

# Agenda

- Introducción
- Presentación de la Familia Cortex M3
- Arquitectura de los Cortex M3
- **Repertorio de Instrucciones y Ejemplos**
- Sistema de Memoria
- Excepciones, Interrupciones y el NVIC
- La familia NXP LPC17xx

## Repertorio de Instrucciones

Thumb User assembly code, compiler generated						Thumb-2 System, OS	
ADC	ADD	ADR	AND	ASR	B	NOP	
BIC	BL		BX	CMN	CMP	SEV	WFE
EOR	LDM	LDR	LDRB	LDRH	LDRSB	WFI	YIELD
LDRSH	LSL	LSR	MOV	MUL	MVN	DMB	
ORR	POP	PUSH	ROR	RSB	SBC	DSB	
STM	STR	STRB	STRH	SUB	SVC	ISB	
TST	BKPT	BLX	CPS	REV	REV16	MRS	
REVSH	SXTB	SXTH	UXTB	UXTH		MSR	

52 instrucciones básicas

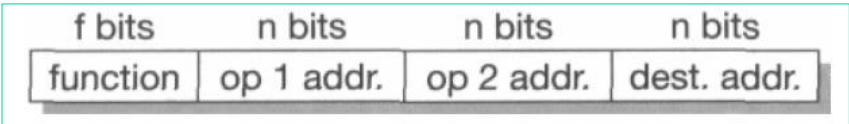
# Registros



3

# Instrucciones

- **Formatos de instrucción de 3 direcciones**
- Consta de “f” bits para el código de operación, “n” bits para especificar la dirección del 1er. operando, “n” bits para especificar la dirección del 2do. operando y “n” bits para especificar la dirección del resultado (el destino).
- n = 4 para instrucciones de 32 bits y n = 3 para instrucciones de 16 bits
- El formato de esta instrucción en *Assembler*, por ejemplo para la instrucción de sumar dos números para producir un resultado, es:
- ADD   d, s1, s2       ; d := s1 + s2.



4

# Instrucciones Condicionales

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond	0	0	1					Opcode	S		Rn		Rd	Operand 2														Data Processing / PSR Transfer			
Cond	0	0	0	0	0	0	0	A	S		Rd		Rn		Rs	1	0	0	1		Rm	Multiply									
Cond	0	0	0	0	1	U	A	S		RdHi		RdLo		Rn		1	0	0	1		Rm	Multiply Long									
Cond	0	0	0	1	0	B	0			Rn		Rd		0	0	0	1	0	0	1		Rm	Single Data Swap								
Cond	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1		Rn	Branch and Exchange						
Cond	0	0	0	P	U	0	W	L		Rn		Rd		0	0	0	0	1	S	H	1		Rm	Halfword Data Transfer: register offset							
Cond	0	0	0	P	U	1	W	L		Rn		Rd		Offset				1	S	H	1		Offset	Halfword Data Transfer: immediate offset							
Cond	0	1	1	P	U	B	W	L		Rn		Rd		Offset														Single Data Transfer			
Cond	0	1	1															1											Undefined		
Cond	1	0	0	P	U	S	W	L		Rn	Register List																Block Data Transfer				
Cond	1	0	1	L	Offset																										Branch
Cond	1	1	0	P	U	N	W	L		Rn		CRd		CP#	Offset												Coprocessor Data Transfer				
Cond	1	1	1	0		CP	Opc			CRn		CRd		CP#		CP	0		CRm	Coprocessor Data Operation											
Cond	1	1	1	0		CP	Opc	L		CRn		Rd		CP#		CP	1		CRm	Coprocessor Register Transfer											
Cond	1	1	1	1	Ignored by processor																										Software Interrupt
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

## Instrucciones Condicionales

cond	Mnemonic extension	Meaning, integer arithmetic	Meaning, floating-point arithmetic <sup>a</sup>	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS <sup>b</sup>	Carry set	Greater than, equal, or unordered	C == 1
0011	CC <sup>c</sup>	Carry clear	Less than	C == 0
0100	MI	Minus, negative	Less than	N == 1
0101	PL	Plus, positive or zero	Greater than, equal, or unordered	N == 0
0110	VS	Overflow	Unordered	V == 1
0111	VC	No overflow	Not unordered	V == 0
1000	HI	Unsigned higher	Greater than, or unordered	C == 1 and Z == 0
1001	LS	Unsigned lower or same	Less than or equal	C == 0 or Z == 1
1010	GE	Signed greater than or equal	Greater than or equal	N == V
1011	LT	Signed less than	Less than, or unordered	N != V
1100	GT	Signed greater than	Greater than	Z == 0 and N == V
1101	LE	Signed less than or equal	Less than, equal, or unordered	Z == 1 or N != V
1110	None (AL) <sup>d</sup>	Always (unconditional)	Always (unconditional)	Any

a. Unordered means at least one NaN operand.

b. HS (unsigned higher or same) is a synonym for CS.

c. LO (unsigned lower) is a synonym for CC.

