

Sistemas Empotrados

Tema 2: Procesador y mapa de memoria

Lección 6:
Introducción al repertorio de instrucciones de la arquitectura ARMv4T



Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Características de la arquitectura ARM

- Application Binary Interface* de la arquitectura ARM

- El hola mundo de un sistema empotrado

Repertorio de instrucciones de la arquitectura ARMv4T

- Instrucciones de procesamiento de datos

- Instrucciones de salto

- Instrucciones de carga y almacenamiento

- Interrupciones software

- Otras instrucciones

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Transferencia entre registros

Copia un registro o un valor inmediato en otro registro

Sintaxis

<instrucción>{<cond>}{s} Rd, N

Rd: registro de destino

N: registro o valor inmediato

Instrucciones

mov	Mueve un valor de 32 bits a un registro	$Rd \leftarrow N$
mvn	Mueve el complemento de un valor de 32 bits a un registro	$Rd \leftarrow \sim N$

Ejemplos de uso

```
mov    r7, r5    @ r7 <- r5
```

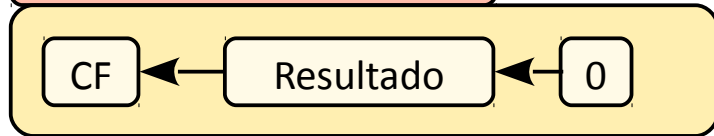
```
movs   r3, #4    @ r3 <- 4  
        @ afecta a los flags
```

```
mvngt r1, #0    @ r1 <- -1 si los  
                @ flags indican  
                @ que el último  
                @ resultado fue >0
```

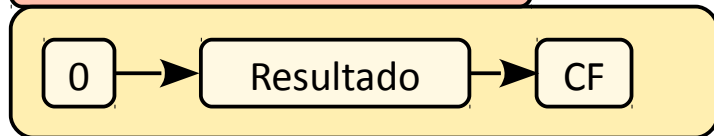
```
mvnles r2, r9   @ r2 <- ~r9 si los flags  
                @ indican que el último  
                @ resultado fue <=0  
                @ afecta a los flags
```

El desplazador de bloques

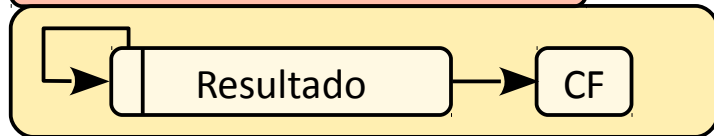
LSL (Logical Shift Left)



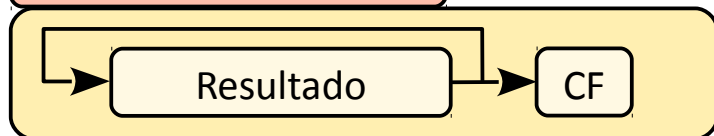
LSR (Logical Shift Right)



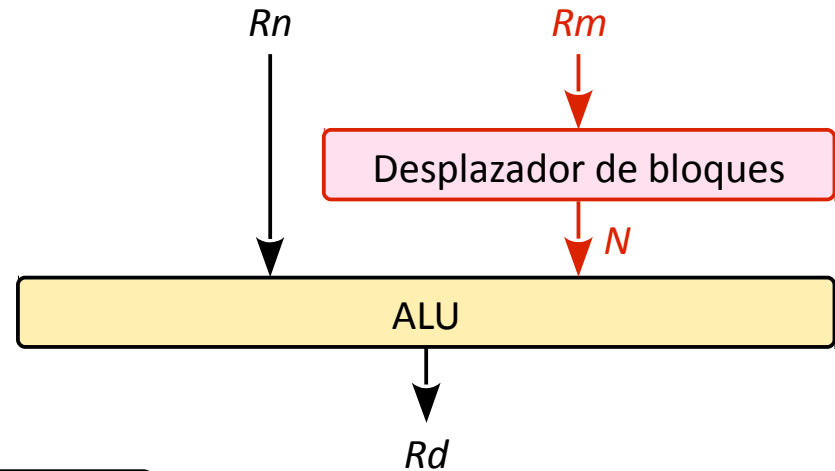
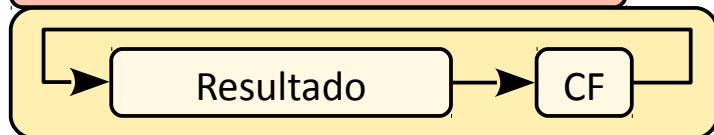
ASR (Arithmetic Shift Right)



ROR (ROtate Right)



RRX (Rotate Right eXtended)



Sintaxis

<instrucción>{<cond>}{s} Rd, Rn, #imm
<instrucción>{<cond>}{s} Rd, Rn, Rm
<instrucción>{<cond>}{s} Rd, Rn, Rm, LSL #shift_imm
<instrucción>{<cond>}{s} Rd, Rn, Rm, LSR Rs
<instrucción>{<cond>}{s} Rd, Rn, Rm, LSR #shift_imm
<instrucción>{<cond>}{s} Rd, Rn, Rm, LSR Rs
<instrucción>{<cond>}{s} Rd, Rn, Rm, ASR #shift_imm
<instrucción>{<cond>}{s} Rd, Rn, Rm, ASR Rs
<instrucción>{<cond>}{s} Rd, Rn, Rm, ROR #shift_imm
<instrucción>{<cond>}{s} Rd, Rn, Rm, ROR Rs
<instrucción>{<cond>}{s} Rd, Rn, Rm, RRX

El segundo operando

Registro desplazado

El desplazamiento puede ser indicado por:

- Un valor inmediato de 5 bits → No produce penalización
- El último byte de un registro (que no sea PC) → Añade 1 ciclo de penalización

Si no se indica desplazamiento, se asume el desplazamiento por defecto → LSL #0

Ejemplos de uso

$r1 = r2 / 4$

```
mov    r1, r2, ASR #2
```

$r4 = r3$ rotado 5 bits a la izquierda

```
mov    r4, r3, ROR #27    @ rotamos 27 a la derecha (32 - 5)
```

Indicamos el desplazamiento en un registro

```
mov    r2, r3, LSL r5    @ Se usa el último byte de r5, penaliz:1 ciclo
```

