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

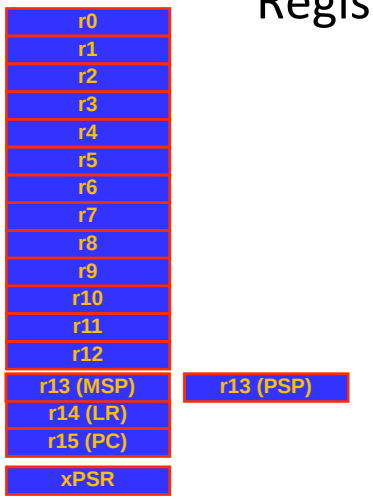
Thumb
User assembly code, compiler generated

ADC	ADD	ADR	AND	ASR	B
BIC	BL		BX	CMN	CMP
EOR	LDM	LDR	LDRB	LDRH	LDRSB
LDRSH	LSL	LSR	MOV	MUL	MVN
ORR	POP	PUSH	ROR	RSB	SBC
STM	STR	STRB	STRH	SUB	SVC
TST	BKPT	BLX	CPS	REV	REV16
REVSH	SXTB	SXTH	UXTB	UXTH	

Thumb-2
System, OS

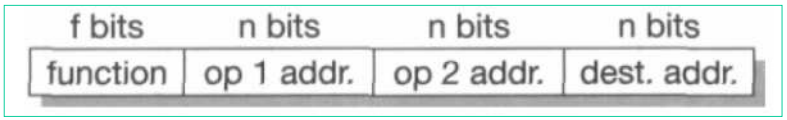
NOP	
SEV	WFE
WFI	YIELD
DMB	
DSB	
ISB	
MRS	
MSR	

Registros

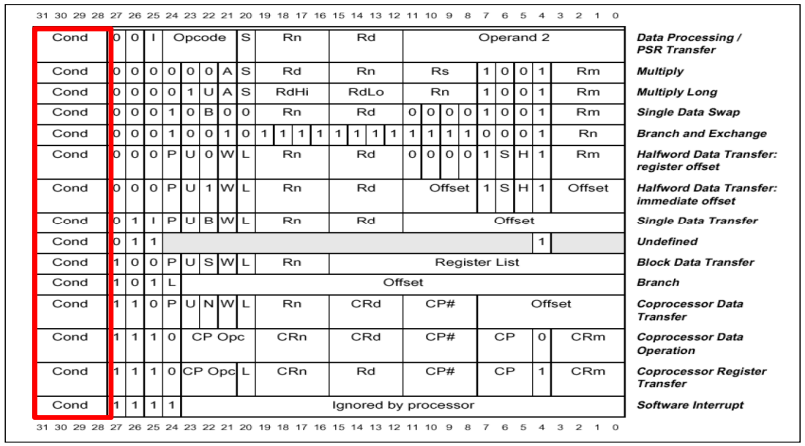


Instrucciones

- **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.



Instrucciones Condicionales



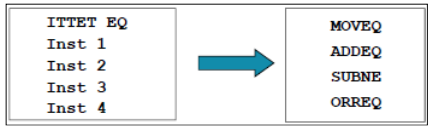
Instrucciones Condicionales

cond	Mnemonic extension	Meaning, integer arithmetic	Meaning, floating-point arithmetic[a]	Condition flags
0000	EQ	Equal	Equal	Z == 1
0001	NE	Not equal	Not equal, or unordered	Z == 0
0010	CS[b]	Carry set	Greater than, equal, or unordered	C == 1
0011	CC[c]	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)[d]	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.

