ARM
Cortex-M

White Paper ARM: Cortex-M for Beginners

Libro: Cortex-M3 Guide

Web DaveSpace
Introduction to ARM (programación en assembler)
Efficient C for ARM

Empresa con base en Cambridge, UK fundada en 1990.
Arquitectura ARM de 32 bits (RISC) **licenciable**. No fabrican ni venden ICs. Entregan (deliverables) el core en RTL (register transfer level) sintetizable (por ejemplo Verilog). Modular, reconfigurable, tools.

Fue crucial para el desarrollo de sistemas portátiles de alta performance (smartphones). Android/iOS, **Cortex-A** (application) de 32 bits (2005) y 64 bits (2012).

Cortex-M (2005, embedded) de 32 bits..

En 2016 fue adquirida por el holding japonés SoftBank (32G$)
Desde 2020 NVidia (US) está intentando comprarla (40G$)

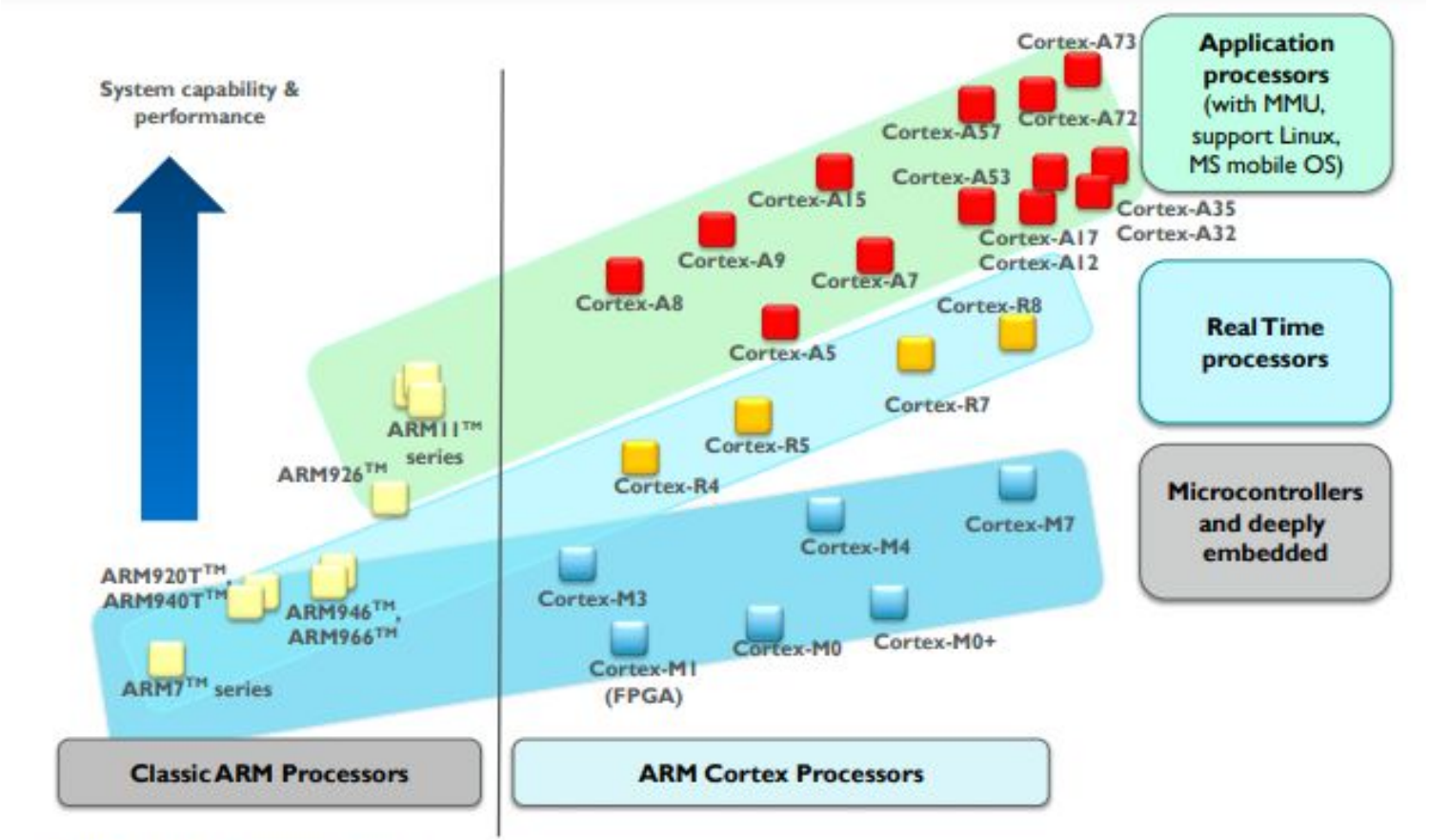
China se opone (Huawei)
Googlear noticias… 2021/22

Empresas licenciatarias de ARM:
Apple, Intel, DEC, NVIDIA, Qualcomm, Samsung, NXP, IBM, Texas Instruments, Nintendo, Philips, Huawei, Sharp, Samsung, Atmel y STMicroelectronics.

The Architecture for the Digital World®

ARM®

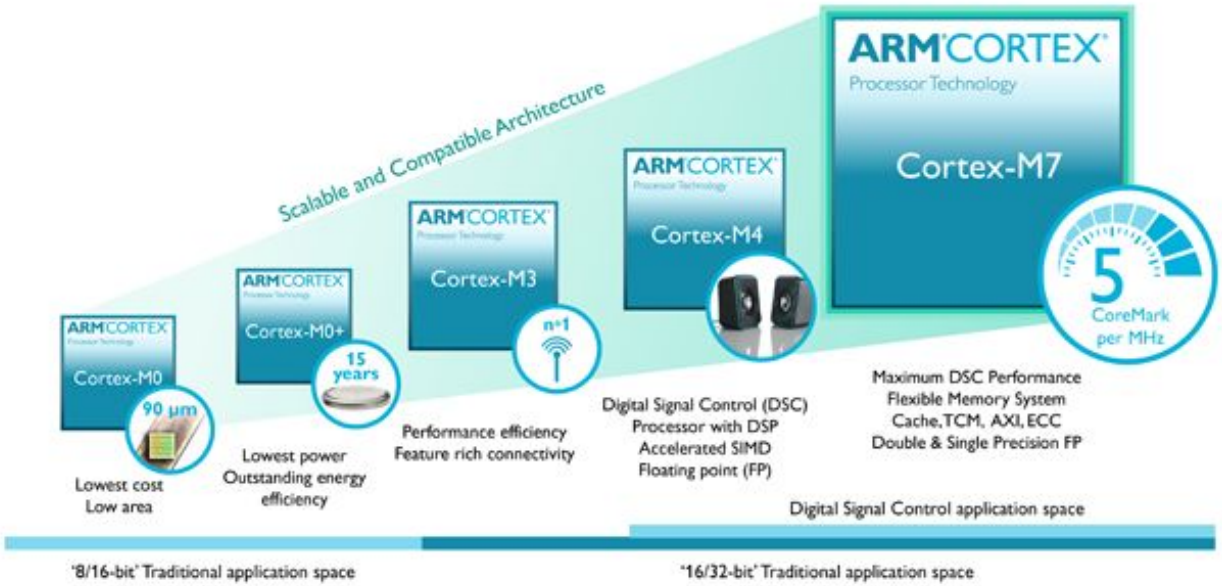
Los procesadores de ARM



Cortex M

Procesadores ARM para aplicaciones embebidas

Son una familia de MCU (Micro-Controller Unit). **No tienen MMU (Memory Management Unit),** necesaria para correr un sistema operativo de propósitos generales (por ejemplo Android en Cortex A). Tienen **MPU** (Memory Protection Unit) útil para RTOS.



Wikipedia:
Cortex-M have become a popular replacements for 8-bit chips in applications that benefit from 32-bit math operations.