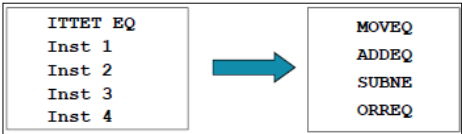
d. AL is an optional mnemonic extension for always, except in IT instructions. See *IT* on page A7-277 for details.

# Ejemplos condicionales

Fuente C	ARM instructions	
	Incondicional	Condicional
<pre>if (r0 == 0) {     r1 = r1 + 1; } else {     r2 = r2 + 1; }</pre>	<pre>... CMP r0, #0 BNE else ADD r1, r1, #1 B end else     ADD r2, r2, #1 end</pre>	<pre>CMP    r0, #0 ADDEQ  r1, r1, #1 ADDNE  r2, r2, #1 ...</pre>
	<ul style="list-style-type: none"><li>▪ 5 instrucciones</li><li>▪ 5 palabras</li><li>▪ 5 o 6 ciclos</li></ul>	<ul style="list-style-type: none"><li>▪ 3 instrucciones</li><li>▪ 3 palabras</li><li>▪ 3 ciclos</li></ul>

## Instrucciones Condicionales

- If – Then (IT) se agrega la instrucción (16 bit)
  - Hasta 4 condiciones “then” o “else” adicionales pueden ser especificadas (T or E)



- Puede utilizarse cualquier código de operación ARM normal
- Las instrucciones de 16 bits, en bloque, no afectan los flags
  - Si lo hacen las instrucciones de comparación
  - Las instrucciones de 32 bits (según las reglas generales) pueden afectar los flags
- El estado actual del “if-then” se almacena en el CPSR
  - El bloque condicional puede ser interrumpido y retornado con seguridad
  - NO se deben realizar branch dentro o fuera del bloque ‘if-then’

## Instrucciones

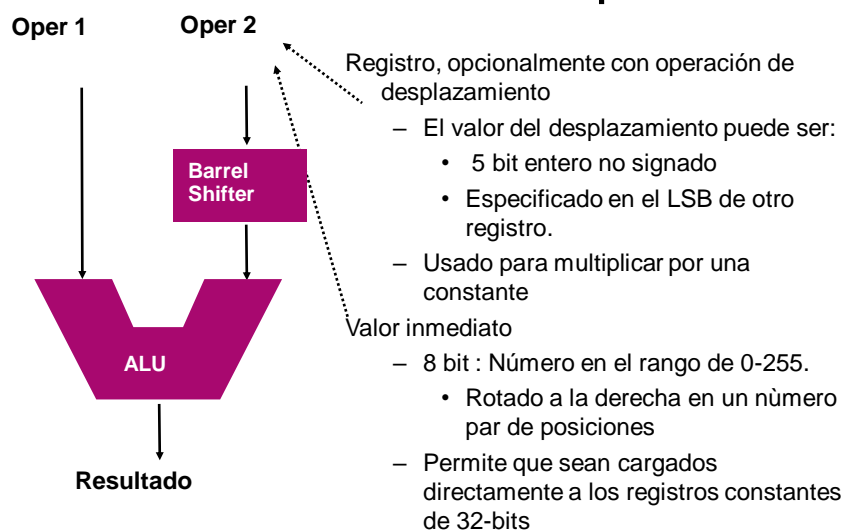
- **Casi todas las instrucciones se ejecutan en un ciclo de reloj**
- **Modos de direccionamiento simples**
- El procesamiento de datos solo opera con contenidos de registros, no directamente en memoria.
- **Control sobre la unidad aritmética lógica (ALU, *Arithmetic Logic Unit*) y el “shifter”, en cada instrucción de procesamiento de datos para maximizar el uso de la ALU y del “shifter”.**
- **Modos de direccionamiento con incremento y decremento automático de punteros**, para optimizar los lazos de los programas.
- **Carga y almacenamiento de múltiples registros**, para maximizar el rendimiento de los datos.

