endianness o Extremidad define el orden en que los bytes de un número son guardado en memoria, cuando el mismo supera el tamaño del byte.

De acuerdo a este orden, se establece tres configuraciones:

- Big-Endian 0x11223344 es guardado como {0x11,0x22,0x33,0x44}
- Little-Endian 0x11223344 es guardado como {0x44,0x33,0x22,0x11}
- Middle-Endian Adopta ambos formatos

Little-Endian

Se guardan de manera mas sencilla para su acceso (el menos significativo se encuentra en la posición mas baja), adoptado por INTEL



Endianness

Endianness o Extremidad define el orden en que los bytes de un número son guardado en memoria, cuando el mismo supera el tamaño del byte.

De acuerdo a este orden, se establece tres configuraciones:

- Big-Endian 0x11223344 es guardado como {0x11,0x22,0x33,0x44}
- Little-Endian 0x11223344 es guardado como {0x44,0x33,0x22,0x11}
- Middle-Endian Adopta ambos formatos

Middle-Endian

No es un formato espacial, suelen nombrarse así a las arquitecturas que aceptan ambos órdenes, entre estas arquitecturas se encuentran ARM, MIPS, PowePC, etc. En el caso de ARM, si bien la arquitectura acepta cualquier formato, la implementación suele realizarse con un solo tipo y en general es la Little-Endian

Instrucciones de Carga y Almacenamiento en Memoria

- Característico de la Arquitectura Load/Store, son las únicas instrucciones que acceden a memoria.
- Permiten leer desde memoria y cargar en registros (load).
- Permiten escribir el contenido de registros en la memoria (store).

Instrucciones de Carga y Almacenamiento en Memoria

Se dividen en dos grupos de instrucciones

Instrucciones de Carga y Almacenamiento de UN REGISTRO

Permite la carga y almacenamiento de un solo registro desde y hacia la memoria

Instrucciones de Carga y Almacenamiento de múltiples registros

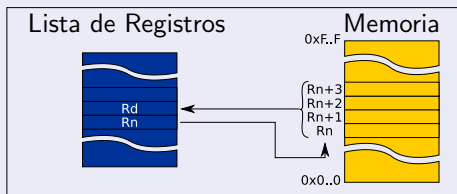
Permite la carga y almacenamiento de un listado de registros desde y hacia la memoria

Instrucciones de Carga y Almacenamiento de un registro

Carga

`LDR{cond}{B|SB|H|SH} Rd,direc`

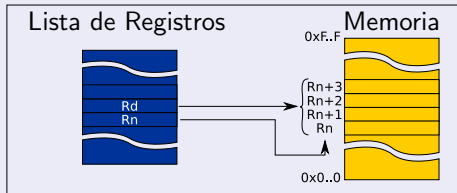
Leer desde la posición indicada en "direc" un word y lo guarda en el registro Rd



Almacenamiento

`STR{cond}{B|SB|H|SH} Rd,direc`

Escribe el contenido del registro Rd a partir de la posición indicada en "direc"



Modos de Direcccionamiento

Las instrucciones de carga y almacenamiento de un registro, requieren para definir la dirección de memoria donde se leerá o escribirá el dato por lo menos de un registro base

A este registro se le sumará o restará otro valor (offset), el tipo de este valor definirá el modo de direccionamiento:

- Direcccionamiento Inmediato. `LDR Rd, [Rn, #offset]`
- Offset por registro. `LDR Rd, [Rn, Rm]`
- Offset por registro escalado. `LDR Rd, [Rn, Rm, lsl #2]`

Modos de Direcccionamiento

Las instrucciones de carga y almacenamiento de un registro, requieren para definir la dirección de memoria donde se leerá o escribirá el dato por lo menos de un registro base

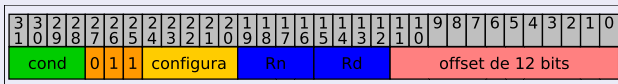
A este registro se le sumará o restará otro valor (offset), el tipo de este valor definirá el modo de direccionamiento:

- Direcccionamiento Inmediato. `LDR Rd, [Rn, #offset]`
- Offset por registro. `LDR Rd, [Rn, Rm]`
- Offset por registro escalado. `LDR Rd, [Rn, Rm, lsl #2]`

Inmediato

Al registro base se le suma o resta un valor numérico.

`LDR Rd, [Rn, #offset]` $Rd = [Rn + \#offset]$



Modos de Direcccionamiento

Las instrucciones de carga y almacenamiento de un registro, requieren para definir la dirección de memoria donde se leerá o escribirá el dato por lo menos de un registro base

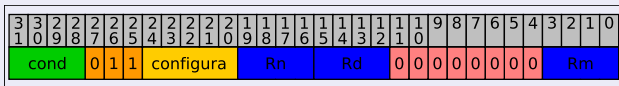
A este registro se le sumará o restará otro valor (offset), el tipo de este valor definirá el modo de direccionamiento:

- Direcccionamiento Inmediato. LDR Rd, [Rn, #offset]
- Offset por registro. LDR Rd, [Rn, Rm]
- Offset por registro escalado. LDR Rd, [Rn, Rm, lsl #2]

Offset por registro

Al registro base se le suma o resta otro registro.

LDR Rd, [Rn, Rm] $Rd = [Rn + Rm]$



Modos de Direcccionamiento

Las instrucciones de carga y almacenamiento de un registro, requieren para definir la dirección de memoria donde se leerá o escribirá el dato por lo menos de un registro base

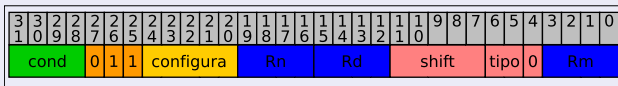
A este registro se le sumará o restará otro valor (offset), el tipo de este valor definirá el modo de direccionamiento:

- Direcccionamiento Inmediato. LDR Rd, [Rn, #offset]
- Offset por registro. LDR Rd, [Rn, Rm]
- Offset por registro escalado. LDR Rd, [Rn, Rm lsl #2]

Offset por registro escalado

Al registro base se le suma o resta un registro escalado (barrel shifter).

LDR Rd, [Rn, Rm lsl #2] $Rd = [Rn + Rm \cdot 4]$



Tipos de Direccionamiento

El tipo vínculo entre el registro base y offset se pueden vincular de varias maneras, esto define el tipo de direccionamiento.

- Direccionamiento por corrimiento. `LDR Rd, [Rn, #offset]`
- Direccionamiento pre-indexado. `LDR Rd, [Rn, #offset] !`
- Direccionamiento post-indexado. `LDR Rd, [Rn], #offset`

Tipos de Direcccionamiento

El tipo vinculo entre el registro base y offset se pueden vincular de varias maneras, esto define el tipo de direccionamiento.

- Direccionamiento por corrimiento. `LDR Rd, [Rn, #offset]`
- Direccionamiento pre-indexado. `LDR Rd, [Rn, #offset] !`
- Direccionamiento post-indexado. `LDR Rd, [Rn], #offset`

Direccionamiento por corrimiento

La dirección de memoria está formada por la suma o resta del corrimiento con el registro base.

`LDR Rd, [Rn, #offset]`

Luego de la operación los registros quedan:

`Rd = [Rn + #offset]`

`Rn = Rn`

Tipos de Direcccionamiento

El tipo vinculo entre el registro base y offset se pueden vincular de varias maneras, esto define el tipo de direccionamiento.

- Direccionamiento por corrimiento. `LDR Rd, [Rn, #offset]`
- Direccionamiento pre-indexado. `LDR Rd, [Rn, #offset] !`
- Direccionamiento post-indexado. `LDR Rd, [Rn], #offset`

Direccionamiento pre-indexado

La dirección de memoria está formada de la misma forma que la anterior, pero esta dirección de memoria calculada, es escrita luego en el registro base.

`LDR Rd, [Rn, #offset] !`

Luego de la operación los registros quedan:

`Rd = [Rn + #offset]`

`Rn = Rn + #offset`

Tipos de Direccionamiento

El tipo vínculo entre el registro base y offset se pueden vincular de varias maneras, esto define el tipo de direccionamiento.