ARM Cortex-M Instruction Sets[6][7]

ARM Cortex-M	Thumb	Thumb-2	Hardware multiply	Hardware divide	Saturated math	DSP extensions	Floating-Point Unit (FPU)	ARM architecture
Cortex-M0[1]	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M
Cortex-M0+[2]	Most	Subset	1 or 32 cycle	No	No	No	No	ARMv6-M
Cortex-M1[3]	Most	Subset	3 or 33 cycle	No	No	No	No	ARMv6-M
Cortex-M3[4]	Entire	Entire	1 cycle	Yes	Yes	No	No	ARMv7-M
Cortex-M4[5]	Entire	Entire	1 cycle	Yes	Yes	Yes	Optional, SP	ARMv7E-M
Cortex-M7	Entire	Entire	1 cycle	Yes	Yes	Yes	Yes, SP & DP	ARMv7E-M

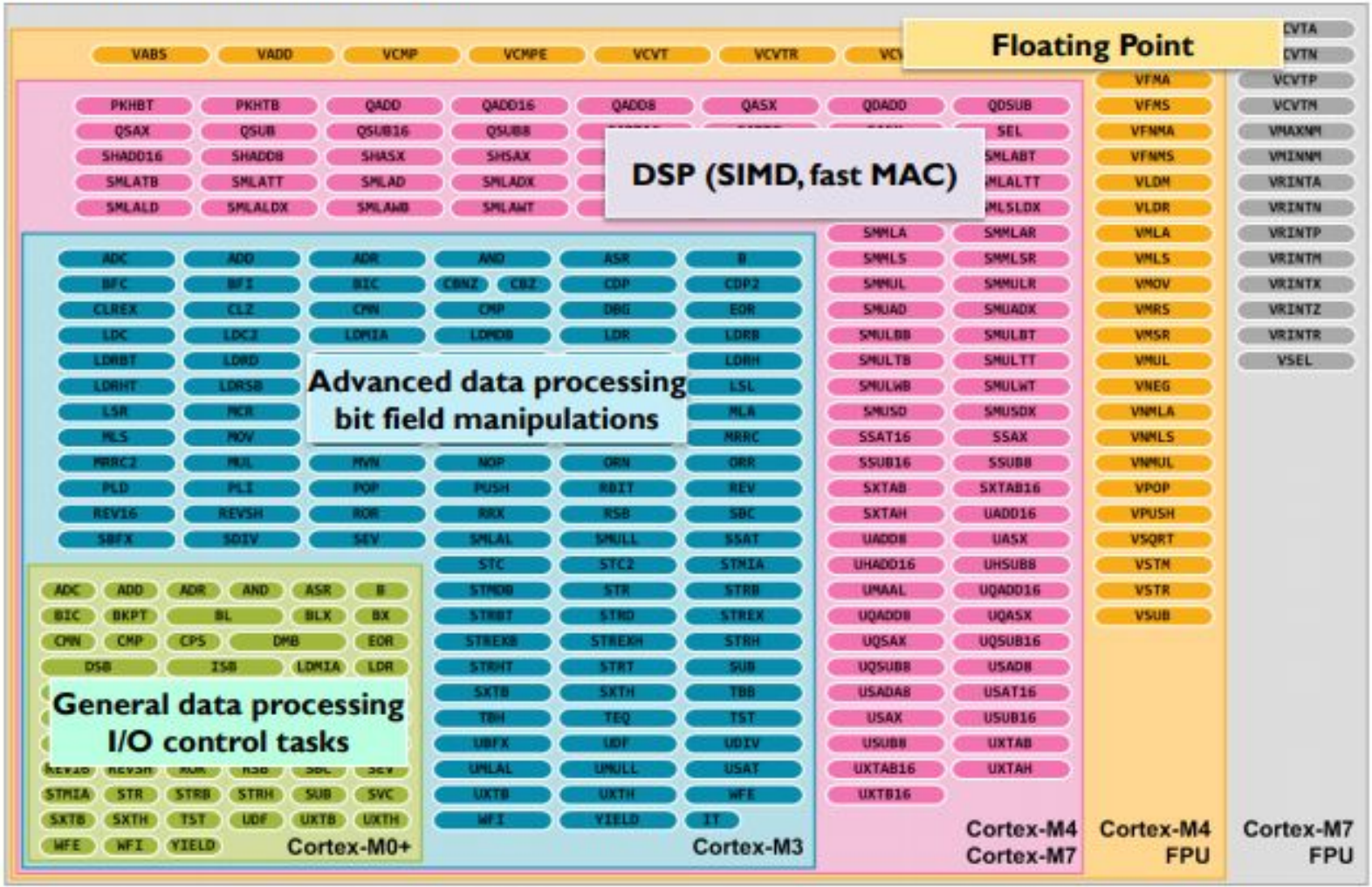
M4 DSP: Harvard, Single cycle MAC, Barrel shifter, SIMD, aritmética con saturación
M4 no-DSP: Circular and bit-reversed addressing, zero overhead loops, load and store operations in parallel with math operations

BENCHMARK: Decodifica MP3 con menos de 10 MHz de clock (un DSP común requiere 15 MHz y uno especializado de audio 5 MHz).

TOP
DSP y FPU
SIMD

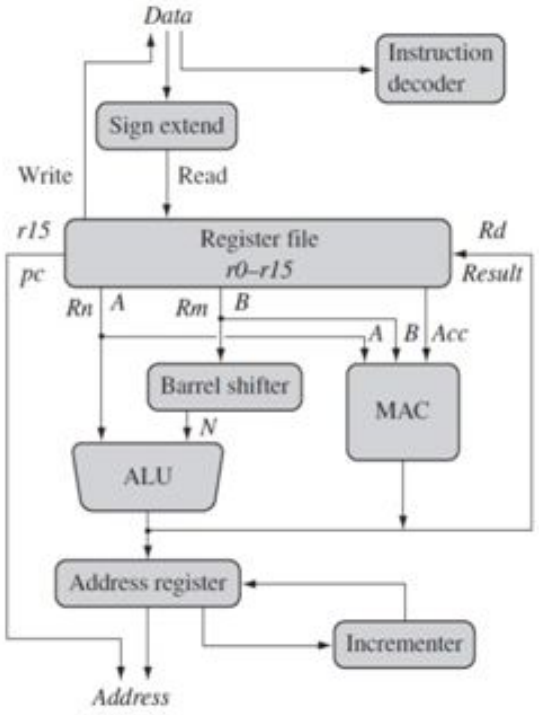
AVANZADO
División, MAC
multiciclo,
barrel shifter,
etc.

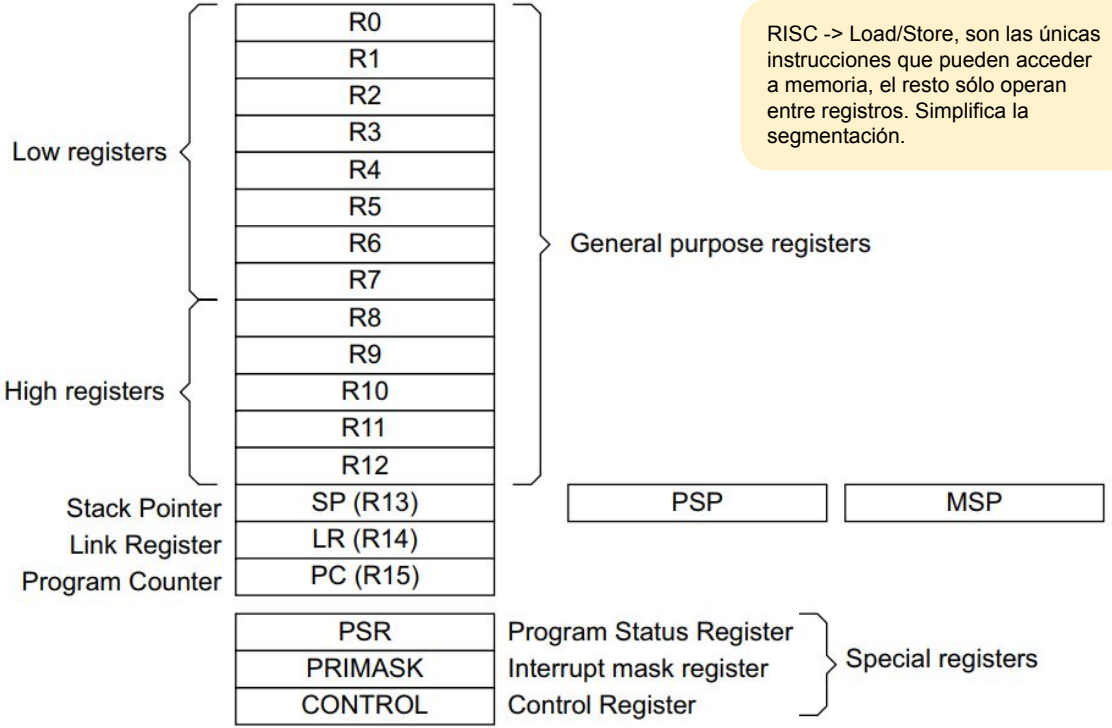
MINIMO
Algunas
instrucciones
de 16 bits



Arquitectura ARM básica

- RISC de 32-bit (load/store), von Neumann (4 GB memoria)
- 16 registros de 32 bits - r13/SP: stack pointer, r14/LR: link register (return address) y r15/PC + Status register: PSR
- Todos los registros pueden usarse para direccionamiento indirecto.
- 56 instrucciones base (++). Instrucciones de largo fijo de 32 bits + Thumb, algunas de 16-bits para mejorar la densidad
- **Barrel shifter en el datapath**
- **Ejecución condicional de todas las instrucciones**





RISC -> Load/Store, son las únicas instrucciones que pueden acceder a memoria, el resto sólo operan entre registros. Simplifica la segmentación.

