

# ARM Overview

# Introducción

- La última versión es ARMv8.
- Hay instrucciones enteras, de punto flotante y SIMD.
- Soporta estados de ejecución de 32 (AArch32) y 64 bits (AArch64).
- AArch32 usa registros y direcciones de memoria de 32 bits y usa el ISA A32.
- AArch64 usa registros y direcciones de memoria de 64 bits y usa el ISA A64.
- En ambos casos, las instrucciones son de tamaño fijo de 4 bytes.

# Introducción

- A32 y A64 usan operandos de registros distintos y algunos opcodes distintos.
- Los programas escritos para el estado de ejecución AArch64 no son compatibles con AArch32 y viceversa.
- AArch32 y AArch64 se pueden configurar como little-endian o como big-endian.
- Las instrucciones se codifican en little-endian.
- ARM utiliza arquitectura load-store.

# Similitudes ARM y MIPS

|                                         | ARM                     | MIPS                    |
|-----------------------------------------|-------------------------|-------------------------|
| Date announced                          | 1985                    | 1985                    |
| Instruction size (bits)                 | 32                      | 32                      |
| Address space (size, model)             | 32 bits, flat           | 32 bits, flat           |
| Data alignment                          | Aligned                 | Aligned                 |
| Data addressing modes                   | 9                       | 3                       |
| Integer registers (number, model, size) | 15 GPR $\times$ 32 bits | 31 GPR $\times$ 32 bits |
| I/O                                     | Memory mapped           | Memory mapped           |

