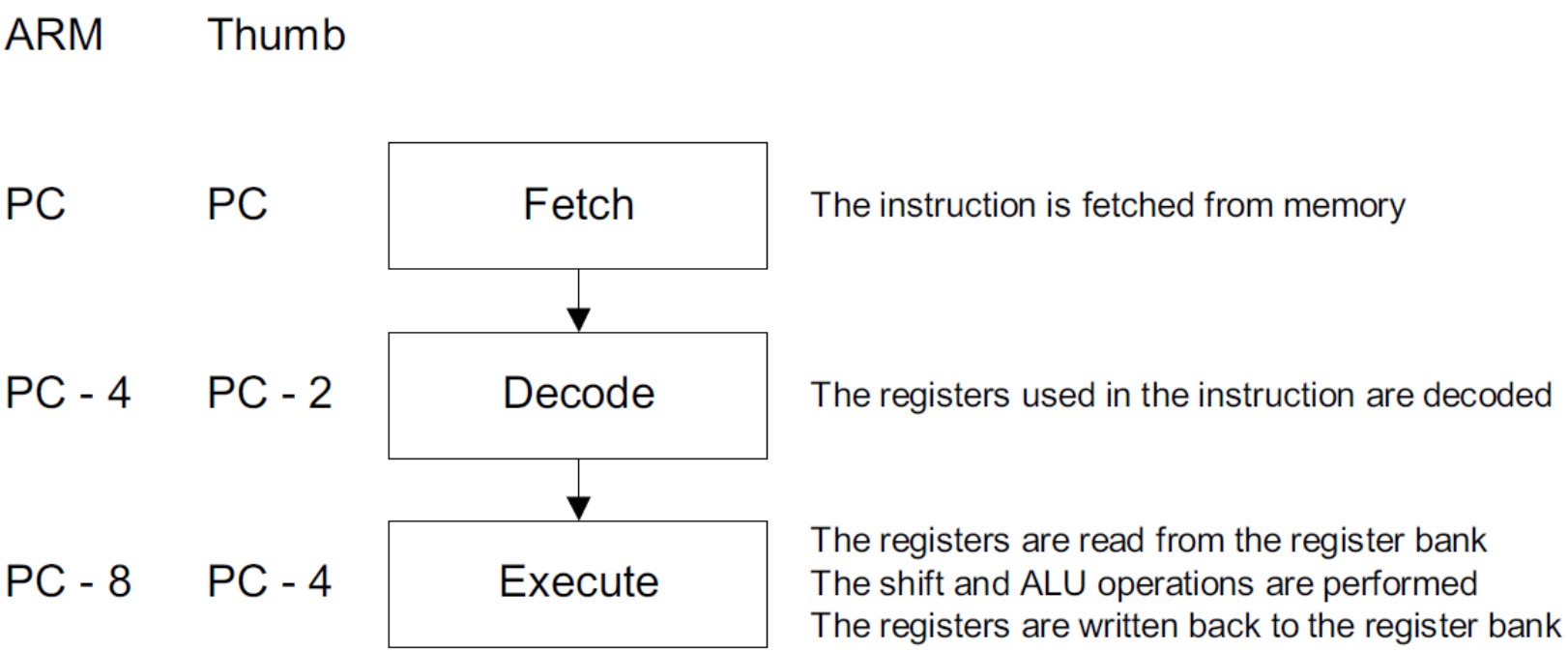
Set de instrucciones ARM-7

Introducción

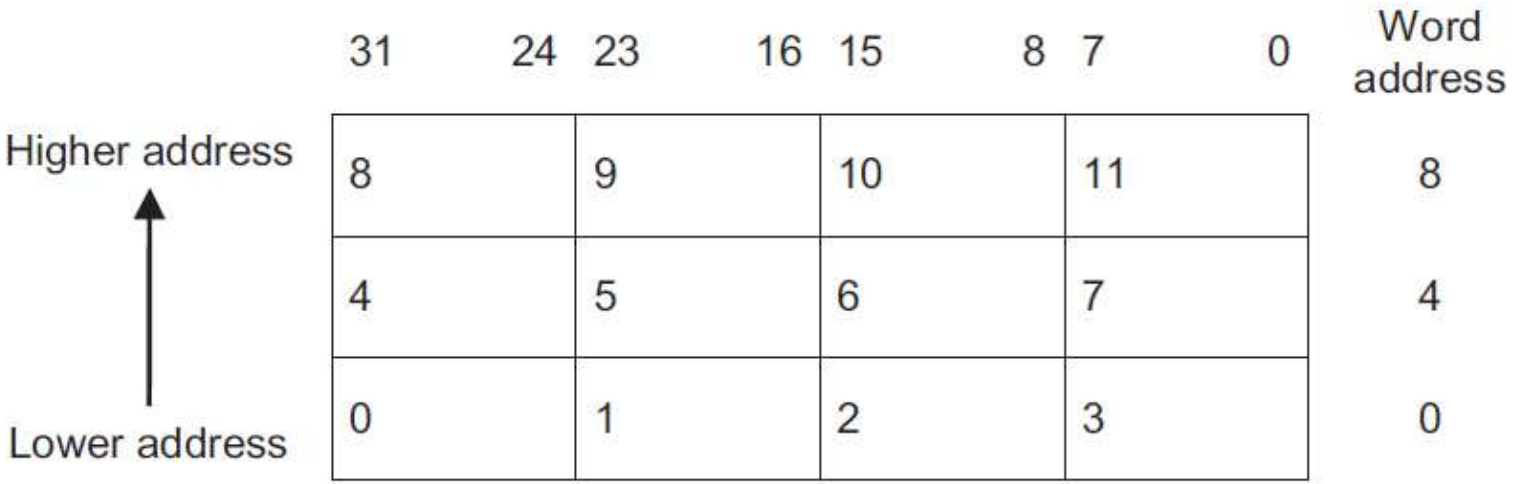
Tipos de operandos que soporta ARM

Byte	8 bits
Halfword	16 bits