ARM
Repertorio de instrucciones

Mnemonic	Instruction	Action	See Section:
ADC	Add with carry	Rd := Rn + Op2 + Carry	4.5
ADD	Add	Rd := Rn + Op2	4.5
AND	AND	Rd := Rn AND Op2	4.5
B	Branch	R15 := address	4.4
BIC	Bit Clear	Rd := Rn AND NOT Op2	4.5
BL	Branch with Link	R14 := R15, R15 := address	4.4
BX	Branch and Exchange	R15 := Rn, T bit := Rn[0]	4.3
CDP	Coprocesor Data Processing	(Coprocessor-specific)	4.14
CMN	Compare Negative	CPSR flags := Rn + Op2	4.5
CMP	Compare	CPSR flags := Rn - Op2	4.5
EOR	Exclusive OR	Rd := (Rn AND NOT Op2) OR (op2 AND NOT Rn)	4.5
LDC	Load coprocessor from memory	Coprocessor load	4.15
LDM	Load multiple registers	Stack manipulation (Pop)	4.11
LDR	Load register from memory	Rd := (address)	4.9, 4.10
MCR	Move CPU register to coprocessor register	cRn := rRn {<op>cRm}	4.16
MLA	Multiply Accumulate	Rd := (Rm * Rs) + Rn	4.7, 4.8
MOV	Move register or constant	Rd : = Op2	4.5
MRC	Move from coprocessor register to CPU register	Rn := cRn {<op>cRm}	4.16
MRS	Move PSR status/flags to register	Rn := PSR	4.6
MSR	Move register to PSR status/flags	PSR := Rm	4.6
MUL	Multiply	Rd := Rm * Rs	4.7, 4.8
MVN	Move negative register	Rd := 0xFFFFFFFF EOR Op2	4.5
ORR	OR	Rd := Rn OR Op2	4.5
RSB	Reverse Subtract	Rd := Op2 - Rn	4.5
RSC	Reverse Subtract with Carry	Rd := Op2 - Rn - 1 + Carry	4.5

Table 4-1: The ARM Instruction set

```
MLA r1,r2,r3,r4  ; R1:=R2*R3+R4 o sea MAC(0,4)
```

Mnemonic	Instruction	Action	See Section:
SBC	Subtract with Carry	Rd := Rn - Op2 - 1 + Carry	4.5
STC	Store coprocessor register to memory	address := CRn	4.15
STM	Store Multiple	Stack manipulation (Push)	4.11
STR	Store register to memory	<address> := Rd	4.9, 4.10
SUB	Subtract	Rd := Rn - Op2	4.5
SWI	Software Interrupt	OS call	4.13
SWP	Swap register with memory	Rd := [Rn], [Rn] := Rm	4.12
TEQ	Test bitwise equality	CPSR flags := Rn EOR Op2	4.5
TST	Test bits	CPSR flags := Rn AND Op2	4.5

Table 4-1: The ARM Instruction set (Continued)