Fuente: COD5, p.146

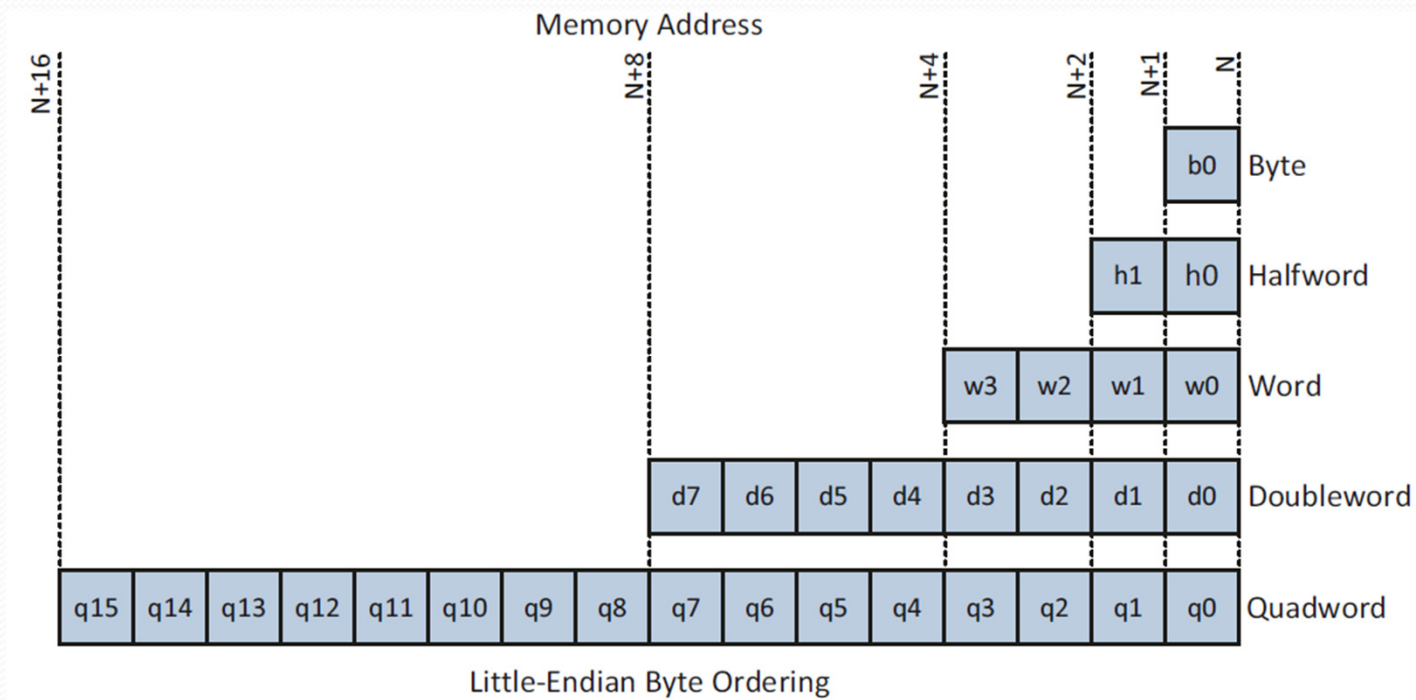
# Tipos de datos

*Table 1-1. AArch32 and AArch64 fundamental data types*

| Data Type  | Size (bits) | Typical Use                                                                                                                                                                                    |
|------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Byte       | 8           | Characters<br>Byte integers                                                                                                                                                                    |
| Halfword   | 16          | Halfword integers<br>Half-precision floating-point                                                                                                                                             |
| Word       | 32          | Wide characters<br>Word integers<br>Single-precision floating-point                                                                                                                            |
| Doubleword | 64          | Doubleword integers<br>Double-precision floating-point<br>Packed integers<br>Packed half-precision floating-point<br>Packed single-precision floating-point                                    |
| Quadword   | 128         | Quadword integers (AArch64 only)<br>Packed integers<br>Packed half-precision floating-point<br>Packed single-precision floating-point<br>Packed double-precision floating-point (AArch64 only) |