- Direccionamiento por corrimiento. `LDR Rd, [Rn, #offset]`
- Direccionamiento pre-indexado. `LDR Rd, [Rn, #offset] !`
- Direccionamiento post-indexado. `LDR Rd, [Rn], #offset`

Direccionamiento post-indexado

Se toma el dato de la dirección apuntada por el registro base y luego se actualiza este sumándole o restándole el corrimiento.

`LDR Rd, [Rn], #offset`

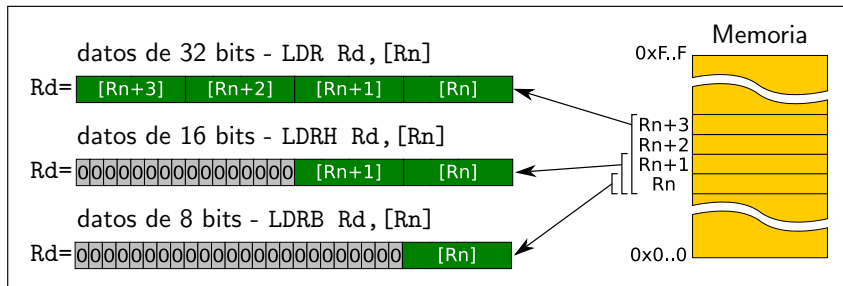
Luego de la operación los registros quedan:

`Rd = [Rn]`

`Rn = Rn + #offset`

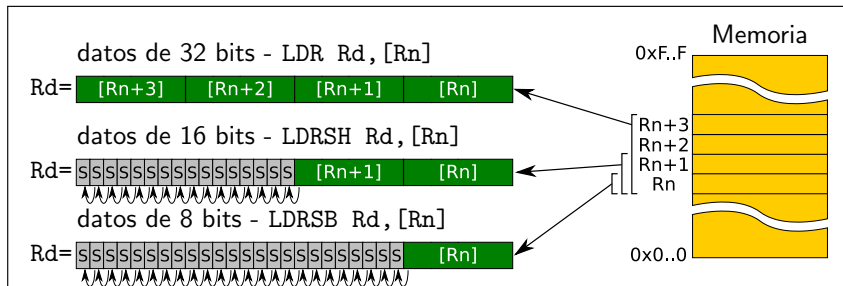
Instrucción LDR - tipo de dato

La instrucción de carga desde memoria a registro, admite tres tipos distinto de tamaño de dato.



Instrucción LDR - tipo de dato

El mismo conjunto de tipos de datos, será tratado de distinta forma si son números CON SIGNO.



Instrucción STR - tipo de dato

La instrucción para guardar de un registro a memoria, admite los mismos tipos de datos que para la carga.

En formatos menores a 32 bits, el dato es recortado al tamaño adecuado, este procedimiento no requiere conocer si el número es con signo o sin signo

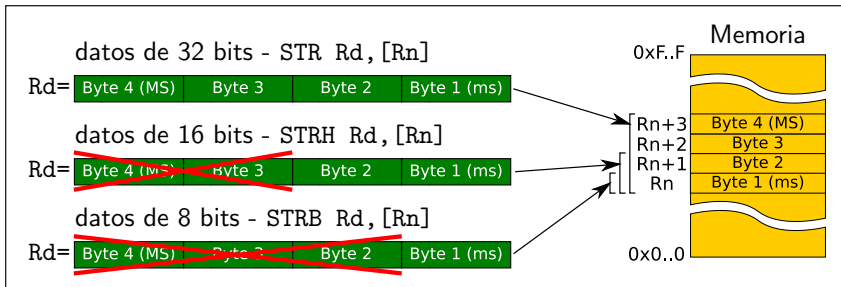


Tabla de Direcccionamiento

Tabla

De acuerdo a los modos de direccionamiento y a los tipos de direccionamientos, se define la siguiente tabla.

- WB - Word o Byte sin signo.
- HB - Halfword con y sin signo o Byte con signo.

T.Dato	Modo	Descripción	Ejemplo
WB	[<Rn>, #+/-<o_12>]	inmediato	LDR R1, [R2, #+1024]
HB	[<Rn>, #+/-<o_8>]	inmediato	LDRH R1, [R2, #+100]
todos	[<Rn>, +/-<Rm>]	por registro	LDR R1, [R2, R3]
WB	[<Rn>, +/-<Rm>, <sh>#<imm>]	por registro escalado	LDR R1, [R2, -R4, LSL #3]
WB	[<Rn>, #+/-<o_12>]	inmediato pre indexado	LDR R1, [R2, #-1024] !
HB	[<Rn>, #+/-<o_8>]	inmediato pre indexado	LDRH R1, [R2, #+100] !
todos	[<Rn>, +/-<Rm>]	por registro pre indexado	LDR R1, [R2, -R4] !
WB	[<Rn>, +/-<Rm>, <sh>#<imm>]	por reg. escalado pre-ind.	LDR R2, [R4, R3, LSL #3] !
WB	[<Rn>], #+/-<o_12>	inmediato post-indexado	LDR R1, [R2], #+1024
HB	[<Rn>], #+/-<o_8>	inmediato post-indexado	LDRSB R1, [R2], #+100
todos	[<Rn>], +/-<Rm>	por registro post-indexado	LDR R1, [R2], R3
WB	[<Rn>], +/-<Rm>, <sh>#<imm>	por reg. escalado post-ind.	LDR R1, [R2], R3, LSL #2

Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

```

reset:  mov r1,#VALOR    20: e3a01034    mov    r1, #52 ; 0x34
        ldr r2,[r1]      24: e5912000    ldr     r2, [r1]

        ldr r1,=VALOR    28: e59f1008    ldr     r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]      2c: e5912000    ldr     r2, [r1]

loop:   b loop           30: eaffffff    b       30 <loop>

00000034 <VALOR>:       34: 00000064    .word   0x00000064

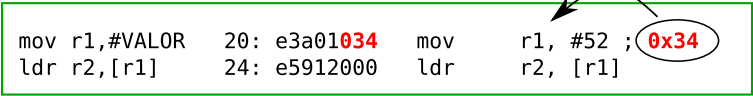
                               38: 00000034    .word   0x00000034
  
```

Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

Un MOV es utilizado para asignada a un registro la posición de la variable, la cual es guardada en la misma instrucción MOV como un literal.



```

reset:  mov r1,#VALOR    20: e3a01034    mov r1, #52 ; 0x34
        ldr r2,[r1]     24: e5912000    ldr r2, [r1]

        ldr r1,=VALOR   28: e59f1008    ldr r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]     2c: e5912000    ldr r2, [r1]

loop:   b loop          30: eaffffff    b 30 <loop>

00000034 <VALOR>:      34: 00000064    .word 0x00000064

                          38: 00000034    .word 0x00000034
  
```

Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

El registro con la posición de la variable, es utilizado como registro base en un LDR leyendo de esta forma el contenido de la variable.

```

reset:  mov r1,#VALOR    20: e3a01034    mov r1, #52 ; 0x34
        ldr r2,[r1]      24: e5912000    ldr r2, [r1]

        ldr r1,=VALOR    28: e59f1008    ldr r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]      2c: e5912000    ldr r2, [r1]

loop:   b loop           30: eaffffff    b 30 <loop>

00000034 <VALOR>:      34: 00000064    .word 0x00000064
                        38: 00000034    .word 0x00000034
  
```

Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

El LDR accede al puntero de la variable guardada en una posición asignada por el compilador.

```

reset:  mov r1,#VALOR    20: e3a01034    mov     r1, #52 ; 0x34
        ldr r2,[r1]      24: e5912000    ldr     r2, [r1]

        ldr r1,=VALOR    28: e59f1008    ldr     r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]      2c: e5912000    ldr     r2, [r1]

loop:   b loop           30: eaffffff    b       30 <loop>

00000034 <VALOR>:      34: 00000064    .word   0x00000064
        38: 00000034    .word   0x00000034
  
```


Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

El cálculo de la posición será $pc + 8 = (0x28 + 8) + 8 = 40 + 16 = 56 = 0x38$

```

reset:  mov r1,#VALOR    20: e3a01034    mov     r1, #52 ; 0x34
        ldr r2,[r1]      24: e5912000    ldr     r2, [r1]

        ldr r1,=VALOR    28: e59f1008    ldr     r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]      2c: e5912000    ldr     r2, [r1]

loop:   b loop           30: eaffffff    b       30 <loop>

00000034 <VALOR>:       34: 00000064    .word   0x00000064
        38: 00000034    .word   0x00000034
  
```

Puntero a variable

Las etiquetas son utilizadas como referencia a la posición de una variable. Podemos acceder entonces de dos maneras diferentes

- Utilizando MOV
- Utilizando LDR

El registro con la posición de la variable, es utilizado como registro base en un LDR leyendo de esta forma el contenido de la variable.

```

reset:  mov r1,#VALOR    20: e3a01034    mov    r1, #52 ; 0x34
        ldr r2,[r1]      24: e5912000    ldr     r2, [r1]

        ldr r1,=VALOR    28: e59f1008    ldr     r1, [pc, #8]; 38<VALOR+4>
        ldr r2,[r1]      2c: e5912000    ldr     r2, [r1]

loop:   b loop           30: eaffffff    b       30 <loop>

00000034 <VALOR>:      34: 00000064    .word   0x00000064
                        38: 00000034    .word   0x00000034
  
```

Instrucciones de Carga y Almacenamiento Múltiple

Carga

LDM Rd,{listado de registros}

Cargar una serie de registros de propósitos generales desde la memoria.