Word	32 bits
Doubleword	64 bits.

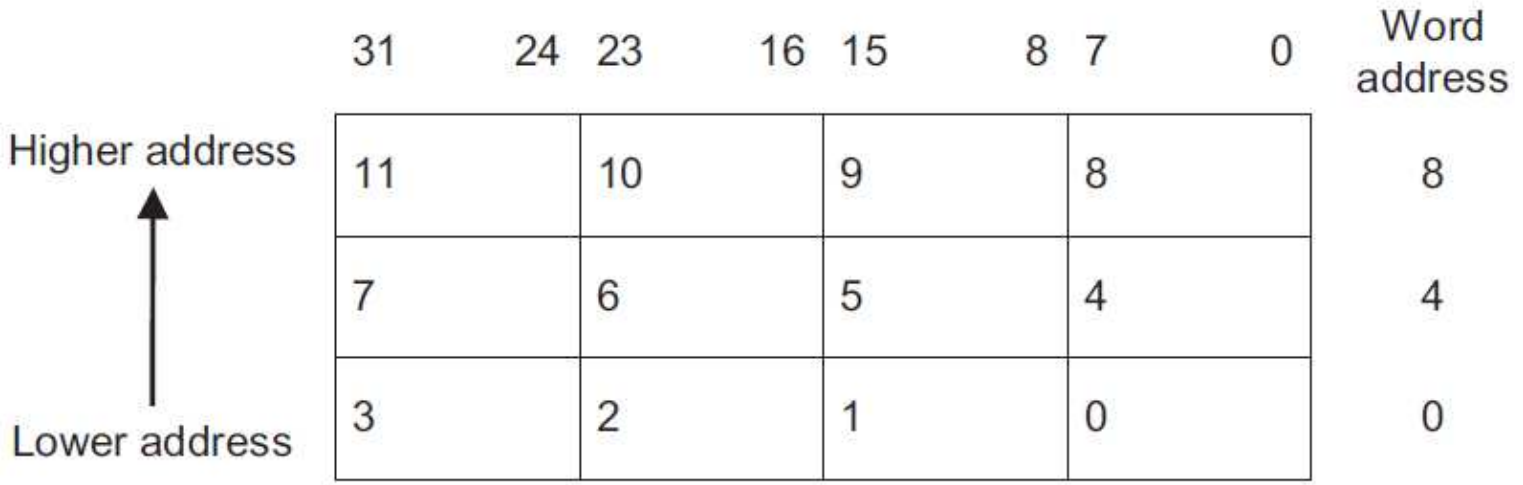
La cola de ejecución



Datos en formato Big Endian



Datos en formato Little Endian



Tipos de instrucciones. Sumario.