Repertorio instrucciones y assembler

9

9

## Barrel Shifter: El 2º Operando

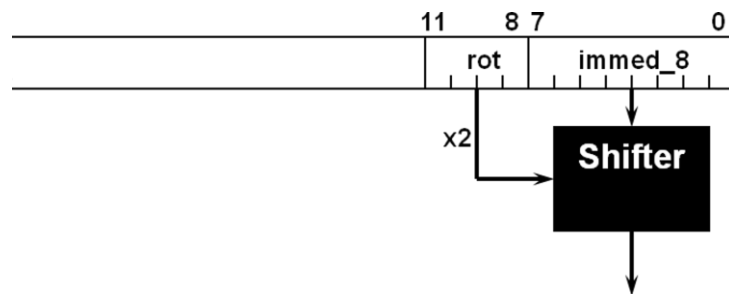


Repertorio instrucciones y assembler

10

10

# Operandos inmediatos



## • Instrucciones de procesamiento de datos

# Instrucciones de procesamiento de datos

Instruction	Function
ADC	Add with carry
ADD	Add
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear (Logical AND one value with the logic inversion of another value)
CMN	Compare negative (compare one data with two's complement of another data and update flags)
CMP	Compare (compare two data and update flags)
CPY	Copy (available from architecture v6; move a value from one high or low register to another high or low register)
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MOV	Move (can be used for register-to-register transfers or loading immediate data)

13

# Instrucciones de procesamiento de datos

MUL	Multiply
MVN	Move NOT (obtain logical inverted value)
NEG	Negate (obtain two's complement value)
ORR	Logical OR
ROR	Rotate right
SBC	Subtract with carry
SUB	Subtract
TST	Test (use as logical AND; Z flag is updated but AND result is not stored)
REV	Reverse the byte order in a 32-bit register (available from architecture v6)
REVH	Reverse the byte order in each 16-bit half word of a 32-bit register (available from architecture v6)
REVSH	Reverse the byte order in the lower 16-bit half word of a 32-bit register and sign extends the result to 32 bits. (available from architecture v6)
SXTB	Signed extend byte (available from architecture v6)
SXTH	Signed extend half word (available from architecture v6)
UXTB	Unsigned extend byte (available from architecture v6)
UXTH	Unsigned extend half word (available from architecture v6)

14

# Procesamiento de Datos

- Consiste de:
  - Aritméticas: **ADD**    **ADC**    **SUB**    **SBC**    **RSB**    **RSC**
  - Logicas: **AND**    **ORR**    **EOR**    **BIC**
  - Comparaciones: **CMP**    **CMN**    **TST**    **TEQ**
  - Movimiento Datos: **MOV**    **MVN**

- Estas instrucciones operan sobre registros y NO en memoria.

- Sintaxis:

**<Operación>{<cond>}{S} Rd, Rn, Operand2**

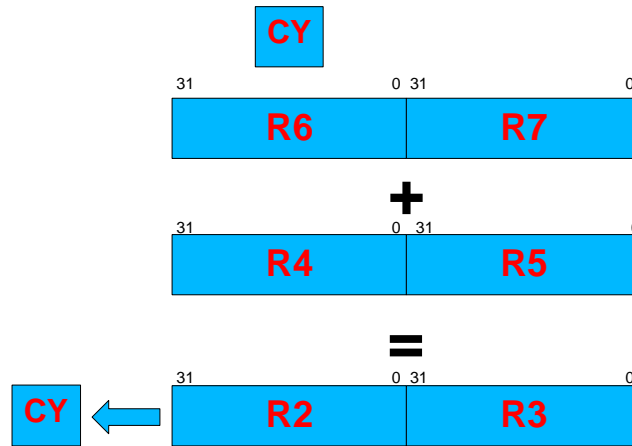
- Comparaciones sólo afectan flags – no especifican Rd
  - Movimiento de datos no especifican Rn
- El segundo operando se envía a la ALU a través del desplazador en barril.

# Procesamiento de Datos

## Aritméticas

- **ADD r1, r2, r3**            ;  $r1 = r2 + r3$
- **ADC r1, r2, r3**            ;  $r1 = r2 + r3 + C$
- **SUB r1, r2, r3**            ;  $r1 = r2 - r3$
- **SUBC r1, r2, r3**          ;  $r1 = r2 - r3 + C - 1$
- **RSB r1, r2, r3**            ;  $r1 = r3 - r2$  (inversa)
- **RSC r1, r2, r3**            ;  $r1 = r3 - r2 + C - 1$





## Aritmeticas – Suma Multibyte

Inicio

; Inicialización de Registros

```
mov    r4,#0x50
mov    r5,#0x70
mov    r6,#0xc0
mov    r7,#0xf0
mov    r1,#0      ;Acarreo inicial = 0
```

; Suma

```
adds  r3,r5,r7
adds  r2,r4,r6
addc   r1,#1
```

# Procesamiento de Datos

- Aritméticas
  - ADD r3,r2,#1
  - ADD r3,r2,r1, lsl #3 (lsl, asl, asr, ror, rrx)
  - ADD r5,r5,r3, LSL r2
  - MUL r4,r3,r2
  - MLA r1,r3,r2,r1 ;r4:=(r3 x r2 + r1)
  - RSB r0,r0,r0, LSL #3
    - » ; Multiplicar por 7

