Microcontroladores de 16 bits

MICROCONTROLADORES DE 8 bits

1. Motorola/Philips/NXP **CPU08**
2. Intel/Atmel **8051**
3. Atmel/Microchip **AVR**
4. STMicroelectronics **STM8**
5. Microchip **PIC16**

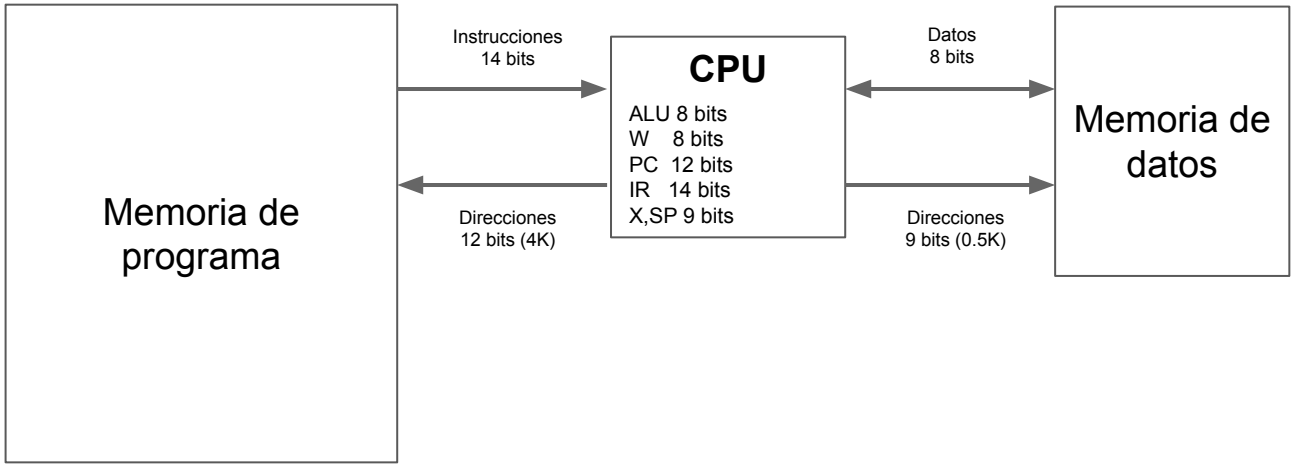
MICROCONTROLADORES DE 16 bits

1. Texas Instruments **MSP430**
2. Microchip **PIC24** y **dsPIC**

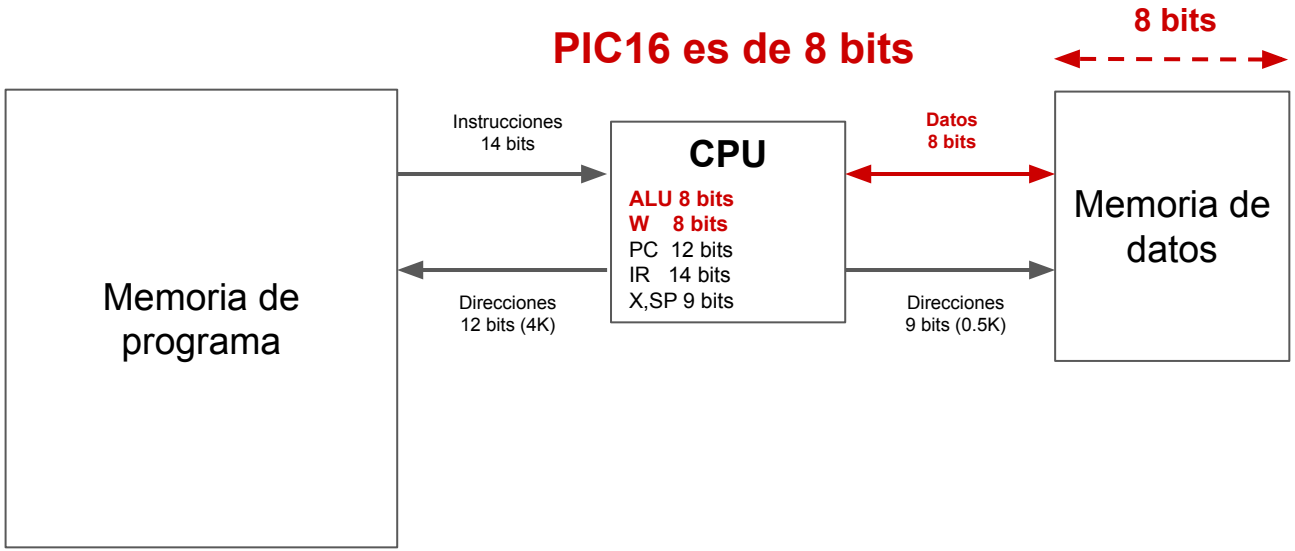
MICROCONTROLADORES DE 32 bits

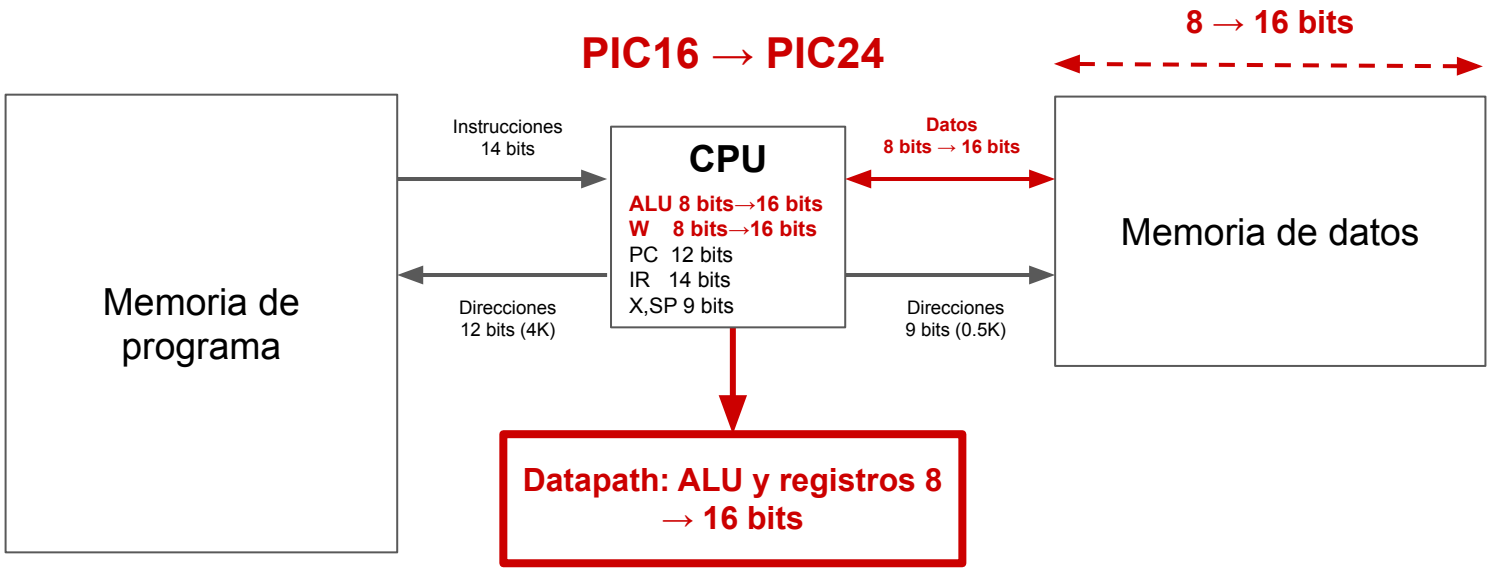
1. ARM **Cortex-M** -->> NXP, ST, Texas, Microchip, etc.
2. Otros: TI **C2000** (DSP FPU), **PIC32**, etc.