Instrucciones Condicionales

- If – Then (IT) se agrega la instrucción (16 bit)
  - Hasta 4 condiciones “then” o “else” adiconales 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’

Ejemplos condicionales

Fuente C

```
if (r0 == 0)
{
  r1 = r1 + 1;
}
else
{
  r2 = r2 + 1;
}
```

ARM instructions

Incondicional

```
...
CMP r0, #0
  BNE else
  ADD r1, r1, #1
  B end
else
  ADD r2, r2, #1
end
```

- 5 instrucciones
- 5 palabras
- 5 o 6 ciclos

Condicional

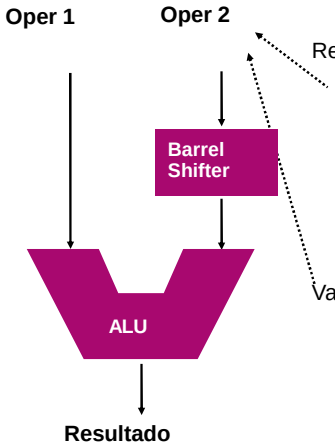
```
  CMP   r0, #0
ADDEQ r1, r1, #1
ADDNE r2, r2, #1
...
```

- 3 instrucciones
- 3 palabras
- 3 ciclos

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.

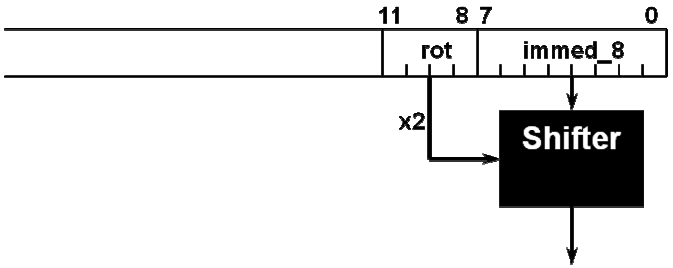
Barrel Shifter: El 2º Operando



Registro, opcionalmente con operación de desplazamiento
- El valor del desplazamiento puede ser:
  - 5 bit entero no signado
  - Especificado en el LSB de otro registro.
- Usado para multiplicar por una constante

Valor inmediato
- 8 bit : Número en el rango de 0-255.
  - Rotado a la derecha en un nùmero par de posiciones
- Permite que sean cargados directamente a los registros constantes de 32-bits

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)

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)

Procesamiento de Datos

- Consiste de:
  - Aritmeticas: **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
        adcs    r2,r4,r6
        bcc     no_hubo_acarreo
        add     r1,r1,#0x1
no_hubo_acarreo
```


Procesamiento de Datos

- Aritméticas
  - ADD r3,r2,#1
  - ADD r3,r2,r1, lsl #3 (lsr, asl, asr, ror, rrx)
  - ADD r5,r5,r3, LSL r2
  - MUL r4,r3,r2
  - MLA r4,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

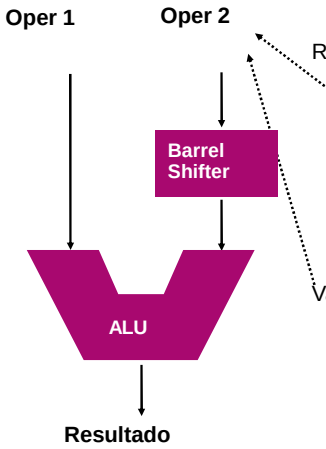
Solo afectan los Flags

```
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



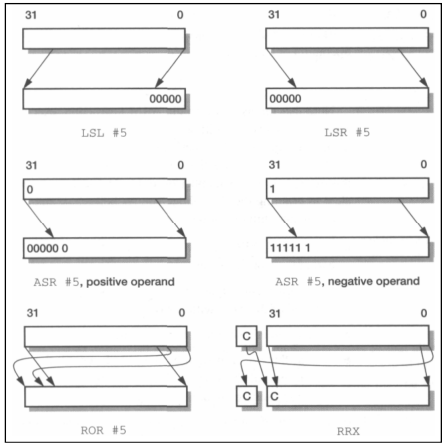
Registro, opcionalmente con operación de desplazamiento
- El valor del desplazamiento puede ser:
  - 5 bit entero no signado
  - Especificado en el LSB de otro registro.
- Usado para multiplicar por una constante

Valor inmediato
- 8 bit : Número en el rango de 0-255.
  - Rotado a la derecha en un número par de posiciones
- Permite que sean cargados directamente a los registros constantes de 32-bits