# Procesamiento de Datos

- Lógicas
  - AND r0,r1,r2 ; r0:= r1.r2
  - ORR r0,r1,r2 ; r0:= r1 + r2
  - EOR r0,r1,r2 ; r0:= r1 xor r2
  - BIC r0,r1,r2 ; r0:= r1 and not r2
  - AND r8,r7,#0xff ; r8:= r7 . 0x000000ff

## Procesamiento de Datos

Solo afectan los Flags

CMP	r1, r2	; cc por r1 - r2
CMN	r1, r2	; cc por r1 + r2
TST	r1, r2	; cc por r1 and r2
TEQ	r1, r2	; cc por r1 xor r2

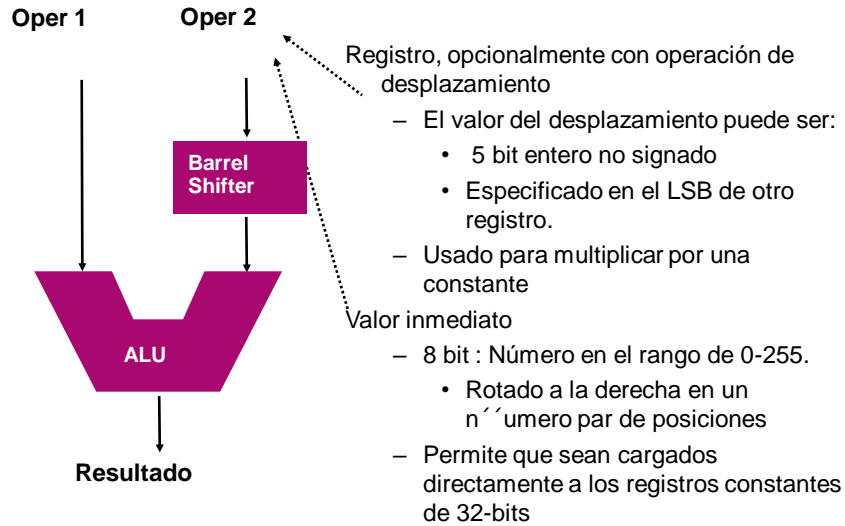
## Procesamiento de Datos

Inmediatas

- ADD r3, r3,#1 ; r3 := r3 + 1
- AND r8, r7,#0xff ; r8 := r7[7:0]

# Procesamiento de Datos

## Barrel Shifter: El 2º Operando



Repertorio instrucciones y assembler

23

23

# Procesamiento de Datos

## Desplazamientos

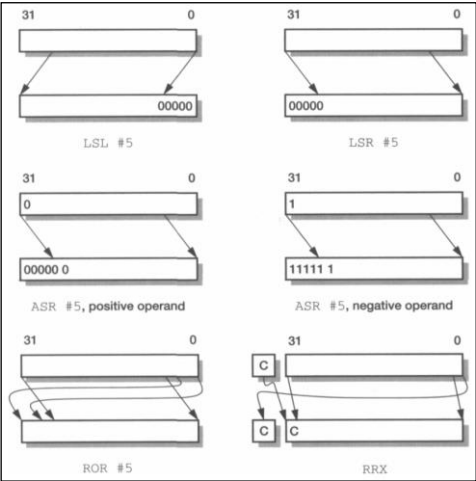
- ADD       $r3, r2, r1, \text{LSL } \#3 ; r3 := r2 + 8 \times r1$
- ADD       $r5, r5, r3, \text{LSL } r2 ; r5 := r5 + r3 \times 2^{r2}$
- RSB       $r0, r0, r0, \text{LSL } \#3$
- ; Multiplicar por 7