El segundo operando

Valor inmediato

- Los valores inmediatos se codifican mediante 8 bits en la instrucción (0 – 255)
- Estos valores se pueden **rotar un número par de veces** sin penalización
- El ensamblador calcula la rotación y el inmediato automáticamente a partir de una constante

Inmediatos generables

0 – 255	[0x00 – 0xFF]
256, 260, 264,..., 1020	[0x40 – 0xFF ror 30 → 0x100 – 0x3FC, salto 4]
1024, 1040, 1056,..., 4080	[0x40 – 0xFF ror 28 → 0x400 – 0xFF0, salto 16]
4060, 4160, 4224,..., 16320	[0x40 – 0xFF ror 26 → 0x1000 – 0x3FC0, salto 64]

Ejemplos de uso

`mov r0 #0x40, 26 @ => mov r0, #4096 (4096 = 0x1000 = 0x40 ror 26)`

`mov r0, #-1 @ => mvn r0, #0 (-1 = 0xffffffff)`

`mov r0, #0xFFFFFD3 @ => mvn r0, #44`

`mov r0, #16321 @ ERROR: Constante no generable`

Instrucciones aritméticas

Implementan sumas y restas de valores de 32 bits con o sin signo

Sintaxis

<instrucción>{<cond>}{s} Rd, Rn, N

Instrucciones

add	Suma de dos valores de 32 bits	$Rd \leftarrow Rd + N$
adc	Suma de dos valores de 32 bits y el acareo	$Rd \leftarrow Rd + N + \text{Carry}$
sub	Resta de dos valores de 32 bits	$Rd \leftarrow Rd - N$
sbc	Resta de dos valores de 32 bits y el acareo	$Rd \leftarrow Rd - N - \text{!Carry}$
rsb	Resta inversa de dos valores de 32 bits	$Rd \leftarrow N - Rd$
rsc	Resta inversa de dos valores de 32 bits y el acareo	$Rd \leftarrow Rd - N - \text{!Carry}$

Ejemplos de uso: **Multiplicación por una constante igual a $2^x \pm 1$**

add r0, r1, r1, LSL #2 @ r0 <- r1 + r1*4 = r1*5

rsb r2, r3, r3, LSL #4 @ r2 <- r3*16 - r3 = r3*15

rsb r2, r2, r2, LSL #3 @ r2 <- r2*8 - r2 = r2*7 * r3*15*7 = r3*105

Instrucciones lógicas

Implementan operaciones bit a bit entre dos valores de 32 bits

Sintaxis

<instrucción>{<cond>}{s} rd, rn, n

Instrucciones

and	Producto lógico bit a bit de dos valores de 32 bits	$Rd \leftarrow Rd \& N$
orr	Suma lógica bit a bit de dos valores de 32 bits	$Rd \leftarrow Rd \mid N$
eor	Operación OR exclusiva entre dos valores de 32 bits	$Rd \leftarrow Rd \wedge N$
bic	<i>Bit Clear</i> (and not)	$Rd \leftarrow Rd \& \sim N$

Ejemplo de uso: Intercambio de los bytes centrales de un registro

```
and    r1, r0, #0x0000ff00    @ r1 <- 0x0000bb00 (asumimos que r0=0xddccbbaa)
and    r2, r0, #0x00ff0000    @ r2 <- 0x00cc0000
bic    r0, r0, #0x0000ff00    @ r0 <- 0xddcc00aa
bic    r0, r0, #0x00ff0000    @ r0 <- 0xdd0000aa
orr    r0, r2, LSR #8          @ r0 <- 0xdd00ccaa
orr    r0, r1, LSL #8          @ r0 <- 0xddbbccaa
```


Instrucciones de comparación

Comparan o prueban un registro con un valor de 32 bits
Sólo actualizan los flags de condición (no modifican el resto de registros)

Sintaxis

<instrucción>{<cond>} rn, n

Instrucciones

cmp	Comparación	Se actualizan los flags al resultado de $Rn - N$
cmn	Comparación negada	Se actualizan los flags al resultado de $Rn + N$
tst	Prueba de un patrón de 32 bits	Se actualizan los flags al resultado de $Rn \& N$
teq	Prueba de la igualdad de dos valores	Se actualizan los flags al resultado de $Rn \wedge N$

Ejemplos de uso

cmp r8, r2 @ Se activan los flags N,V,Z o C para saltos posteriores

tst r1, #0x0000ff00 @ Están todos los bits de la máscara a cero?

teq r1, r2 @ r1 == r2 ?

Instrucciones de multiplicación

Multiplican dos registros de 32 bits y truncan el resultado a 32 bits. Dependiendo de la instrucción, puede que acumulen el resultado

Sintaxis

`mul{<cond>}{s} Rd, Rm, Rs`
`mula{<cond>}{s} Rd, Rm, Rs, Rn`

Restricciones

Rd y Rm no pueden ser el mismo registro
No se puede usar el registro PC

Instrucciones

mul	Multiplicación	$Rd \leftarrow Rm * Rs$
mula	Multiplicación y acumulación	$Rd \leftarrow (Rm * Rs) + Rn$

Ejemplos de uso

`muls r7, r3, r8` @ `r7 <- r3 * r8`, se fijan los flags según el resultado

`mlage r7, r8, r9, r3` @ `r7 <- (r8 * r9) + r3` si el anterior resultado fue `>=0`

Limitación

Las multiplicaciones consumen mucha energía. Se deben evitar si se puede conseguir el mismo resultado mediante otras instrucciones

Instrucciones de multiplicación extendida

Multiplican dos registros de 32 bits y devuelven un resultado de 64 bits en dos registros. Dependiendo de la instrucción, puede que acumulen el resultado

Puesto que ahora no se descartan las 32 bits más significativos, hay que indicar si los operandos son **signed** o **unsigned**

Sintaxis

<instrucción>{<cond>}{s} RdLo, RdHi, Rm, Rs

Restricciones

Sólo están implementadas en las variantes M de la arquitectura ARM

Instrucciones

smull	Multiplicación extendida con signo	$[RdHi, RdLo] \leftarrow Rm * Rs$
smlal	Multiplicación extend. y acum. con signo	$[RdHi, RdLo] \leftarrow [RdHi, RdLo] + (Rm * Rs)$
umull	Multiplicación extendida sin signo	$[RdHi, RdLo] \leftarrow Rm * Rs$
umlal	Multiplicación extend. y acum. sin signo	$[RdHi, RdLo] \leftarrow [RdHi, RdLo] + (Rm * Rs)$