Procesamiento de Datos
Desplazamientos

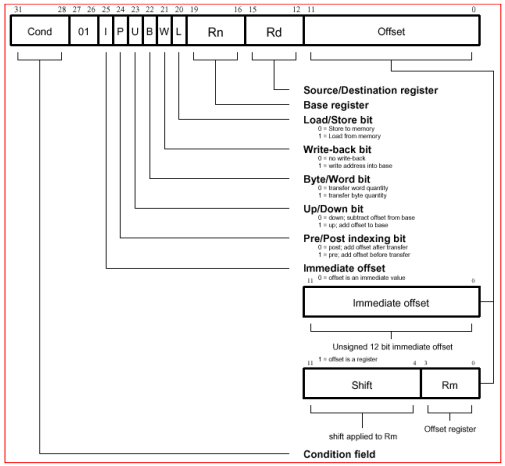
- ADD r3, r2, r1, LSL #3 ; r3 := r2 + 8 x r1
- ADD r5, r5, r3, LSL r2 ; r5 : = r5 + r3 x 2^{r2}
- RSB r0,r0,r0, LSL #3
- ; Multiplicar por 7

Desplazamientos



• INSTRUCCIONES DE TRANSFERENCIA DE DATOS

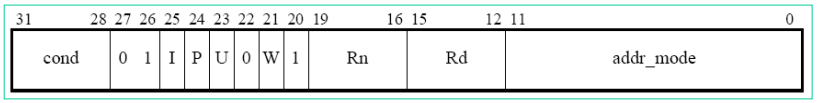
Transferencias de Datos



Transferencias de Datos

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

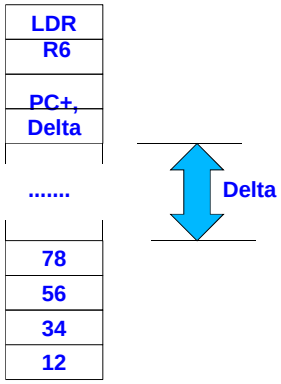
Transferencias de Datos
LDR




```
LDR   r0,[r1]          ; r0 := mem32[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
```



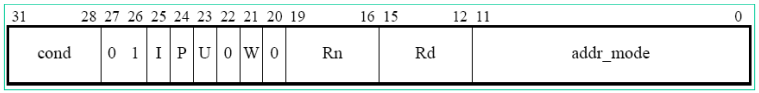
Transferencias de Datos
Pre y post indexado

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

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

Transferencias de Datos

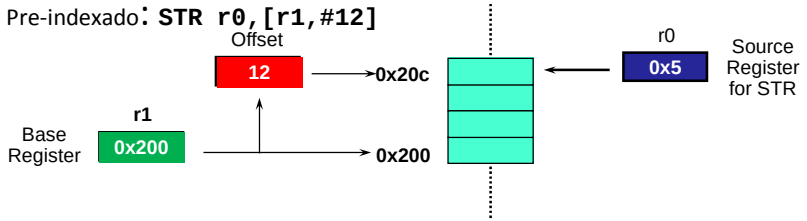
STR



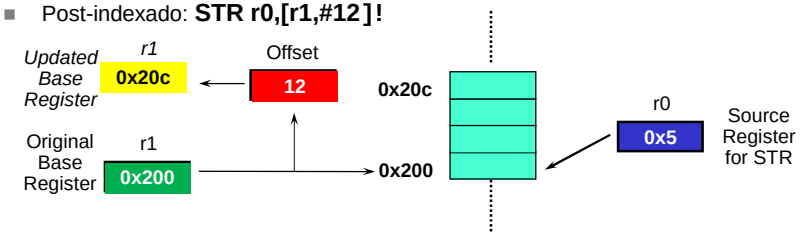

```
STR     r0,[r1]              ; mem32[r1] := r0
STR     R0, [R7, #0x7C] ; Guarda word desde R0 a direc R7 + 124
STRB    R1, [R5, #31] ; Guarda byte desde R1 a direc R5 + 31
STRH    R4, [R2, R3] ; Guarda halfword desde R4 a dir R2 + R3
STR     R4, [SP, #0x260] ; Guarda R5 a direc. SP + 0x260
```