PIC16


Memoria de programa
Instrucciones
14 bits
Direcciones
12 bits (4K)
CPU
ALU 8 bits
W 8 bits
PC 12 bits
IR 14 bits
X,SP 9 bits
Datos
8 bits
Direcciones
9 bits (0.5K)
Memoria de datos

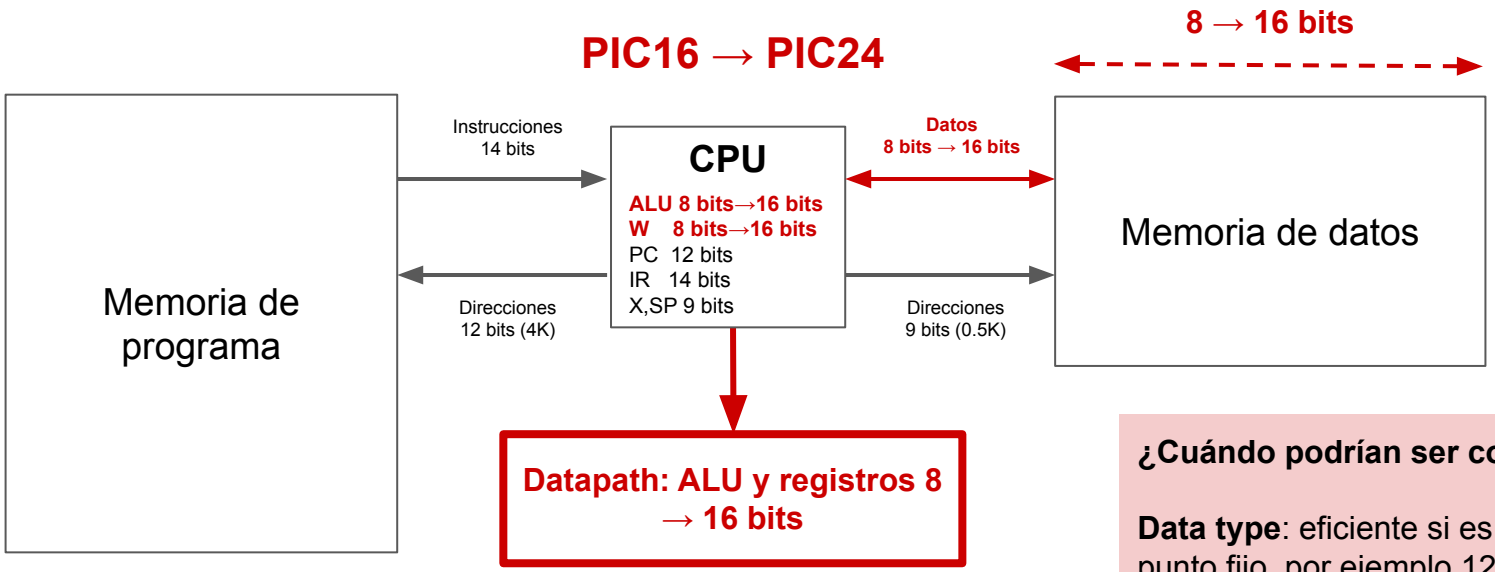
PIC16 es de 8 bits





No son necesariamente más veloces que los de 8-bit. Más hw, más potencia. Y no direccionan necesariamente más memoria (bank switching).

No son siempre más baratos que los de 32-bits



No son necesariamente más veloces que los de 8-bit. Más hw, más potencia. Y no direccionan necesariamente más memoria (bank switching).

No son siempre más baratos que los de 32-bits

¿Cuándo podrían ser convenientes?

Data type: eficiente si es mayor que 8-bits, punto fijo, por ejemplo 12-bit ADC/DAC que no serán mejorados.
"natural word length" → "more efficient code"

Los **DSP** se empiezan a parecer a los MCU y viceversa → MAC, etc.
Core en assembler y el resto en C.

Mayor capacidad para **OS.**

ESTO MÁS ALLÁ DE SI EXISTE O NO EL MODELO CORRECTO EN EL MERCADO

Discusión hipotética: ¿qué se necesitaría para “convertir” AVR a 16 bits?

Ensanchar los 32 registros. **Duplica el HW**.

ALU de 16 bits, cuidado de no enlentecer la etapa de ejecución, predictor de carry.
Más que duplica el HW.

Podría dejar la multiplicación como está. Pero lo ideal sería implementar 16x16=32 en r0-r1.

Memoria de datos, la puedo dejar direccionable por byte, pero debería poder escribir dos bytes a la vez. Idem la pila. **Duplica el ancho de banda del bus de datos.**

No cambia el sistema de entrada/salida, pueden usarse registros de 16 bits (mapeados en memoria de datos).

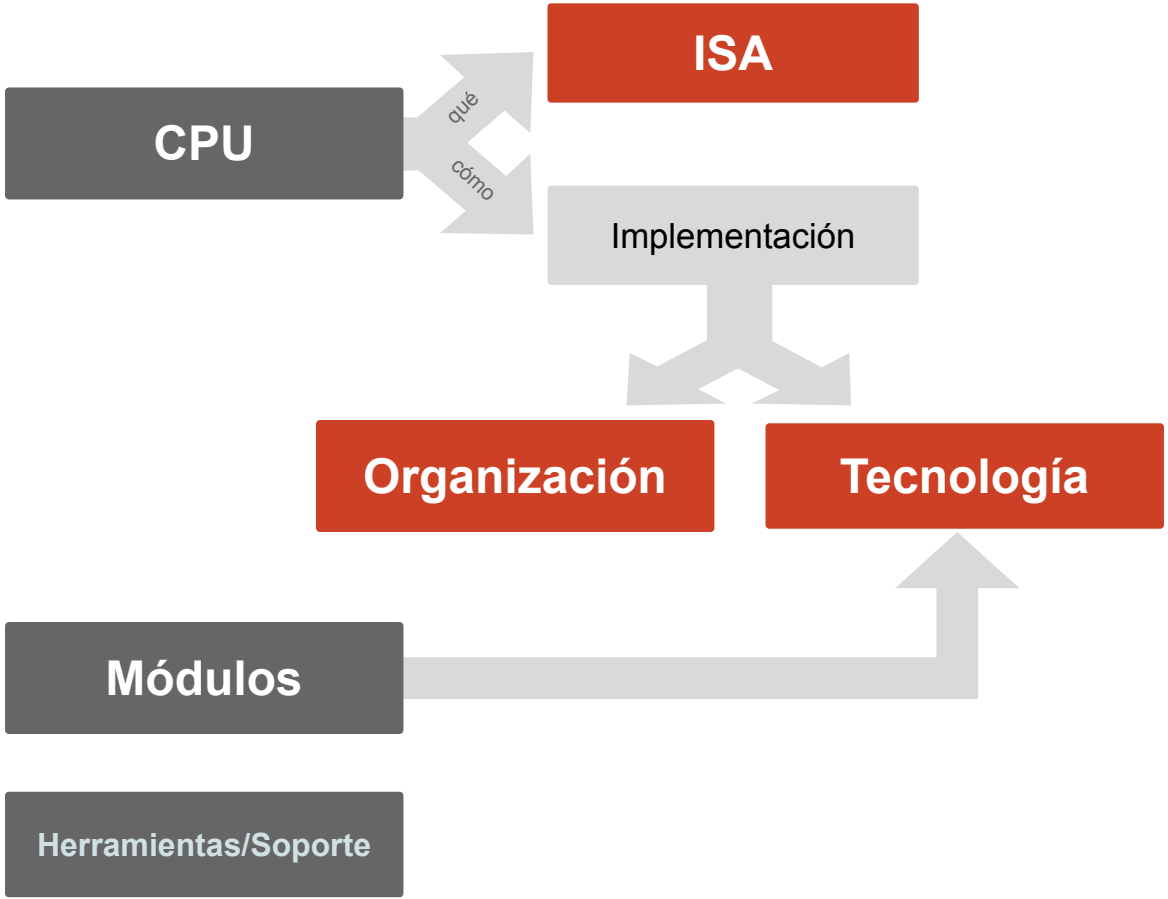
No afecta la memoria de programa (PC, IR, saltos) ni los punteros de datos (X, Y y Z ya no necesitarían ocupar dos registros).

La instrucciones no cambiarían significativamente. Habría que ampliar el modo de direccionamiento inmediato para que sea útil en 16 bits.

Lectura recomendada:

Embedded.com
16-Bit: The Good, The Bad, Your Options
https://www.embedded.com/16-bit-the-good-the-bad-your-options-special-report/

Digikey.com
Take Advantage of a 16-bit Microcontroller's Performance and Low Power
https://www.digikey.com/en/articles/take-advantage-of-16-bit-mcu-performance-and-low-power



HAY QUE SER MUY CUIDADOSO AL COMPARARLOS CON LOS DE 8 BITS

Medibles

Parámetros eléctricos (rango) —> V, I, T, f

Tamaño del código para un determinado problema y RAM necesaria —> Memoria de programa y de datos.