Limitación

Estas instrucciones no se generan automáticamente por el compilador
El resultado es impredecible en procesadores ARM que no implementen la variante M

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Características de la arquitectura ARM

- Application Binary Interface* de la arquitectura ARM

- El hola mundo de un sistema empujado

Repertorio de instrucciones de la arquitectura ARMv4T

- Instrucciones de procesamiento de datos

- Instrucciones de salto

- Instrucciones de carga y almacenamiento

- Interrupciones software

- Otras instrucciones

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Instrucciones de salto

Cambian el flujo del programa (llamadas a subrutinas, retornos, bucles, if-then-else, switch,...)

Sintaxis

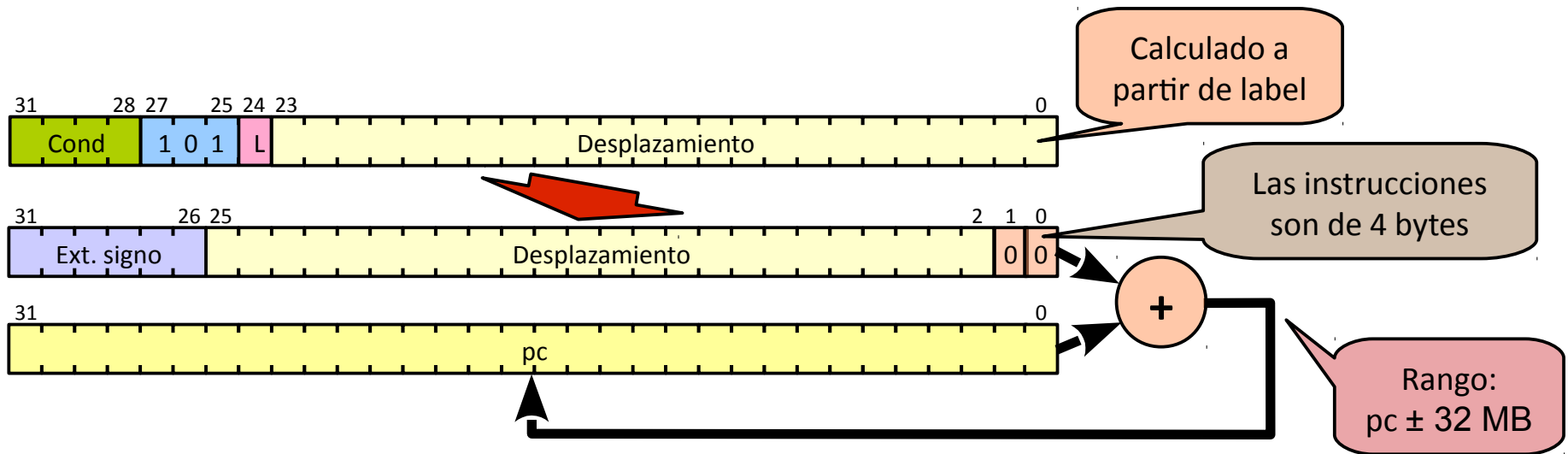
b{<cond>} Label
bl{<cond>} Label
bx{<cond>} Rm
blx{<cond>} Label | Rm

Instrucciones

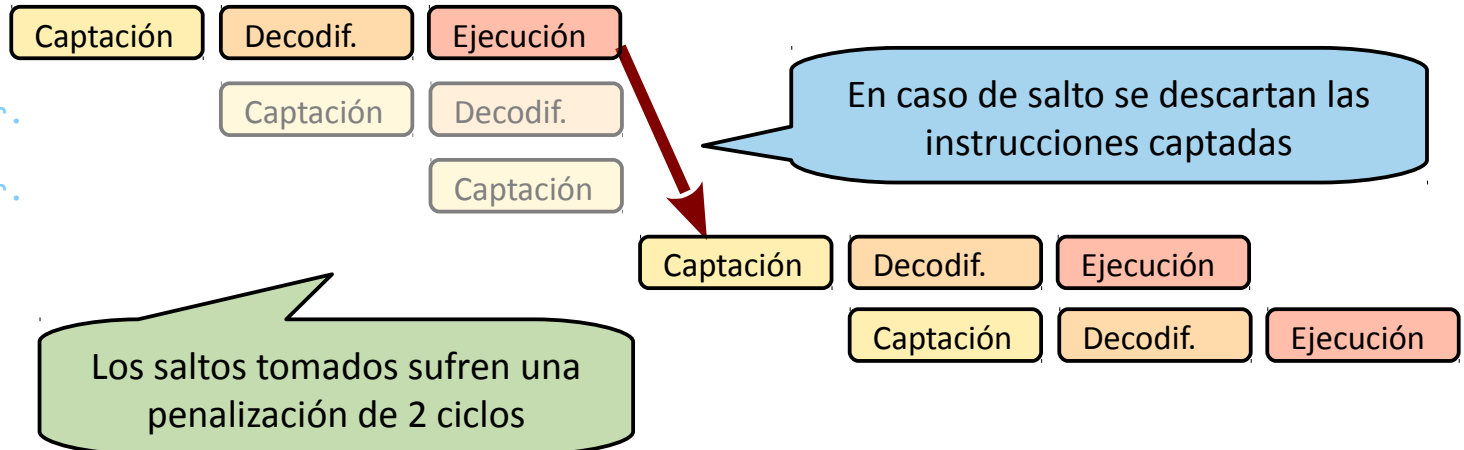
b	Salto	$PC \leftarrow \text{Label}$
bl	Salto con enlace	$PC \leftarrow \text{Label}$ $LR \leftarrow \text{Siguiente instrucción al salto}$
bx	Salto y cambio de modo	$PC \leftarrow Rm \ \& \ 0\text{fffffffe}, T = Rm \ \& \ 1$
blx	Salto y cambio de modo con enlace	$PC \leftarrow \text{Label}, T = 1$ $PC \leftarrow Rm \ \& \ 0\text{fffffffe}, T = Rm \ \& \ 1$ $LR \leftarrow \text{Siguiente instrucción al salto}$