Preindexado y postindexado

Pre-indexado: **`STR r0,[r1,#12]`**



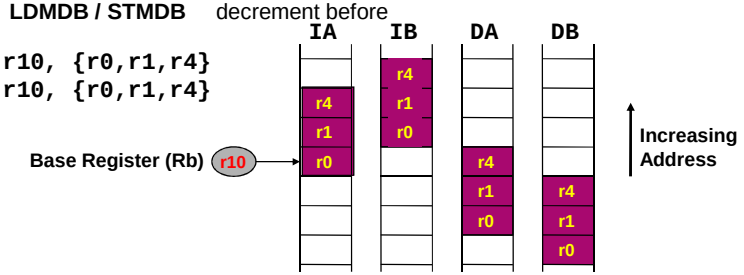
- Post-indexado: **STR r0,[r1,#12]!**



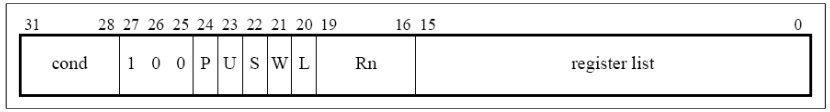
Multiples Load y Store Registros

- 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

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



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

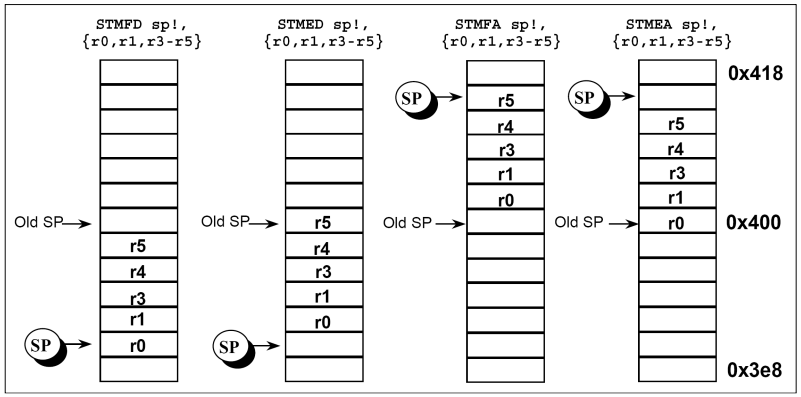
```
LDMIA  r0!, {r2-r9}
STMIA  r1,  {r2-r9}
```

```
STMFD   r13!,   {r2-r9}
LDMIA   r0!,    {r2-r9}
STMIA   r1,     {r2-r9}
LDMFD   r13!,   {r2-r9}
```

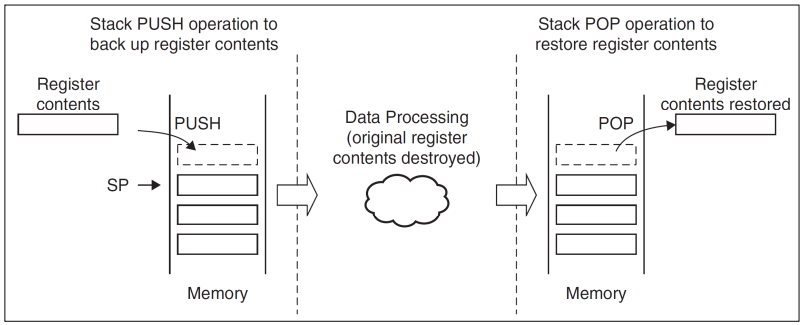
Uso de la Pila

- STMFD / LDMFD : Full Descending stack
- STMFA / LDMFA : Full Ascending stack.
- STMED / LDMED : Empty Descending stack
- STMEA / LDMEA : Empty Ascending stack