Repertorio instrucciones y assembler

24

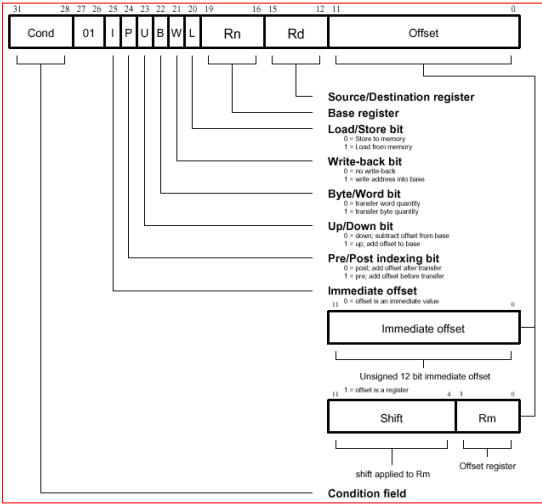
24

# Desplazamientos



## • INSTRUCCIONES DE TRANSFERENCIA DE DATOS

# Transferencias de Datos



Repertorio instrucciones y assembler

27

27

# Transferencias de Datos

- Movimiento
  - MOV r0,r2
  - MVN r0,r2 ; r0:= not r2

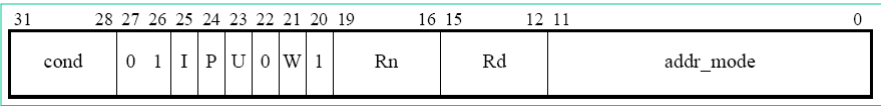
Repertorio instrucciones y assembler

28

28

# Transferencias de Datos

## LDR

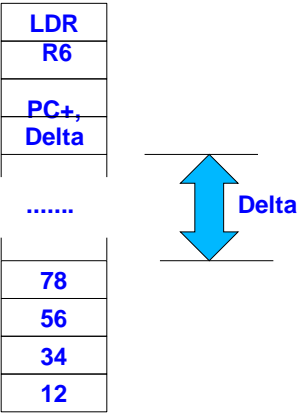


LDR r0,[r1] ; r0 := mem<sub>32</sub>[r1]  
LDR R4, [R2, #4] ; Carga word en R4 desde direc R2 + 4  
LDR R4, [R2, R1] ; Carga word en R4 desde direc R2 + R1  
LDRH R3, [R6, R5] ; Carga half word en R3 desde R6 + R5  
LDRB R2, [R1, #5] ; Carga byte en R2 desde R1 + 5  
LDR R6, [PC, #0x3FC] ; Carga R6 desde PC + 0x3FC  
LDR R5, [SP, #64] ; Carga R5 desde SP + 64

# Transferencias de Datos

## LDR

LDR R6,=0x12345678



# Transferencias de Datos

## Pre y post indexado

```
LDR r0,[r1,#4]    ; r0 := mem32[r1+ 4]

LDR r0,[r1,#4]!   ; r0 := mem32[r1] ; r1 := r1 + 4
```

# Transferencias de Datos

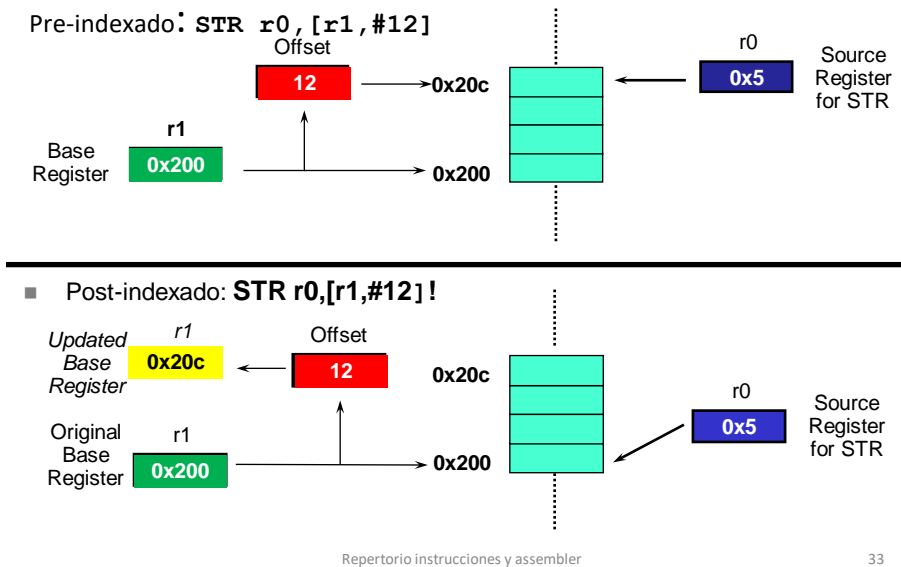
## STR

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	0
cond		0	1	I	P	U	0	W	0	Rn	Rd		addr_mode		

```
STR    r0,[r1]          ; mem32[r1] := r0
STR    R0, [R7, #0x1C] ; Guarda word desde R0 a direc R7 + 28
STRB   R1, [R5, #31] ; Guarda byte desde R1 a direc R5 + 31
STRH   R4, [R2, R3] ; Guarda halfword desde R4 a dir R2 + R3
STR    R4, [SP, #0x26] ; Guarda R4 en la direc. SP + 0x26
```