Potencia a fmax y en stand-by —> Watt y Watt/MHz

Performance a fmax —> Dhrystone/s y Coremark

Performance relativa —> DMIPS

Organización —> DMIPS/MHz y Coremark/MHz

Eficiencia —> DMIPS/Watt

Costo unitario —> $ MercadoLibre x1, u$s DigiKey x3000

Costo del sistema de desarrollo —> $ Kit, instrumentos y herramientas

No-tan-medibles

Tiempo de desarrollo y experiencia necesaria

Futuras migraciones y mejoras, soporte

Texas Instruments MSP430

Texas Instruments MSP430 (1994)

Arquitectura de 16-bits, **von Neumann**
Instrucciones de largo fijo 16-bits
Repertorio ortogonal, 27 instrucciones: 7 de 1 operando, 12 de 2 y 8 saltos. (+24 emuladas)
7 modos de direccionamiento
RISC (0,2) 1 ciclo - (1,2) 2-4 ciclos - (2,2) 6 ciclos

16 registros de 16 bits
- R0 → PC
- R1 → SP
- R2 → SR/CG1
- R3 → CG2
- R4-R15 → GPR

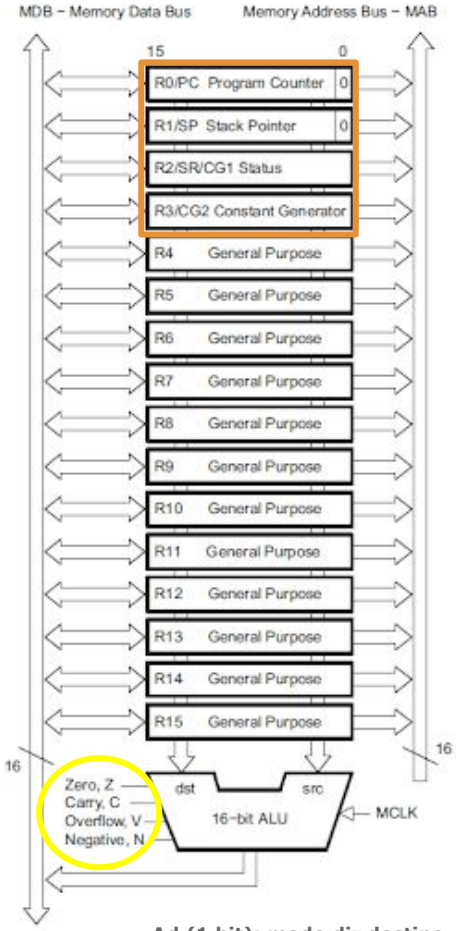
Constant generator, CG1 y CG2
I/O mapeado en memoria

Ultra-low-power modes

Los 7 modos de direccionamiento (ver, 2 para el destino y 4 para la fuente) están disponibles para TODAS las instrucciones y pueden acceder al mapa de memoria completo, **repertorio muy ortogonal**

Todas las instrucciones disponibles en formato .B (8-bit byte) and .W (16-bit word), dependiendo del bit B/W (1: 8-bit, 0: 16-bit), **ahorro de energía**

Las instrucciones **registro-registro** ocupan 1 palabra de programa (16-bits) y son ejecutadas en **1 ciclo**.
Las instrucciones con **operandos en memoria** pueden ocupar 2 o 3 palabras y pueden tomar hasta **6 ciclos**.
Los **saltos** ocupan 1 palabra de programa y son ejecutadas en **2 ciclos**, se tome o no el salto.
MSP430 User guide

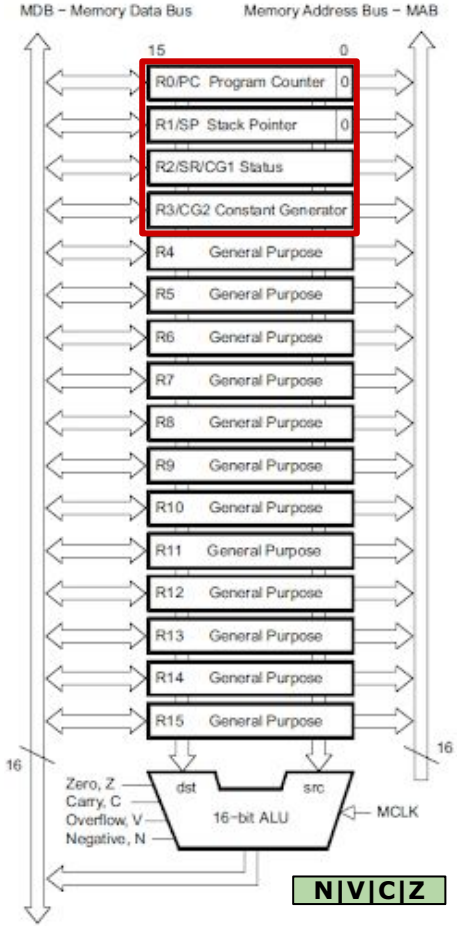


Ad (1 bit): modo dir destino
As (2 bits): modo dir origen
B/W: Byte o word

core instruction mnemonics	core instruction binary															
Single-operand arithmetic	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	1	0	0	opcode			B/W	As		source			
RRC Rotate right through carry	0	0	0	1	0	0	0	0	0	B/W	As		source			
SWPB Swap bytes	0	0	0	1	0	0	0	0	1	0	As		source			
RRA Rotate right arithmetic	0	0	0	1	0	0	0	1	0	B/W	As		source			
SXT Sign extend byte to word	0	0	0	1	0	0	0	1	1	0	As		source			
PUSH Push value onto stack	0	0	0	1	0	0	1	0	0	B/W	As		source			
CALL Subroutine call; push PC and move source to PC	0	0	0	1	0	0	1	0	1	0	As		source			
RETI Return from interrupt; pop SR then pop PC	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0

Conditional jump; PC = PC + 2×offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	condition			10-bit signed offset									
JNE/JNZ Jump if not equal/zero	0	0	1	0	0	0	10-bit signed offset									
JEQ/JZ Jump if equal/zero	0	0	1	0	0	1	10-bit signed offset									
JNC/JLO Jump if no carry/lower	0	0	1	0	1	0	10-bit signed offset									
JC/JHS Jump if carry/higher or same	0	0	1	0	1	1	10-bit signed offset									
JN Jump if negative	0	0	1	1	0	0	10-bit signed offset									
JGE Jump if greater or equal (N == V)	0	0	1	1	0	1	10-bit signed offset									
JL Jump if less (N != V)	0	0	1	1	1	0	10-bit signed offset									
JMP Jump (unconditionally)	0	0	1	1	1	1	10-bit signed offset									

Two-operand arithmetic	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	opcode				source				Ad	B/W	As		destination			
MOV Move source to destination	0	1	0	0	source				Ad	B/W	As		destination			
ADD Add source to destination	0	1	0	1	source				Ad	B/W	As		destination			
ADDC Add w/carry: dst += (src+C)	0	1	1	0	source				Ad	B/W	As		destination			
SUBC Subtract w/ carry: dst -= (src+C)	0	1	1	1	source				Ad	B/W	As		destination			
SUB Subtract; dst -= src	1	0	0	0	source				Ad	B/W	As		destination			
CMP Compare; (dst-src); discard result	1	0	0	1	source				Ad	B/W	As		destination			
DADD Decimal (BCD) addition: dst += src	1	0	1	0	source				Ad	B/W	As		destination			
BIT Test bits; (dst & src); discard result	1	0	1	1	source				Ad	B/W	As		destination			
BIC Bit clear; dest &= ~src	1	1	0	0	source				Ad	B/W	As		destination			
BIS "Bit set" - logical OR; dst |= src	1	1	0	1	source				Ad	B/W	As		destination			
XOR Bitwise XOR; dst ^= src	1	1	1	0	source				Ad	B/W	As		destination			
AND Bitwise AND; dst &= src	1	1	1	1	source				Ad	B/W	As		destination			