Uso de la Pila

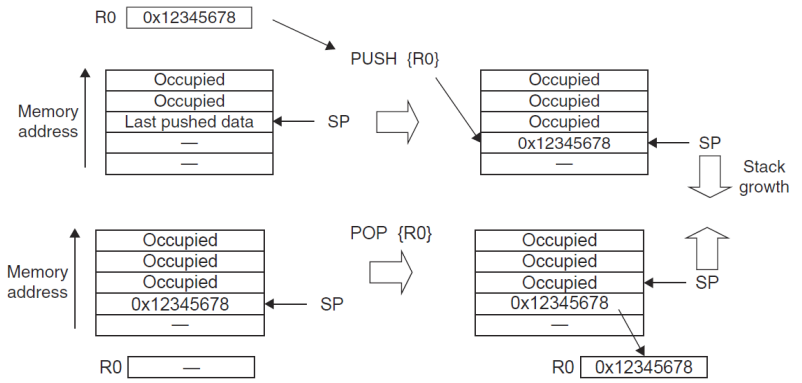


Push y Pop



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

Push y Pop Ampliados



Uso de la pila en subrutinas

```
; R0 = X, R1 = Y, R2 = Z
BL funcion1
; Regresa al prgma 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
```


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
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

Condensado Almacenado en el stack al Inicio de excepción

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)[1]
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

Saltos - Branch

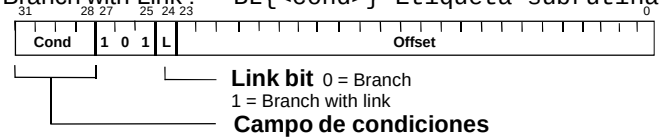
- Instrucciones de salto (Branching): BX, B, BL
- B: salto con desplazamiento de 24 bits con signo
- BL: enlace (link) PC -> R14
- Instrucciones de transferencia de datos: LDR, STR, LDRH, STRH, LDRSB, LDRSH, LDM, STM, SWP.

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 BLO	Carry clear Lower	Arithmetic operation did not give carry-out Unsigned comparison gave lower
BCS BHS	Carry set Higher or same	Arithmetic operation gave carry-out Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT BGE	Greater than Greater or equal	Signed integer comparison gave greater than 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

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?

Subrutinas

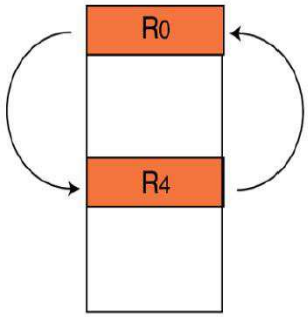
```
   BL      subru
…
Subru: ….
   mov     pc,r14
```

Instrucciones de Swap

También llamadas Instrucciones de semáforos

SWP R12, R10, [R9]
; cargar R12 desde la dirección apuntada
; por R9 y 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

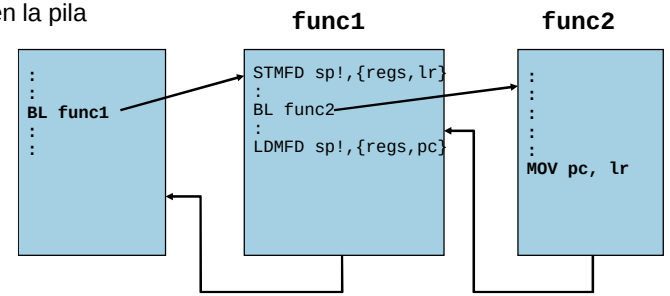
Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this 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.

Uso de Registros

	Register	
Argumentos a funcion Resultado(s) de functon alterables ((Parámetros pasados en la pila)	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/sl r11	Si se vinculan tutinas 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 serusado una vez que se almacenó en pila**

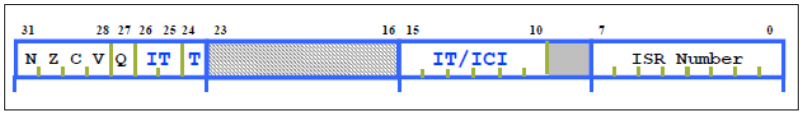
Branches y subrutinas en ARM

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



Instrucciones reservadas de control

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 lcambiar los modos escribiendo al CPSR
  - Típicamente se deben emplear estragias 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 midificado al byte‘c’
```

- En modo usuario sólo pueden modificarse los flags

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.

Registros especiales

Registro	Función
xPSR	Provee los flags de la ALU (zero, carry), status de la ejecución y número de la interrupción actuamente ejecutada
PRIMASK	Deshabilita todas las interrupciones salvo la no enmascarable (NMI) y HardFault
FAULTMASK	Deshabilita todas las interrupcones 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