T se refiere al bit Thumb en el registro cpsr. Por tanto, las instrucciones bx y blx se pueden usar para cambiar el modo de ejecución de 32 bits a Thumb

Limitaciones de las instrucciones de salto



0x8000 bl
0x8004 Sig. Instr.
0x8008 Sig. Instr.
0x8fec add
0x8ff0 mov



Ejemplos: bucles y sentencias condicionales

Cálculo del máximo común divisor usando saltos

```
mcd:    cmp    r0, r1    @ Hemos llegado al final?    1 ciclo
        beq    fin      1 ciclo (menos en la última iter.)
        blt    menor     @ si r0 > r1                1 ciclo + 2 si r0 > r1
        sub    r0, r0, r1 @ restamos r1 de r0         1 ciclo
        b      mcd       1 ciclo + 2 cada vez que se tome
menor:  sub    r1, r1, r0 @ restamos r0 de r1         1 ciclo
        b      mcd       1 ciclo + 2 cada vez que se tome
fin:
```

Cálculo del máximo común divisor usando instrucciones condicionales

```
mcd:    cmp    r0, r1    @ si r0 > r1                1 ciclo
        subgt  r0, r0, r1 @ restamos r1 de r0         1 ciclo
        sublt  r1, r1, r0 @ si no, restamos r0 de r1  1 ciclo
        bne    mcd       1 ciclo + 2 cada vez que se tome
```

Conclusiones

Reducción del tamaño del código: 43%

Reducción del tiempo de ejecución: 14% – 33%

Hay que evitar el uso instrucciones de salto siempre que se pueda

Ejemplos: llamadas a subrutinas

Subrutinas

<código del programa principal>

```
bl    subrutina
cmp   r1, #5
moveq r1, #0
```

Salto relativo a pc

subrutina:

```
<código de la subrutina>
mov   pc, lr
```

Salto global

Salto globales

<código del programa principal>

```
adr    r0, muy_lejos
blx    r0
cmp    r1, #5
```

Salto global

muy_lejos:

```
<código de la subrutina>
mov    pc, lr
```

Llamadas a código Thumb

<código del programa principal>

```
adr    r0, subrutina_thumb + 1
blx    r0
cmp    r1, #5
moveq  r1, #0
```

Provocamos el
cambio de modo

subrutina_thumb:

```
<código thumb>
bx     lr
```

Salto global y
cambio de modo

Salto global y
cambio de modo

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Características de la arquitectura ARM

- Application Binary Interface* de la arquitectura ARM

- El hola mundo de un sistema empotrado

Repertorio de instrucciones de la arquitectura ARMv4T

- Instrucciones de procesamiento de datos

- Instrucciones de salto

- Instrucciones de carga y almacenamiento

- Interrupciones software

- Otras instrucciones

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Transferencia de un solo registro

Mueven un valor desde o hacia un registro

Sintaxis

<ldr | str>{<cond>}{<size>} Rd, <address>

Restricciones

La dirección debe estar alineada al tamaño del dato

Instrucciones

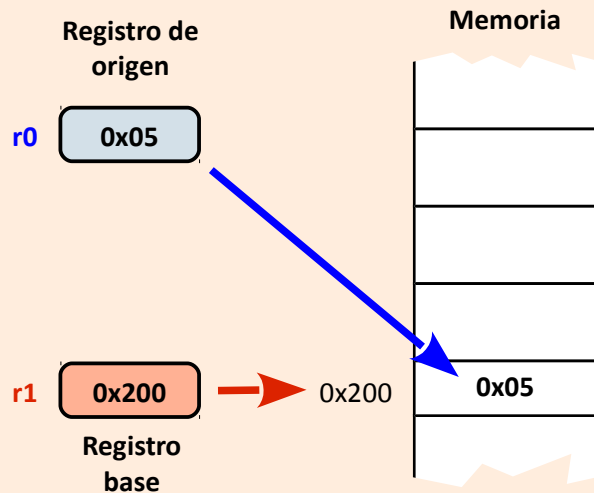
ldrb	Carga un byte en un registro	$Rd \leftarrow \text{mem}_8[\text{address}]$
ldrsb	Carga un byte con signo en un registro	$Rd \leftarrow \text{SignExtend}(\text{mem}_8[\text{address}])$
ldrh	Carga media palabra en un registro	$Rd \leftarrow \text{mem}_{16}[\text{address}]$
ldrsh	Carga media palabra con signo en un registro	$Rd \leftarrow \text{SignExtend}(\text{mem}_{16}[\text{address}])$
ldr	Carga una palabra en un registro	$Rd \leftarrow \text{mem}_{32}[\text{address}]$
strb	Almacena un byte desde un registro	$Rd \rightarrow \text{mem}_8[\text{address}]$
srh	Almacena media palabra desde un registro	$Rd \rightarrow \text{mem}_{16}[\text{address}]$
str	Almacena una palabra desde un registro	$Rd \rightarrow \text{mem}_{32}[\text{address}]$

Direcccionamiento mediante un registro base

La dirección efectiva está almacenada en un registro

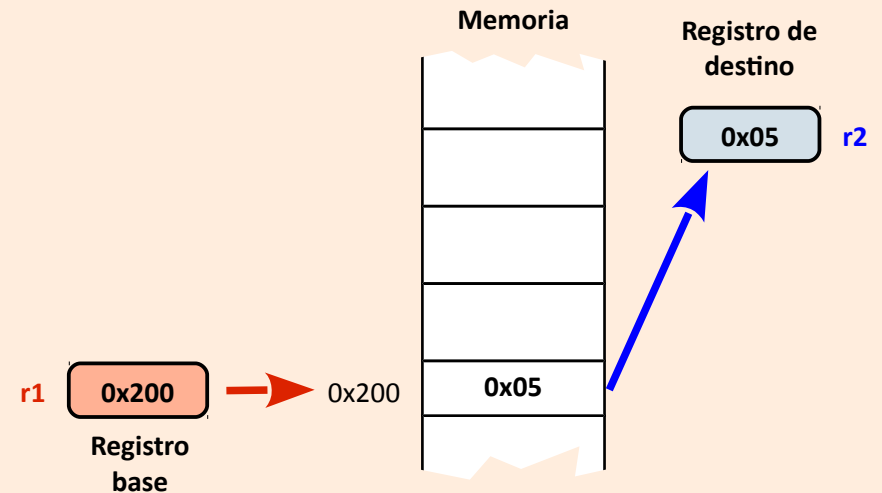
Ejemplo de uso

str r0, [r1]