ARM
Codificación del repertorio de instrucciones

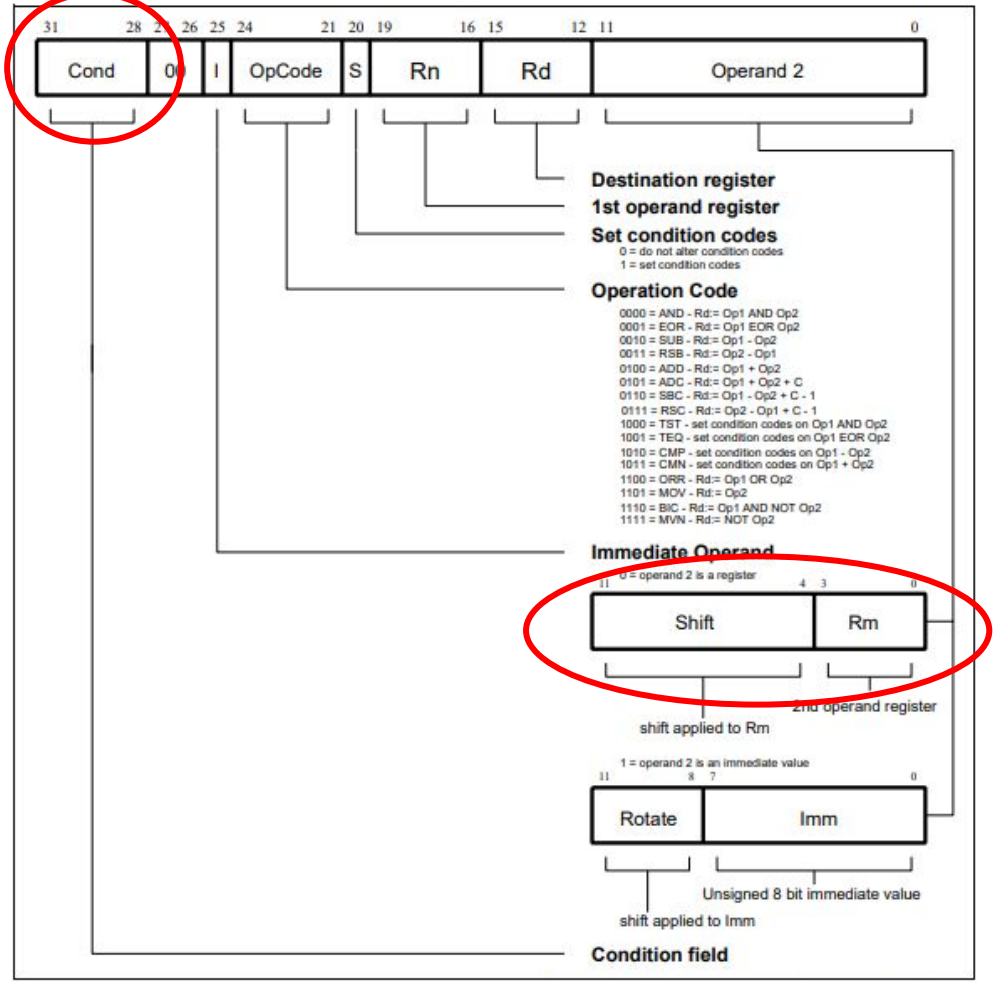


Figure 4-4: Data processing instructions

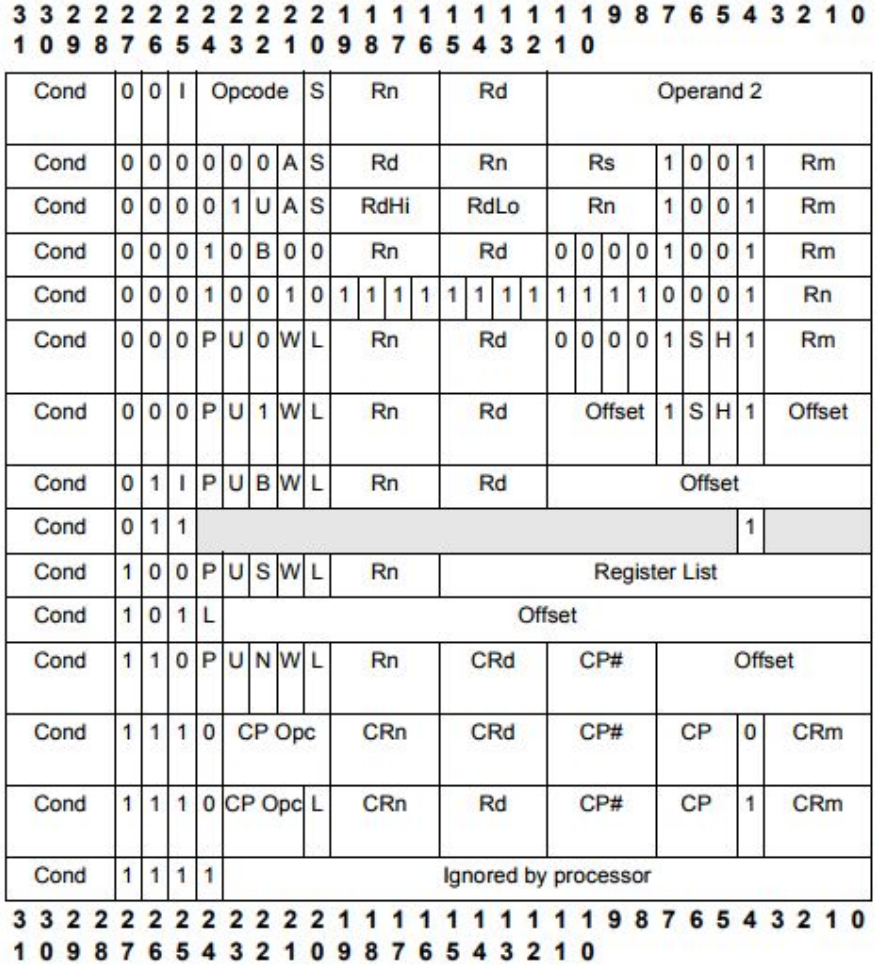
<operation>{cond}{S} Rd,Rn,Op2

<operation> A three-letter mnemonic
{cond} An optional two-letter condition code
S: set condition codes if S present
Rd: The destination register.
Rn: The first source register.
Op2: A flexible second operand.

```
ADDEQ R0,R1,R2

ADDS R3,R5,#15

ANDGTS R1,R2,R3,LSR#3
```

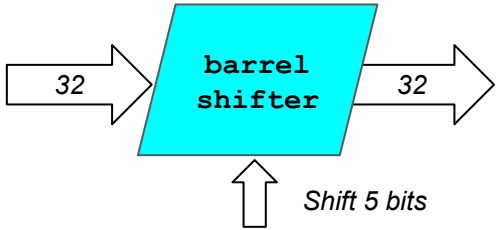
ver
[ARM Instruction Set]

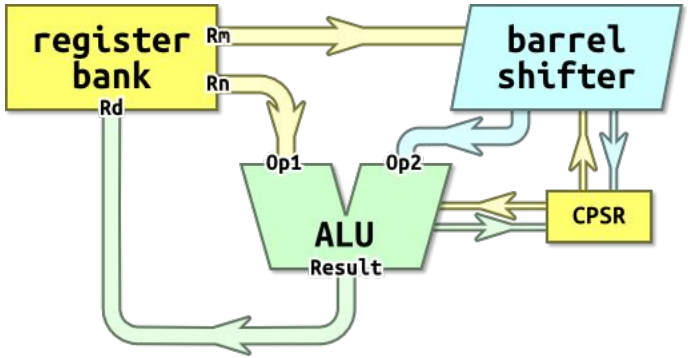
ARM
Codificación del repertorio de instrucciones



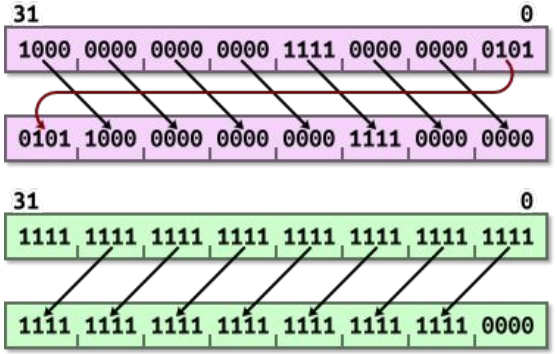
Barrel Shifter

Circuito combinatorio puro (arreglo de multiplexores)





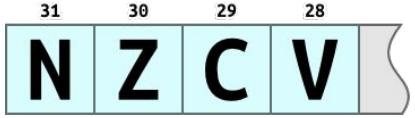

```
MOV r3, r4, LSL #1

SUB r3, r4, r5, ROR #4
```




```
; Multiplicar R4 por 17 (16+1)
ADD r4, r4, r4, LSL #4

; Swap the top and bottom halves of
R3
MOV r3, r3, ROR #16
```

Ejecución condicional

```
; TODO PUEDE SER CONDICIONAL,
; incluidos los saltos

; R1:=R2*R3+R4 condicional
MLAEQ r1,r2,r3,r4

; Multiplicar por 17 condicional
ADDGT r4,r4,r4,LSL #4

; DOS SALTOS: B y BL
back:
    ...             ;
    B back          ; jump to label 'back'

; Llamada a función
    ...
    BL  calc        ; call 'calc', store retu
    ...             ; returns to here

calc:               ; function body
    ADD r0, r1, r2  ; do some work here
    MOV pc, r14     ; PC = R14 to return
```



Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Table 4-2: Condition code summary

El ejemplo típico es el Máximo común divisor, según el algoritmo de Euclides.
En C:

```
   while (i != j)  // Ingresa en el ciclo cuando i<j o i>j, no cuando i==j
    {
       if (i > j)   // Cuando i>j realiza lo siguiente
            i -= j;
       else         // en otro caso, realiza lo siguiente
            j -= i;
     }
```

Una estrategia más eficiente para implementar en ensamblador::

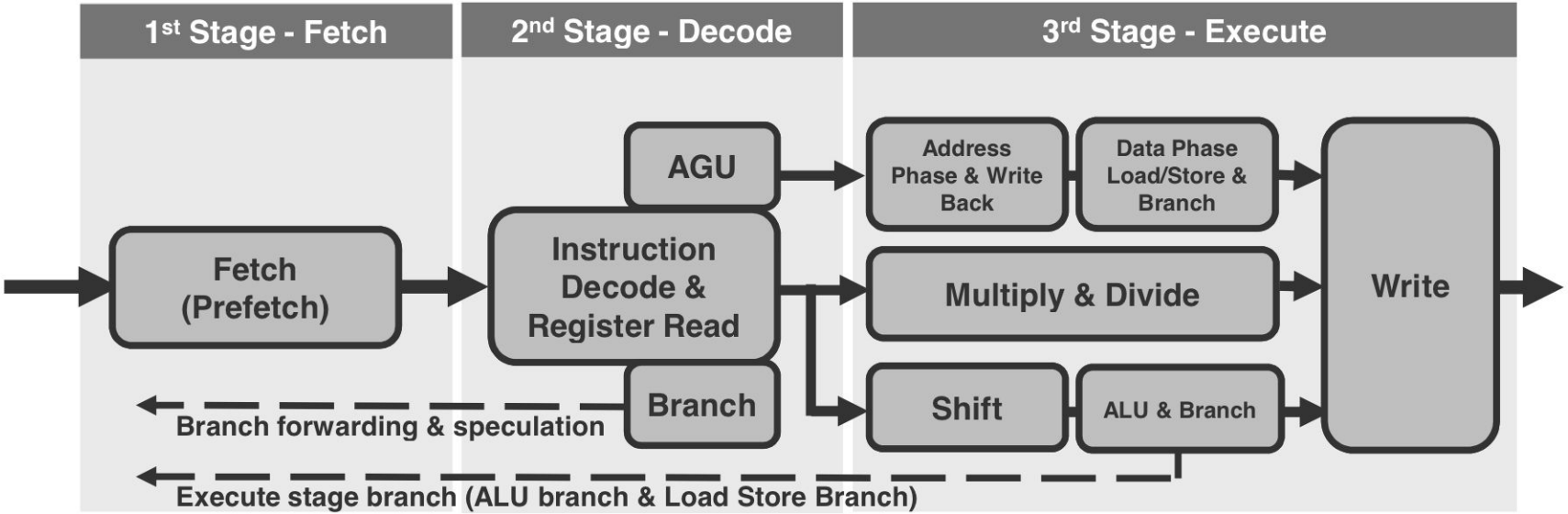
```
loop:
    // Compara i y j
    GT = i > j;
    LT = i < j;
    NE = i != j;

    // Operaciones mejoradas usando resultados de flags
    if(GT) i -= j;    // Sustrae *solo* si es mayor
    if(LT) j -= i;    // Sustrae *solo* si es menor
    if(NE) goto loop; // Ciclo *solo* si los valores comparados no son iguales
```