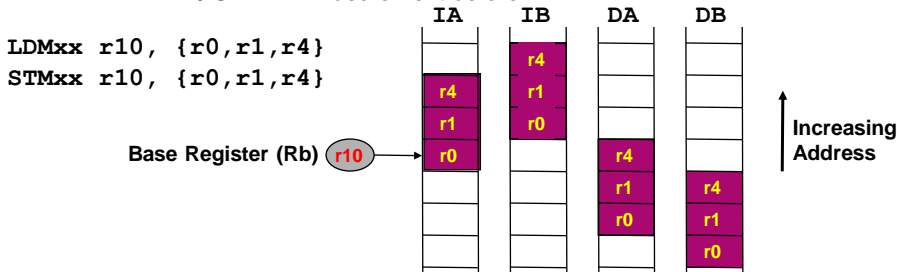
# Preindexado y postindexado



33

# Multiples Load y Store Registros

- Sintaxis:
  - <LDM|STM>**{<cond>}<addressing\_mode> Rb{!}, <register list>
- 4 modos de direccionamiento:
  - LDMIA / STMIA** increment after
  - LDMIB / STMIB** increment before
  - LDMDA / STMDA** decrement after
  - LDMDB / STMDB** decrement before

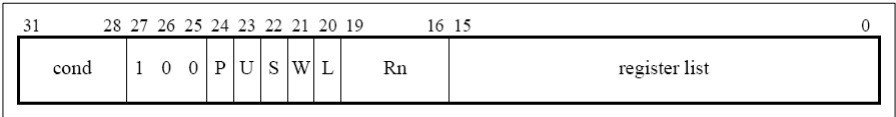


Repertorio instrucciones y assembler

34

34

# LDMIA y STMIA



LDMIA R7!, {R0-R3, R5} ; Load R0 to R3-R5 desde R7, add 20 to R7  
STMIA R0!, {R3, R4, R5} ; Store R3-R5 to R0: add 12 to R0

# PUSH y POP

Funcion:

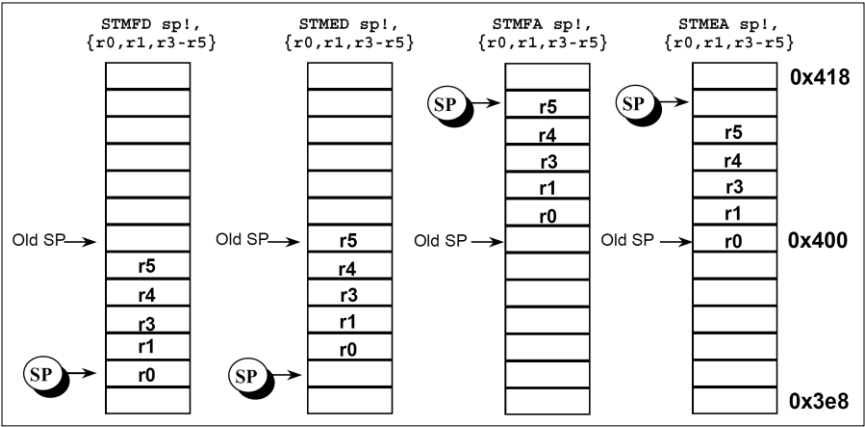
PUSH {R0-R7, LR} ; push al stack (R13) R0-R7 y la  
; dirección de retorno

... ; Cuerpo de la función

...

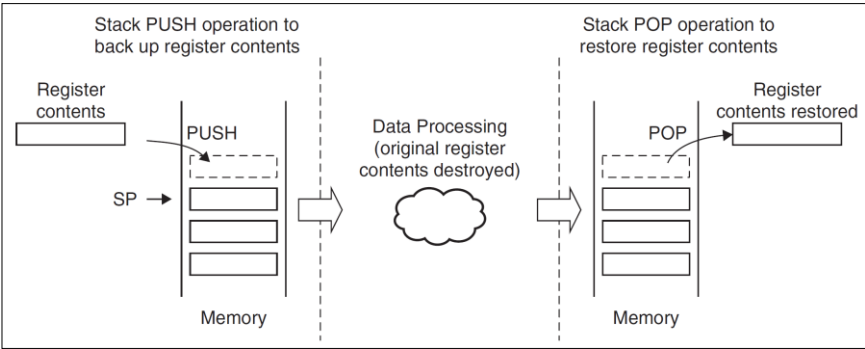
POP {R0-R7, PC} ; restaura R0-R7 del stack  
; y el PC y retorna