Ejemplo de uso

ldr r2, [r1]

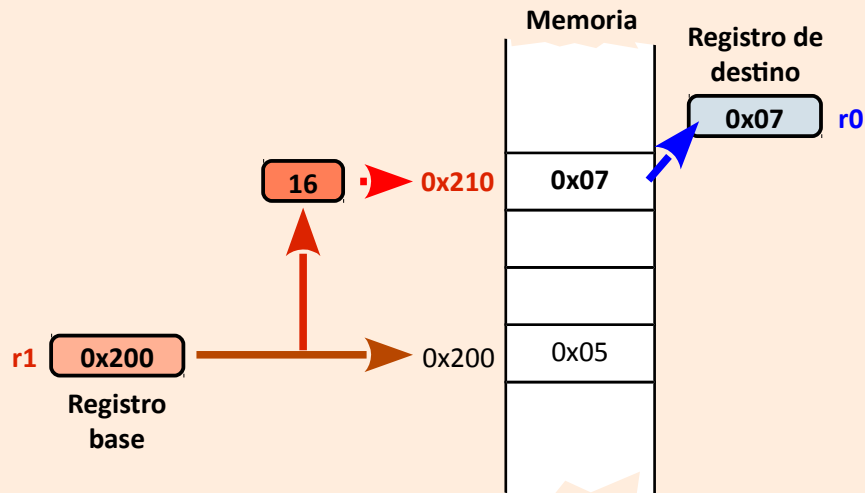


Direccionamiento pre-indexado

Se suma un desplazamiento a la dirección del registro base antes de acceder a memoria

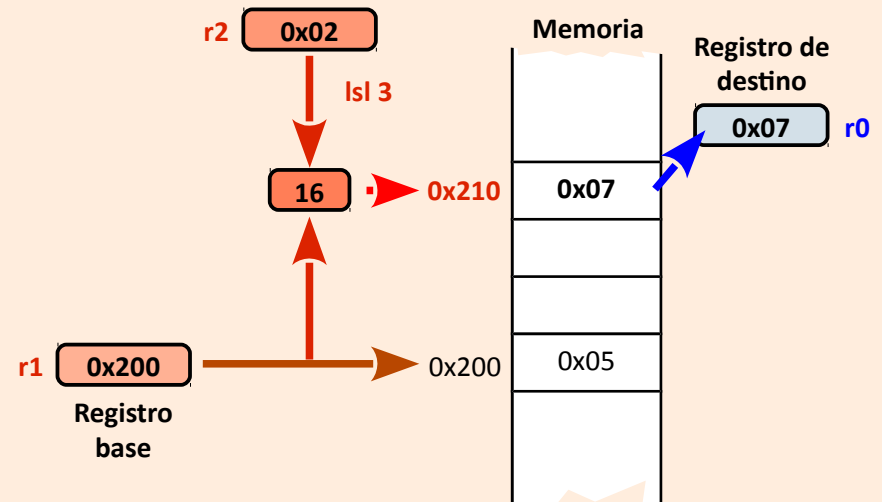
Uso de un inmediato (hasta 12 bits)

```
ldr r0, [r1, #16]
```



Uso de un registro

```
ldr r0, [r1, r2, lsl #3]
```

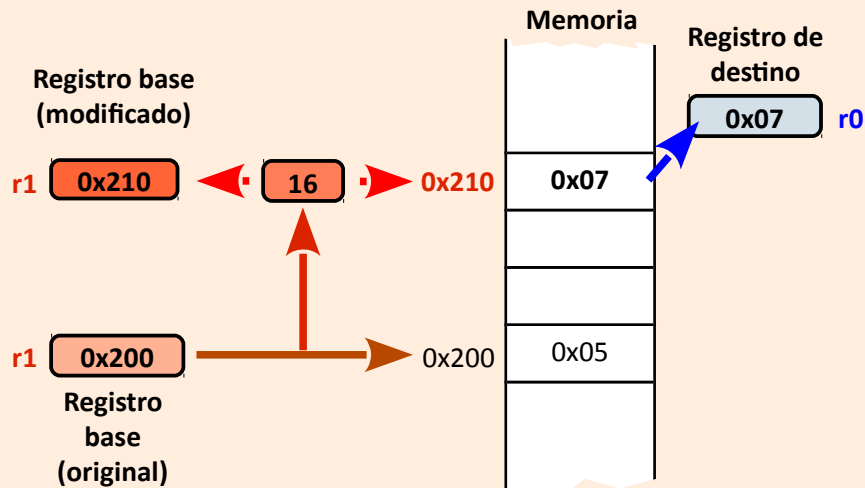


Direccionamiento pre-indexado con modificación

Se suma un desplazamiento a la dirección del registro base antes de acceder a memoria y se modifica el registro base con la dirección obtenida

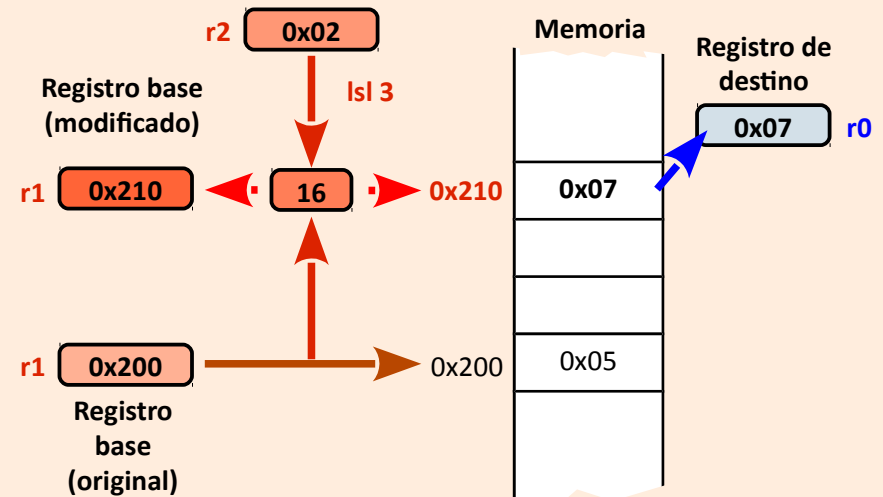
Uso de un inmediato (hasta 12 bits)

```
ldr r0, [r1, #16]!
```