Esto en ARM se codifica en 4 instrucciones:

```
loop:   CMP    Ri, Rj       ; establece la condicion "NE" if (i != j),
                            ;               "GT" si (i > j),
                            ;            o  "LT" si (i < j)
        SUBGT  Ri, Ri, Rj   ; si "GT" (Mayor que), i = i-j;
        SUBLT  Rj, Rj, Ri   ; si "LT" (Menor que), j = j-i;
        BNE  loop           ; si "NE" (No igual), entonces realiza el ciclo
```

ORGANIZACIÓN Cortex-M

3-stage pipeline core with von Neumann architecture (Harvard bus architecture (M3/M4/M7)
Speculative fetch of branch targets

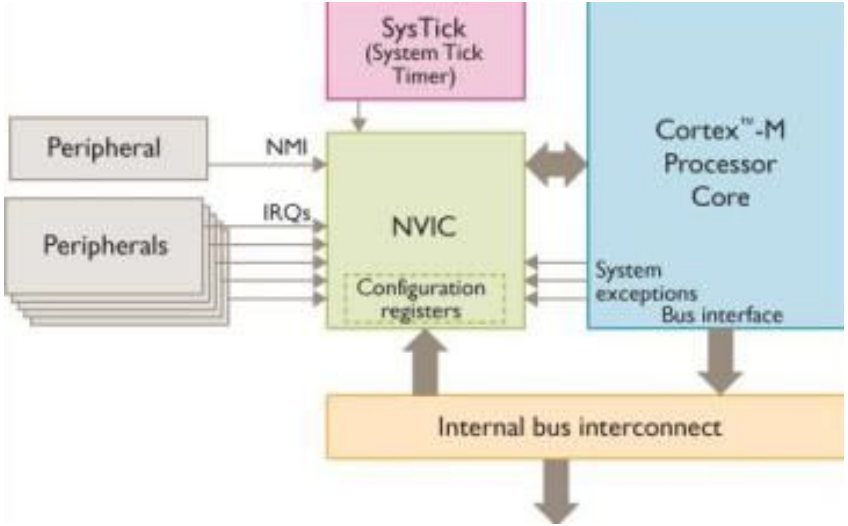


M7 incluye “six-stage dual issue pipeline” (allows execution of up to two instructions in the same clock cycle), branch prediction, data caches, etc. **ARQII**

INTERRUPT HANDLING

One Non-Maskable Interrupt (INTNMI)
16 prioritizable interrupts
Interrupts can be masked
Implementation option selects number of interrupts supported

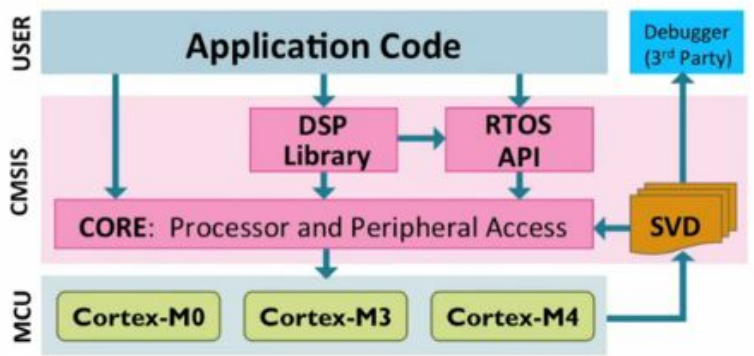
Nested Vectored Interrupt Controller (NVIC) is tightly coupled with processor core
Interrupt inputs are active HIGH
Tail-chaining supported



The NVIC supports a number of interrupt inputs from peripherals, a Non-Maskable Interrupt request, an interrupt request from a built-in timer called SysTick (see section3.3) and a number of system exceptions. The NVIC handles the priority management and masking of these interrupt and exceptions.

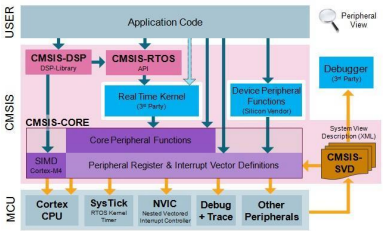
Programación en C

Para facilitar el desarrollo de software y permitir una mejor reusabilidad de código y portabilidad, ARM desarrolló **CMSIS-Core** (Cortex Microcontroller Software Interface Standard) que provee un Hardware Abstraction Layer (HAL), estandarizado para varios recursos del procesador, como el manejo de interrupciones. Suele estar integrado en las librerías de los diversos fabricantes.



También desarrollaron una biblioteca **CMSIS-DSP** que provee varias funciones optimizadas para Cortex-M4 y que también soporta otros procesadores Cortex-M processors.

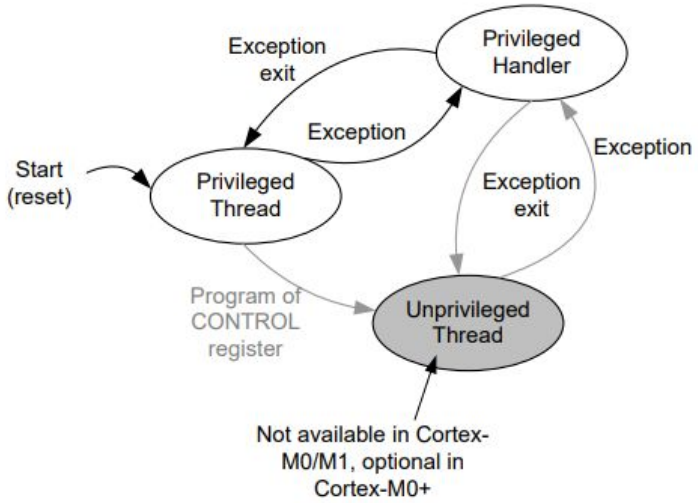
Tanto CMSIS-Core como CMSIS-DSP son gratuitas, pueden descargarse del sitio oficial de ARM (www.arm.com/cmsis), y son soportadas por la mayoría de las herramientas de desarrollo, para Cortex.



Soporte para Sistema Operativo (OS)

La arquitectura Cortex-M fue diseñada pensando en un OS. Algunas prestaciones relacionadas:

- Shadowed stack pointer.
- SVC (supervisor call) and PendSV (pendable service call) exceptions, interfaz para proveer acceso al HW a las aplicaciones.
- SysTick timer – a 24-bit down counter for generating periodic OS exception for time keeping and task management.
- Unprivileged execution level and Memory Protection Unit (MPU) ir Cortex-M0+/M3/M4 and M7.



CMSIS-RTOS provee la interfaz a sistemas operativos de terceros: https://os.mbed.com/handbook/CMSIS-RTOS
Tread, osDelay, Mutex, Semaphore, Signals, Message Queue, Memory Pool, Mail Queue, Timer, Interrupt Service Routines, Status and Error Codes, etc.
mbed OS es un sistema operativo desarrollado por ARM usando la interfaz CMSIS-RTOS: https://os.mbed.com/

Mbed Online Compiler (free)

FreeRTOS https://www.freertos.org/
Simba https://simba-os.readthedocs.io/en/latest/index.html
Trampoline https://github.com/TrampolineRTOS/trampoline
DuinOS https://github.com/DuinOS/DuinOS

Materia optativa 2° cuatrimestre
Sistemas Embebidos

Sistemas operativos de tiempo real (RTOS)

Mejor llamados “Embedded Programming Platforms”.

Beneficios de programar en el entorno de un RTOS:

- Threads scheduled by a priority based cooperative or preemptive scheduler.
- Channels for inter-thread communication (Queue, Event).
- Timers.
- Counting semaphores.
- Device drivers (SPI, UART, …)
- A simple shell.
- Logging.
- Internet protocols (TCP, UDP, HTTP, …).
- File systems (FAT16, SPIFFS).