Ad (1 bit): modo dir destino
As (2 bits): modo dir origen
B/W: Byte o word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	**0**	**0**	**1**	**0**	**0**	**opcode**			**B/W**	**As**		**source**				**7 Single-operand arithmetic**
0	0	0	1	0	0	0	0	0	B/W	As		source				**RRC** Rotate right through carry; Flags:N,Z,C;V=0
0	0	0	1	0	0	0	0	1	0	As		source				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		source				**RRA** Rotate right arithmetic; Flags:N,Z,C;V=0
0	0	0	1	0	0	0	1	1	0	As		source				**SXT** Sign extend byte to word; Flags=NZ, C=!Z, V=0
0	0	0	1	0	0	1	0	0	B/W	As		source				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		source				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC
0	**0**	**1**	**condition**			**10-bit signed offset**										**8 Conditional jump; PC = PC + 2×offset**
0	0	1	0	0	0	10-bit signed offset										**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset										**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset										**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset										**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset										**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset										**JGE** Jump if greater or equal (N == V)
0	0	1	1	1	0	10-bit signed offset										**JL** Jump if less (N != V)
0	0	1	1	1	1	10-bit signed offset										**JMP** Jump (unconditionally)
opcode				**source**		**Ad**	**B/W**	**As**	**destination**							**12 Two-operand arithmetic**
0	1	0	0	source		Ad	B/W	As	destination							**MOV** Move source to destination; Flags:unchanged
0	1	0	1	source		Ad	B/W	As	destination							**ADD** Add source to destination; Flags:all
0	1	1	0	source		Ad	B/W	As	destination							**ADDC** Add w/carry: dst += (src+C); Flags:all
0	1	1	1	source		Ad	B/W	As	destination							**SUBC** Subtract w/ carry: dst -= (src+C); Flags:all
1	0	0	0	source		Ad	B/W	As	destination							**SUB** Subtract; dst -= src; Flags:all
1	0	0	1	source		Ad	B/W	As	destination							**CMP** Compare; (dst-src); discard result; Flags:all
1	0	1	0	source		Ad	B/W	As	destination							**DADD** Decimal (BCD) addition+C: dst += src; Flags:ZCN
1	0	1	1	source		Ad	B/W	As	destination							**BIT** Test bits; (dst & src); discardresult;Flags:N,Z;C=!Z;V=0
1	1	0	0	source		Ad	B/W	As	destination							**BIC** Bit clear; dest &= ~src; Flags:unchanged
1	1	0	1	source		Ad	B/W	As	destination							**BIS** "Bit set" - logical OR; dst |= src; Flags:unchanged
1	1	1	0	source		Ad	B/W	As	destination							**XOR** Bitwise XOR; dst ^= src; Flags:N,Z, C=!Z, Z=1 if
1	1	1	1	source		Ad	B/W	As	destination							**AND** Bitwise AND; dst &= src; Flags:N,Z; C=!Z; V=0

MSP430 - Modos de direccionamiento

(0,2) 1 ciclo
(1,2) 2-4 ciclos
(2,2) 6 ciclos

```
ADD source,dest    ; dest = source + dest
```

4 Modos de direccionamiento para **source**

- **Rs** Register (Direct)
- **@Rs** Register Indirect
- **@Rs+** Indirect Auto Increment
- **x(Rs)** Indexed Register length 2

2 Modos de direccionamiento para **destination**

- **Rd** Register (Direct)
- **x(Rd)** Indexed Register length 2

Cualquier registro puede ser puntero, origen o destino. Incluídos SP y PC (porque son R0 y R1, todos de 16 bits). Modos "adicionales".

MUY ORTOGONAL

Ejemplos:

```
ADD R5,R6          ; single-cycle!
ADD @R5,R6         ; no usa palabra adicional, 2 ciclos
ADD @R5+,R6        ; no usa palabra adicional, 2 ciclos
ADD 3(R5),R6       ; el offset está en una palabra adicional, 3 ciclos

ADD #100,R6        ; el inmediato está en una palabra adicional, 2 ciclos

MOV 55(R5),73(R6)  ; extremo (2,2): 3 palabras de instrucción, 6 ciclos !
```

MSP430 - Modos de direccionamiento

5.2 Addressing Modes

All seven addressing modes for the source operand and all four addressing modes for the destination operand can address the complete address space. The bit numbers show the contents of the As resp. Ad mode bits.

As	Ad	Addressing Mode	Syntax	Description
00	0	Register Mode	Rn	Register contents are operand
01	1	Indexed Mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word
01	1	Symbolic Mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed Mode X(PC) is used
01	1	Absolute Mode	&ADDR	The word following the instruction contains the absolute address.
10	-	Indirect Register Mode	@Rn	Rn is used as a pointer to the operand
11	-	Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards
11	-	Immediate Mode	#N	The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ is used

Length 2 words

Length 2 words

MSP430 - Modos de direccionamiento

As/Ad	Reg	Syntax	Description
0	*n*	**R*n***	Register direct. The operand is the contents of R*n*.
1	*n*	***x*(R*n*)**	Indexed. The operand is in memory at address R*n*+*x*.
2	*n*	**@R*n***	Register indirect. The operand is in memory at the address held in R*n*.
3	*n*	**@R*n*+**	Indirect autoincrement. As above, then the register is incremented by 1 or 2.
Addressing modes using R0 (PC)			
3	0	**#*x***	Immediate. @PC+ The operand is the next word in the instruction stream.
Addressing modes using R2 (SP) and R3 , special-case decoding			
1	2	**&*x***	Absolute. The operand is in memory at address *x*.
Addressing modes using the constant generator			
2	2	**#4**	Constant. The operand is the constant 4.
3	2	**#8**	Constant. The operand is the constant 8.
0	3	**#0**	Constant. The operand is the constant 0.
1	3	**#1**	Constant. The operand is the constant 1. There is no index word.
2	3	**#2**	Constant. The operand is the constant 2.
3	3	**#-1**	Constant. The operand is the constant -1.

MSP430 - Instrucciones emuladas

La versatilidad de los modos de direccionamiento hace que algunas instrucciones resulten repetidas. No hace falta implementarlas, el ensamblador se encarga.

Ejemplos:

```
RET == MOV @SP+,PC
    == MOV @R2+,R1

POP R5 == MOV @SP+,R5
```

emulated	core instructions	instructions	
ADC.x dst	ADDC.x #0,dst	add carry to destination	
CLRC	BIC #1,SR	clear carry bit	0xc312
CLRN	BIC #4,SR	clear negative bit	0xc222
CLRZ	BIC #2,SR	clear zero bit	0xc322
DADC.x dst	DADD.x #0,dst	decimal add carry to destination	
DEC.x dst	SUB.x #1,dst	decrement	
DECD.x dst	SUB.x #2,dst	double decrement	
DINT	BIC #8,SR	disable interrupts	0xc232
EINT	BIS #8,SR	enable interrupts	0xd232
INC.x dst	ADD.x #1,dst	increment	
INCD.x dst	ADD.x #2,dst	double increment	
INV.x dst	XOR.x #-1,dst	invert	
NOP	MOV #0,R3	no operation	0x4303
POP dst	MOV @SP+,dst	pop from stack	
RET	MOV @SP+,PC	return from subroutine	0x4130
RLA.x dst	ADD.x dst,dst	rotate left arithmetic (shift left 1 bit)	
RLC.x dst	ADDC.x dst,dst	rotate left through carry	
SBC.x dst	SUBC.x #0,dst	subtract borrow (1-carry) from destination	
SETC	BIS #1,SR	set carry bit	0xd312
SETN	BIS #4,SR	set negative bit	0xd222
SETZ	BIS #2,SR	set zero bit	0xd322
TST.x dst	CMP.x #0,dst	test destination	

MSP430 - Constant generator

`MOV R1,R2 == ADD R0,R1,R2`

R3 es un “falso registro” que al leerlo devuelve siempre cero (clásico en los diseños RISC que tienen R0). Siempre R3 ignora las escrituras (`NOP == MOV #0,R3`). Pero en este caso también devuelve otras constantes cuando se usa otro modo de direccionamiento. **Esto es MUY novedoso.**
Lo mismo se implementó para **R2**, que es el Status Register. Cuando se lee en modo directo (register mode) devuelve el contenido de SR. Con otros modos devuelve constantes.
Para utilizar estas constantes, que supuestamente son las más utilizadas, no hace falta utilizar el modo de direccionamiento inmediato (costoso, ocupa dos palabras de instrucción y dos ciclos de reloj).
CONTRIBUYE A CÓDIGO MÁS COMPACTO.