Uso de un registro

```
ldr r0, [r1, r2, lsl #3]!
```

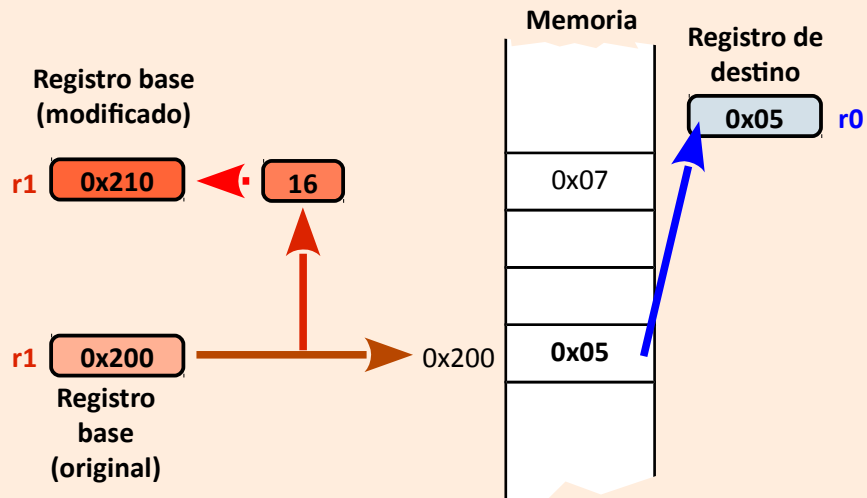


Direccionamiento post-indexado

Se suma un desplazamiento al registro base después de acceder a memoria

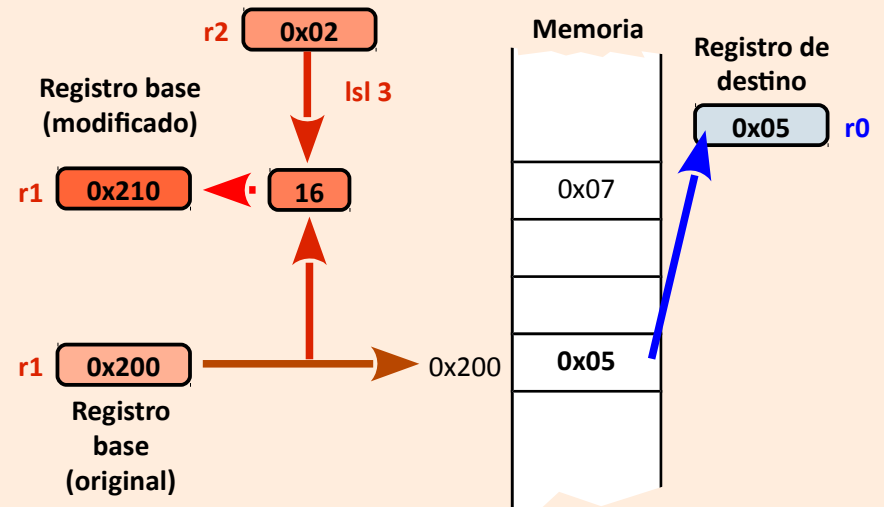
Uso de un inmediato (hasta 12 bits)

```
ldr r0, [r1], #16
```



Uso de un registro

```
ldr r0, [r1], r2, lsl #3
```



Transferencia de múltiples registros

Transferen múltiples valores entre la memoria y varios registros (entre 1 y 16 registros)

Sintaxis

<ldm | stm>{<cond>}<addressing mode> Rn{!}, <registers>{^}

Para afectar al banco de registros del modo USER desde un modo privilegiado

Instrucciones

ldm	Carga múltiples registros	<registers> \leftarrow bloque de memoria
stm	Almacena múltiples registros	<registers> \rightarrow bloque de memoria

Modos de direccionamiento

Modo	Descripción	Dirección Inicial	Dirección Final	Rn!
ia	Incrementar después	Rn	$Rn + 4 * N + 4$	$Rn + 4 * N$
ib	Incrementar antes	$Rn + 4$	$Rn + 4 * N$	$Rn + 4 * N$
da	Decrementar después	$Rn - 4 * N + 4$	Rn	$Rn - 4 * N$
db	Decrementar antes	$Rn - 4 * N$	$Rn - 4$	$Rn - 4 * N$

Ejemplo: Movimiento de bloques de memoria

Inicialmente

- R0: Puntero al búfer de origen
- R1: Puntero al búfer de destino
- R2: Tamaño del búffer, N (en palabras de 32 bits)

Primera aproximación

```
memcpy: ldr  r3, [r0]    @ leemos una palabra
        str  r3, [r1]    @ la escribimos
        add  r0, r0, #4  @ incrementamos r0
        add  r1, r1, #4  @ incrementamos r1
        sub  r2, r2, #1  @ queda 1 menos
        cmp  r2, #0      @ hemos terminado?
        bne  memcpy     @ a por la siguiente
```

12N
ciclos

Direccionamiento post-indexado

```
memcpy: ldr  r3, [r0], #4 @ leemos 1 palabra
        str  r3, [r1], #4 @ la escribimos
        sub  r2, r2, #1   @ hemos terminado?
        bne  memcpy     @ a por la siguiente
```

9N
ciclos

0.33 veces más rápido

LDM / STM

```
memcpy: ldmbia r0!, {r3,r5-r7,r9-r12} @ leemos 8 palabras
        stmbia r1!, {r3,r5-r7,r9-r12} @ las escribimos
        subs r2, r2, #8                @ quedan 8 palabras menos
        bne  memcpy                    @ saltamos si quedan más palabras
```

4 veces más rápido!