Materia optativa 2° cuatrimestre
Sistemas Embebidos

ARM Cortex-M4 DSP Instructions Compared

		Cycle counts		
CLASS	INSTRUCTION	ARM9E-S	CORTEX-M3	Cortex-M4
Arithmetic	ALU operation (not PC)	1 - 2	1	1
	ALU operation to PC	3 - 4	3	3
	CLZ	1	1	1
	QADD, QDADD, QSUB, QDSUB	1 - 2	n/a	1
	QADD8, QADD16, QSUB8, QSUB16	n/a	n/a	1
	QDADD, QDSUB	n/a	n/a	1
	QASX, QSAX, SASX, SSAX	n/a	n/a	1
	SHASX, SHSAX, UHASX, UHSAX	n/a	n/a	1
	SADD8, SADD16, SSUB8, SSUB16	n/a	n/a	1
	SHADD8, SHADD16, SHSUB8, SHSUB16	n/a	n/a	1
	UQADD8, UQADD16, UQSUB8, UQSUB16	n/a	n/a	1
	UHADD8, UHADD16, UHSUB8, UHSUB16	n/a	n/a	1
	UADD8, UADD16, USUB8, USUB16	n/a	n/a	1
	UQASX, UQSAX, USAX, UASX	n/a	n/a	1
	UXTAB, UXTAB16, UXTAH	n/a	n/a	1
	USAD8, USADA8	n/a	n/a	1
Multiplication	MUL, MLA	2 - 3	1 - 2	1
	MULS, MLAS	4	1 - 2	1
	SMULL, UMULL, SMLAL, UMLAL	3 - 4	5 - 7	1
	SMULBB, SMULBT, SMULTB, SMULTT	1 - 2	n/a	1
	SMLABB, SMLBT, SMLATB, SMLATT	1 - 2	n/a	1
	SMULWB, SMULWT, SMLAWB, SMLAWT	1 - 2	n/a	1
	SMLALBB, SMLALBT, SMLALTB, SMLALTT	2 - 3	n/a	1
	SMLAD, SMLADX, SMLALD, SMLALDX	n/a	n/a	1
	SMLSD, SMLSDX	n/a	n/a	1
	SMLSLD, SMLSLD	n/a	n/a	1
	SMMLA, SMMLAR, SMMLS, SMMLSR	n/a	n/a	1
	SMMUL, SMMULR	n/a	n/a	1
	SMUAD, SMUADX, SMUSD, SMUSDX	n/a	n/a	1
	UMAAL	n/a	n/a	1
Division	SDIV, UDIV	n/a	2 - 12	2 - 12

Single cycle MAC

ARM Cortex-M4 Single Cycle MAC Instructions

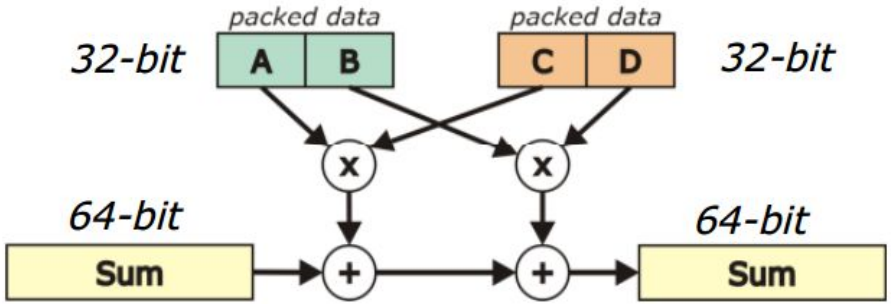
OPERATION	INSTRUCTIONS
16 x 16 = 32	SMULBB, SMULBT, SMULTB, SMULTT
16 x 16 + 32 = 32	SMLABB, SMLABT, SMLATB, SMLATT
16 x 16 + 64 = 64	SMLALBB, SMLALBT, SMLALTB, SMLALTT
16 x 32 = 32	SMULWB, SMULWT
(16 x 32) + 32 = 32	SMLAWB, SMLAWT
(16 x 16) ± (16 x 16) = 32	SMUAD, SMUADX, SMUSD, SMUSDX
(16 x 16) ± (16 x 16) + 32 = 32	SMLAD, SMLADX, SMLSD, SMLSDX
(16 x 16) ± (16 x 16) + 64 = 64	SMLALD, SMLALDX, SMLSLD, SMLSLDX
32 x 32 = 32	MUL
32 ± (32 x 32) = 32	MLA, MLS
32 x 32 = 64	SMULL, UMULL
(32 x 32) + 64 = 64	SMLAL, UMLAL
(32 x 32) + 32 + 32 = 64	UMAAL
32 ± (32 x 32) = 32 (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR
(32 x 32) = 32 (upper)	SMMUL, SMMULR

All the above operations are single cycle on the Cortex-M4 processor

ARM Cortex-M4 SIMD Instructions

SIMD extensions perform multiple operations in one cycle

$$Sum = Sum + (A \times C) + (B \times D)$$



SIMD techniques operate with packed data

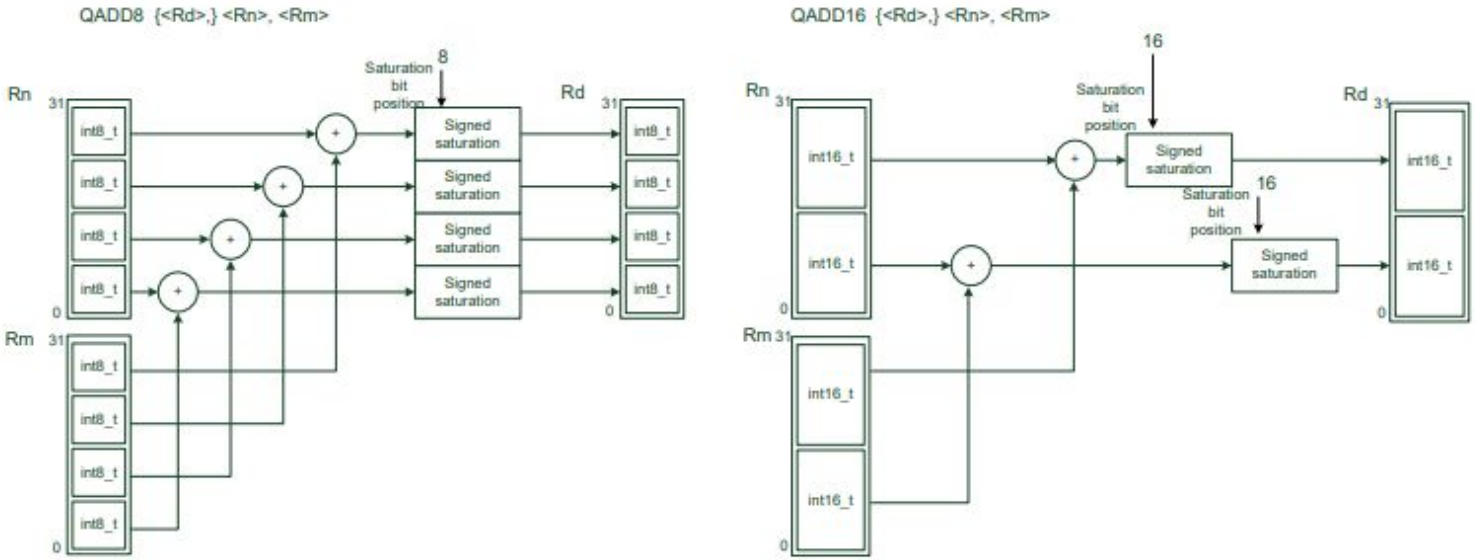


Figure 4: Example of SIMD instructions: QADD8 and QADD16

Performance

ARM Cortex	CoreMark/MHz	DMIPS/MHz
M7	5.04	2.14 / 2.55 / 3.23 DMIPS/MHz**
M4	3.40	1.25 / 1.52 / 1.91 DMIPS/MHz**
M3	3.32	1.25 / 1.50 / 1.89 DMIPS/MHz**
M0	2.33	0.87 / 1.02 / 1.27 DMIPS/MHz**

*** The first result abides by all of the “ground rules” laid out in the Dhrystone documentation, the second permits inlining of functions, not just the permitted C string libraries, while the third additionally permits simultaneous (”multi-file”) compilation. All are with the original (K&R) v2.1 of Dhrystone*

https://www.eembc.org/coremark/scores.php
(no está STM8)