Move

Move	`MOV{cond}{S} Rd, <Oprnd2>`
Move NOT	`MVN{cond}{S} Rd, <Oprnd2>`
Move SPSR to register	`MRS{cond} Rd, SPSR`
Move CPSR to register	`MRS{cond} Rd, CPSR`
Move register to SPSR	`MSR{cond} SPSR{field}, Rm`
Move register to CPSR	`MSR{cond} CPSR{field}, Rm`
Move immediate to SPSR flags	`MSR{cond} SPSR_f, #32bit_Imm`
Move immediate to CPSR flags	`MSR{cond} CPSR_f, #32bit_Imm`

Instrucción MOV

Es la mas simple de las instrucciones de ARM. Copia *N* al registro destino *Rd.*

MOV
Copia un valor de 32 bits a un registro
MVN
Copia el complemento a uno de un valor de 32 bits a un registro

Se la usa para inicializar valores y para transferir datos entre registros.

Sintaxis:

<instrucción>{<cond>}{S} Rd, N

N es usualmente un registro Rm o una constante precedida por #.

Ejemplo:

Valores previos:

r5 = 8
r4 = 5

MOV r5, r4 *;* let r5 = r4

Valores posteriores:

r5 = 5
r4 = 5

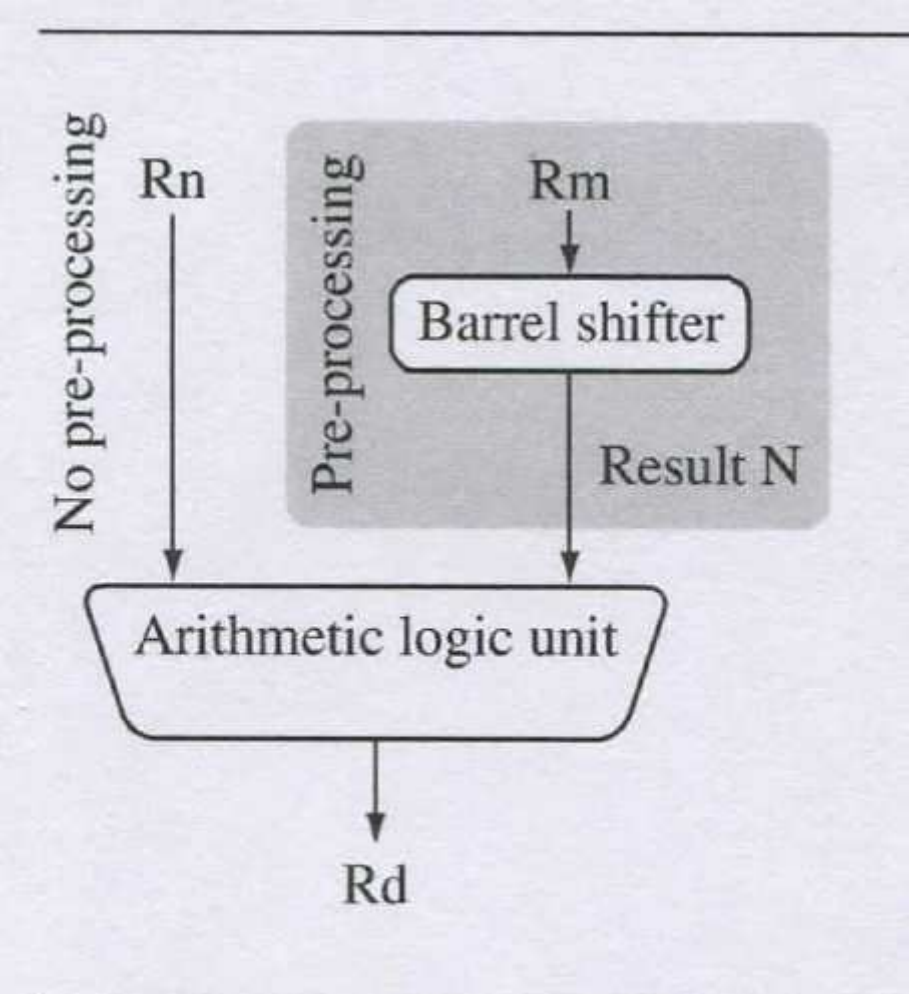
Uso del rotador por hardware (barrel shifter)

La instrucción MOV puede ser combinada con el barrel shifter para producir resultados mas interesantes:

```
MOV     r5, r4, LSL#2     ; let r5 = r4 * 4  (r4 << 2)
```