3N
ciclos

Ejemplo: Gestión de la pila

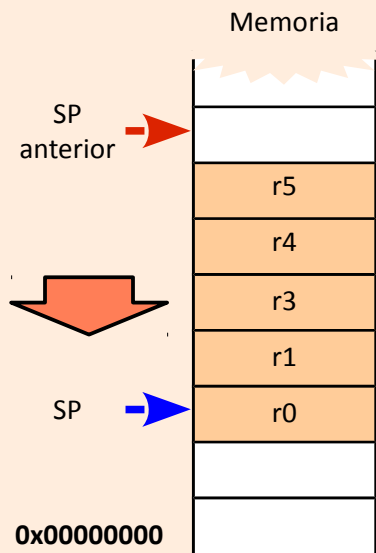
ARM soporta 4 formas de gestionar una pila

Uso de la pila	POP	=LDM	PUSH	=STM
Full Ascending	LDMFA	LMDA	STMFA	STMIB
Full Descending	LDMFD	LDMIA	STMFD	STMDB
Empty Ascending	LDMEA	LDMDB	STMEA	STMIA
Empty Descending	LDMED	LDMIB	STMED	STMDA

Es el modo que genera el compilador por defecto

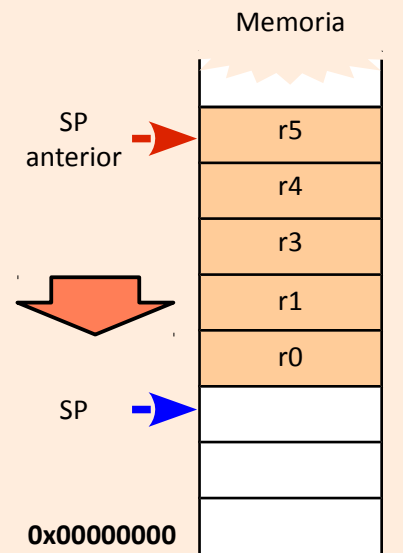
Full Descending

`stmfd sp!, {r0,r1,r3-r5}`



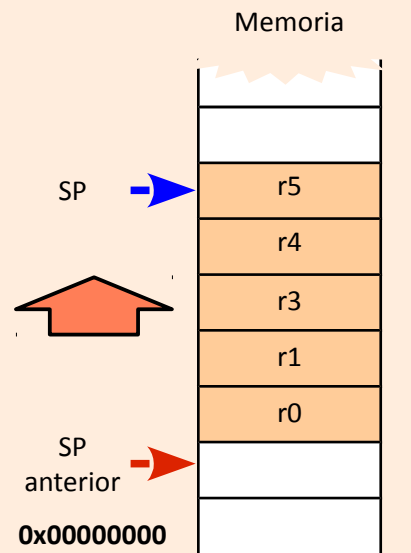
Empty Descending

`stmед sp!, {r0,r1,r3-r5}`



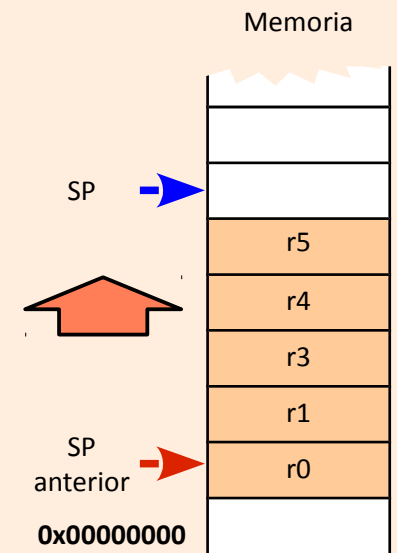
Full Ascending

`stmfa sp!, {r0,r1,r3-r5}`



Empty Ascending

`stmea sp!, {r0,r1,r3-r5}`



Swap

Intercambia el contenido de un registro con el de una posición de memoria de forma **atómica**

Sintaxis

Soporte para semáforos y exclusión mutua

swp{b}{<cond>} Rd, Rm, [Rn]

Instrucciones

swp	Intercambia una palabra	$tmp \leftarrow mem_{32}[Rn]$ $mem_{32}[Rn] \leftarrow Rm$ $Rd \leftarrow tmp$
swpb	Intercambia un byte	$tmp \leftarrow mem_8[Rn]$ $mem_8[Rn] \leftarrow Rm$ $Rd \leftarrow tmp$

Implementación del acceso exclusivo a un bus

adr r1, semaforo	@ r1 = dirección del semáforo
mov r2, #1	@ Valor para bloquear el semáforo
esperar: swp r3, r2, [r1]	@ Consultamos el semáforo
cmp r3, #1	@ Comprobamos si estaba previamente bloqueado
beq esperar	@ Si es así, continuamos esperando

Carga de constantes en los registros

Como las instrucciones son de 32 bits, no es posible especificar un valor inmediato de 32 bits

Solución

Se añaden dos pseudo-instrucciones:

adr y **ldr**

Sintaxis

adr Rd, label

ldr Rd, =constante

pseudo-instrucciones

adr	Carga una dirección	$Rd \leftarrow \text{dirección relativa de 32 bits}$
ldr	Carga una constante	$Rd \leftarrow \text{constante de 32 bits}$

Funcionamiento

adr

Genera la dirección en el registro como una suma o una resta relativa al PC

ldr

Trata de generar la constante directamente en el registro con **mov** o **mvn**, y si no es posible la almacena en un **literal pool** para que sea cargada en tiempo de ejecución

Ejemplos

adr

Lo que escribe el programador/compilador

```
adr r0, buffer    @ r0 = dir. del buffer
ldr r1, [r0]      @ r1 = buffer[0]
```

Código generado

```
sub r0, pc, #0x21c @ r0 = dir. del buffer
ldr r1, [r0]      @ r1 = buffer[0]
```

ldr

Lo que escribe el programador/compilador

```
bl func1
bl func2
stop: b stop

func1: ldr r0, =16320
      ldr r1, =0x55555555
      mov pc, lr

func2: ldr r2, =0xffffffff
      ldr r3, =0x66666666
      mov pc, lr

.end
```