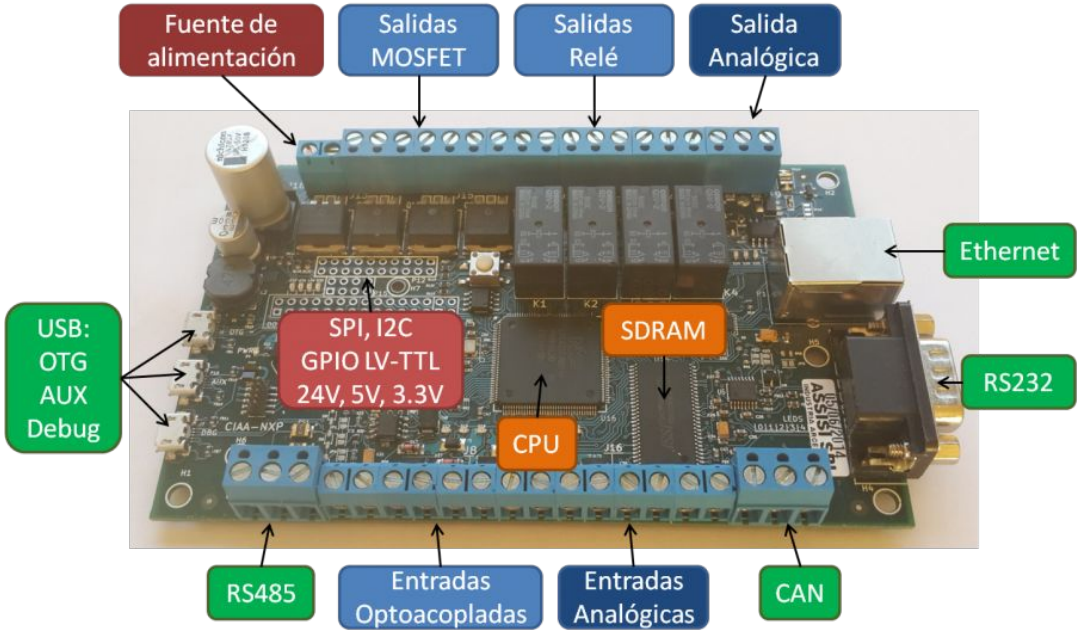
CoreMark®
An EEMBC Benchmark

	Compiler	Execution Memory	MHz	CoreMark	CoreMark / MHz↑	DMIPS/MHz
Atmel AT89C51RE2 in X2 mode (6 clocks/machine cycle)	Keil C51 v8.18	Internal RAM & flash (static)	22	2.36	**0.11**	
Microchip PIC18F46K22	Microchip MPLAB XC8 v1.32	Code in Flash, Data in RAM	64	7.23	**0.11**	
STM8	Origen dudoso		24	5	**0.21**	**0.29**
Atmel ATmega644	avr-gcc-4.3.2	Flash & SRAM 20 MHz (Static)	20	10.81	**0.54**	**0.33**
Texas Instruments MSP430F5438	Code Composer Studio 4.1.2	Internal flash and RAM (static)	18	11.10	**0.62**	**0.31**
Microchip PIC24FJ64GA004	gcc 4.0.3	Code Flash. Data SRAM (Static)	32	23.87	**0.75**	**0.50**
STMicroelectronics STM32F103	GCC 4.4.1	Internal Flash	72	108.26	**3.32**	**1.25**

CIAA

Computadora Industrial Abierta Argentina

http://www.proyecto-ciaa.com.ar/

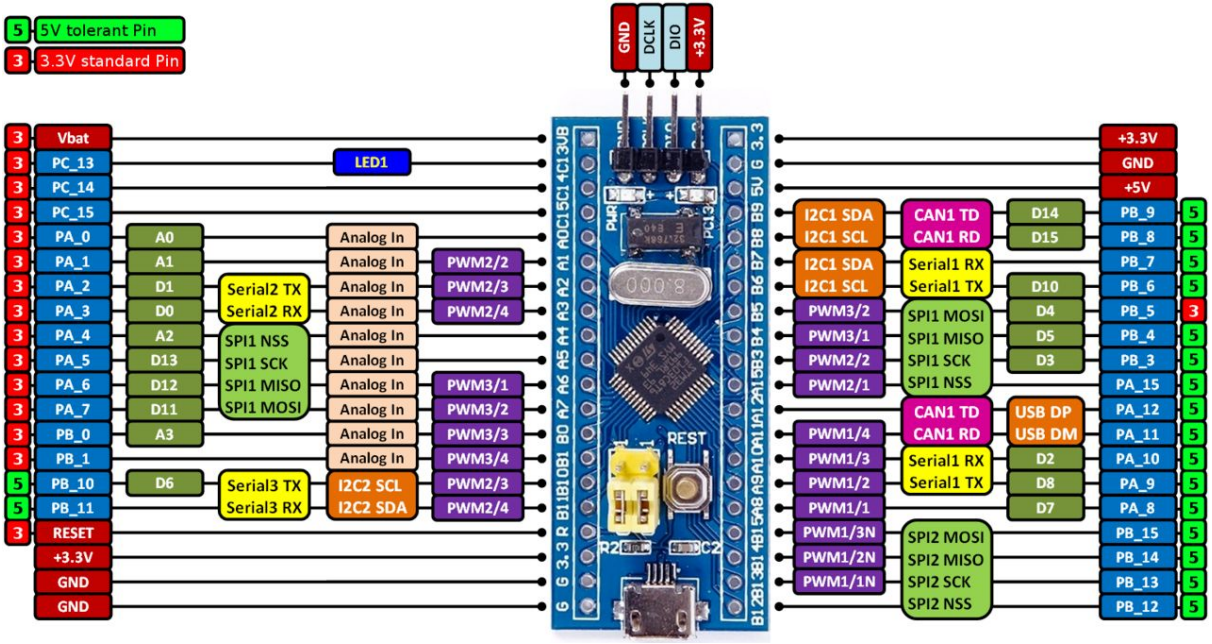


NXP LPC4337
Cortex M4F + Cortex M0
@ 204 MHz
1 MB Flash
136 KB RAM

Blue Pill

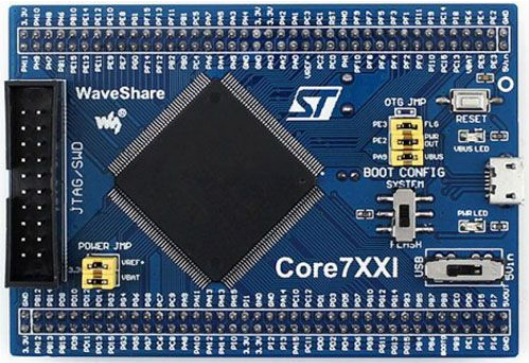
ST STM32F103C8T6 - Cortex M3 @ 72 MHz

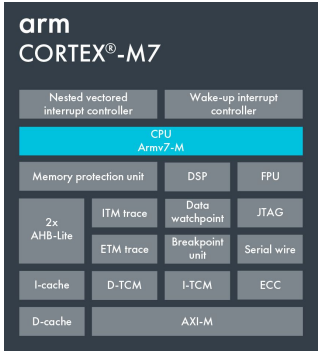
$500 en Mercadolibre + $700 Programador ST-Link V2 (el mismo que para STM8)



SPECS/BOARD	STM32F103C	ARDUINO UNO
Number of Cores	1	1
Architecture	32 Bit	8 Bit
CPU Frequency	72 MHz	16 MHz
WiFi	NO	NO
BLUETOOTH	NO	NO
RAM	20 KB	2 KB
FLASH	64 KB	32 KB
GPIO PINS	37	14
Busses	SPI, I2C, UART, CAN	SPI, I2C, UART
ADC Pins	10	6
DAC Pins	0	0

Cortex M7 (2017)





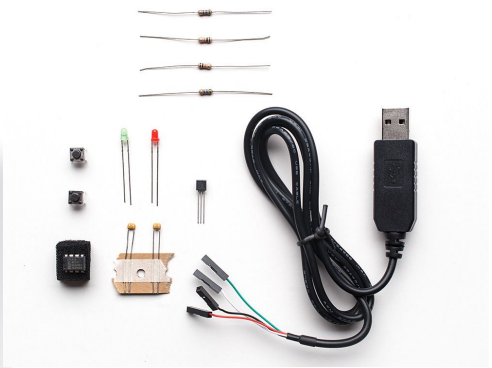
eBay $37

ISA Support	ARMv7-M
DSP Extensions	Single cycle 16/32-bit MAC Single cycle dual 16-bit MAC 8/16-bit SIMD arithmetic Hardware Divide (2-12 Cycles)
Floating Point Unit	Single and double precision floating point unit IEEE 754 compliant
Pipeline	6-stage 2-way superscalar + branch prediction
Performance Efficiency	5.04 CoreMark/MHz*
Performance Efficiency	2.14 / 2.55 / 3.23 DMIPS/MHz**