Register	As	Constant	Remarks
R2	00	-	Register mode
R2	01	(0)	Absolute mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	-1, word processing

```
DECD == SUB @R3,R5    ; double decrement
```

MSP430 - Mapa de memoria

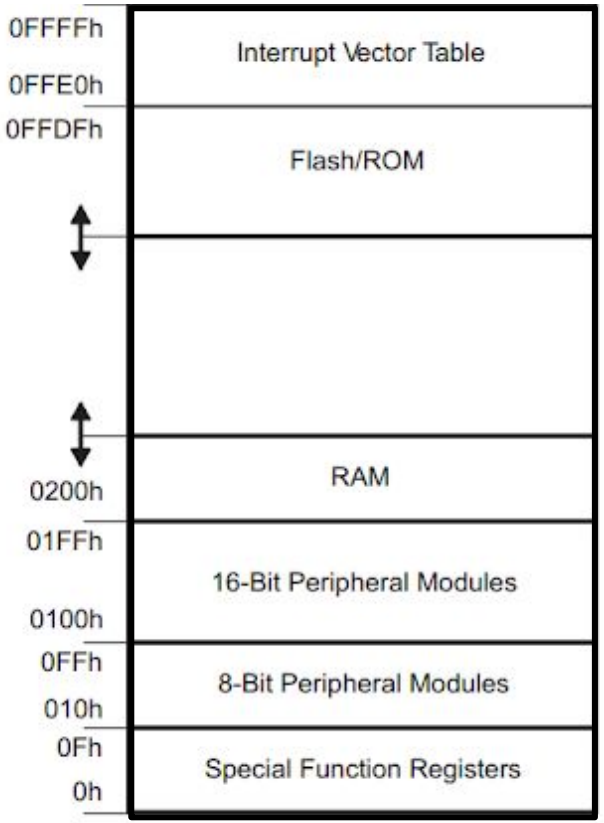
Espacio de memoria unificado: **von Neumann**.
Los registros no están mapeados en este espacio.
Los periféricos si, al principio. Los primeros 256 se direccionan utilizando sólo un byte (idem AVR).

La RAM comienza en 0x0200.

Como es von Neumann (único espacio de memoria) y las direcciones son de 16 bits, y dado que la memoria de programa suele comenzar en 0x1000, FLASH < 60KB.

MSP430X extiende los registros a 20 bits (ver).

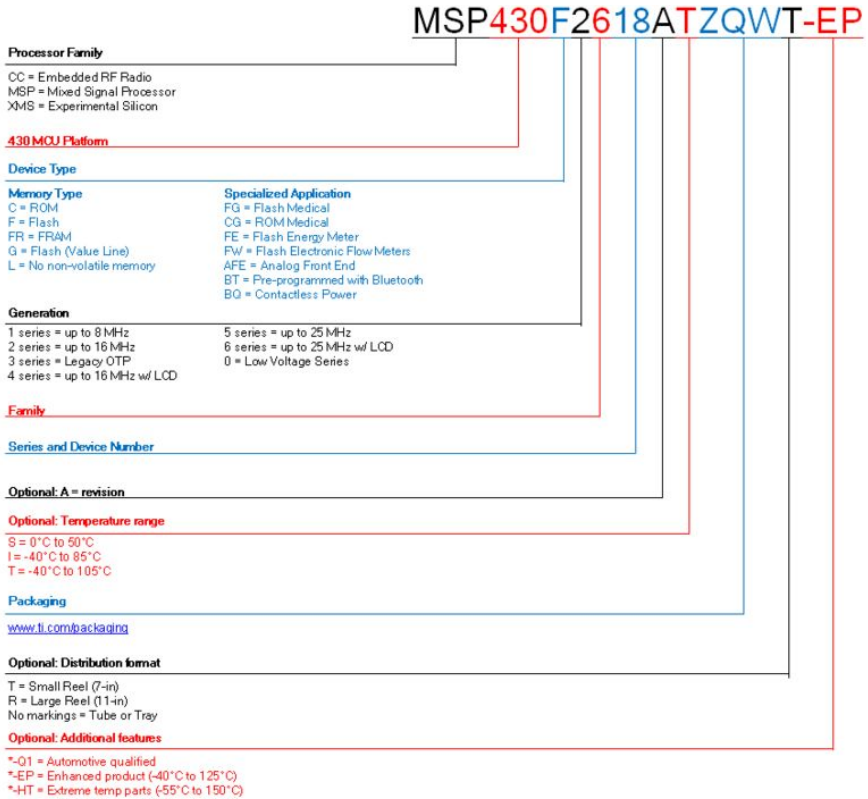
La memoria es direccionable por bytes (**byte-addressable**). Pares consecutivos de bytes se combinan (**little-endian**) para formar **16-bit words**. El último bit del PC es siempre 0.



MSP430 - Módulos y diferentes productos

Memory-mapped **hardware multiplier** peripheral which performs various 16×16+32→33-bit multiply-accumulate operations. MAC DSP
+
Direct Memory Access (DMA) Controller
Advanced Encryption Standard (AES)
Memory Protection Unit (MPU)
Power management module (PMM)
Supply-Voltage Supervisor (SVS)

566 modelos diferentes
Todos con la misma CPU
https://www.ti.com/msp430



ATmega328 US$ 1.50

MSP430F2131
16 MHz MCU with 8KB FLASH, 256B SRAM, 1 Timer - 20-pin plastic
1ku | US$ 0.957 / **ML 10x$850**
https://www.ti.com/lit/ds/symlink/msp430f2131.pdf

MSP430FG6626
25 MHz MCU with 128KB FLASH, 10KB SRAM, 16-bit Sigma-Delta ADC, Dual DAC, DMA, 2 OpAmp, 160 Seg LCD - 100-pin LQFP
1ku | US$ 2.44
https://www.ti.com/lit/ds/symlink/msp430fg6626.pdf
UN SISTEMA DE MEDICIÓN Y CONTROL DE 16-bits COMPLETO

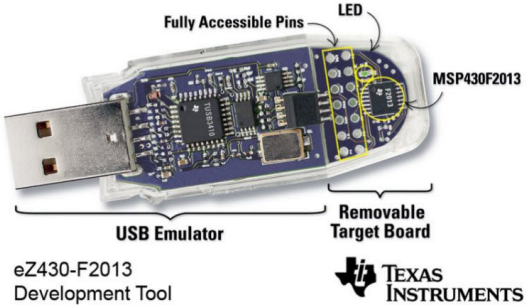
MSP430F2013-EP
Enhanced Product 16-bit Ultra-Low-Power Microcontroller, 2kB Flash, 128B RAM, 16-Bit Sigma-Delta A/D - 16-pin PVQFN
1ku | US$ 3.08
https://www.ti.com/lit/ds/symlink/msp430f2013-ep.pdf
SUPPORTS DEFENSE, AEROSPACE, AND MEDICAL APPLICATIONS
(rango extendido de temperatura, control de calidad, trazabilidad, etc.)

Texas Instruments

MSP430 - Herramientas

UNICO PROVEEDOR Texas Instruments

Kits no muy caros
MSP430 LaunchPad original - ML $4500 (2021)
Los más baratos US$12 en Digikey, no hay chinos



ML n/a (2021)

Toolchains

TI combines a version of its own compiler and tools with its Eclipse-based **Code Composer Studio** IDE ("CCS"). It sells full-featured versions, and offers a free version for download which has a **code size limit of 16 KB**.

$$$
IAR C/C++ compiler and IDE
https://en.wikipedia.org/wiki/IAR_Systems
Empresa sueca especializada en herramientas para embedded

MSPgcc?

ML $6800 (2021)

MPS430FR

FRAM
(ferroelectric, no volátil, exclusiva de TI)
++ menor tiempo de acceso
++ menor consumo
- - menor densidad
- - más cara
https://en.wikipedia.org/wiki/Ferroelectric_RAM

Ultra-low power
CACHE!
32-b multiplier
Low Energy Accelerator (LEA)
co-processor

MSP430FR microcontroller core has performance enhancements that include a **two-way associative cache** with four cache lines of 64-bit line size for better **FRAM** performance.

A **32-bit hardware multiplier** enhances performance for math-intensive operations. It also has a **Low Energy Accelerator (LEA) co-processor** that operates independently of the main MSP430 core.

The LEA can perform a 256 point complex fast Fourier transform (FFT), a finite impulse response (FIR) filter, and matrix multiplication that, according to TI, is up to 40x faster than an Arm® Cortex®-M0+. The LEA improves performance for sensor fusion operations, enhancing images, and processing ultrasonic sensor data. These are all applications where a developer will usually first think of a 32-bit core, not an ultra-low-power 16-bit microcontroller.

Herramientas bajo consumo: LaunchPad KIT supports TI's EnergyTrace™ technology and can connect to a computer using Texas Instruments' EnergyTrace graphical user interface (GUI). This lets developers observe real-time power consumption figures from the MSP430 microcontroller and the application, allowing developers to fine tune their application for power consumption.