16320 = 255 ROR 26

0xffffffff = ~0

El compilador coloca el *literal pool* a partir de aquí

Código generado

```
bl func1
bl func2
stop: b stop

func1: mov r0, #0xff, 0x1a
      ldr r1, [pc, #12]
      mov pc, lr

func2: mvn r2, #0
      ldr r3, [pc, #4]
      mov pc, lr

      .word 0x55555555
      .word 0x66666666
```

EX

ID

IF ← PC

← PC + 12

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Características de la arquitectura ARM

- Application Binary Interface* de la arquitectura ARM

- El hola mundo de un sistema empotrado

Repertorio de instrucciones de la arquitectura ARMv4T

- Instrucciones de procesamiento de datos

- Instrucciones de salto

- Instrucciones de carga y almacenamiento

- Interrupciones software

- Otras instrucciones

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Interrupciones software

Facilitan que los programas de usuario puedan hacer llamadas al sistema

Sintaxis

`swi{<cond>} SWI_number`

Funcionamiento

- Si estaba en estado Thumb (16 bits) pasa a estado ARM (32 bits)
- $\text{SPSR_SVC} \leftarrow \text{CPSR}$
- $\text{CPSR mode} \leftarrow \text{SVC}$ (modo privilegiado)
- $\text{CPSR I} \leftarrow 1$ (deshabilita Interrupciones)
- $\text{LR_SVC} \leftarrow$ Siguierte instrucción a SWI
- $\text{PC} \leftarrow$ Offset 0x8 de la tabla de vectores

Cada vector suele ser un salto a la ISR correspondiente

Ejemplo de uso: Llamada al servicio write de linux

```
hello_str: .ascii "Hello, World!\n"

mov  r0, #1          @ Descriptor de fichero para STDOUT
adr  r1, hello_str    @ Puntero a la buffer
mov  r2, #14         @ Tamaño del buffer
mov  r12, #4          @ Write es el servicio 4 del kernel
swi  0x80             @ Llamada al kernel de linux
```

Tabla de vectores

0x00000000

Reset

0x00000004

Undefined instruction

0x00000008

Software interrupt

0x0000000C

Prefetch abort

0x00000010

Data abort

0x00000014

Reserved

0x00000018

IRQ

0x0000001C

FIQ

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Características de la arquitectura ARM

- Application Binary Interface* de la arquitectura ARM

- El hola mundo de un sistema empujado

Repertorio de instrucciones de la arquitectura ARMv4T

- Instrucciones de procesamiento de datos

- Instrucciones de salto

- Instrucciones de carga y almacenamiento

- Interrupciones software

- Otras instrucciones

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

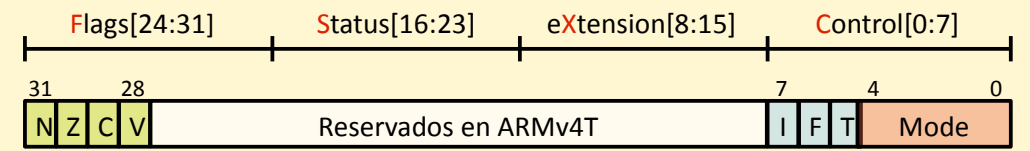
Gestión del estado del procesador

Copian el contenido de los registros de control y estado o modifican su contenido

Sintaxis

```
mrs{<cond>} Rd, <cprs | sprs>  
msr{<cond>} <cprs | sprs>_<fields>, Rm  
msr{<cond>} <cprs | sprs>_<fields>, N
```

Campos



Cualquier combinación de **f**, **s**, **x** y **c**

Program status register

Instrucciones

mrs	Copia el registro de estado a uno de propósito general	$Rd \leftarrow psr$
msr	Mueve el contenido de un registro a un registro de estado	$psr[field] \leftarrow Rn$
msr	Mueve un inmediato a un registro de estado	$psr[field] \leftarrow N$

Ejemplo: Cómo habilitar las interrupciones IRQ

```
mrs  r1, cpsr      @ r1 <- cpsr  
bic  r1, r1, #(1<<7) @ ponemos a 0 el bit 7 (habilitamos las IRQ)  
msr  cpsr_c, r1    @ cpsr[0:7] <- r1[0:7]
```


Manejo de coprocesadores

Gestionan los coprocesadores

Sintaxis

```
cdp{<cond>} cp, opcode1, Cd, Cn {, opcode2}q  
<mrc | mcr>{<cond>} cp, opcode1, Rd, Cn, Cm {, opcode2}  
<ldc | stc>{<cond>} cp, Cd, addressing
```

cp: número de coprocesador (p0-p15)

opcode: operación a realizar

Cn, Cm, Cd: registros del coprocesador

Instrucciones

cdp	Realiza una operación de cómputo en el coprocesador
mrc mcr	Mueven datos desde/hacia los registros del coprocesador
ldc stc	Cargan o almacenan bloques de memoria desde/hacia un coprocesador

Ejemplo: Copiar a R1 el contenido del registro C1 del coprocesador CP15

```
mrc p15, 0, r1, c1, c0, 0
```

```
@ r1 <- CP15:c1
```

System control coprocessor

Gestiona las caches y la memoria virtual

Coprocesador

Registro
primario

Registro
secundario

Modificador
del registro
secundario

Lecturas recomendadas

Arquitectura ARMv4T:

W. Hohl. *ARM Assembly Language. Fundamentals and Techniques*. CRC Press, 2009.

Capítulos 1 y 2

A. N. Sloss, D. Symes, C. Wright. *ARM System Developer's Guide*. Morgan Kaufmann, 2004.

Capítulos 1 y 2

P. Knaggs. *ARM Assembly Language Programming*, 2006. <http://www.rigwit.co.uk/ARMBook/>

ARM University Program. *The ARM Instruction Set, v1.0*.

http://simplemachines.it/doc/arm_inst.pdf

ARM. *ARM Architecture Reference Manual*, DDI 0100i. Capítulos 1 y 2.

<http://infocenter.arm.com/help/topic/com.arm.doc.ddi0100i/>

ARM. *ARM7TDMI-S Technical Reference Manual (Rev 3)*, DDI 0084f.

<http://infocenter.arm.com/help/topic/com.arm.doc.ddi0084f/>