Almacenamiento

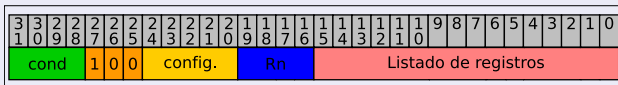
STM Rd,{listado de registros}

Almacenar una serie de registros de propósitos generales en la memoria

Instrucciones de Carga y Almacenamiento Múltiple

Estas Instrucciones poseen el siguiente formato:

LDM|STM{cond}modo_direccion Rn{!},Listado de Registros



Donde:

- modo_direccion es el modo como se guardarán o leerán los datos.
- Rn el registro base de donde se obtiene la dirección inicial a guardar o leer los datos.
- Signo !, su presencia indica que al finalizar la instrucción Rn será actualizado con el valor final de memoria.
- Listado de Registros, sub conjunto de los 16 registros generales.

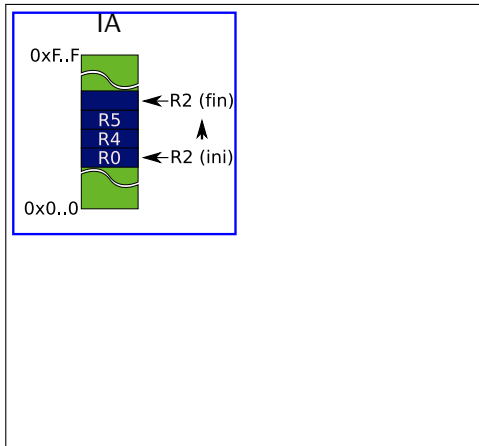
Modo de dirección

Indica de que forma los datos se irán leyendo o guardando en la memoria.
Se tiene 4 maneras distintas de realizar la operación

- IA(Increment After)
- IB(Increment Before)
- DA(Decrement After)
- DB(Decrement Before)

STMIA R2!, {R0,R4,R5}

El menor registro se lee o guarda en R2, luego los registros posteriores operarán en posiciones contiguas incrementadas de a 4 bytes.



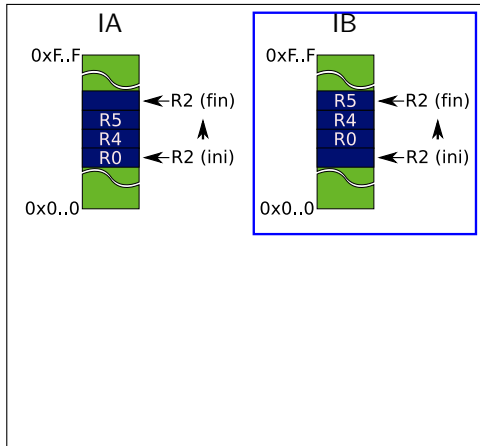
Modo de dirección

Indica de que forma los datos se irán leyendo o guardando en la memoria.
Se tiene 4 maneras distintas de realizar la operation

- IA(Increment After)
- IB(Increment Before)
- DA(Decrement After)
- DB(Decrement Before)

STMIB R2!, {R0,R4,R5}

El menor registro se lee o guarda en $R2 + 4$, los registros posteriores operarán en posiciones contiguas incrementadas de a 4 bytes.



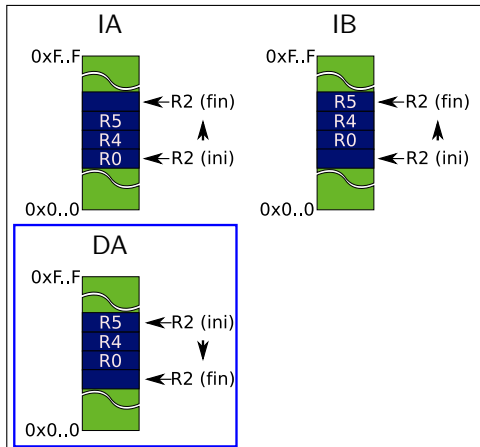
Modo de dirección

Indica de que forma los datos se irán leyendo o guardando en la memoria.
Se tiene 4 maneras distintas de realizar la operation

- IA(Increment After)
- IB(Increment Before)
- DA(Decrement After)
- DB(Decrement Before)

STMDA R2!, {R0,R4,R5}

El registro mas alto se lee o guarda en R2, luego los registros anteriores operarán en posiciones contiguas decrementadas de a 4 bytes.



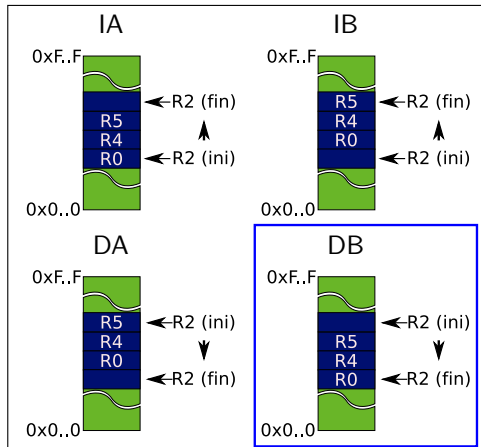
Modo de dirección

Indica de que forma los datos se irán leyendo o guardando en la memoria.
Se tiene 4 maneras distintas de realizar la operación

- IA(Increment After)
- IB(Increment Before)
- DA(Decrement After)
- DB(Decrement Before)

STMDB R2!, {R0,R4,R5}

El registro mas alto se lee o guarda en R2 - 4, luego los registros anteriores operarán en posiciones contiguas decrementadas de a 4 bytes.

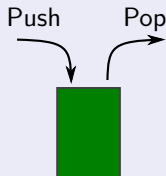


STACK

La utilización de las instrucciones STM y LDM para el manejo de bloques de datos es solo una aplicación, estas instrucciones se utilizan además para las operaciones en el STACK

El STACK es una porción de memoria en la cual se encuentra una lista ordenada que permite almacenar y recuperar datos, esa lista es de tipo LIFO (Last Input First Output).

Posee dos funciones principales PUSH (colocar algo en la pila) POP (Sacarlo)



STACK

En un programa, este tipo de estructuras es utilizada para almacenar datos temporarios y luego recuperarlos.

Durante la ejecución de un sistema, el uso del Stack se puede presentar en diferentes situaciones.

Ejemplos

- Cuando una función llama a otra, guardar el estado de la primera para recuperar su estado original a la salida de la segunda.
- Variables Locales de una función.
- Pase de parámetros a una función.

La arquitectura ARM, utiliza las instrucciones LDM y STM para implementar 4 modelos diferentes de STACK.

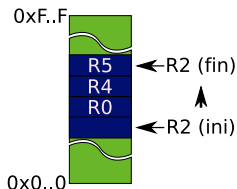
Cada uno de estos STACK, difiere en la manera de como se actualiza el puntero luego de cada lectura o almacenamiento de datos.

Podemos encontrar entonces:

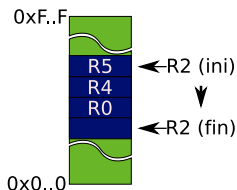
- FA(Full Ascending)
- EA(Empty Ascending)
- FD(Full Descending)
- ED(Empty Descending)

El menor registro es guardado en $R2 + 4$, y los demás registro en posiciones posteriores. La lectura comienza del registro mas alto apuntado por R2, y los demás en posiciones anteriores.