MSP430FR2110

16 MHz MCU with 2KB FRAM, 1KB SRAM, Comparator, 8 ch 10-bit ADC, UART/SPI, Timer, RT Clock

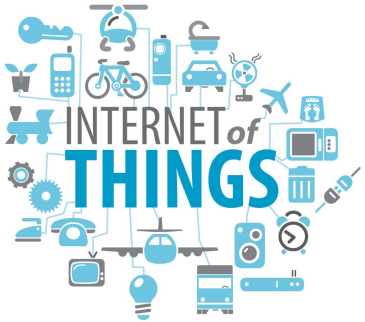
Package TSSOP | 16, VQFN | 24

1ku | **US$ 0.23 (TI.com) <- ??? Digikey US$ 0.96**

https://www.ti.com/lit/ds/symlink/msp430fr2110.pdf

Optimized low-power modes (at 3 V)

– Active mode: 120 µA/MHz
– Standby Real-time clock (RTC) counter: 1 µA
– Shutdown: 34 nA



Applications

- Appliance battery packs
- Smoke and heat detectors
- Door and window sensors
- Lighting sensors
- Power monitoring
- Personal care electronics
- Portable health and fitness devices

Es “demasiado” barato

ATmega328p
US$1.50 (x1000)

CoreMark®
An EEMBC Benchmark

	Compiler	Execution Memory	MHz	CoreMark	CoreMark / MHz↑	DMIPS/MHz
Microchip PIC18F46K20 (4 clocks/instruction cycle)	Microchip MPLAB XC8 v1.32	Code Flash. Data SRAM (Static)	64 / 4	7.23	0.11 x 4	
Atmel AT89C51RE2 in X2 mode	Keil C51 v8.18	Internal RAM & flash (static)	22	2.36	**0.10**	
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** [1] [2]
Texas Instruments MSP430F5438	IAR C/C++ Compiler	Internal flash and RAM (static)	25	19.56	**0.78**	
Texas Instruments MSP430F5438	IAR EW430 V.5.52.1	STACK [3]	25	27.70	**1.11**	

1. https://www.ftdichip.com/Support/Documents/AppNotes/AN_304%20FT900%20Microcontroller%20Benchmark.pdf
2. http://www.ecrostech.com/Other/Resources/Dhrystone.htm
3. http://embedded-funk.net/running-c-function-in-ram-on-msp430-devices/

16 MHz
2KB FRAM
1KB SRAM
16-pin TSSOP
$0.96

MSP430
([MSP430FR2110 datasheet](#) pag.1)
Optimized low-power modes (at 3 V)
– Active mode: 120 µA/MHz
– Standby Real-time clock (RTC) counter: 1 µA
– Shutdown: 34 nA

0.62
CoreMark/MHz

20 MHz
16KB FRAM
1KB SRAM
14-pin SOIC
$0.66

AVR
([ATtiny1604 datasheet](#) pag.474)
- Active mode: 2.4 mA at 3V/10MHz
- Stand-by RTC: 0.7 µA at 3V
- Power-down mode: 0.1µA at 3V

0.54
CoreMark/MHz

MSP430
([MSP430FR2110 datasheet](#) pag.1)
Optimized low-power modes (at 3 V)
– Active mode: 120 µA/MHz
– Standby Real-time clock (RTC) counter: 1 µA
– Shutdown: 34 nA

TEXAS INSTRUMENTS

0.62
CoreMark/MHz

AVR
([ATtiny1604 datasheet](#) pag.474)
- Active mode: 2.4 mA at 3V y 10MHz
- Stand-by RTC: 0.7 µA at 3V
- Power-down mode: 0.1µA at 3V

0.54
CoreMark/MHz

Atmel AVR®

Rendimiento energético:
MSP430
0.62/120 = **5.16** CoreMark/mA @3V
AVR
0.54/(2.4/10) = **2.25** CoreMark/mA @3V

16 MHz
2KB FRAM
1KB SRAM
16-pin TSSOP
$0.96

MSP430
([MSP430FR2110 datasheet](#) pag.1)
Optimized low-power modes (at 3 V)
– Active mode: 120 µA/MHz
– Standby Real-time clock (RTC) counter: 1 µA
– Shutdown: 34 nA

TEXAS INSTRUMENTS

0.62
CoreMark/MHz

PROBLEMA

Ambos operando a 3V y 10 MHz, con una batería de litio de **2000 mAh**, ejecutando cada **10 segundos** una carga equivalente a **1 Coremark** y el resto del tiempo en el modo de más bajo consumo posible.
¿Cuánto dura la batería en cada caso?

20 MHz
16KB FRAM
1KB SRAM
14-pin SOIC
$0.66

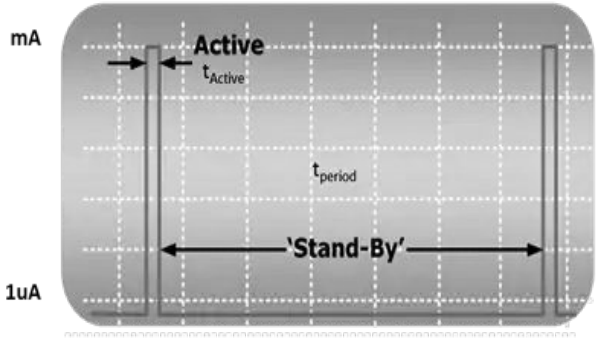
AVR
([ATtiny1604 datasheet](#) pag.474)
- Active mode: 2.4 mA at 3V/10MHz
- Stand-by RTC: 0.7 µA at 3V
- Power-down mode: 0.1µA at 3V

0.54
CoreMark/MHz

Atmel AVR®

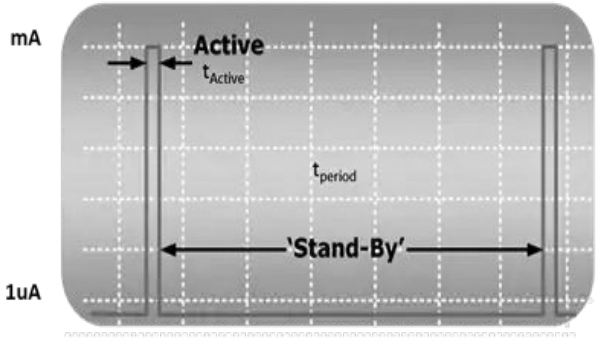
$$Imedia = (Tact * Iact + (T - Tact) * Isb) / T$$

$$Duración = 2000\ mAh / Imedia$$



$$Imedia = (Tact * Iact + (T - Tact) * Isb) / T$$

$$Duración = 2000\ mAh / Imedia$$



MSP430 @10 MHz ejecuta 6.2 Coremark/s
Tact = 1 / 6.2 = 0.161 s
Iact = 1.2 mA
Isb = 0.001 mA
Resulta: Imedia = 0.02 mA y una duración de la batería de **11 años**

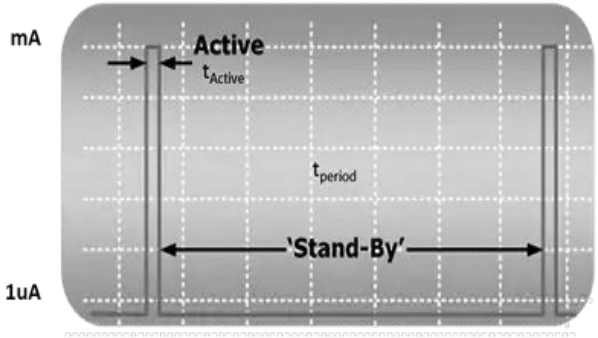
AVR @10 MHz ejecuta 5.4 Coremark/s
Tact = 1 / 5.4 = 0.185 s
Iact = 2.4 mA
Isb = 0.0007 mA
Resulta: Imedia = 0.045 mA y una duración de la batería de **5 años**

11 / 5 = 2.2

La batería dura más del doble en MSP430

$$Imedia = (Tact * Iact + (T - Tact) * Isb) / T$$

$$Duración = 2000\ mAh / Imedia$$



MSP430 @10 MHz ejecuta 6.2 CoreMark/s
Tact = 1 / 6.2 = 0.161 s
Iact = 1.2 mA
Isb = 0.001 mA
Resulta: Imedia = 0.02 mA y una duración de la batería de **11 años**

AVR @10 MHz ejecuta 5.4 CoreMark/s
Tact = 1 / 5.4 = 0.185 s
Iact = 2.4 mA
Isb = 0.0007 mA
Resulta: Imedia = 0.045 mA y una duración de la batería de **5 años**