Fuente: MAALP, p. 3

# Little-endian



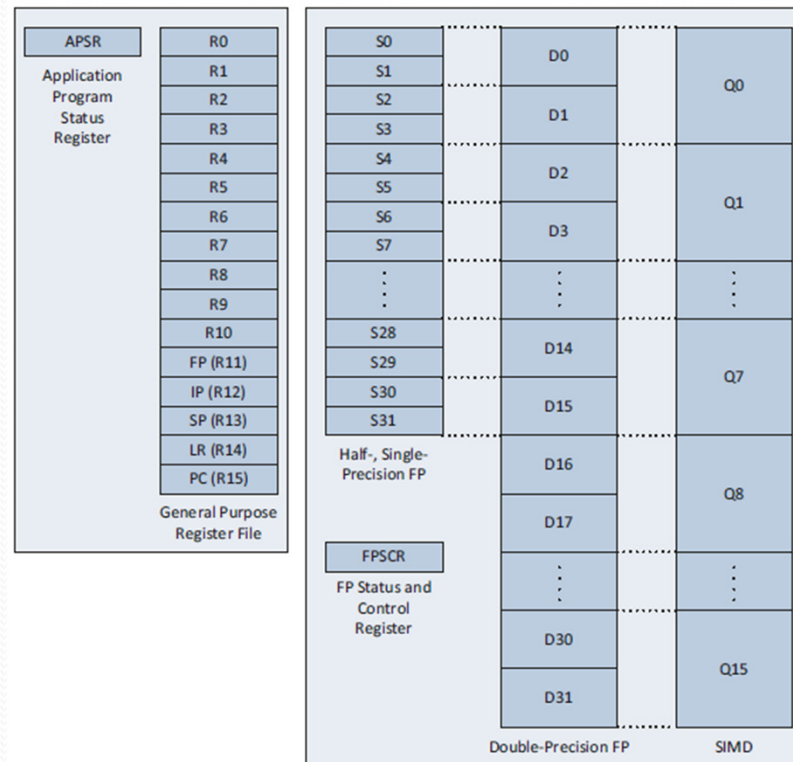
Fuente: MAALP, p. 4

# Big-endian



Fuente: MAALP, p. 4

# Registros AArch32



Fuente: MAALP, p. 6

# Registros AArch32

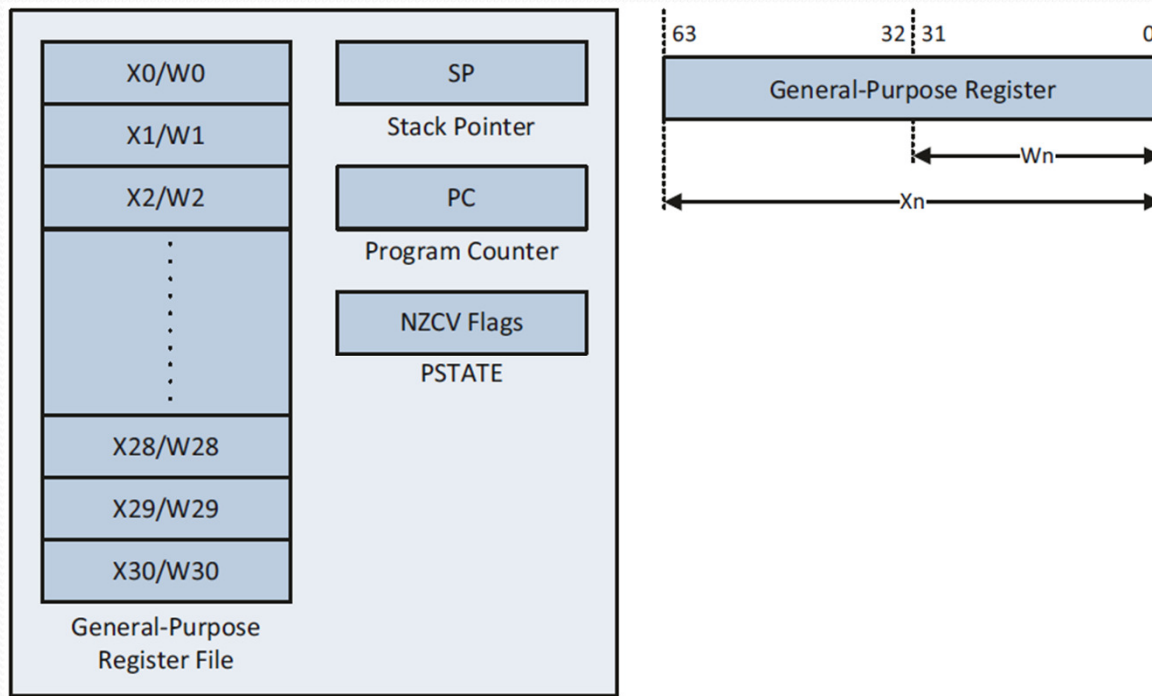
- R0 a R12 son registros de propósito general.
- R13 es el apuntador al tope de pila (stack pointer).
- R14 guarda la dirección de retorno de una subrutina (link register).
- R15 es el contador de programa y es accesible por el programador.

# Registro APSR

| Bit Position | Symbol  | Name                                             |
|--------------|---------|--------------------------------------------------|
| 31           | N       | Negative condition flag                          |
| 30           | Z       | Zero condition flag                              |
| 29           | C       | Carry condition flag                             |
| 28           | V       | Overflow condition flag                          |
| 27           | Q       | Overflow or saturation flag                      |
| 26:24        |         | Reserved 0                                       |
| 23:20        |         | Reserved for system features or future expansion |
| 19:16        | GE[3:0] | Parallel add/sub greater than or equal flags     |
| 15:0         |         | Reserved for system features or future expansion |