STMFA R2!, {R0,R4,R5} (IB)



LDMFA R2!, {R0,R4,R5} (DA)



La arquitectura ARM, utiliza las instrucciones LDM y STM para implementar 4 modelos diferentes de STACK.

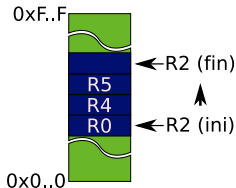
Cada uno de estos STACK, difiere en la manera de como se actualiza el puntero luego de cada lectura o almacenamiento de datos.

Podemos encontrar entonces:

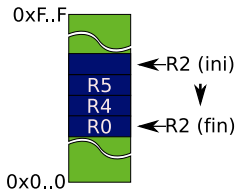
- FA(Full Ascending)
- EA(Empty Ascending)
- FD(Full Descending)
- ED(Empty Descending)

El menor registro es guardado en R2, y los demás registro en posiciones posteriores. La lectura comienza del registro mas alto apuntado por R2 - 4, y los demás en posiciones anteriores.

STMEA R2!, {R0,R4,R5} (IA)



LDMEA R2!, {R0,R4,R5} (DB)



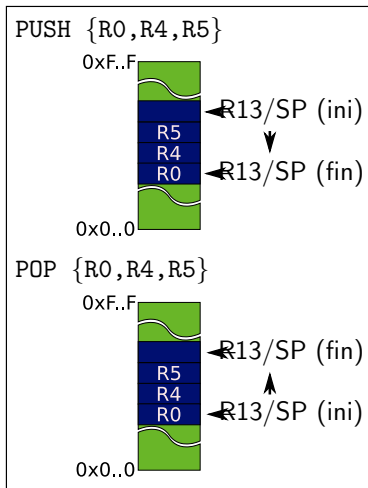
La arquitectura ARM, utiliza las instrucciones LDM y STM para implementar 4 modelos diferentes de STACK.

Cada uno de estos STACK, difiere en la manera de como se actualiza el puntero luego de cada lectura o almacenamiento de datos.

Podemos encontrar entonces:

- FA(Full Ascending)
- EA(Empty Ascending)
- FD(Full Descending)
- ED(Empty Descending)

El mayor registro es guardado en R2 - 4, y los demás registro en posiciones anteriores. La lectura comienza del registro mas bajo apuntado por R2, y los demás en posiciones posteriores.



La arquitectura ARM, utiliza las instrucciones LDM y STM para implementar 4 modelos diferentes de STACK.

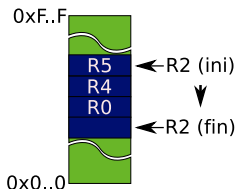
Cada uno de estos STACK, difiere en la manera de como se actualiza el puntero luego de cada lectura o almacenamiento de datos.

Podemos encontrar entonces:

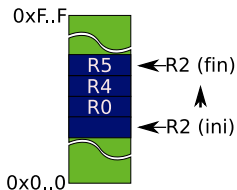
- FA(Full Ascending)
- EA(Empty Ascending)
- FD(Full Descending)
- ED(Empty Descending)

El mayor registro es guardado en R2, y los demás registro en posiciones anteriores. La lectura comienza del registro mas bajo apuntado por $R2 + 4$, y los demás en posiciones posteriores.

STMED R2!, {R0,R4,R5} (DA)



LDMED R2!, {R0,R4,R5} (IB)



STACK Push y Pop

Las Instrucciones clásicas PUSH y POP para el manejo del Stack, son en ARM un caso especial de STM y LDM, pero no son instrucciones nuevas.

Las instrucciones LDM y STM configuradas como:

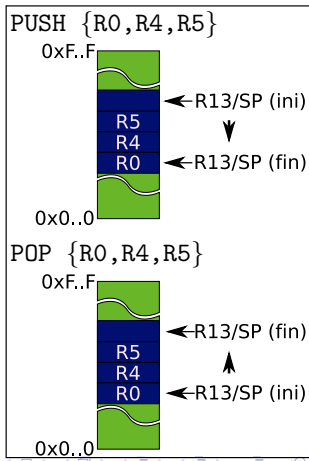
- Tipo de Stack FULL DESCENDING.
- Registro puntero R13.

Pueden ser reemplazada por PUSH y POP.

STMF_D R13!, {R0,R4,R5} por PUSH {R0,R4,R5}
 LDMF_D R13!, {R0,R4,R5} por POP {R0,R4,R5}

Finalmente

- FULL DESCENDING es el modo por defecto.
- el R13 pasa a llamarse SP y por esta condición es guardado en los cambios de contexto.



Manejo de Funciones en Ensamblador

Utilizar funciones en ensamblador, requiere aplicar las siguientes funcionalidades de la arquitectura ARM

- Salto con link (BL etiqueta).
- Utilización del Stack para guardar estados, pase de parámetros y variables locales.

La instrucción BL es idéntica a la instrucción branch (B) vista anteriormente, a excepción de que antes de saltar guarda el valor del registro PC en el registro R14/LR

Esto permite conservar la dirección de retorno para luego con una simple copia de este registro al PC retornar a la posición desde donde se realizó el llamado y continuar con el programa.

Manejo de Funciones en Ensamblador

Utilizar funciones en ensamblador, requiere aplicar las siguientes funcionalidades de la arquitectura ARM

- Salto con link (BL etiqueta).
- Utilización del Stack para guardar estados, pase de parámetros y variables locales.

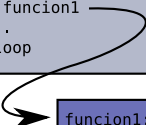
El uso del STACK, como fue desarrollado anteriormente, permite

- Guardar el estado de los registros.
- Variables Locales de una función.
- Pase de parámetros a una función.

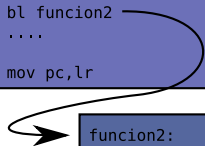
STACK

Utilización del STACK para guardar el estado de los registros.

```
reset: @ uso de registros r5 y r6
      ....
      bl funcion1
      ....
loop:  b loop
```



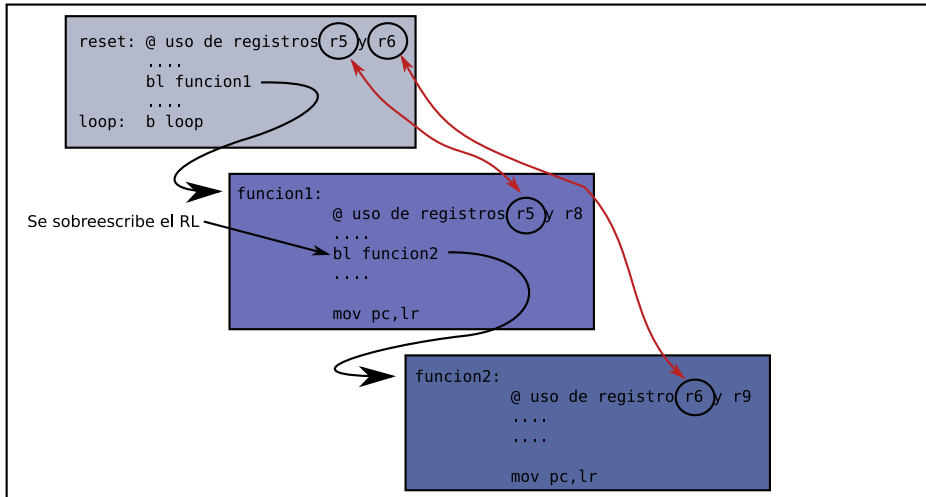
```
funcion1:
      @ uso de registros r5 y r8
      ....
      bl funcion2
      ....
      mov pc,lr
```



```
funcion2:
      @ uso de registro r6 y r9
      ....
      ....
      mov pc,lr
```

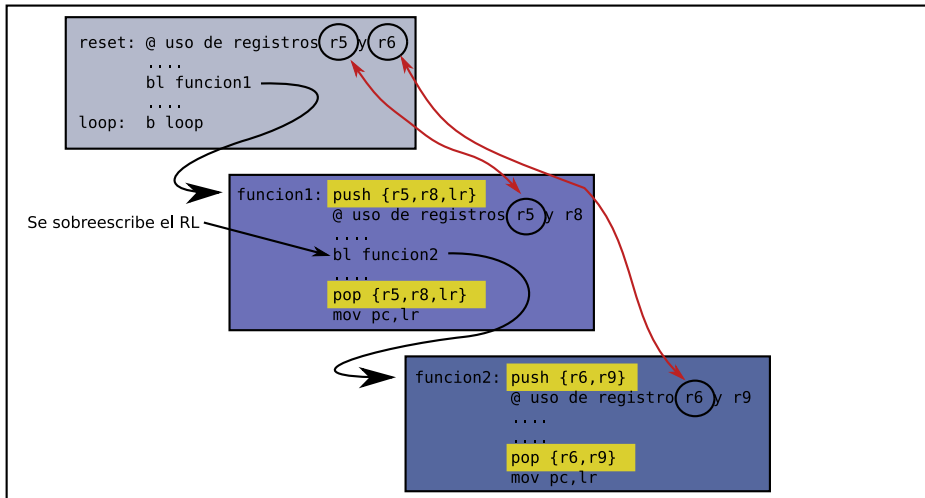
STACK

Utilización del STACK para guardar el estado de los registros.