Posteriores:

r5 = 20
r4 = 5

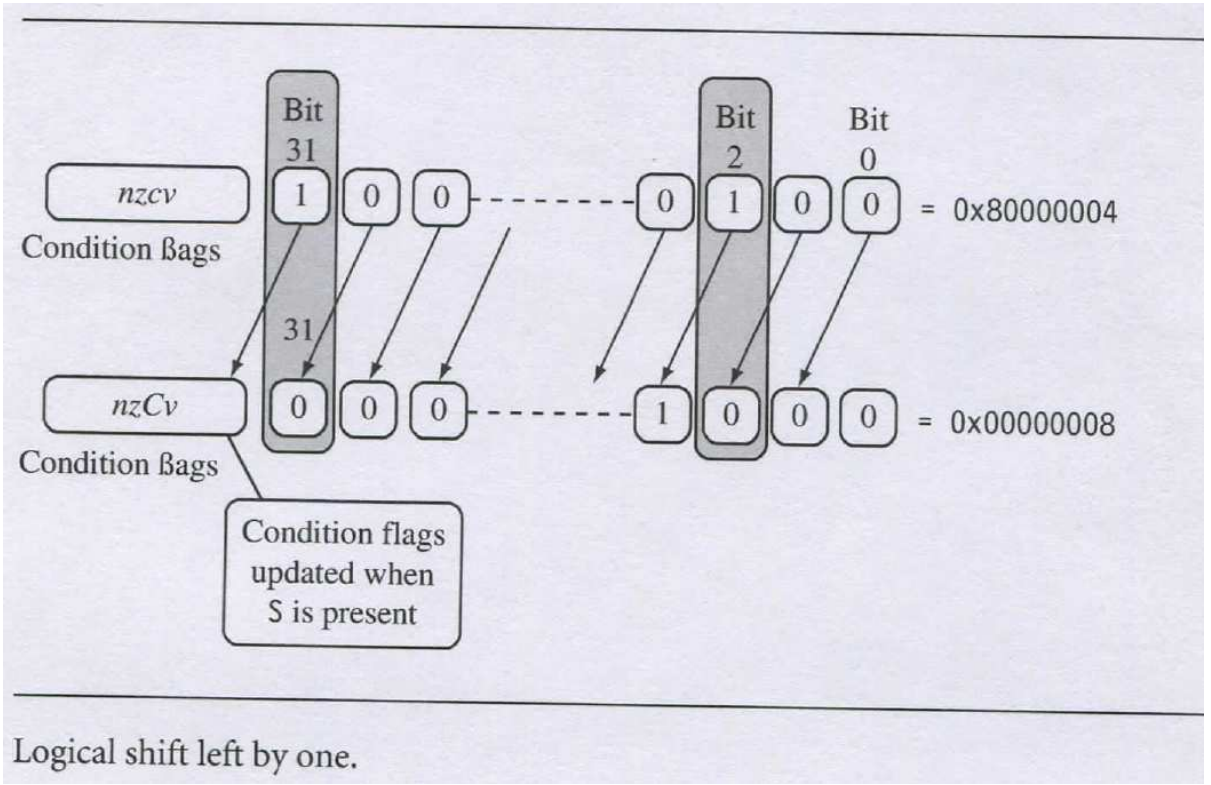


Barrel shifter and ALU.

Table 3.2 Barrel shifter operations.

Mnemonic	Description	Shift	Result	Shift amount y
LSL	logical shift left	*x*LSL *y*	$x \ll y$	#0–31 or *Rs*
LSR	logical shift right	*x*LSR *y*	(unsigned)$x \gg y$	#1–32 or *Rs*
ASR	arithmetic right shift	*x*ASR *y*	(signed)$x \gg y$	#1–32 or *Rs*
ROR	rotate right	*x*ROR *y*	((unsigned)$x \gg y$) | ($x \ll (32 - y)$)	#1–31 or *Rs*
RRX	rotate right extended	*x*RRX	(*c* flag $\ll 31$) | ((unsigned)$x \gg 1$)	none

Note: *x* represents the register being shifted and *y* represents the shift amount.



Logical shift left by one.

Arithmetic	Add	`ADD{cond}{S} Rd, Rn, <Oprnd2>`
	Add with carry	`ADC{cond}{S} Rd, Rn, <Oprnd2>`
	Subtract	`SUB{cond}{S} Rd, Rn, <Oprnd2>`
	Subtract with carry	`SBC{cond}{S} Rd, Rn, <Oprnd2>`
	Subtract reverse subtract	`RSB{cond}{S} Rd, Rn, <Oprnd2>`
	Subtract reverse subtract with carry	`RSC{cond}{S} Rd, Rn, <Oprnd2>`
	Multiply	`MUL{cond}{S} Rd, Rm, Rs`
	Multiply accumulate	`MLA{cond}{S} Rd, Rm, Rs, Rn`
	Multiply unsigned long	`UMULL{cond}{S} RdLo, RdHi, Rm, Rs`
	Multiply unsigned accumulate long	`UMLAL{cond}{S} RdLo, RdHi, Rm, Rs`
	Multiply signed long	`SMULL{cond}{S} RdLo, RdHi, Rm, Rs`
	Multiply signed accumulate long	`SMLAL{cond}{S} RdLo, RdHi, Rm, Rs`
	Compare	`CMP{cond} Rd, <Oprnd2>`
	Compare negative	`CMN{cond} Rd, <Oprnd2>`

Operaciones aritméticas: suma y resta.