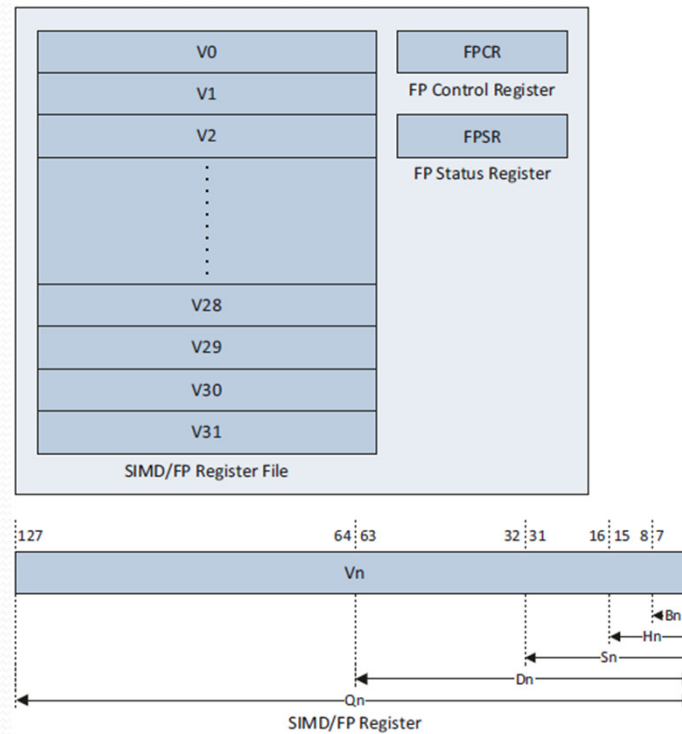
Fuente: MAALP, p. 8

# Registros enteros AArch64



Fuente: MAALP, p. 218

# Registros FP y SIMD AArch64



Fuente: MAALP, p. 219

# Instrucciones A32 vs MIPS

|                   | Instruction name              | ARM                | MIPS         |
|-------------------|-------------------------------|--------------------|--------------|
| Register-register | Add                           | add                | addu, addiu  |
|                   | Add (trap if overflow)        | adds; swivs        | add          |
|                   | Subtract                      | sub                | subu         |
|                   | Subtract (trap if overflow)   | subs; swivs        | sub          |
|                   | Multiply                      | mul                | mult, multu  |
|                   | Divide                        | —                  | div, divu    |
|                   | And                           | and                | and          |
|                   | Or                            | orr                | or           |
|                   | Xor                           | eor                | xor          |
|                   | Load high part register       | —                  | lui          |
|                   | Shift left logical            | lsl <sup>1</sup>   | sllv, sll    |
|                   | Shift right logical           | lsr <sup>1</sup>   | srlv, srl    |
|                   | Shift right arithmetic        | asr <sup>1</sup>   | srav, sra    |
|                   | Compare                       | cmp, cmn, tst, teq | slt/l,slt/lu |
| Data transfer     | Load byte signed              | ldrsb              | lb           |
|                   | Load byte unsigned            | ldrb               | lbu          |
|                   | Load halfword signed          | ldrsh              | lh           |
|                   | Load halfword unsigned        | ldrh               | lhu          |
|                   | Load word                     | ldr                | lw           |
|                   | Store byte                    | strb               | sb           |
|                   | Store halfword                | strh               | sh           |
|                   | Store word                    | str                | sw           |
|                   | Read, write special registers | mrs, msr           | move         |
|                   | Atomic Exchange               | swp, swpb          | ll;sc        |

Fuente: COD5, p. 146

# Addressing

| Name                                   | Alternative Name                          | ARM Examples                   |
|----------------------------------------|-------------------------------------------|--------------------------------|
| Register to register                   | Register direct                           | MOV R0, R1                     |
| Absolute                               | Direct                                    | LDR R0, MEM                    |
| Literal                                | Immediate                                 | MOV R0, #15<br>ADD R1, R2, #12 |
| Indexed, base                          | Register indirect                         | LDR R0, [R1]                   |
| Pre-indexed,<br>base with displacement | Register indirect<br>with offset          | LDR R0, [R1, #4]               |
| Pre-indexed,<br>autoindexing           | Register indirect<br>pre-incrementing     | LDR R0, [R1, #4]!              |
| Post-indexing,<br>autoindexed          | Register indirect<br>post-increment       | LDR R0, [R1], #4               |
| Double Reg indirect                    | Register indirect<br>Register indexed     | LDR R0, [R1, R2]               |
| Double Reg indirect<br>with scaling    | Register indirect<br>indexed with scaling | LDR R0, [R1, R2, LSL #2]       |
| Program counter relative               |                                           | LDR R0, [PC, #offset]          |

Fuente: <https://www.labs.cs.uregina.ca/301/ARM-addressing/lecture.html>

# Addressing modes

| Addressing Mode             | Assembly Mnemonic | Effective address | FinalValue in R1 |
|-----------------------------|-------------------|-------------------|------------------|
| Pre-indexed, base unchanged | LDR R0, [R1, #d]  | $R1 + d$          | R1               |
| Pre-indexed, base updated   | LDR R0, [R1, #d]! | $R1 + d$          | $R1 + d$         |
| Post-indexed, base updated  | LDR R0, [R1], #d  | R1                | $R1 + d$         |

Fuente: <https://www.labs.cs.uregina.ca/301/ARM-addressing/lecture.html>

# Brincos condicionales

- MIPS utiliza el contenido de los registros para evaluar brincos condicionales.  
    `beq $s0, $s1, Label`
- ARM utiliza 4 banderas de un bit almacenadas en el APSR (Application Program Status Register): negativo, cero, carry y overflow.
- Las instrucciones aritméticas y lógicas cambian estos bits.