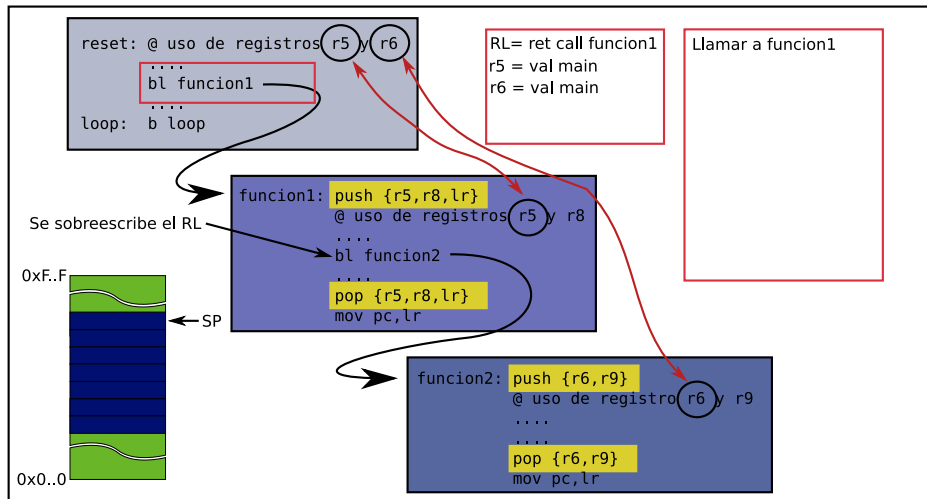
STACK

Utilización del STACK para guardar el estado de los registros.



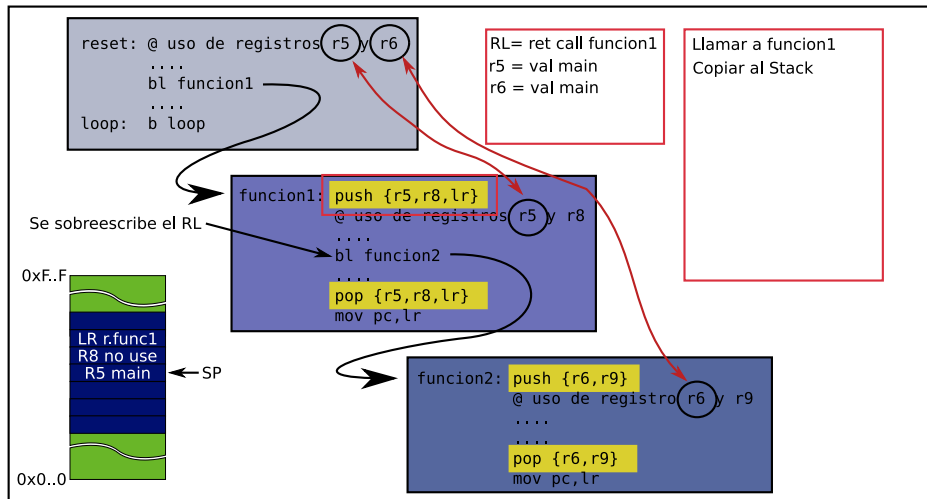
STACK

Utilización del STACK para guardar el estado de los registros.



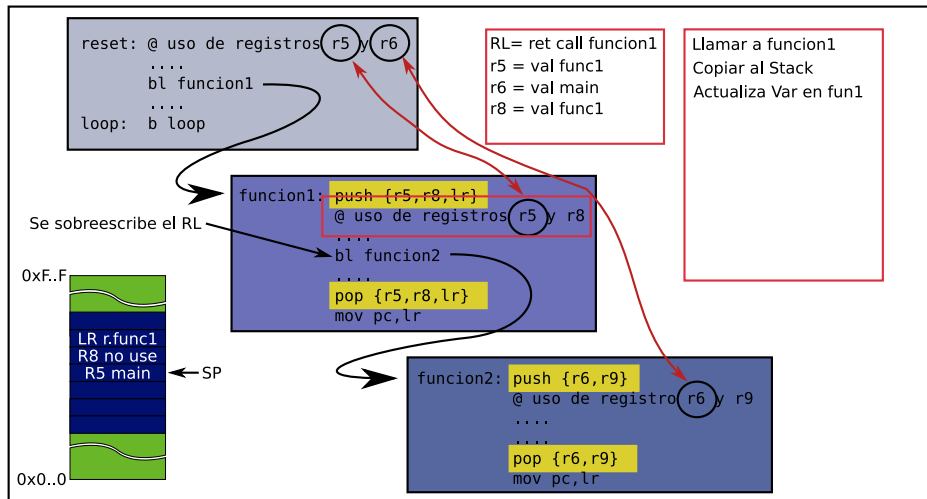
STACK

Utilización del STACK para guardar el estado de los registros.



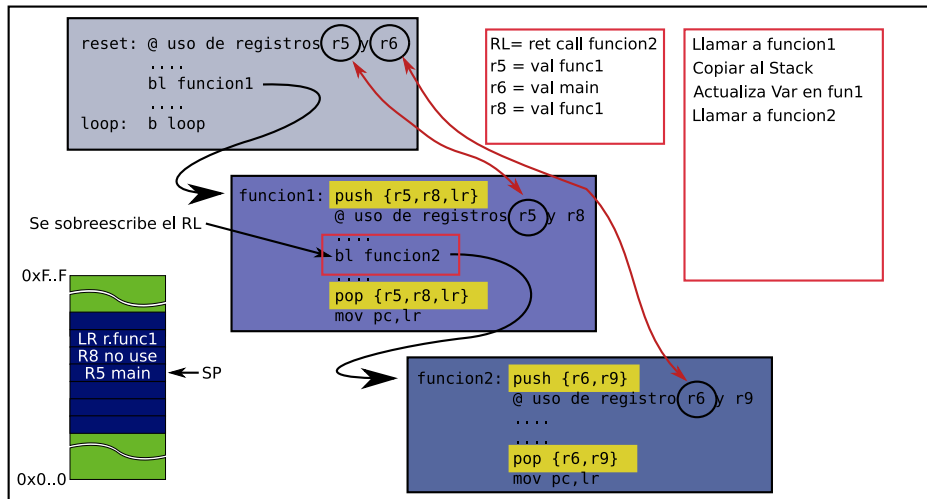
STACK

Utilización del STACK para guardar el estado de los registros.



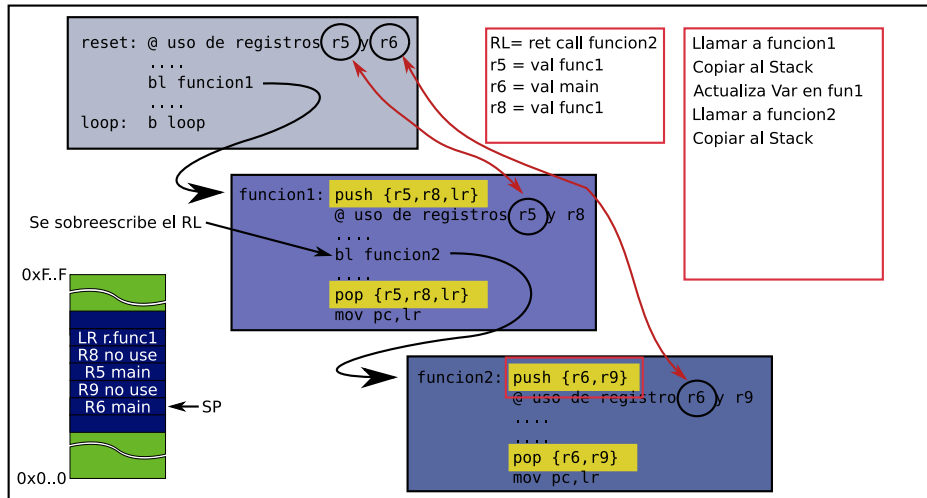
STACK

Utilización del STACK para guardar el estado de los registros.