400 MHz (40 nm)
Can scale up to 800MHz at 28nm.

Mismo repertorio de instrucciones que Cortex-M4F

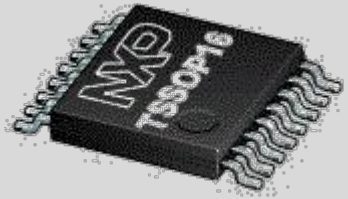
Cortex DIP-8

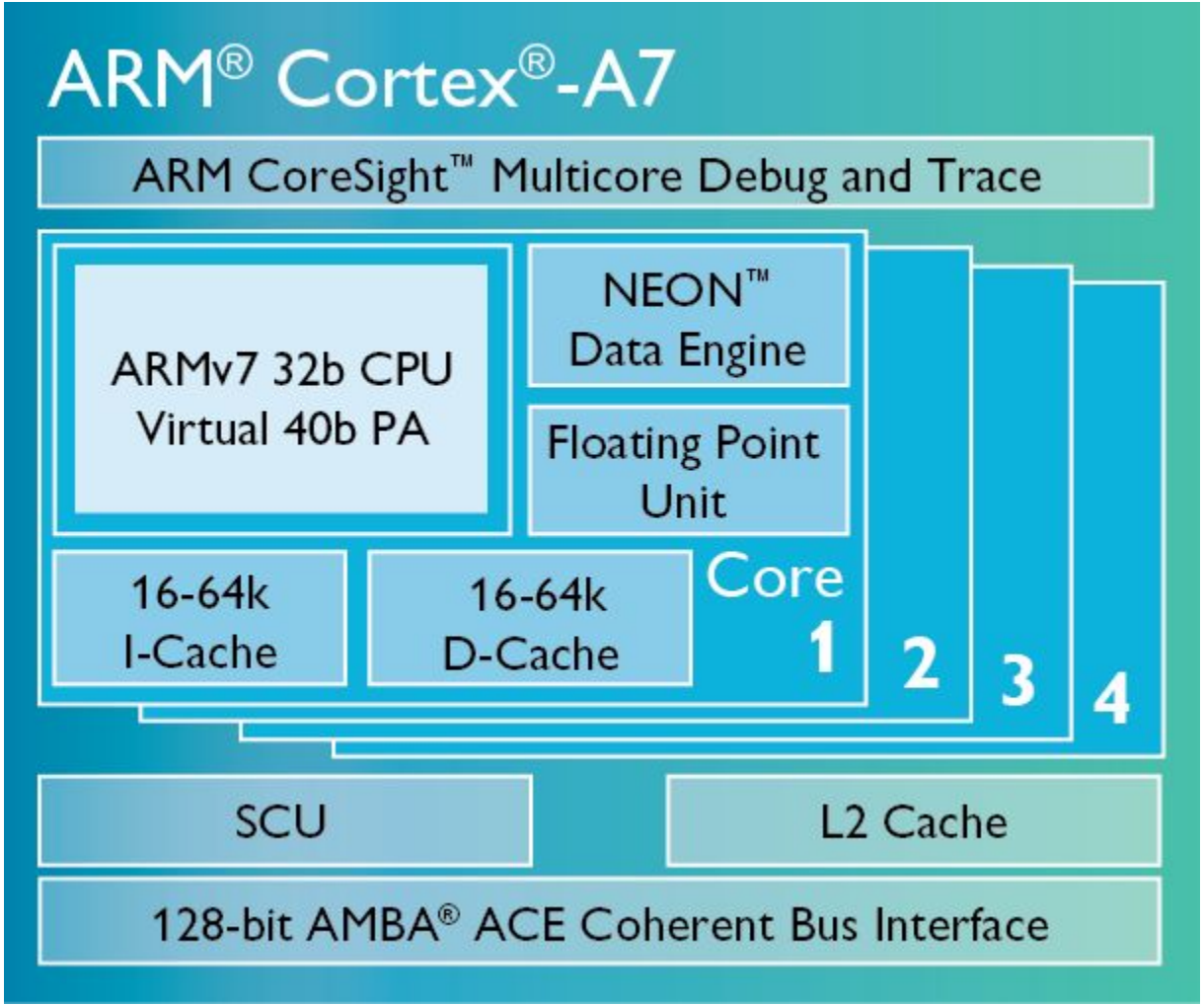


NXP LPC800: Cortex-M0
Up to 16KB flash and 2KB SRAM
https://learn.adafruit.com/getting-started-with-the-lpc810

Toolchain Windows:
LPCXpresso IDE para escribir el programa y cross-compilar
Flashmagic para grabar - Adaptador UART-USB y unos pocos componentes

UPDATE:
LPC801 (DIP-8) dejó de fabricarse?
El más chico es LPC802 (TSSOP16)





La arquitectura **ARMv7-A**

- MMU (memory management unit)
- Out-of-order execution (superescalar)
- Deep pipeline
- FPU
- Cache
- Multicore
- SIMD (Neon)

Following are the architectural features which makes Cortex-M3 architecture a low power device:

1. It has sleep mode and deep sleep mode supports, which can work with various system-design methodologies to reduce power consumption during idle period.
2. Its low gate count and design techniques reduce circuit activities in the processor to allow active power to be reduced.
3. Since Cortex-M3 has high code density, it has lowered the program size requirement. At the same time, it allows processing tasks to be completed in a short time, so that the processor can return to sleep modes as soon as possible to cut down energy use.
4. Starting from Cortex-M3 revision 2, a new feature called Wakeup Interrupt Controller (WIC) is available. This feature allows the whole processor core to be powered down, while processor states are retained and the processor can be returned to active state almost immediately when an interrupt takes place.

The following below explains the various modes available in Cortex M3 for power management.

Sleep Modes:

1. During sleep mode, the system clock can be stopped, but the free-running clock input could still be running to allow the processor to be woken by an interrupt.
2. The two sleep modes are as follows:
   a. Sleep: Indicated by the SLEEPING signal from the Cortex-M3 processor
   b. Deep sleep: Indicated by the SLEEPDEEP signal from the Cortex-M3 processor
3. The sleep modes are invoked by Wait-For-Interrupt (WFI) or Wait-For-Event (WFE) instructions.
4. The events for invoking the sleep modes can be interrupts, a previously triggered interrupt, or an external event signal pulse via the Receive Event (RXEV) signal.
5. To decide which sleep mode is to be invoked in case of an event can be set by setting the SLEEPDEEP bit field of the Nested Vectored Interrupt Controller (NVIC) control register
6. The sleep mode operation of the processor depends on chip design. In some cases the clock signals can be stopped to reduce power consumption. The chip can also be designed to shut down part of the chip to further reduce power, or it is also possible that a design can shut down the chip completely. In a case where the chip is shut down completely, the only way to wake the system from sleep is via a system reset.

Sleep-On-Exit Feature:

1. Processor can be programmed to go back to sleep automatically after the interrupt routine exit. In this way, we can make the core sleep all the time unless an interrupt needs to be served
2. To use this feature, we need to set the SLEEPONEXIT bit in the System Control register.
3. If the Sleep-On-Exit feature is enabled, the processor can enter sleep at any exception return to thread level, even if no WFE/WFI instruction is executed.

Wake-up Interrupt Controller:

1. A new unit called the Wakeup Interrupt Controller (WIC) is available as an optional component. This controller is coupled to the existing NVIC and is used to generate a wakeup request when an interrupt arrives.
2. By using the technology called State Retention Power Gating (SRPG) and WIC together, most portions of the Cortex-M3 processor can be powered down during deep sleep, leaving a small amount of logic for state retention.
3. During this power down state, the WIC remains operational and generates a wakeup request to power up and restore the system state when an interrupt arrives. Maximum interrupt

https://controllerstech.com/low-power-modes-in-stm32/