# Brincos condicionales

- La instrucción `cmp` (compara) resta un operando del otro y la diferencia cambia las banderas.
- Ejemplo:
- MIPS  
`beq $s0, $s1, Label`
- ARM  
`cmp r0, r1`  
`b.eq Label`

# Brincos condicionales

- La instrucción `cmn` (compara negativo) suma un operando al otro y la suma cambia las banderas.
- La instrucción `tst` hace un AND lógico de sus dos operandos y cambia todas las banderas, excepto overflow.
- La instrucción `teq` hace un XOR lógico de sus dos operandos y cambia todas las banderas, excepto overflow.

# Códigos de condición

| Encoding | Name (& alias)  | Meaning (integer)                      | Meaning (floating point)          | Flags           |
|----------|-----------------|----------------------------------------|-----------------------------------|-----------------|
| 0000     | EQ              | Equal                                  | Equal                             | Z==1            |
| 0001     | NE              | Not equal                              | Not equal, or unordered           | Z==0            |
| 0010     | HS<br>(CS)      | Unsigned higher or same<br>(Carry set) | Greater than, equal, or unordered | C==1            |
| 0011     | LO<br>(CC)      | Unsigned lower<br>(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 set                           | Unordered                         | V==1            |
| 0111     | VC              | Overflow clear                         | Ordered                           | V==0            |
| 1000     | HI              | Unsigned higher                        | Greater than, or unordered        | C==1 && Z==0    |
| 1001     | LS              | Unsigned lower or same                 | Less than or equal                | !(C==1 && Z==0) |
| 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 && N==V    |
| 1101     | LE              | Signed less than or equal              | Less than, equal, or unordered    | !(Z==0 && N==V) |
| 1110     | AL              |                                        |                                   |                 |
| 1111     | NV <sup>†</sup> | Always                                 | Always                            | Any             |

Fuente: <https://www.cs.princeton.edu/courses/archive/spr19/cos217/reading/ArInstructionSetOverview.pdf>

# Instrucciones únicas de ARM

| Name                              | Definition                                                 | ARM      | MIPS      |
|-----------------------------------|------------------------------------------------------------|----------|-----------|
| Load immediate                    | $Rd = Imm$                                                 | mov      | addi \$0, |
| Not                               | $Rd = \sim(Rs1)$                                           | mvn      | nor \$0,  |
| Move                              | $Rd = Rs1$                                                 | mov      | or \$0,   |
| Rotate right                      | $Rd = Rs \ll i$<br>$Rd_{0 \dots i-1} = Rs_{31-i \dots 31}$ | ror      |           |
| And not                           | $Rd = Rs1 \& \sim(Rs2)$                                    | bic      |           |
| Reverse subtract                  | $Rd = Rs2 - Rs1$                                           | rsb, rsc |           |
| Support for multiword integer add | CarryOut, $Rd = Rd + Rs1 + OldCarryOut$                    | adcs     | —         |
| Support for multiword integer sub | CarryOut, $Rd = Rd - Rs1 + OldCarryOut$                    | sbcs     | —         |

Fuente: COD5, p. 149

# Ejemplo

**; <http://kerseykyle.com/articles/ARM-assembly-hello-world>**

.text

.global \_start

\_start:

mov r0, #1

**; stdout is 0**

ldr r1, =message

ldr r2, =len

mov r7, #4

**; SYS\_WRITE is 4**

swi 0

**; software interrupt**

mov r7, #1

**; SYS\_EXIT is 1**

swi 0

# Ejemplo

.data

message:

.asciz "hello world\n"

len = .-message



# Bibliografía

## **Modern Arm Assembly Language Programming**

Daniel Kusswurm

Apress, 2020