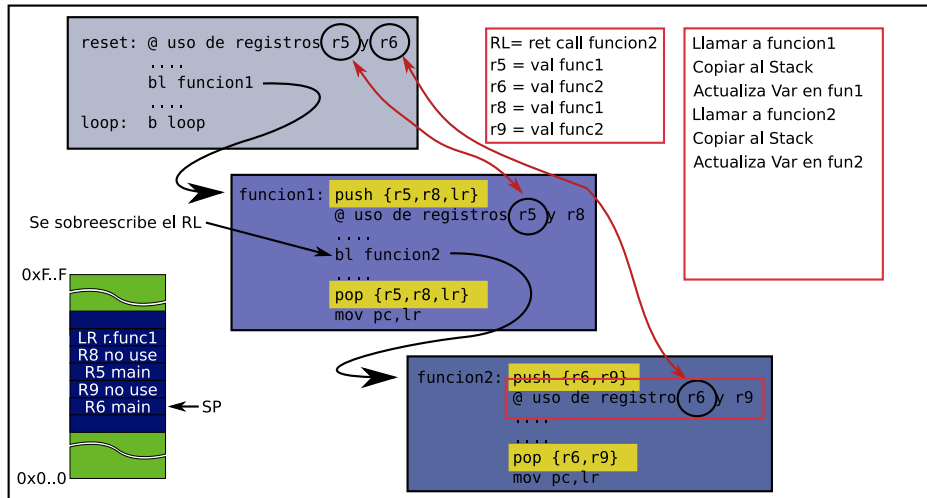
STACK

Utilización del STACK para guardar el estado de los registros.



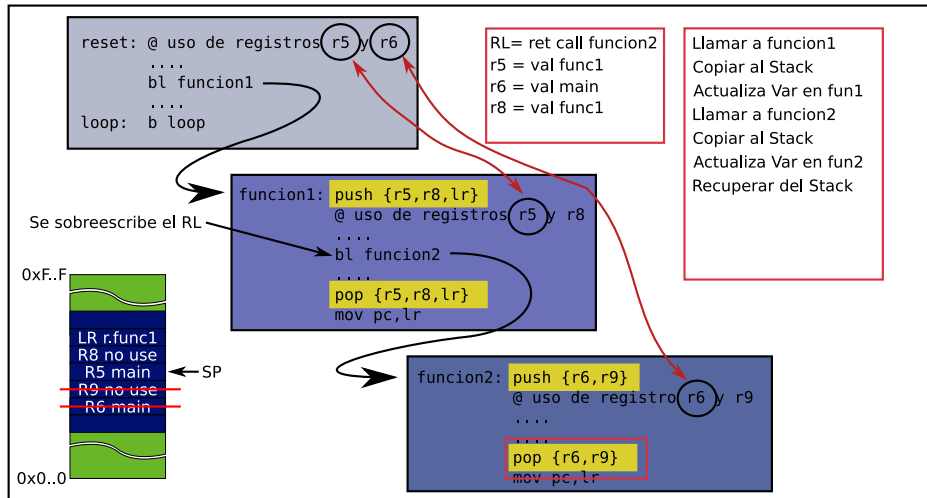
STACK

Utilización del STACK para guardar el estado de los registros.



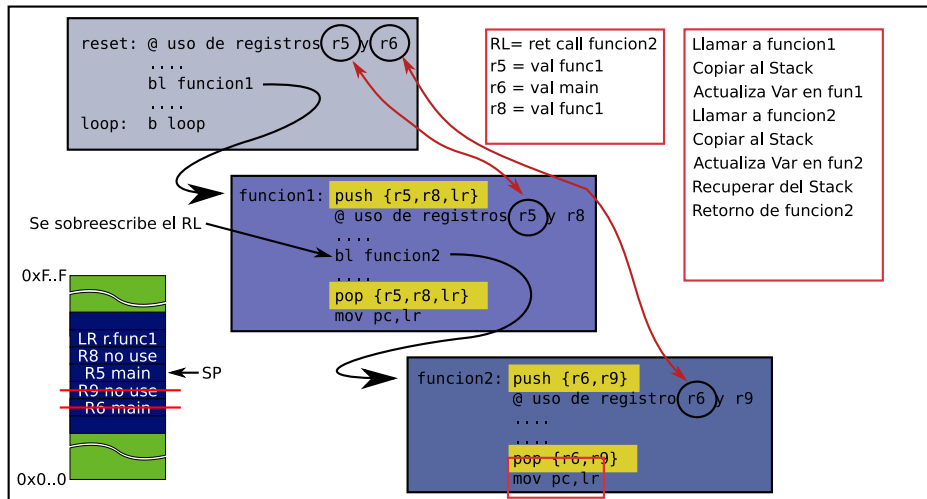
STACK

Utilización del STACK para guardar el estado de los registros.



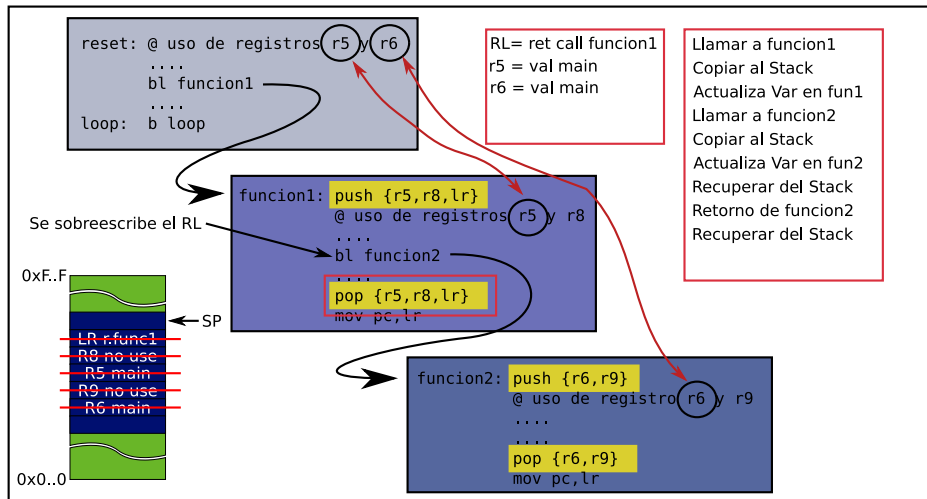
STACK

Utilización del STACK para guardar el estado de los registros.



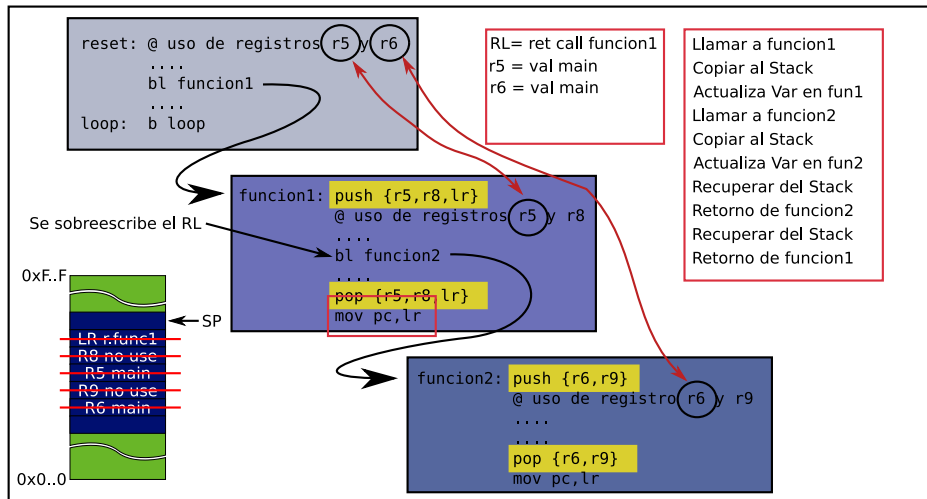
STACK

Utilización del STACK para guardar el estado de los registros.