# Uso de la Pila



39

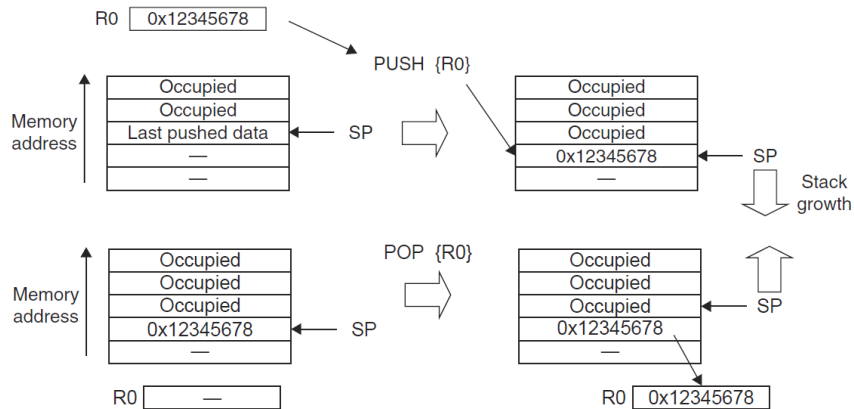
# Push y Pop



```
PUSH {R0} ; R13 = R13 - 4, luego Memory[R13] <- R0
POP {R0} ; R0 <- Memory[R13], luego R13 = R13 + 4
```

40

# Push y Pop Ampliados



Repertorio instrucciones y assembler

41

41

# Uso de la pila en subrutinas

; R0 = X, R1 = Y, R2 = Z

BL funcion1

; Regresa al programa ppal

; R0 = X, R1 = Y, R2 = Z

... ;

funcion1

PUSH {R0} ; almacena R0 en pila y ajusta SP

PUSH {R1} ; almacena R1 en pila y ajusta SP

PUSH {R2} ; almacena R2 en pila y ajusta SP

... ; Ejecuta tarea. (R0, R1 y R2 pueden cambiar)

POP {R2} ; restaura R2 y reajusta SP

POP {R1} ; restaura R1 y reajusta SP

POP {R0} ; restaura R0 y reajusta SP

BX LR ; R

Repertorio instrucciones y assembler

42

42

# Program Status Register – Ampliado y Condensado

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT		T				ICI/IT				

- APSR - Application Program Status Register – ALU flags
- IPSR - Interrupt Program Status Register – Interrupt/Exception No.
- EPSR - Execution Program Status Register
  - IT field – If/Then block information
  - ICI field – Interruptible-Continuable Instruction information

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0	Condensado Almacenado en el stack al Inicio de excepción
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number					

# Instrucciones de control de flujo

# Instrucciones de control de flujo

Instruction	Function
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR
BLX	Branch with link and change state (BLX <reg> only) <sup>1</sup>
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

## Saltos - Branch

- Instrucciones de salto (Branching): BX, B, BL
- B: salto con desplazamiento de 24 bits con signo
- BL: enlace (link) PC -> R14
- Instrucciones de transferencia de datos: LDR, STR, LDRH, STRH, LDRSB, LDRSH, LDM, STM, SWP.

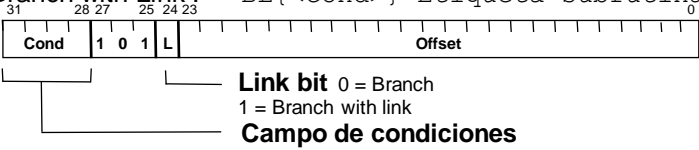
# Control de flujo

Branch	Interpretation	Normal uses
B BAL	Unconditional Always	Always take this branch Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

47

## Instrucciones de Branch

- Branch : B{<cond>} etiqueta
- Branch with Link : BL{<cond>} Etiqueta subrutina



- El procesador desplaza el campo del offset a la izquierda en dos lugares, extiende el signo y lo suma al PC
  - ± 32 Mbyte de rango
  - ¿Cómo se pueden implementar Branches mayores?

48

# Subrutinas

```
BL      subru
...
Subru: ....
      mov    pc,r14
```

## Instrucciones de Swap

También llamadas Instrucciones de semáforo

### **SWP R12, R10, [R9]**

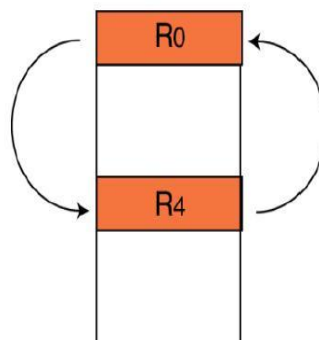
; cargar R12 desde la dirección apuntada  
; por R9 y almacenar R10 en la dirección  
; Apuntada por R9

### **SWPB R3, R4, [R8]**

; cargar byte en R3 desde la dirección  
; apuntada por R8 y almacenar byte desde  
; R4 en la dirección apuntada por R8