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

This simple subtract instruction subtracts a value stored in register *r2* from a value stored in register *r1*. The result is stored in register *r0*.

```
PRE     r0 = 0x00000000
        r1 = 0x00000002
        r2 = 0x00000001

        SUB r0, r1, r2

POST    r0 = 0x00000001
```

Instrucciones de multiplicación:
Realizan el producto entre dos registros de 32 bits, pudiendo acumular el resultado con otro registro.
El resultado se puede guardar en un registro o en un par de ellos (64 bits)

```
Syntax: MLA{<cond>}{S} Rd, Rm, Rs, Rn
        MUL{<cond>}{S} Rd, Rm, Rs
```

MLA	multiply and accumulate	$Rd = (Rm^* Rs) + Rn$
MUL	multiply	$Rd = Rm^* Rs$

Logical	Test	`TST{cond} Rn, <Oprnd2>`
	Test equivalence	`TEQ{cond} Rn, <Oprnd2>`
	AND	`AND{cond}{S} Rd, Rn, <Oprnd2>`
	EOR	`EOR{cond}{S} Rd, Rn, <Oprnd2>`
	ORR	`ORR{cond}{S} Rd, Rn, <Oprnd2>`
	Bit clear	`BIC{cond}{S} Rd, Rn, <Oprnd2>`

Syntax: <instruction>{<cond>}{S} Rd, Rn, N

AND	logical bitwise AND of two 32-bit values	$Rd = Rn \,\&\, N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn \mid N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \,\hat{}\, N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \,\&\, \sim N$

This example shows a logical OR operation between registers *r1* and *r2*. *r0* holds the result.

```
PRE     r0 = 0x00000000
        r1 = 0x02040608
        r2 = 0x10305070
```

```
        ORR   r0, r1, r2

POST    r0 = 0x12345678
```

Instrucciones de comparación

Syntax: <instruction>{<cond>} Rn, N

CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \,\&\, N$

Branch	Branch	`B{cond} label`
	Branch with link	`BL{cond} label`
	Branch and exchange instruction set	`BX{cond} Rn`

Store	Word	`STR{cond} Rd, <a_mode2>`
	Word with User-mode privilege	`STR{cond}T Rd, <a_mode2P>`
	Byte	`STR{cond}B Rd, <a_mode2>`
	Byte with User-mode privilege	`STR{cond}BT Rd, <a_mode2P>`
	Halfword	`STR{cond}H Rd, <a_mode3>`
	Multiple	-
	Block data operations	-
	Increment before	`STM{cond}IB Rd{!}, <reglist>{^}`
	Increment after	`STM{cond}IA Rd{!}, <reglist>{^}`
	Decrement before	`STM{cond}DB Rd{!}, <reglist>{^}`
	Decrement after	`STM{cond}DA Rd{!}, <reglist>{^}`
	Stack operations	`STM{cond}<a_mode4S> Rd{!}, <reglist>`
	User registers	`STM{cond}<a_mode4S> Rd{!}, <reglist>^`

Stack operations and restore CPSR	`LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^`
User registers	`LDM{cond}<a_mode4L> Rd{!}, <reglist>^`

Swap	Word	`SWP{cond} Rd, Rm, [Rn]`
	Byte	`SWP{cond}B Rd, Rm, [Rn]`

Coprocessors

Data operations	`CDP{cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>`
Move to ARM register from coprocessor	`MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`
Move to coprocessor from ARM register	`MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>`
Load	`LDC{cond} p<cpnum>, CRd, <a_mode5>`
Store	`STC{cond} p<cpnum>, CRd, <a_mode5>`

Software Interrupt

```
SWI 24bit_Imm
```

Table 1-11 Condition fields

Suffix	Description
EQ	Equal
NE	Not equal
CS	Unsigned higher, or same
CC	Unsigned lower
MI	Negative
PL	Positive, or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower, or same

Códigos de condición para las operaciones