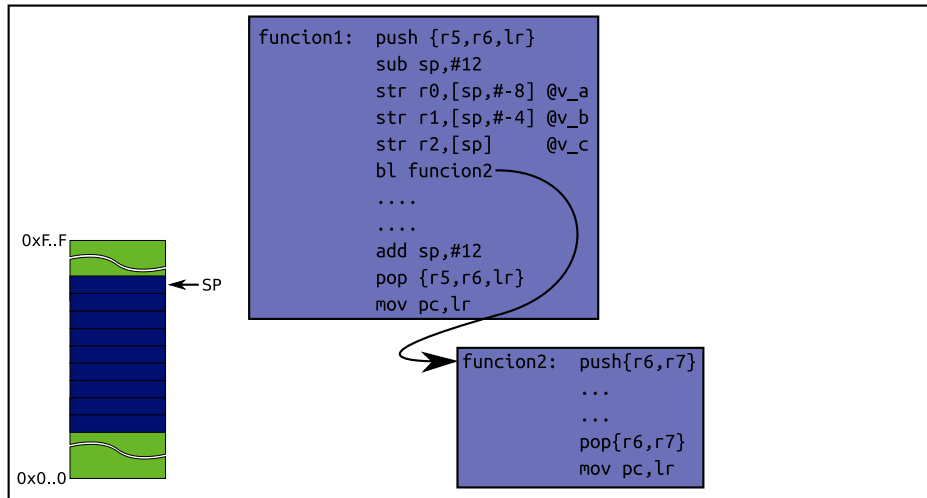
STACK

Utilización del STACK para guardar el estado de los registros.