11 / 5 = 2.2

La batería dura más del doble en MSP430

ENTONCES… PARA QUÉ SIRVEN LOS BENCHMARKS?
Habíamos dicho que en modo activo:
MSP430 = **5.16** CoreMark/mA @3V
AVR = **2.25** CoreMark/mA @3V
Como el consumo en stand-by es despreciable (menos del 1%), **5.16 / 2.25 = 2.3**

Home
https://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html

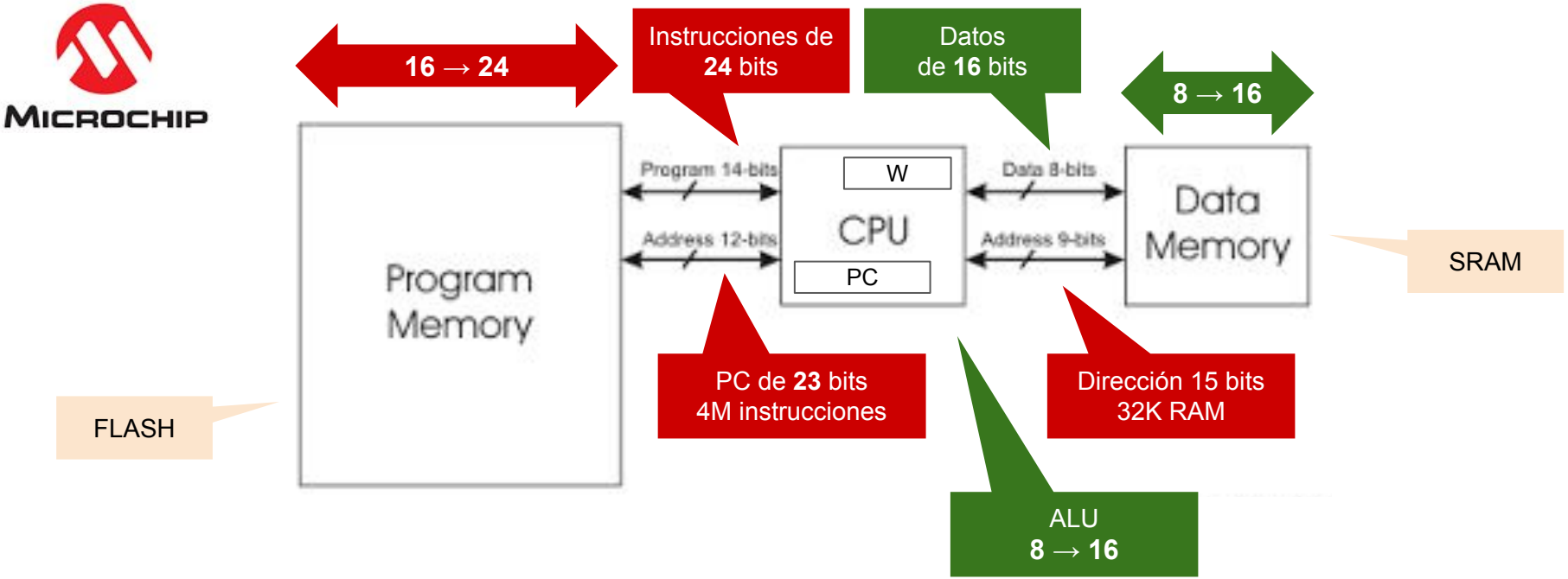
Arquitectura
https://www.ti.com/sc/data/msp/databook/chp8.pdf
https://www.ti.com/sc/docs/products/micro/msp430/userguid/as_5.pdf

Comparación rápida de MSP430FR vs. PIC24
https://www.digikey.es/en/articles/take-advantage-of-16-bit-mcu-performance-and-low-power

MAXQ vs.MSP430
https://www.maximintegrated.com/en/design/technical-documents/app-notes/3/3593.html

Microchip PIC24/dsPIC

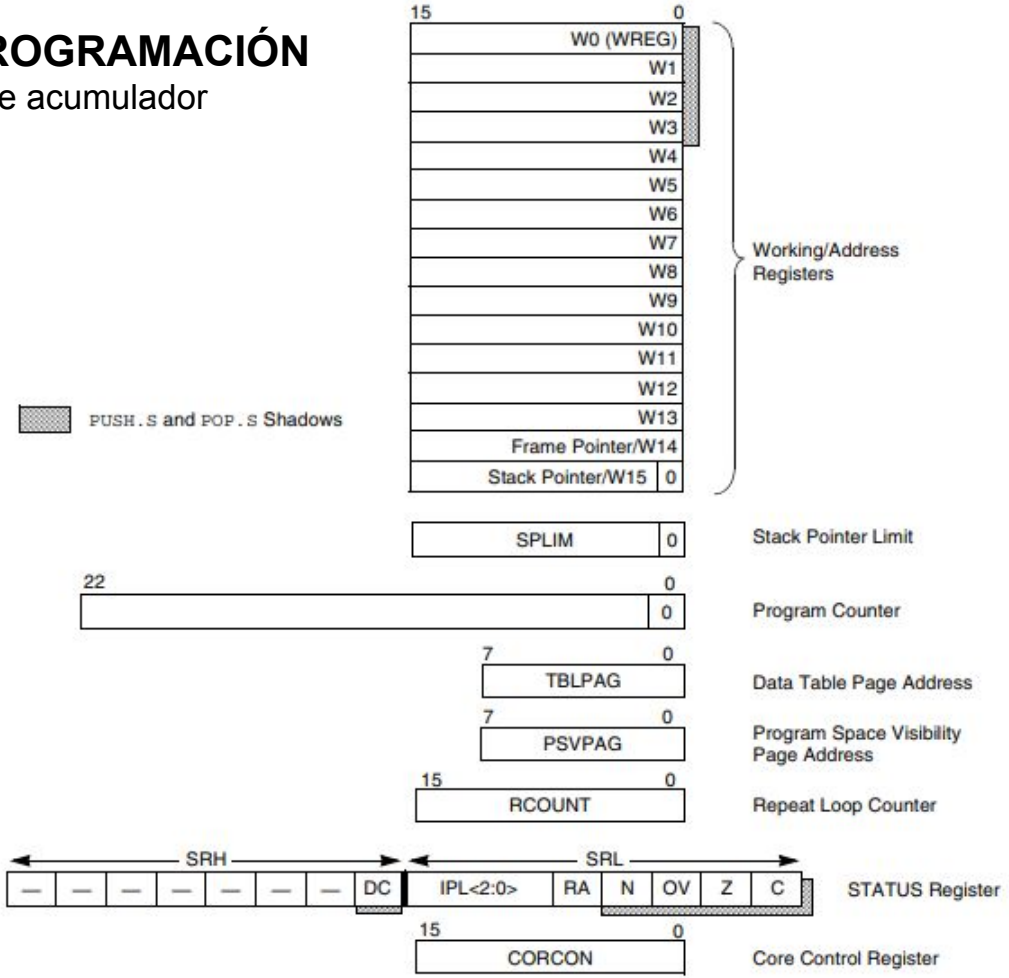
https://microchipdeveloper.com/16bit:architecture
http://ww1.microchip.com/downloads/en/devicedoc/39703a.pdf



	Largo de la instrucción	Program words (ROM)	Registros 8-b (RAM)	Stack
PIC 10/12	12	512	32	2
PIC 16	14	8K (14KB)	368	8
PIC 18	16	2M	16x256	31
PIC 24	24	4M	32K	RAM