### **SWP R1, R1, [R2]**

; Intercambiar valores entre R1 y el  
; contenido de la memoria apuntada por R2



La instrucción de swap nos permite intercambiar el contenido de dos registros.  
Toma dos ciclos pero es tratada como una sola instrucción atómica de manera que el intercambio no puede ser corrompido por una interrupción



# AAPCS

## Procedure Call Standard for the ARM® Architecture

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Repertorio instrucciones y assembler

51

51

# Uso de Registros

Argumentos a funcion  
Resultado(s) de funcion  
alterables  
((Parámetros pasados  
en la pila)

Register
r0
r1
r2
r3

El compilador tiene un conjunto de reglas conocidas como Procedure Call Standard que determina como pasar parámetros a las funciones (ver **AAPCS**)

Los flags del CPSR pueden corromperse por una llamada inapropiada a función

Variables en registros  
deben ser preservadas

r4
r5
r6
r7
r8
r9/sb
r10/s1
r11

Si se vinculan rutinas en assembler con rutinas en C deben seguir el protocolo

El AAPCS es parte del nuevo ABI de la arquitectura ARM  
ABI = Application Binary Interface

- Stack base
- Stack limit e

Scratch register  
(alterable)

r12
-----

Stack Pointer  
Link Register  
Program Counter

r13/sp
r14/lr
r15/pc

- SP debe ser 8 bytes (2 palabras alineadas)
- R14 puede reusado una vez que se almacenó en pila

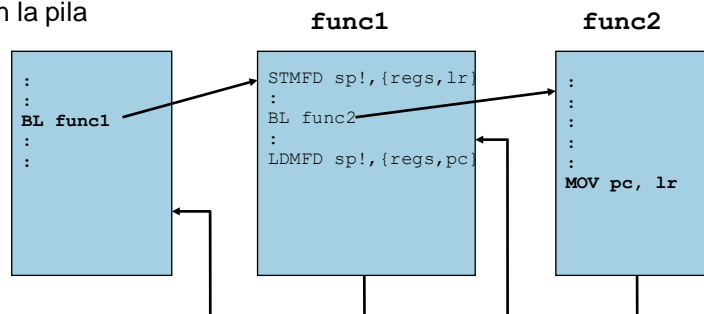
Repertorio instrucciones y assembler

52

52

## Branches y subrutinas en ARM

- B <etiqueta>
  - Relativo al PC  $\pm 32$  Mbyte range.
- BL <subrutina>
  - Almacena la dirección de retorno en LR
  - Retorna restaurando el PC desde LR
  - Para llamadas a subrutina desde una subrutina, LR debe ser guardada en la pila



Repertorio instrucciones y assembler

53

53

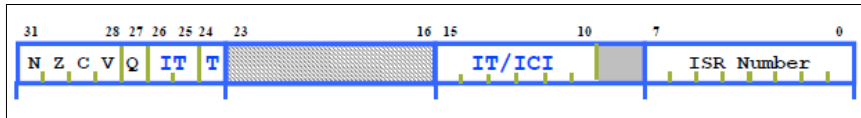
## Instrucciones reservadas de control

Repertorio instrucciones y assembler

54

54

## Acceso PSR - Reservadas



- MRS y MSR permiten transferir CPSR / SPSR de/a un registro o tomar un valor inmediato.
    - MSR permite actualizar todo el registro o una parte del mismo
  - Las interrupciones pueden ser habilitadas/deshabilitadas y cambiar los modos escribiendo al CPSR
    - Típicamente se deben emplear estrategias de read/modify/write :
- ```

MRS r0,CPSR          ; copia el CPSR a r0
BIC r0,r0,#0x10000000; ; limpiar bit V
MSR CPSR,r0          ; Escribir el valor modificado al byte 'c'
    
```
- En modo usuario sólo pueden modificarse los flags

Repertorio instrucciones y assembler

55

55

## Instrucciones de control

- Instrucciones de excepciones: SVC
- Instrucciones del Coprocesador: CDP, LDC, STC, MRC, MCR.
- Cortex no ejecuta estas instrucciones pero deja al coprocesador la manipulación de ellas.

Repertorio instrucciones y assembler

56

56

# Registros especiales

| Registro  | Función                                                                                                            |
|-----------|--------------------------------------------------------------------------------------------------------------------|
| xPSR      | Provee los flags de la ALU (zero, carry), status de la ejecución y número de la interrupción actualmente ejecutada |
| PRIMASK   | Deshabilita todas las interrupciones salvo la no enmascarable (NMI) y HardFault                                    |
| FAULTMASK | Deshabilita todas las interrupciones salvo la no enmascarable (NMI)                                                |
| BASEPRI   | Deshabilita todas las interrupciones de un nivel específico de prioridad o menor                                   |
| CONTROL   | Define estado privilegiado y selecciona el SP                                                                      |