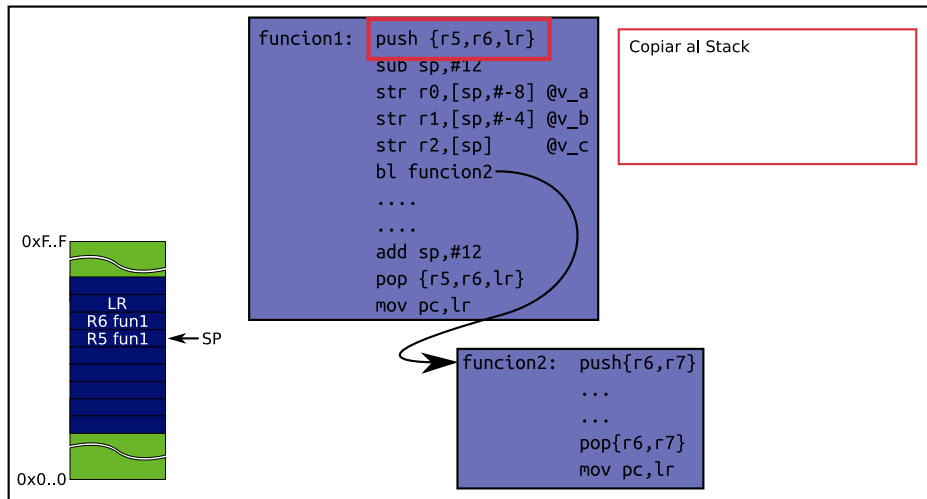
STACK

Variables Locales



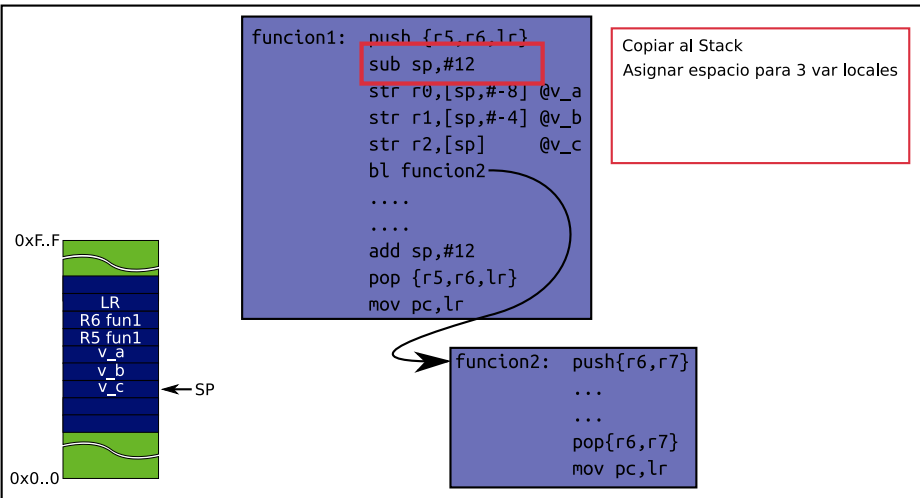
STACK

Variables Locales



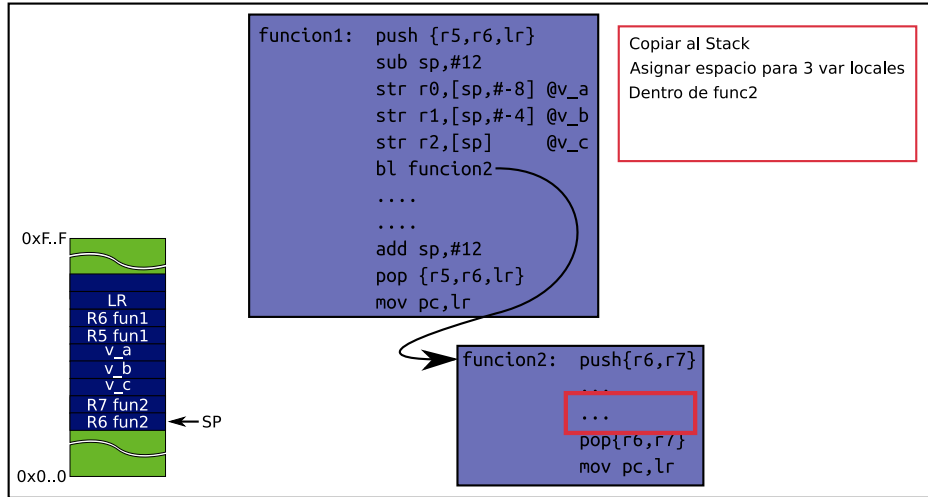
STACK

Variables Locales



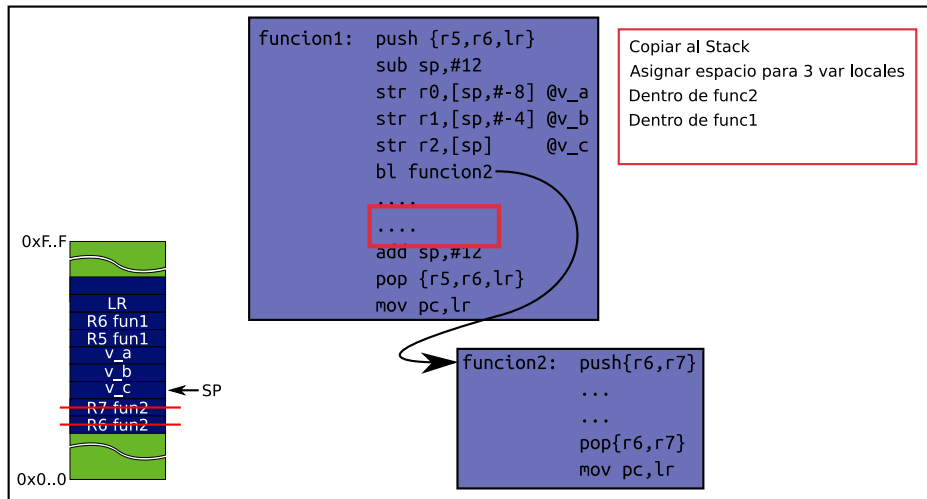
STACK

Variables Locales



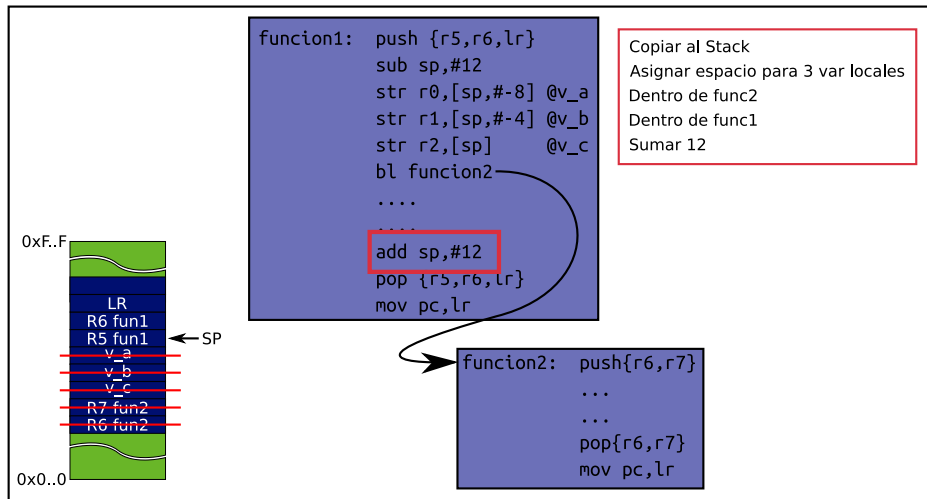
STACK

Variables Locales



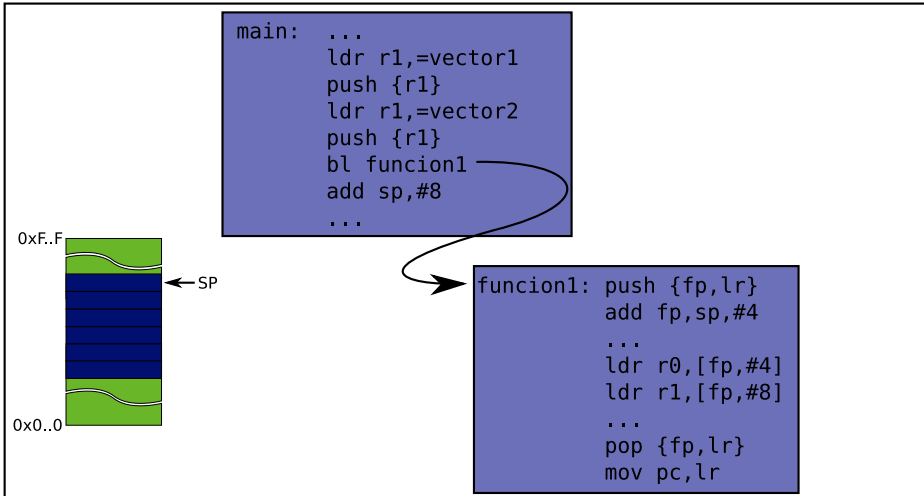
STACK

Variables Locales



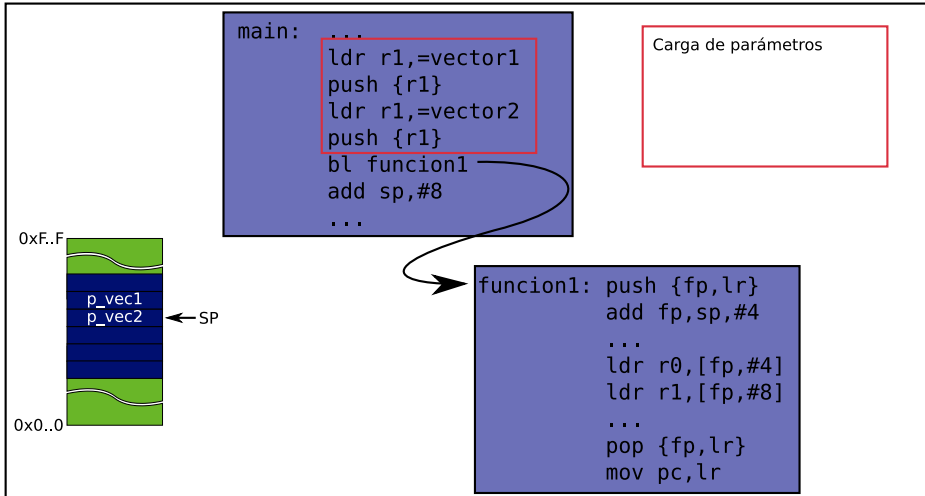
STACK

Pase de Parámetros



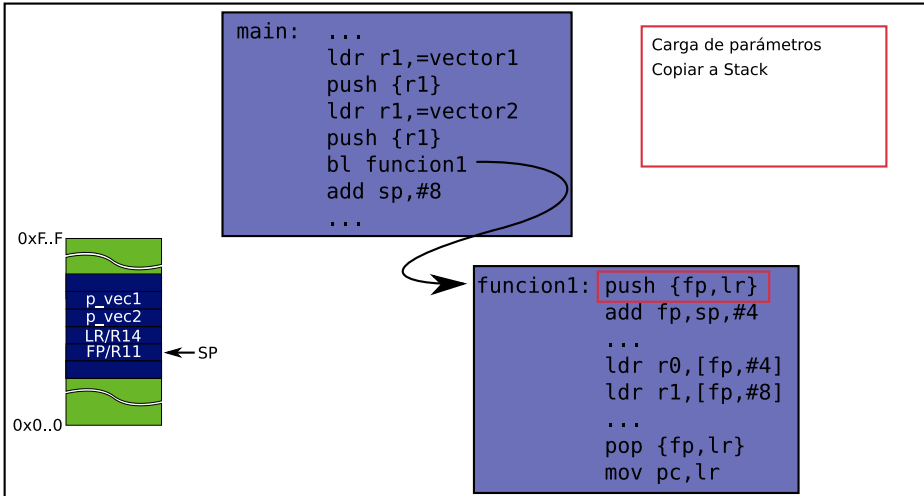
STACK

Pase de Parámetros



STACK

Pase de Parámetros

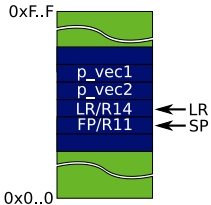


STACK

Pase de Parámetros

```
main:  ...
        ldr r1,=vector1
        push {r1}
        ldr r1,=vector2
        push {r1}
        bl funcion1
        add sp,#8
        ...
```

Carga de parámetros
Copiar a Stack
Asignar a FP base de param



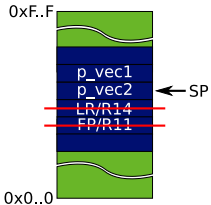
```
funcion1: push {fp,lr}
           add fp,sp,#4
           ...
           ldr r0,[fp,#4]
           ldr r1,[fp,#8]
           ...
           pop {fp,lr}
           mov pc,lr
```


STACK

Pase de Parámetros

```
main:  ...
        ldr r1,=vector1
        push {r1}
        ldr r1,=vector2
        push {r1}
        bl funcion1
        add sp,#8
        ...
```

Carga de parámetros
Copiar a Stack
Asignar a FP base de param
Salir de funcion1



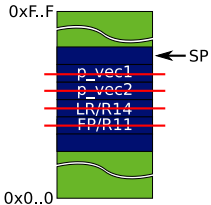
```
funcion1: push {fp,lr}
          add fp,sp,#4
          ...
          ldr r0,[fp,#4]
          ldr r1,[fp,#8]
          ...
          pop {fp,lr}
          mov pc,lr
```

STACK

Pase de Parámetros

```
main:  ...
      ldr r1,=vector1
      push {r1}
      ldr r1,=vector2
      push {r1}
      bl funcion1
      add sp,#8
      ...
```

Carga de parámetros
Copiar a Stack
Asignar a FP base de param
Salir de funcion1
Recuperar espacio param



```
funcion1: push {fp,lr}
          add fp,sp,#4
          ...
          ldr r0,[fp,#4]
          ldr r1,[fp,#8]
          ...
          pop {fp,lr}
          mov pc,lr
```

STACK

APCS (ARM Procedure Call Standard)

- El retorno de un valor se realiza siempre con el registro R0 (scratch register).
- Los Registros R0-R3 no requieren que sean salvados entre funciones.
- Los Registros R4-R11 deben ser salvados.
- Cuando se envían parámetros los primeros 4 se envían con los registros R0 a R3