PIC24 - EL MODELO DE PROGRAMACIÓN

Muuuy diferente a PIC16 que era de acumulador



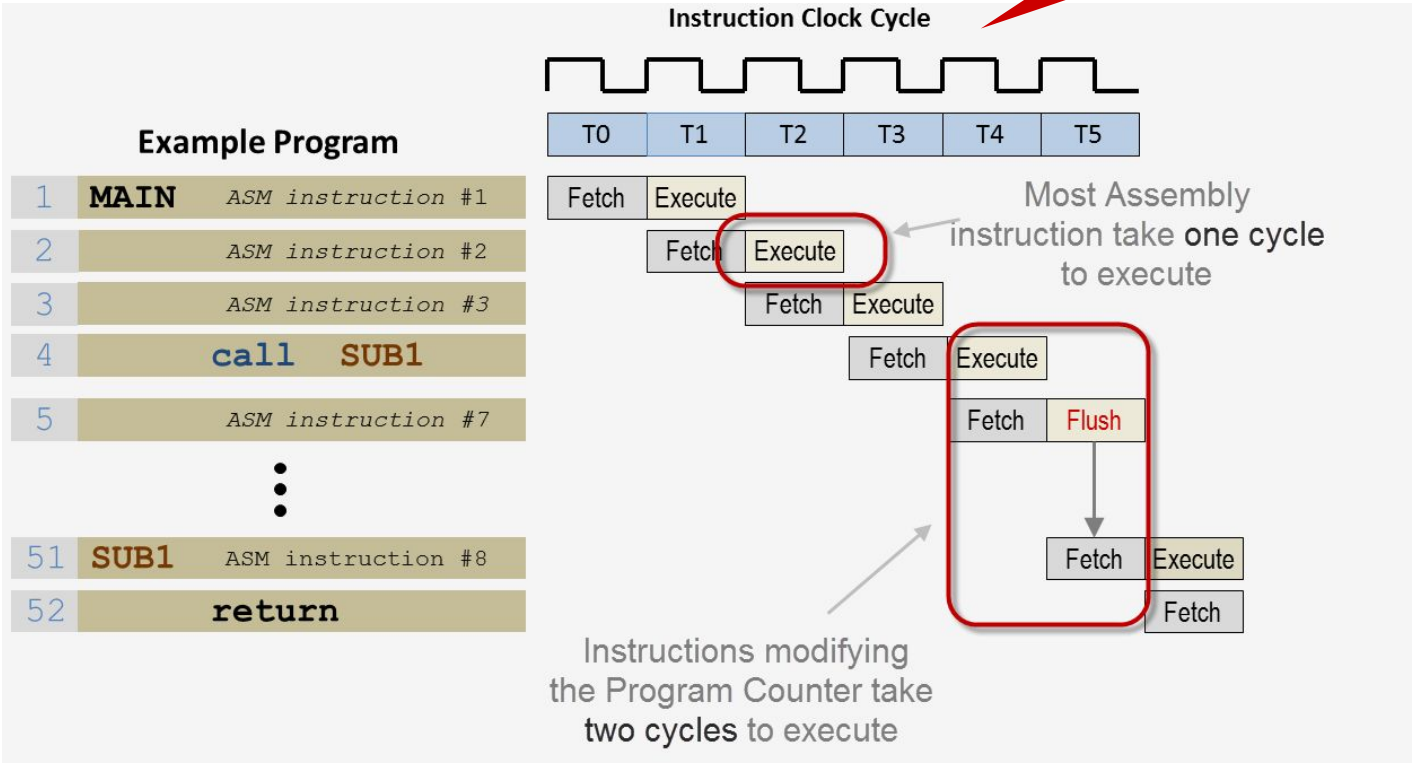
PIC24 - ORGANIZACIÓN Y TECNOLOGÍA

Harvard - Pipeline 2 etapas FETCH-EXEC

Idem PIC16/18 pero clock = inst @ 32 MHz

PIC24 32 MHz

PIC16
64 MHz / 4 = 16 MHz



dsPIC tiene el mismo core que PIC24, con mucho más hardware para implementar instrucciones DSP
http://ww1.microchip.com/downloads/en/DeviceDoc/70165d.pdf

The dsPIC33F device family employs a powerful 16-bit architecture that seamlessly integrates the control features of a Microcontroller (MCU) with the computational capabilities of a Digital Signal Processor (DSP). The resulting functionality is ideal for applications that rely on high-speed, repetitive computations, as well as control. The DSP engine, **dual 40-bit accumulators**, **hardware support for division operations**, **barrel shifter**, **17 x 17 multiplier**, a large array of 16-bit working registers and a wide variety of data addressing modes, together provide the dsPIC33F Central Processing Unit (CPU) with extensive mathematical processing capability…

Poco queda ya del PIC16.

El manejo de memoria es endemoniado. Registros especiales. Puede tener hasta 4M de instrucciones y 32K de datos.

Maxim MAXQ

Transfer-triggered architecture (una única instrucción, implicita obviamente, ejecuta en un ciclo)
https://www.maximintegrated.com/en/design/technical-documents/app-notes/3/3222.html

INTERESANTE: Para demostrar que es mejor que MS430 usaron un benchmark de TI:
https://www.maximintegrated.com/en/design/technical-documents/app-notes/3/3593.html

CoreMark®
An EEMBC Benchmark

	Compiler	Execution Memory	MHz	CoreMark	CoreMark / MHz↑	DMIPS/MHz
Atmel AT89C51RE2 in X2 mode (8051, 6 clocks/instruction cycle)	Keil C51 v8.18	Internal RAM & flash (static)	22	2.36	**0.10**	
Microchip PIC18F46K20 (4 clocks/instruction cycle)	Microchip MPLAB XC8 v1.32	Code Flash. Data SRAM (Static)	64 [/4]	7.23	**0.11 [x4]**	
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** [3]
Texas Instruments MSP430F5438	Code Composer Studio 4.1.2	Internal flash and RAM (static)	18	11.10	**0.62**	**0.31** [1,2]
Microchip PIC24FJ64GA004	gcc 4.0.3	Code Flash. Data SRAM (Static)	32	23.87	**0.75**	**0.50** [1]
MAXQ2000	CrossWorks		20			**0.48** [2]
AMD Am386DX (32-bit)	GCC4.2.4	FPM DRAM 80ns; Stack	40	24.27	**0.61**	

https://www.eembc.org/coremark/scores.php

1 . https://www.ftdichip.com/Support/Documents/AppNotes/AN_304%20FT900%20Microcontroller%20Benchmark.pdf
2. http://www.ecrostech.com/Other/Resources/Dhrystone.htm
3. http://fpgalibre.sourceforge.net/SASE2011/poster_avr.pdf

Más allá de la performance

Perdón, falta traducir...

OVERALL COST

https://predictabledesigns.com/atmega-versus-stm32-which-microcontroller-is-best-for-your-application/

“Several, somewhat conflicting areas need to be considered when choosing a given microcontroller for an application.

Among them are overall cost, not just of the microcontroller itself, but the total system cost.

For example, if a microcontroller already has a built-in peripheral that removes the need for an external one, then that’s a much better way of approaching a design.

On the other hand, choosing an expensive microcontroller, with lots of extra peripherals that won’t be used, simply because it has one that is actually needed is not a wise choice.

Another thing to consider is the upward migration path of a given microcontroller family. It is important to consider the availability of tools, both hardware and software, and the ease of writing and debugging the application firmware…”

LA IMPORTANCIA DE LOS PERIFÉRICOS

https://hackaday.com/2016/06/28/avr-vs-pic-round-223-fight/

“I wish people understood how powerful the integrated peripherals are. Too many high level language programmers on this site. Code is their hammer. Moar speed, moar memory, moar better.

I recently redesigned a customer’s circuit from a 100MIPS software controlled four channel PWM motor controller to a 4MIPS 8bit PIC using it’s integrated modules. I was even able to add a bunch of stuff the previous controller couldn’t handle because it was so over-worked.

I did a similar thing last year. Previous engineer was trying to measure two fast pulses by simply polling the inputs with a very fast processor. I used the PICs capture compare module instead of code and was able to eliminate both ~$5 processors on the board. That alone reduced BOM cost by 50%, but since it consumed much less power and took up less space, required fewer support components, etc… the final BOM cost reduction was more like 80%...”

Herramientas de MÁS alto nivel

Formas diferentes de subir...

El Proyecto Arduino

https://www.arduino.cc/
Arduino is an open-source electronics platform based on easy-to-use hardware and software.
UTIL PARA PROTOTIPADO RAPIDO. Ej: probar un sensor.

Arduino programming language
https://www.arduino.cc/reference/en/
(basado en Wiring)

“Wiring is an open-source programming framework for microcontrollers” http://wiring.org.co/

Arduino Software IDE
https://www.arduino.cc/en/Main/Software
(basado en Processing)

“Processing is a flexible software sketchbook and a language for learning how to code within the context of the **visual arts**”
https://processing.org/

Programación Orientada a Objetos

C++ Programación, Electrónica plan nuevo
Materias de Ingeniería en Computación
MicroPython

PYTHON

Sistemas operativos de tiempo real (RTOS)

Mejor llamados “Embedded Programming Platforms”. Un tema interesante para los que quieren seguir investigando en la capa de aplicación. Más en la asignatura optativa Sistemas Operativos y Redes...

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

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).